U N I V E R Z I T E T U B E O G R A D U
Prirodno-matematički fakultet

P R I L O G
T E O R I J I P R E V O D J E N J A
P R O G R A M S K I H J E Z I K A
I
K O N S T R U I S A N J U P R E V O D I L A C A

Doktorska teza

Vojislav Stojković

Beograd 1981.

PREDGOVOR

Od nastanka prvih programskih jezika i njihovih prevodilaca pa sve do današnjih dana, izmedju ostalih, sledeća tri problema su u teoriji i praksi programskih jezika i prevodilaca neprekidno aktuelna:

1. Utvrdjivanje sintaksne ispravnosti naredbe odnosno programa.
2. Raznovrsnost prevodilaca.
3. Relativno česte izmene postojećih i razvijanje novih programskih jezika i prevodilaca.

Definisanjem niza sintaksnih pravila i njihovim poštovanjem dobija se sintaksno ispravan program. Sintaksno ispravan program se može pomoću prevodioca prevesti sa izvornog jezika na ciljni jezik. Program koji sadrži sintaksne greške ne može se pomoću prevodioca korektno prevesti sa izvornog jezika na ciljni jezik.

Program se neposredno može izvršiti na računaru isključivo ako je napisan na mašinskom jeziku dotičnog računara u suprotnom program se pomoću prevodioca mora prevesti na mašinski jezik. Znači, za svaki programski jezik različit od mašinskog jezika postoji prevodilac ili niz prevodilaca koji ga u jednom ili više poteza prevode na mašinski jezik.

Razvoj računarskih nauka, zahtevi korisnika računara, nove oblasti primene računara itd. uzrokuju manje ili veće modifikacije - poboljšanja postojećih programskih jezika kao i nastanak novih programskih jezika. Svaka promena programskog jezika zahteva modifikaciju odgovarajućeg prevodioca. Svaka nova generacija računara, u slučaju promene skupa mašinskih naredbi, zahteva modifikaciju postojećeg ili čak konstruisanje novog prevodioca.

Navedeni problemi se u principu mogu rešiti na sledeći način:

1. Uvode se metajezici za opisivanje sintakse i semantike programskih jezika. Dati opisi omogućavaju strogo, a relativno jednostavno definisanje i razumevanje programskih jezika što je od posebnog značaja kako za primenjene programere - korisnike jezika tako i za sistemske programere - konstruktore prevodilaca.
2. Konstruiše se univerzalni prevodilac čiji su ulazni podaci opisi sintakse i semantike izvornog jezika, a rezultat je prevodilac izvornog jezika.
3. Izmene programskog jezika zahtevaju samo modifikaciju opisa programskog jezika, a ne univerzalnog prevodioca. Sav posao oko stvaranja novog programskog jezika sastoji se u definisanju njegove sintakse i semantike. Pomoću univerzalnog prevodioca jednostavno se generiše odgovarajući prevodilac. Nova generacija računara zahteva konstruisanje novog univerzalnog prevodioca i modifikaciju semantike postojećih programskih jezika što je ipak daleko lakše od konstruisanja niza novih posebnih prevodilaca.

Izloženi način rešavanja prethodno navedena tri problema naziva se sintaksno-orijentisano /vodjeno/ prevodjenje ili kraće SO /SV/ -prevodjenje.

Cheatham i Sattley [17] su 1964. godine prvi izložili filozofiju i tehniku sintaksno-orijentisanog prevodjenja.

Ovaj Rad je doprinos daljem razvoju i primeni sintaksno-orijentisanog prevodjenja.

Rad se sastoji iz sledećih 6 glava:

1. Programski jezici
2. Prevodioci
3. Beleženje sintakse programskih jezika
4. Sintaksno-orijentisano prevodjenje
5. Programska realizacija šeme sintaksno-orijentisanog prevodjenja
6. Istorijske napomene o programskim jezicima i prevodiocima

Literature i

Programa univerzalnog kompilatora.

Na početku prve glave date su neformalne definicije: uopšte jezika, prirodnog, veštačkog, formalnog i programskog jezika.

Programski jezik je znakovni sistem koji se koristi za opisivanje procesa rešavanja zadataka - algoritama.

Programski jezici se dele na:

1. /Formalne/ algoritamske jezike,
2. /Formalne/ nealgoritamske jezike i
3. Ne sasvim formalizovane znakovne sisteme. [104]

Algoritamski jezici pružaju mogućnost jednoznačnog opisivanja algoritama u kompaktnom, preglednom i lako razumljivom obliku, bliskom matematičkom zapisu.

Nealgoritamski jezici pružaju mogućnost opisivanja procesa rešavanja zadataka bez eksplicitnog navodjenja operacija koje se trebaju izvršiti. Naziv nealgoritamski i ako je uobičajen u literaturi i praksi nije najsrećnije odabran jer može da izazove zabunu i pogrešan zaključak. Možda bi prikladniji naziv bio funkcionalni jezici.

Algoritamski jezici se dele na:

1. Mašinske jezike,
2. Mašinski-orijentisane /simboličke, niže programske/ jezike,
3. Proceduralno-orijentisane /više programske/ jezike i
4. Problemski-orijentisane jezike.

Navedene klasifikacije programskih odnosno algoritamskih jezika, kao i svaka druga klasifikacija, su u izvesnom stepenu uslovne. Postoje jezici koji imaju osobine jezika različite vrste. Neki mašinski-orijentisani jezici dozvoljavaju upotrebu aritmetičkih i/ili logičkih izraza što je inače osobina proceduralno-orijentisanih i problemski-orijentisanih jezika. Pojedini proceduralno-orijentisani jezici su veoma specijalizovani pa su bliski problemski-orijentisanim jezicima. Na kraju, postoje višeciljni - univerzalni algoritamski jezici. Takodje, mnogi autori ne prave razliku izmedju programskih i algoritamskih jezika ili izmedju algoritamskih i nealgoritamskih jezika polazeći od činjenice da je svaki programski jezik u suštini algoritamski itd. [105]

U nastavku glave razmatra se gramatička struktura programskih prvenstveno proceduralno-orijentisanih jezika.

Programski jezik definisan je:

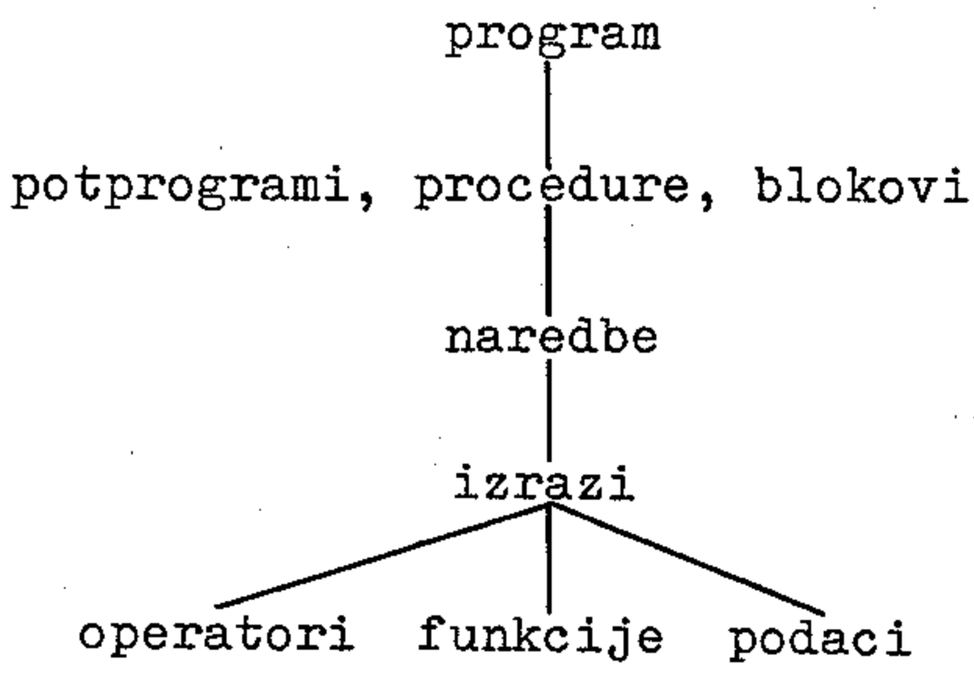
1. Azbukom,
2. Sistemom pravila obrazovanja sintaksnih konstrukcija /sintaksom/ jezika i

3. Sistemom pravila značenja sintaksnih konstrukcija /semantikom/ jezika.

Od slova azbuke po odredjenim gramatičkim pravilima konstruišu se niske slova - sintaksne konstrukcije jezika: reči, izrazi, rečenice /naredbe/ i tekstovi /programi/.

Sintaksne konstrukcije jezika nalaze se u relaciji medjusobne podčinjenosti i obrazuju hijerarhijsku strukturu jezika.

Proceduralno-orijentisani jezici imaju sledeću hijerarhijsku strukturu:



U razmatranje, najčešće uporedo, uzeti su svi danas značajniji proceduralno-orijentisani jezici: FORTRAN, ALGOL, BASIC, PASCAL, COBOL, PL/I, APL, LISP, SNOBOL, BLISS i C.

Izlaganje je propraćeno većim brojem primera.

Na početku druge glave data je definicija prevodilaca, a zatim je izvršena njihova klasifikacija.

Prevodioci se dele na:

- asemblere,
- kompilatore i
- generatore.

Ulazni jezik asemblera je mašinski-orijentisan jezik, kompilatora - proceduralno-orijentisan jezik, a generatora - problemski-orijentisan jezik.

Dalje se razmatra struktura kompilatora.

Kompilator se u principu sastoji iz:

- leksičkog analizatora,
- sintaksnog analizatora,
- generatora kôda i eventualno
- optimizatora kôda.

Sledi detaljan opis procesa kompilacije.

Da bi izlaganje bilo potpuno, ukratko se razmatraju i interpretatori.

Realizacija programa na računaru u principu se sastoji iz dva procesa:
- prevodjenja izvornog programa i
- izvršavanja prevedenog programa.

Proces prevodjenja izvornog programa i proces izvršavanja prevedenog programa mogu vremenski da budu:
- razdvojeni,
- povezani.

Interpretator istovremeno - vremenski povezano izvršava:
1. Prevodjenje naredbe izvornog programa i
2. Realizaciju /interpretaciju/ mašinskih naredbi /ili naredbi na medjujeziku/ koje predstavljaju dobijeni prevod naredbe izvornog programa.

Interpretator se obično sastoji iz tri dela:
- pripremnog,
- izvršnog i
- upravljačkog.

Struktura interpretatora ilustrovana je na primeru BASIC-interpretatora.

Druga glava se završava opisom najpoznatijih kompilatora:
- ALGOL-ALFA,
- ALGOL-60 TA-2,
- C,
- FORTRAN H i
- BLISS-11.

Navedeni kompilatori su u svoje vreme imali značajnu ulogu i predstavljali su korak napred u teoriji i praksi konstruisanja kompilatora.

U trećoj glavi su razmotreni razni načini beleženja sintakse programskih jezika koji se mogu definisati kontekstno-slobodnim gramatikama.

Obradjene su:
- notacija Chomsky-og,

- Backus*-ova notacija,
- razne modifikacije Backus-ove notacije,
- Ingerman-ova notacija,
- Iverson-ova notacija,
- Obrnuta notacija,
- Berkhardt-ova notacija,
- Lukas-ova notacija,
- van Wijngaarden-ova notacija i
- Grafička notacija.

Takodje su izloženi:

- originalan** način opisa Backus-ove notacije pomoću Backus-ove notacije i
- jedno poboljšanje Modifikovane Backus-ove notacije.

Navedene notacije ilustrovane su na primeru jezika jednostavnih aritmetičkih izraza.

Četvrta glava je posvećena sintaksno-orijentisanom /SO/ prevodjenju. U ovoj glavi je izloženo više originalnih** rezultata.

Definisani su sledeći pojmovi:

- šema SO-prevodjenja i SO-prevodjenje,
- biunivoko, identičko, inverzno, prosto, složeno itd. SO-prevodjenje,
+ uslovi kompozicije i jednakosti SO-prevodjenja,
+ potkresano drvo prevodjenja itd.

Dokazane su, a označene sa + i formulisane su sledeće teoreme:

- U opštem slučaju prevodjenje nije homomorfizam.
- Prevodjenje izraza iz prefiksnog zapisa u postfiksni zapis i obratno je jednoznačno.
- Oblast definisanosti i skup vrednosti SO-prevodjenja su KS-jezici.
- Problem ekvivalentnosti dva SO-prevodjenja je nerešiv.
+ Kompozicija SO-prevodjenja je SO-prevodjenje.
+ Ne važi zakon komutacije za SO-prevodjenja.
+ Važi zakon asocijacije za SO-prevodjenja.
+ Inverzno prevodjenje biunivokog SO-prevodjenja je jedinstveno.
+ Kompozicija konačno mnogo:

---

* Uobičajeni su još i nazivi: Backus-ova normalna forma, Backus-Naur-ova forma ili samo BNF.

** Dobijeni rezultati su po mišljenju autora originalni.

složenih,
složenih i prostih,
prostih i složenih
SO-prevodjenja je složeno SO-prevodjenje.*

+ Kompozicija konačno mnogo prostih SO-prevodjenja je prosto SO-prevodjenje.
- Inverzno prevodjenje prostog SO-prevodjenja je prosto SO-prevodjenje.
- Desno-linearno SO-prevodjenje je prosto SO-prevodjenje.

Zatim, više osobina desno-linearnih SO-prevodjenja.

+ Potkresano drvo izvodjenja i potkresano drvo prevodjenja su izomorfna i jednako obeležena.
+ Drvo izvodjenja i drvo prevodjenja su različita.

Obradjeno je SO-prevodjenje jednog proceduralno-orijentisanog BASIC-olikog jezika na mašinski-orijentisan jezik nastavr.og računara. Navedeni jezici odabrani su zbog toga jer se koriste u nastavi iz programiranja.

Dalje je izvršena prirodna - dalja modifikacija šeme SO-prevodjenja koja takodje predstavlja originalan rezultat.

Dat je algoritam svodjenja šeme SO-prevodjenja na normalan oblik. Dokazana je ekvivalentnost nesvedene i svedene šeme SO-prevodjenja.

Na kraju, dato je jedno uopštenje šeme SO-prevodjenja.

Cetvrta glava se završava primenom SO-prevodjenja u teoriji formalnih gramatika i jezika. Navedeni primeri i teoreme su takodje originalni.

U petoj glavi je obradjena programska realizacija šeme SO-prevodjenja. Izloženi su principi konstruisanja i rada univerzalnog kompilatora.

Univerzalni kompilator se sastoji iz:

- sintaksnog punioca,
- semantičkog punioca i
- jezgra.

Navedeni način opisa semantike jezika je takodje originalan i ima posebnu vrednost jer se može dalje razvijati.

Univerzalni kompilator napisan je na FORTRAN-jeziku i realizovan je na računaru PDP 11/70.

---

* Izuzev u jednom specifičnom slučaju.

Šesta glava sadrži niz značajnih i interesantnih istorijskih podataka o programskim jezicima i prevodiocima.

Na kraju Rada navedeni su:

1. Literatura na engleskom, ruskom i srpsko-hrvatskom jeziku i
2. Program univerzalnog kompilatora.

Najsrdačnije se zahvaljujem mentoru, dr Nedeljku Parezanoviću, redovnom profesoru Prirodnomatematičkog fakulteta, u Beogradu na vodjenju i pomoći u izradi Rada.

Takodje se zahvaljujem dr Dušanu Velaševiću, docentu Elektrotehničkog fakulteta, u Beogradu i dr Žarku Mijajloviću, docentu Prirodnomatematičkog fakulteta, u Beogradu za pregled Rada i korisne sugestije i savete koje su mi dali.

S A D R Ž A J

1. PROGRAMSKI JEZICI

1.1. Jezik kao objekat izučavanja matematike

Jezik je sredstvo za predstavljanje i prenos informacija, komunikaciju izmedju dva ili više korisnika. Korisnici jezika su ljudi i/ili mašine.

1.1.1. Prirodni i veštački jezici

Postoje prirodni i veštački jezici.

Prirodni jezik je sredstvo za komunikaciju medju ljudima. Primeri prirodnih jezika su: srpsko-hrvatski jezik, ruski jezik, engleski jezik itd. Za svaki prirodan jezik postoji skup ljudi koji ga razume, govori i razmišlja na njemu, jednom rečju koristi ga. Prirodni jezici su univerzalni jer se pomoću njih može predstaviti proizvoljna informacija. Medjutim, prirodni jezici nisu podesni za predstavljanje specifičnih na primer matematičkih informacija. U takvim slučajevima, prirodni jezici su se pokazali nedovoljno strogim jer dopuštaju izvesnu nejednoznačnost i neprecizност. Iz navedenih razloga izmišljeni su veštački jezici za predstavljanje informacija, po pravilu relativno uske klase, sa osobinama koje odražavaju specifičnosti takvih informacija. Primeri veštačkih jezika su: jezici za beleženje matematičkih, fizičkih ili hemijskih formula, jezik eksperanto itd.

Veštački jezici se koriste zajedno sa prirodnim jezicima uzajamno se dopunjujući.

Kvalitativno nova etapa u razvoju veštačkih jezika nastaje pojavom programskih jezika. Specifičnost programskih jezika sastoji se u tome što su prvenstveno namenjeni za komunikaciju izmedju čoveka i mašine /računara/.

1.1.2. Aspekti jezika

Pri izučavanju jezika neophodno je razmotriti sledeća dva aspekta [100]:
- na koji se način informacije mogu preneti jezikom i
- kakve se informacije mogu preneti jezikom.

Navedeni aspekti jezika nazivaju se:
- plan izražavanja i
- plan sadržavanja.

Plan izražavanja je niz fizičkih signala /simbola, glasova itd./ koji predstavljaju odredjenu informaciju.

Plan sadržavanja je prenos sredstvima jezika različitih informacija.

Koristeći matematičku terminologiju jezik se u najopštijim crtama može definisati kao objekat koji se sastoji iz:
- skupa tekstova koji karakterišu plan izražavanja,
- skupa značenja koji karakterišu plan sadržavanja i
- preslikavanja koje svakom elementu skupa tekstova pridružuje jedan ili više elemenata skupa značenja.

Razmotrimo detaljnije navedena dva aspekta jezika.

U slučaju prirodnih jezika skup značenja je u velikoj meri neodredjen i teško se može strogo definisati. Pod skupom značenja podrazumeva se skup informacija predatih čoveku, odnosno skup informacija koje je čovek primio.

U slučaju kad je jezik namenjen za komunikaciju izmedju čoveka i mašine stvari stoje drugačije. Uočimo mašinski jezik tj. jezik neposredno prihvatljiv i razumljiv za neku mašinu. Sredstva mašinskog jezika moraju dozvoljavati predavanje poruka mašini koja uzrokuju potpuno odredjena dejstva mašine. Plan sadržavanja mašinskog jezika odgovara mogučnostima mašine u smislu da su obuhvaćena sva dejstva mašine. Pod skupom značenja mašinskog jezika podrazumeva se skup svih mogućih operacija mašine*. Uočimo algoritamski jezik tj. jezik nezavisan od operacija konkretne mašine. Pod skupom značenja algoritamskog jezika podrazumeva se skup svih mogućih operacija odgovarajuće apstraktne mašine*. Da bi se operacije apstraktne mašine mogle izvršiti na konkretnoj računskoj mašini, neophodno je da se operacije apstraktne mašine mogu interpretirati operacijama konkretne računske mašine.

---

* Ovo je jedna od mogućih interpretacija.

Za razliku od skupa značenja, koji se u slučaju prirodnih jezika teško može strogo definisati i proučavati, skup tekstova se relativno lako može definisati. Elementi skupa tekstova su rečenice jezika koje podležu kako u slučaju prirodnih tako i u slučaju veštačkih jezika strogim pravilima konstruisanja. Ova pravila omogućavaju izvodjenje rečenica ili proveru pripadnosti rečenica skupu tekstova.

Uočimo preslikavanje skupa tekstova u skup značenja. Primetimo pre svega, da je stvarna upotreba jezika moguća tada i samo tada ako je funkcija koja izvršava preslikavanje skupa tekstova u skup značenja i obratno efektivno izračunljiva. Drugačije rečeno, mora da postoji postupak neposrednog odredjivanja za dati tekst odgovarajućeg značenja i obratno za dato značenje odgovarajućeg teksta.

U slučaju prirodnih jezika, prethodno rečeno znači da čovek koji zna jezik može:
- razumeti saopštenu poruku,
- formulisati /iskazati/ odredjenu poruku.

Izučavanje svojstava funkcije koja preslikava skup tekstova u skup značenja matematičkim metodama je izuzetno složeno. Pre svega, ništa se nezna o prirodi funkcije osim da je izračunljiva. Čovek podsvesno izračunava vrednosti ove funkcije pa ne može ništa da kaže o algoritmu koji pri tom koristi. Svaki čovek ima vlastite skupove tekstova i značenja pa prema tome i vlastitu funkciju preslikavanja. Individualni jezik tj. jezik pojedinca zavisi od niza faktora kao što su na primer: stepen poznavanja jezika, opšte obrazovanje, specijalnost itd. Velike probleme izučavanju prirodnih jezika formalnim metodama prave osobine prirodnih jezika kao što su na primer: nejednoznačnost, emocionalna obojenost, idiomi i fraze, narečja, promenljivost i druge. Većina navedenih problema ne postoji u slučaju veštačkih jezika. Za razliku od prirodnih jezika, koji su se u velikoj meri razvijali stihijski veštački jezici nastajali su planski sa strogo unapred postavljenim ciljevima. Tvorci veštačkih jezika uvek su težili da jezik bude što prihvatljiviji za čoveka i ne tako težak za formalno prevodjenje na mašinski jezik.

Tekst je niska simbola. Simboli su elementi skupa koji se naziva azbuka. Osnovni problem je utvrditi da li data niska simbola čini ispravan tekst. Skup pravila za utvrdjivanje ispravnosti teksta naziva se sintaksa jezika. Pošto se utvrdi da je tekst

ispravan nameće se pitanje značenja teksta. Skup pravila za odredjivanje značenja teksta naziva se semantika jezika.

1.1.3. Prevodjenje jezika

Prevodjenje jezika A na jezik B je preslikavanje skupa tekstova jezika A u skup tekstova jezika B pri kojem se čuva značenje tekstova. U opštem slučaju, prevodjenje jednog jezika na drugi jezik, nije uvek moguće*. Neophodan uslov postojanja prevodjenja jezika A na jezik B je jednakost skupova značenja ili u krajnjem slučaju sadržavanje skupa značenja jezika A u skup značenja jezika B. Tako na primer besmisleno je govoriti o prevodjenju proizvoljnog teksta sa srpsko-hrvatskog jezika na jezik nota ili programa napisanog na FORTRAN-jeziku na mašinski jezik specijalizovane računske mašine koja recimo ne sadrži osnovne aritmetičke operacije.

Prevodjenje može biti:

- semantičko ili
- formalno.

Označimo sa:

T_A - skup tekstova jezika A,
T_B - skup tekstova jezika B,
S - skup značenja jezika A i B,
f_A - preslikavanje skupa T_A u skup S,
f_B - preslikavanje skupa S u skup T_B.

Preslikavanja f_A i f_B su jednoznačna.

Semantičko prevodjenje sastoji se iz:

- Neka je $t \in T_A$. Izračunava se $f_A(t)$.
- Neka je $s=f_A(t) \in S$. Izračunava se $f_B(s)$. Neka je $t'=f_B(s) \in T_B$.

Prevod teksta t je tekst t'.

Skup značenja S i preslikavanja f_A i f_B se teško formalno definišu, posebno još u slučaju prirodnih jezika. Semantičko prevodjenje je složeno da bi se realizovalo pomoću računara. Mnogo je prihvatljiviji formalan način prevodjenja. Formalan način prevodjenja se sastoji u eksplicitnom zadavanju algoritma izračunavanja preslikavanja I_{AB} jednakog superpoziciji preslikavanja f_A i f_B; $I_{AB}=f_A \circ f_B$. Preslikavanje I_{AB} je definisano na skupu T_A, a uzima vrednosti iz skupa T_B, tj. važi: $I_{AB}(t)=f_B(f_A(t))=f_B(s)=t'$.

Preslikavanje I_{AB} omogućava "zaobilaženje" pri prevodjenju teško formalizovanog skupa značenja S.

---

* Church-ova teza tvrdi suprotno ali pritom isključivo podrazumeva formalno prevodjenje. Zato, ne postoji protivurečnost.

1.2. Klasifikacija programskih jezika

Programski jezik je znakovni sistem - notacija koja se koristi za opisivanje procesa rešavanja zadataka prvenstveno na računaru. Programski jezici se dele na:

- algoritamske jezike,
- nealgoritamske jezike i
- ne sasvim formalizovane znakovne sisteme.

1.2.1. Algoritamski jezici

Algoritamski jezici se dele na:

- mašinske jezike,
- mašinski-orijentisane (simboličke ili niže programske) jezike,
- proceduralno-orijentisane (više programske) jezike i
- problemski-orijentisane jezike.

1.2.1.1. Mašinski jezici

Morfemi su reči binarne azbuke. Od morfema se obrazuju naredbe mašinskog jezika.

Osnovni morfem naredbe mašinskog jezika je kôd operacije. Mesto i dužina kôda operacije naredbe mašinskog jezika su strogo odredjeni. Mesto, broj i dužina drugih morfema zavisi od kôda operacije.

Drugi morfemi naredbe mašinskog jezika su adrese. Postoje razne vrste adresa kao na primer obeležje, adresa indeks registra, adresa operanda itd.

Reč je osnovna adresiva memorijska jedinica. Reč sadrži naredbu ili operand. Zavisno od računara reč se sastoji iz 8, 16, 32 itd. ćelija.

Prikaz je obeležje ili reč.

Program na mašinskom jeziku sastoji se iz niza prikaza.

Naredba programa na mašinskom jeziku nalazi se u memorijskom registru čija je adresa jednaka obeležju naredbe. Odavde sledi da obeležja naredbi moraju biti medjusobno različita.

Specifičnost mašinskih jezika je da svakom kôdu operacije odgovara više elementarnih-mikro operacija. Pri izvršavanju naredbe mašinskog jezika ulazni podaci elementarnih operacija uzimaju se iz registra ukazanim odredjenim adresama, rezultati se smeštaju u registre ukazane adresama.

Pri pisanju prikaza morfemi se obično ne pišu u binarnoj azbuci {0,1} već se kôdiraju brojevima brojnog sistema osnove veće od 2. Najčešće se koristi oktalni ili heksadekadni brojni sistem. Na primer, u slučaju oktalnog brojnog sistema jedna cifra označava grupu od tri binarne cifre.

Algoritam izvršavanja programa na mašinskom jeziku je jednostavan i odražava proces rada računara.

Uopšteno govoreći algoritam izvršavanja programa na mašinskom jeziku sastoji se u sledećem:

- vrednost veličine bn-nazovimo je brojač naredbe,uzima se za obeležje naredbe;
- nalazi se prikaz koji sadrži odgovarajuće obeležje;
- naredba koja odgovara nadjenom prikazu dodeljuje se veličini rn tzv. registru naredbe;
- iz rn se izdvaja kôd operacije koji odredjuje vrstu naredbe i uopšte operacije koja treba da se izvrši;
- po izvršenju naredbe, prethodno opisani ciklus se ponavlja.

Različitost računara ne dozvoljava detaljniji opis algoritma izvršavanja programa na mašinskom jeziku.

Danas se veoma retko pišu programi na mašinskom jeziku, jer to zahteva od programera pamćenje velikog broja detalja bez kojih je nemoguće sastaviti program.

Najčešće se programira na mašinskom jeziku u prvoj etapi razrade operativnog sistema, ako iz nekog razloga nije moguće koristiti računar koji ima razvijen programski sistem.

1.2.1.2. Mašinski-orijentisani jezici

Mašinski-orijentisani (simbolički ili niži programski) jezici omogućavaju da se na simbolički način ukažu sredstva koja treba da izvrše odredjene algoritamske korake. Mnemotehničke skraćenice se koriste za označavanje kôda operacije i adresa.

Primer

Sledeći niz nula i jedinica:

0110 001110 010101

je naredba nekog mašinskog jezika.

0110 je kôd operacije sabiranja,
001110 je adresa operanda X i
010101 je adresa operanda Y.

SAB X,Y

je odgovarajuća naredba nekog mašinski-orijentisanog jezika.

SAB je mnemokôd operacije sabiranja, a
X i Y su simboličke adrese operanada X i Y.

Mnogi mašinski-orijentisani jezici imaju makro mogućnosti tj. dozvoljavaju upotrebu makro-naredbi. Makro-naredbe se razvijaju - prevode nizom naredbi mašinski-orijentisanog jezika pre prevodjenja na mašinski jezik.

Primer

Pretpostavimo da neki mašinski-orijentisan jezik ne sadrži naredbu sabiranja sadržaja dva memorijska registra recimo:

SAB X,Y

a sadrži naredbe:

- prenosa sadržaja odredjenog memorijskog registra u akumulator, recimo:

MUA X

- sabiranja sadržaja akumulatora i odredjenog memorijskog registra sa smeštanjem rezultata u akumulator, recimo:

SAB Y

- prenosa sadržaja akumulatora u odredjen memorijski registar, recimo:

AUM X

Koristeći makro mogućnosti mašinski-orijentisanog jezika možemo definisati sledeću makro-naredbu:

```
MAKRO     SABl X,Y
          MUA  X
          SAB  Y
          AUM  X
          KRAJMAKRO
```

Prva naredba definiše ime SABl i fiktivne argumente X i Y makro-naredbe. Sledeće tri naredbe čine telo makro-naredbe. Poslednja naredba označava kraj definicije makro-naredbe.

Po izvršenom definisanju, makro-naredba se koristi kao naredba simboličkog jezika, na primer:

```
SABl A,B
```

gde su A i B realni parametri.

Makro-procesor će po definiciji razviti datu makro-naredbu u niz naredbi mašinski-orijentisanog jezika:

```
MUA  A
SAB  B
AUM  A
```

1.2.1.3. Proceduralno-orijentisani jezici

Proceduralno-orijentisani (viši programski) jezici se koriste za opisivanje algoritama za rešavanje zadataka široke klase problema. Najpoznatiji proceduralno-orijentisani jezici su FORTRAN, BASIC, PASCAL, COBOL, PL/1, LISP, SNOBOL, ALGOL, APL, C, BLISS itd.

Programiranje na mašinski-orijentisanom jeziku je lakše i jednostavnije od programiranja na mašinskom jeziku. To je zato, što su brojni kodovi operatora i operanada zamenjeni mnemotehničkim-simboličkim oznakama. Uvodjenjem makro-naredbi, još više je olakšano programiranje na mašinski-orijentisanom jeziku. Medjutim, i pored svega, programiranje na mašinski-orijentisanom jeziku praćeno je nizom teškoća. Programer mora da zna i vodi računa o nizu detalja vezanih za organizaciju, arhitekturu i rad konkretnog računara. On mora sam da prevodi složenije operacije

i strukture podataka u niz jednostavnih operacija i primitivnih tipova podataka koje je računar sam u stanju neposredno da prihvati i izvrši. Programer, takodje, mora da se usredsredi gde i u kom obliku mu se nalaze naredbe i podaci. Da bi se prebrodile navedene teškoće izmišljeni su proceduralno-orijentisani jezici. U principu svaki proceduralno orijentisani jezik omogućava izvršavanje algoritama na jedan mnogo prirodniji - uobičajeniji način nego što je to bilo u slučaju mašinski-orijentisanog jezika. Programer više ne mora, skoro ništa da zna, o funkcionisanju računara. Sabiranje dve veličine, umesto niza naredbi, prikazuje se u obliku jedne naredbe A+B.

Danas se nalaze u upotrebi više stotina različitih proceduralno-orijentisanih jezika. Oni se medjusobno razlikuju:

1) u bliskosti prirodnim - matematičkim jezicima, odnosno udaljenosti od mašinskog jezika računara;

2) u prilagodjenosti izražavanju algoritama za rešavanje zadataka.

Sada ćemo ukratko navesti neke odlike proceduralno-orijentisanih jezika po kojima se bitno razlikuju od mašinski-orijentisanih jezika.

1. Jednostavnost

Program na proceduralno-orijentisanom jeziku u principu se jednostavnije piše, čita, dokazuje nego odgovarajući program na mašinski-orijentisanom jeziku. To je zato što su izražajne mogućnosti proceduralno-orijentisanih jezika daleko veće od izražajnih mogućnosti mašinski-orijentisanih jezika. Proceduralno-orijentisani jezici se relativno jednostavno prihvataju, uče i primenjuju. Izmedju pojedinačnih proceduralno-orijentisanih jezika često postoje velike razlike. Smatra se da je moćniji onaj programski jezik koji ima raznovrsnije operatore i bogatije strukture podataka.

2. Prirodnost

Razumljivost proceduralno-orijentisanih jezika zavisi od jednostavnosti izražavanja - zapisa algoritama na njima. Svaki proceduralno-orijentisan jezik pripada klasi programskih jezika podesnih za rešavanje problema odredjene vrste.

Primer

Programski jezici FORTRAN i ALGOL prvenstveno su namenjeni za rešavanje numeričkih zadataka.

Programski jezici BASIC i PASCAL podesni su za obuku iz programiranja.

Programski jezici COBOL i RPG služe za poslovnu obradu podataka.

Programski jezici LISP i SNOBOL koriste se za rad sa znakovnim podacima itd.

Treba naglasiti da su proceduralno-orijentisani jezici univerzalni jezici i mogu se koristiti sa manjom ili većom efikasnošću za rešavanje svih vrsta problema.

Primer

Izraz: $\frac{-b+\sqrt{b^2-4ac}}{2a}$

piše se u FORTRAN-u:

```
(-B+SQRT(B**2-4*A*C))/2*A
```

a u LISP-u:

```
(QUOTIENT(PLUS(MINUS B)(SQRT(DIFFERENCE(TIMES B B)
(TIMES 4 A C))))(TIMES 2 A))
```

3. Prenosivost

Za većinu proceduralno-orijentisanih jezika propisana je standardna verzija. To je učinjeno da bi se zaveo red i izbegle nepotrebne razlike jednog istog programskog jezika.

Programi napisani na proceduralno-orijentisanim jezicima su skoro nezavisni od tehničkih karakteristika računara i u principu se bez većih izmena mogu izvršavati na svakom računaru.

Primer

U programskom jeziku FORTRAN realizovanom na računaru sa memorijskim registrima dužine 32. ćelije dozvoljeno je neposredno korišćenje celobrojnih konstanti reda 10^9. Medjutim, "isti" programski jezik FORTRAN realizovan na računaru sa memorijskim

registrima dužine 16 ćelija dozvoljava neposredno korišćenje celobrojnih konstanti reda 10^4.

Očigledno, FORTRAN-program koji sadrži celobrojne konstante reda 10^n, $5 \leq n \leq 9$ mora se modifikovati pre izvršavanja na 16-to bitnom računaru.

Prethodan problem elegantno je rešen u programskom jeziku PL/I. Programski jezik PL/I sadrži opisnu naredbu za deklarisanje proizvoljne tačnosti konstanti.

4. Efikasnost

Efikasnost proceduralno-orijentisanih jezika podjednako zavisi od realizacije i načina upotrebe.

Jedanput je važnije da proces prevodjenja bude što kraći, a drugi put da je prevedeni - izvršni program što efikasniji.

Sledeće mogućnosti programskih jezika bitno utiču na njihovu pouzdanost i efikasnost upotrebe:

- strukture podataka,
- delokrug naredbi,
- kontrolne konstrukcije i
- potprogrami.

1.2.1.4. Problemski-orijentisani jezici

Problemski-orijentisani jezici su posebno razvijeni programski jezici za opisivanje procesa rešavanja specifične klase zadataka kao što je na primer linearna algebra, statistika, linearno programiranje, mrežno planiranje, teorija grafova itd.

1.2.2. Nealgoritamski jezici

Nealgoritamski jezici omogućavaju zapisivanje metoda za rešavanje zadataka bez tačnog navodjenja niza operacija koje se trebaju izvršiti. Primer nealgoritamskog jezika je jezik parametarskog zadavanja šema. Tu se proces obrade informacija zadaje kao skup parova operacija pri čemu prva operacija predstavlja neki uslov, a druga dejstvo po obradi informacija. Pri rešavanju zadatka izvršavaju se operacije čiji je uslov ispunjen.

Sve dok se uslov ne ispuni, njemu odgovarajuće operacije se ne izvršavaju.

Ako je istovremeno ispunjeno više uslova, odgovarajuće operacije se mogu izvršavati bilo istovremeno, bilo u proizvoljnom redosledu.

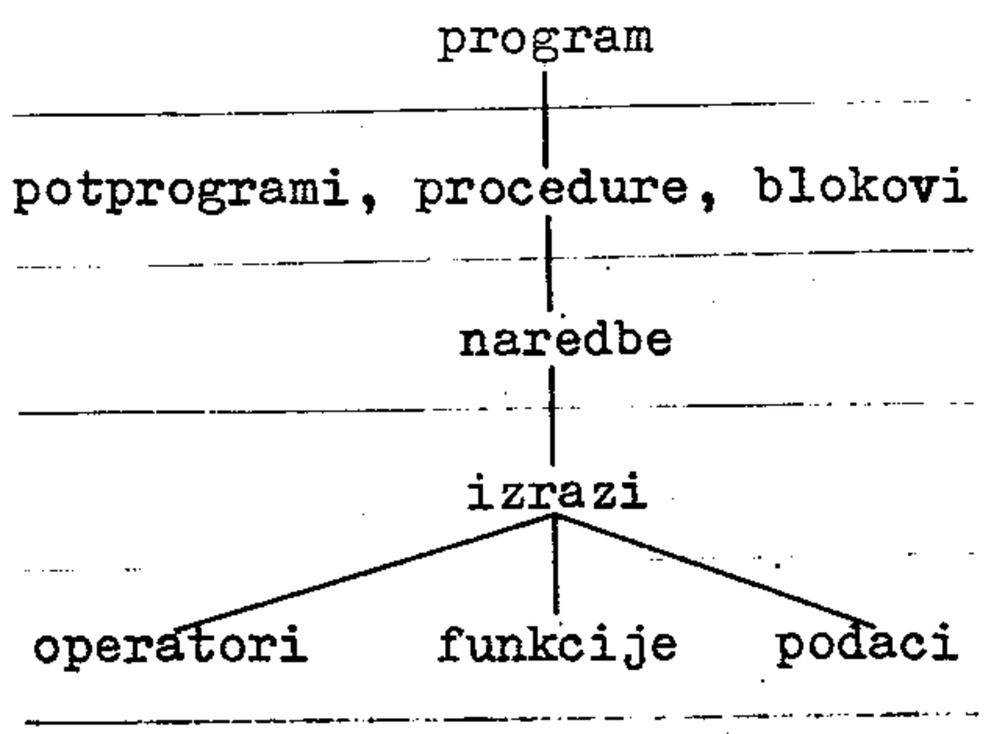
1.2.3. Ne sasvim formalizovani znakovni sistemi

Primer ne sasvim formalizovanog znakovnog sistema je algoritamska šema. Algoritamska šema je grafički zapis algoritma kod kojeg se pojedini delovi algoritma predstavljaju u vidu blokova (pravougaonika, rombova, krugova itd.) unutar kojih se na prirodnom jeziku (na primer srpskohrvatskom) objašnjava značenje algoritamskog koraka. Veze izmedju blokova (algoritamskih koraka) predstavljaju se linijama koje označavaju prenos upravljanja.

1.3. Struktura programskih jezika

1.3.1. Hijerarhijska struktura programskih jezika

Programski jezik je notacija za zapis niza operacija koje će se izvršiti nad podacima. Operacije i podaci se mogu grupisati u hijerarhijsku strukturu oblika drveta.



Na vrhu hijerarhijske strukture nalazi se program. Program je osnovna izvršna jedinica.

Sledeće mesto u hijerarhijiskoj strukturi zauzimaju potprogrami, procedure i blokovi. Ove jedinice imaju odredjeno značenje, mogu se posebno, nezavisno od programa, prevoditi ali se obično ne mogu same izvršavati. Potprogrami i procedure razlikuju se od blokova jer se mogu pozivati iz programa.

Program, potprogrami, procedure i blokovi sastavljeni su iz naredbi. Naredbe su dalje sastavljene iz izraza, a ovi dalje iz operatora, funkcija i podataka.

Svaka konstrukcija programskog jezika ima:

- logičko (apstraktno) značenje i
- realizaciju (implementaciju).

Logičko značenje je shvaćanje konstrukcije u matematičkom smislu. Realizacija je reprezentacija (predstavljanje) konstrukcija unutar računara.

Primer

Ime promenljive je reč, obrazovana u skladu sa odredjenim sintaksnim pravilima, a koristi se za označavanje promenljive.

U mašinskoj interpretaciji imenu promenljive odgovara adresa memorijske lokacije. Sadržaj memorijske lokacije odgovara vrednosti promenljive.

1.3.2. Leksička i sintaksna struktura programskih jezika

Primer

Uočimo nisku A+B*C. Niska se sastoji iz 5 simbola: A,+,B,*,C. Navedeni simboli mogu se grupisati na razne načine. Najčešće se data niska shvata kao izraz koji se sastoji iz tri promenljive razdvojene znacima operacija i interpretira se kao:

(A+B)*C

ili

A+(B*C)

Koja je od navedenih interpretacija tačna zavisi od definicije prioriteta operatora.

Leksička struktura programskog jezika definiše načine grupisanja simbola jezika u leksičke konstrukcije.

Osnovne leksičke konstrukcije su identifikatori i operatori.

Sintaksna struktura programskog jezika definiše načine grupisanja leksičkih konstrukcija u složenije konstrukcije tzv. sintaksne kategorije.

1.3.2.1. Azbuka

Simboli koji se koriste u programskom jeziku čine skup znakova (karaktera), tj. azbuku programskog jezika.

Azbuka mašinskog jezika sastoji se iz dva slova: Ø i 1.

U sledećoj tabeli* navedene su neke karakteristike azbuka najpoznatijih programskih jezika.

Programski jezik	Broj simbola	Slova	Cifre	Specijalni znaci
FORTRAN	48	26	10	12
ALGOL-60	144	52	10	52
BASIC	59	26	10	23
PASCAL	120	52	10	58
COBOL	51	26	10	15
PL/I	60	29	10	21
LISP	68	45	10	13
SNOBOL	62	26	10	26

Primer

Azbuka programskog jezika FORTRAN sastoji se iz:

- 26 velikih štampanih slova engleske azbuke A,B,...,Z;
- 10 cifara dekadnog brojnog sistema 0,1,2,...,9;
- 11 specijalnih znakova: razmak,+,-,*,/,**,=,(,), zarez, tačka i Ø .

Azbuka programskog jezika ALGOL-60 pored uobičajenih simbola sadrži mala štampana slova engleske ili ruske azbuke a,b,...,z i službene reči for, begin itd.

1.3.2.2. Leksičke konstrukcije

Niska koja predstavlja program može se izdeliti na niz podniski koje se nazivaju leksičke konstrukcije.

* Moguća su izvesna odstupanja.

Na koji način će se izvršiti deljenje, odnosno koja će niska biti proglašena za leksičku konstrukciju zavisi od definicije programskog jezika.

Kod većine programskih jezika sledeće celine smatraju se leksičkim konstrukcijama:

- konstante tj. 1,2.3,-4.56E+7 itd.
- imena (promenljivih, podprograma itd.), tj. I,JOT,PP itd.
- znaci operatora, tj. +,-,*,/,↑,**,div,mod itd.
- službene reči tj. LET, IF, GOTO, STOP, END itd.
- interpunkcijski znaci tj. zapete, zarez, tačka zarez itd.

1.3.2.3. Sintaksne konstrukcije

Na osnovu odredjenih sintaksnih pravila, grupisanjem leksičkih konstrukcija dobijaju se:

- elementarne sintaksne konstrukcije i
- složene sintaksne konstrukcije.

Primer

Elementarne sintaksne konstrukcije FORTRAN-jezika su:

- konstante,
- promenljive,
- nizovi i
- izrazi.

Složene sintaksne konstrukcije FORTRAN-jezika su:

- naredbe,
- potprogrami i
- programi.

Elementarna sintaksna konstrukcija ima odredjeno značenje ali sama za sebe ne egzistira, tj. ne može proizvesti nikakvu akciju na računaru.

Složena sintaksna konstrukcija ima odredjeno značenje, sama za sebe egzistira i može proizvesti odredjenu akciju na računaru.

1.3.2.4. Načini pisanja programa

Fizičko predstavljanje programa utiče na složenost leksičke analize programa.

Programski jezici FORTRAN, COBOL zahtevaju poštovanje odredjenih pravila za pisanje programa.

Primer

Naredbe FORTRAN-programa moraju se pisati u sledećem formatu:

1) od 1. do 5. kolone piše se obeležje naredbe.

2) od 7. do 72. kolone piše se naredba.

3) ako naredba sadrži više od 66 simbola i ne može da stane u jednom redu, može se pisati u više redova pri čemu svaki naredni red mora u 6. koloni da sadrži priznak produžetka naredbe.

4) od 73. do 80. kolone piše se tekst koji služi za identifikaciju programa i označavanje redosleda naredbi.

Fiksan format naredbi podseća na mašinsko-orijentisane jezike, kod kojih pozicija osnovne konstrukcije nosi informaciju o značenju konstrukcije. Na primer, osnovna konstrukcija koja počinje od prve kolone je obeležje naredbe itd.

U savremenim programskim jezicima teži se slobodnom formatu naredbi.

Primer

U programskom jeziku PASCAL dozvoljeno je pisati više naredbi u jednom programskom redu, odnosno jednu naredbu u više programskih redova. Ne postoje nikakva ograničenja u pogledu formata pisanja naredbi, već naprotiv u cilju povećanja čitljivosti programa preporučuje se što fleksibilnije pisanje naredbi.

Slobodan format naredbi utiče na smanjenje broja sintaksnih grešaka. Povećana je čitljivost i vidljivost strukture programa. Simbol razmak (blanko) različito se tretira u programskim jezicima. Tako na primer, u programskim jezicima FORTRAN, ALGOL, BASIC i PASCAL simbol razmak nije od značaja izuzev kad se nalazi u okviru literala. Razmak se može slobodno dodati ili

izostaviti iz programa. Jedina svrha mu je povećanje čitljivosti programa. Medjutim, u programskim jezicima LISP i SNOBOL simbol: razmak je od značaja i služi kao razdvajač.

1.3.3. Osnovni (elementarni) podaci

Osnovni gradjevinski materijal programskih jezika čine osnovni (elementarni) podaci. Na skupu osnovnih podataka definišu se standardni operatori. Grupisanjem osnovnih podataka dobijaju se složeni podaci - strukture podataka.

Postoji velika razlika izmedju programskih jezika u pogledu skupova osnovnih podataka. Ipak, kod većine programskih jezika susreću se:

1. Numerički (brojni) podaci. Numerički podaci obuhvataju celobrojne, realne, kompleksne brojeve obične i dvostruke (višestruke) tačnosti.

2. Logički podaci.

3. Znakovni (azbučni) podaci.

Jedni programski jezici dozvoljavaju upotrebu isključivo znakova, a drugi niski znakova. Dužina niski je obično unapred ograničene dužine.

4. Pokazivači (pointeri).

Pokazivači su osnovni podaci koji pokazuju na tražene osnovne podatke. Opšta ideja je da se, recimo, mogu koristiti naredbe:

P:= adresa(X) i

Y:= sadržaj(P)

Pomoću pokazivača mogu se obrazovati složene strukture podataka.

5. Obeležja. Obeležje je podatak čija je vrednost naredba ili pozicija programa kojoj se može pristupiti.

Primer

U sledećoj tabeli prikazani su osnovni tipovi podataka koji postoje u najpoznatijim programskim jezicima.

FORTRAN
celobrojni, realni, kompleksni, logički, dvostruke tačnosti;
ALGOL
celobrojni, realni, logički;
BASIC
celobrojni, realni, znakovni;
PASCAL
celobrojni, realni, logički, znakovni;
COBOL
alfabetski, alfanumerički, numerički, alfanumerički za izdavanje i numerički za izdavanje;
PL/I
brojevi (sa četiri pravougaona svojstva)
- režim : realan ili kompleksan;
- skala : fiksan zarez ili pokretni zarez;
- osnova : decimalna ili binarna;
- tačnost : broj cifara, niske znakova i bitova, pokazivači, nizovi, razlike, obeležja, ulazi, formati, zadaci, nailasci.

1.3.4. Strukture podataka

Strukturu podataka čine:
- skup osnovnih podataka i ranije obrazovanih struktura podataka i
- skup strukturnih relacija definisanih na prethodnom skupu.

Strukture podataka se najčešće definišu rekursivno.

Primer

Lista je po definiciji prazna ili se sastoji iz osnovnog podatka za kojim sledi lista. Osnovne operacije sa listama su: umetanje, izostavljanje i zamena elemenata liste i utvrdjivanje da li je dati podatak elemenat liste.

Primer

Drvo T je skup elemenata (čvorova) za koje važi sledeća strukturna relacija:

a)Odabrani čvor č naziva se koren drveta T.

b)Preostali čvorovi mogu se razdeliti u k$\geq \emptyset$ poddrveta $T_1, T_2, \ldots, T_k$ tako da je koren $č_i$ drveta T_i potomak korena č drveta T.

Najpoznatije i najčešće korišćene strukture podataka su: nizovi, redovi, stogovi, niske i grafovi.

Postoji mnoštvo operatora predvidjenih za rad sa strukturama podataka. U principu, svi ovi operatori mogu se razvrstati u tri grupe:

1) konstruktore, čiji je zadatak obrazovanje strukture podataka,

2) destruktore, koji služe za oslobadjanje struktura podataka za prijem novih podataka i

3) selektore, čiji je zadatak da omoguće što jednostavniji i brži pristup podacima.

Elementi strukture podataka okarakterisani su:

- imenom,
- vrednošću i
- mestom (pozicijom).

Kada je dato ime, možemo se pozivati bilo na vrednost, bilo na poziciju elementa strukture.

Primer

Neka je A ime niza. Onda je A(I) ime i-tog člana niza.

U slučaju naredbe B=A(I), ime A(I) poziva se na vrednost A(I).

U slučaju naredbe A(I)=B, ime A(I) poziva se na poziciju i-tog člana niza A.

Realizacija (implementacija) strukture podataka je aproksimacija logičke definicije strukture podataka. Svaki računar mora da sadrži tehnička ograničenja kao što su na primer: veličina brojeva, nizova, niski itd.

Ovim ograničenjima obično se ne pridaje pažnja pri logičkoj definiciji strukture podataka i programskog jezika uopšte.

1.3.4.1. Nizovi

Niz je skup elemenata istog tipa, uredjenih u k-dimenzionalnu pravougaonu strukturu.

Mera rastojanja duž svake dimenzije naziva se indeks niza. Elementi niza rasporedjeni su po celobrojnim tačkama počev od neke donje granice pa sve do odgovarajuće gornje granice duž svake dimenzije.

Primer

Standardni FORTRAN-jezik dopušta upotrebu jedno, dvo i trodimenzionalnih nizova. Donja granica za svaku dimenziju je 1.

BASIC-jezik dozvoljava korišćenje isključivo jednodimenzionalnih i dvodimenzionalnih nizova. Donja granica indeksa je Ø ili 1.

Savremeni programski jezici dozvoljavaju proizvoljnu donju i gornju granicu svakog od indeksa niza kao i proizvoljan broj dimenzija niza. Navedene proizvoljnosti su na izvestan način ograničene konkretnom realizacijom jezika.

Primer

U PASCAL-jeziku dozvoljeno je pisati:

```
TYPE KOCKA=ARRAY [BROJ] OF ARRAY [BROJ] OF ARRAY [BROJ] OF REAL
      :
VAR  TELO : KOCKA;
```

Elementi niza KOCKA su recimo:
KOCKA[1], KOCKA[2][3], KOCKA[3][2][1] itd.

Element niza je u potpunosti odredjen imenom niza i vrednostima indeksa. Donja i gornja granica indeksa niza, kao i ukupan broj elemenata niza mogu biti poznati:

- u vreme prevodjenja programa,
- u vreme izvršavanja programa.

U prvom slučaju niz je fiksne, a u drugom slučaju promenljive dužine.

1.3.4.1.1. Jednodimenzionalni nizovi fiksne dužine

Jednodimenzionalni nizovi čija je dužina poznata u vreme prevodjenja programa, obično se implementiraju kao niz uzastopnih memorijskih registara. Ako jedan podatak treba k memorijskih registara za registrovanje vrednosti onda će se i-ti element niza A registrovati počev od memorijske lokacije l odredjene formulom:

$$l = \begin{matrix}\text{početna memorijska lokacija} \\ \text{registrovanja niza}\end{matrix} + k * \left(i - \begin{matrix}\text{donja granica} \\ \text{indeksa niza}\end{matrix}\right)$$

Opisivač podataka jednodimenzionalnog niza mora da sadrži sledeće informacije o nizu:

1) vrsta podatka (tj. jednodimenzionalan niz),

2) vrsta elemenata (celobrojni, realni itd.),

3) broj potrebnih memorijskih lokacija za registrovanje jednog elementa,

4) donja granica indeksa i

5) gornja granica indeksa.

Ako su navedene informacije konstante, poznate su u vreme prevodjenja programa i čuvaju se u tabeli simbola.

1.3.4.1.2. Višedimenzionalni nizovi fiksne dužine

Jednostavnosti radi uočimo dvodimenzionalni niz - matricu A dimenzije 3*2.

A(1,1)	A(1,2)
A(2,1)	A(2,2)
A(3,1)	A(3,2)

Dvodimenzionalni niz A registruje se u operativnoj memoriji računara na jedan od sledeća dva načina:

- red po red ili
- kolona po kolona.

U PL/I-jeziku dvodimenzionalni niz A registruje se red po red, tj. na sledeći način:

A(1,1)
A(1,2) prvi red

A(2,1)
A(2,2) — drugi red

A(3,1)
A(3,2) — treći red

U FORTRAN-jeziku dvodimenzionalni niz A registruje se kolona po kolona, tj. na sledeći način:

A(1,1)
A(2,1) — prva kolona
A(3,1)

A(1,2)
A(2,2) — druga kolona
A(3,2)

Druga moguća implementacija dvodimenzionalnog niza je u obliku vektora. U ovoj implementaciji red se predstavlja u obliku jednodimenzionalnog niza. Dvodimenzionalan niz se predstavlja u obliku jednodimenzionalnog niza pokazivača nizova redova.

Ako se dovdimenzionalni niz dimenzija m i n registruje red po red u operativnoj memoriji računara (i,j) elemenat registrovaće se počev od memorijske lokacije l odredjene formulom:

l = OSNOVA+((i-D)*m+j-D)

Slično ako se dvodimenzionalni niz dimenzija m i n registruje - kolona po kolona u operativnoj memoriji računara (i,j) elemenat registrovaće se počev od memorijske lokacije l odredjene formulom:

l = OSNOVA+k*((j-D)*n+i-D)

U prethodnim formulama korišćene su sledeće oznake:

OSNOVA - početna memorijska lokacija registrovanja niza;

k - broj memorijskih registara potrebnih za registrovanje vrednosti jednog člana niza;

D - donja granica indeksa niza.

Navedene formule neposredno se uopštavaju za proizvoljan broj dimenzija. Ovo uopštavanje se ne navodi zbog dužine.

Posmatrajući prethodne formule lako se uočava sledeća zakonitost:

- U slučaju registrovanja niza red po red, desni indeksi elemenata niza se brže menjaju (slično brojevima na kilometarsatu automobila)

- Pri registrovanju niza kolona po kolona, brže se menjaju levi indeksi elemenata niza.

1.3.4.1.3. Nizovi promenljive dužine

Neki programski jezici kao na primer ALGOL-68, PL/I, LISP, SNOBOL itd. ali ne FORTRAN, BASIC i PASCAL dozvoljavaju dinamičko dimenzionisanje nizova, tj. specificiranje dužine nizova u vreme izvršavanja programa. Programski jezik APL čak dozvoljava promenu broja dimenzija niza u vreme izvršavanja programa.

1.3.4.2. Slogovi

Važnu klasu strukture podataka čine slogovi.

Primer

Tipičan primer sloga je poštanska adresa. Pretpostavimo da u adresaru imamo adrese 100 poznanika oblika:

MR. DUŠAN TOŠIĆ
MARKA CREŠKOVIĆA 5
11090 KALUDJERICA

U PL/I-jeziku može se deklarisati sledeći prigodan niz slogova:

```
1  POZNANICI (100),
     2  OSOBA,
          3  TITULA CHARACTER (3),
          3  IME CHARACTER (10),
          3  PREZIME CHARACTER (15)
   2  ADRESA,
          3  ULICA CHARACTER (20),
          3  BROJ FIXED DECIMAL (3),
          3  POSTA FIXED DECIMAL (5),
          3  MESTO CHARACTER (15);
```

U PASCAL-jeziku prethodan niz slogova definisao bi se na sledeći način:

```
CONST N=20;
TYPE  NISKA=PACKED ARRAY [1..N] OF CHAR;
      NASLOV=  RECORD
               TITULA    : NISKA;
               IME       : NISKA;
               PREZIME   : NISKA;
               END;
SEDIŠTE= RECORD
               ULICA     : NISKA;
               BROJ      : INTEGER;
               POŠTA     : INTEGER;
               MESTO     : NISKA
               END;
LICA=    RECORD
               OSOBA     : NASLOV;
               ADRESA    : SEDIŠTE;
               END;
POZNANICI= ARRAY[1..100] OF LICA;
```

Slogovi se implementiraju kao memorijski blok. Da bi se izračunao potreban memorijski prostor za registrovanje sloga i vrednost funkcije pristupa odredjenom polju sloga moraju da budu poznati dužina i udaljenost svakog polja. Dužina polja je broj memorijskih registara potrebnih za registrovanje odgovarajućeg podatka.

Udaljenost polja je zbir dužina prethodnih polja sadržanih u polju višeg nivoa. Tako na primer polje TITULA ima udaljenost Ø od početka polja OSOBA, tj. polja prvog višeg nivoa koje ga sadrži. Dalje, polje IME je udaljeno 3, a polje PREZIME je udaljeno 13 jedinica od početka polja OSOBA.

Dužina polja sastavljenog iz više podpolja jednaka je zbiru dužina podpolja.

Opisivač podataka strukture sloga mora da sadrži sledeće informacije o slogu:

1) udaljenost svakog polja,

2) širinu svakog polja koje je za sebe struktura podataka,

3) vrste podataka sadržanih u slogu.

1.3.4.3. Strukture podataka koje se redje susreću

Postoji niz specifičnih struktura podataka. One se susreću kod pojedinih programskih jezika i za sada nisu od nekog većeg značaja ali su sa teorijskog gledišta interesantne. To su:

- niske znakova (BASIC, SNOBOL, PL/I)
- skupovi (PASCAL)
- liste (LISP)
- stogovi (PL/I) .

1.3.4.3.1. Niske znakova

Niske znakova su ustvari jednodimenzionalni nizovi čiji su elementi znakovi (karakteri). Susreću se u programskim jezicima BASIC, SNOBOL itd. U BASIC-jezicu su ograničene dužine (recimo 256 znakova) dok su u SNOBOL-jeziku proizvoljne dužine. Proizvoljna (ne unapred odredjena) dužina niske znakova omogućena je dinamičkom raspodelom memorijskog prostora. U PL/I-jeziku niske znakova mogu da budu promenljive dužine ali u unapred odredjenim granicama.

1.3.4.3.2. Skupovi

U PASCAL-jeziku postoji mogućnost rada sa skupovima podataka odredjenog tipa. Svi članovi skupova moraju da budu iste vrste i nesmeju da budu strukturirani. Zavisno od implementacije, broj članova skupa - kardinarni broj skupa je iz intervala $[16,256]$.

1.3.4.3.3. Liste

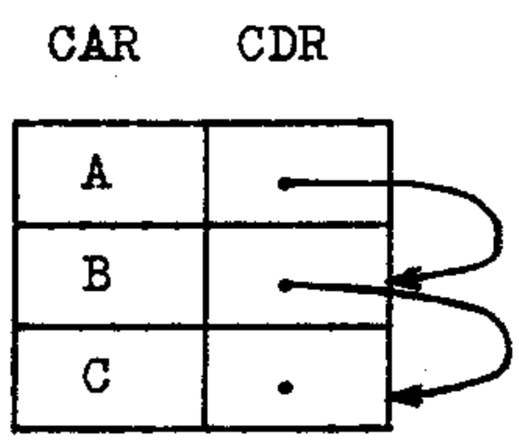
Programski jezik LISP je namenjen za rad sa listama. Elementi liste su slogovi koji se sastoje iz dva polja tzv.

CAR i CDR*. Navedena polja mogu da sadrže:

- atom (osnovni podatak),
- nula pokazivač ili
- adresu sledećeg sloga.

Primer

Linearna lista A,B,C može se obrazovati iz tri sloga. Polje CAR svakog od slogova sadrži redom A,B i C. Polje CDR prvog sloga je pokazivač drugog sloga, slično polje CDR drugog sloga je pokazivač trećeg sloga dok je polje CDR trećeg sloga nula pokazivač. Prethodno rečeno može se grafički prikazati na sledeći način:



Koristeći listu, drvo se može predstaviti na sledeći način. Svakom čvoru drveta pridruži se po jedan slog. Polje CAR svakog od slogova sadrži:

- pokazivač krajnje levog potomka ili
- nula pokazivač ako nema potomaka.

Polje CDR sadrži:

- pokazivač prvog desnog susednog čvora ili
- nula pokazivač ako ne postoje desni susedni čvorovi.

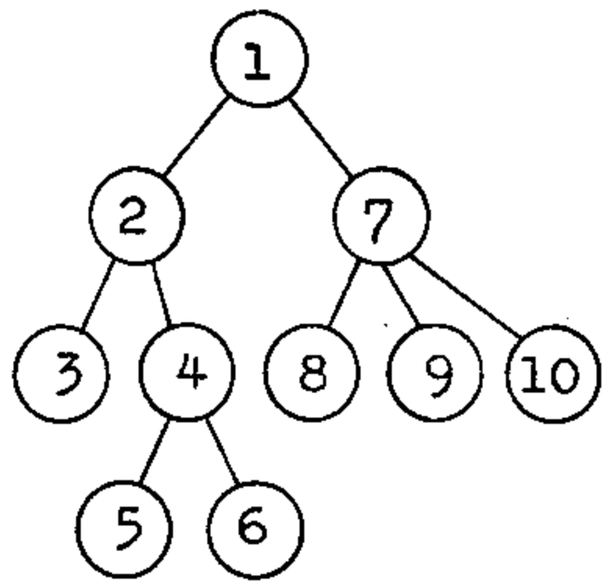
Navedena reprezentacija drveta može, na prvi pogled, da izgleda komplikovana ali u svakom slučaju omogućava jednostavno silaženje niz drvo.

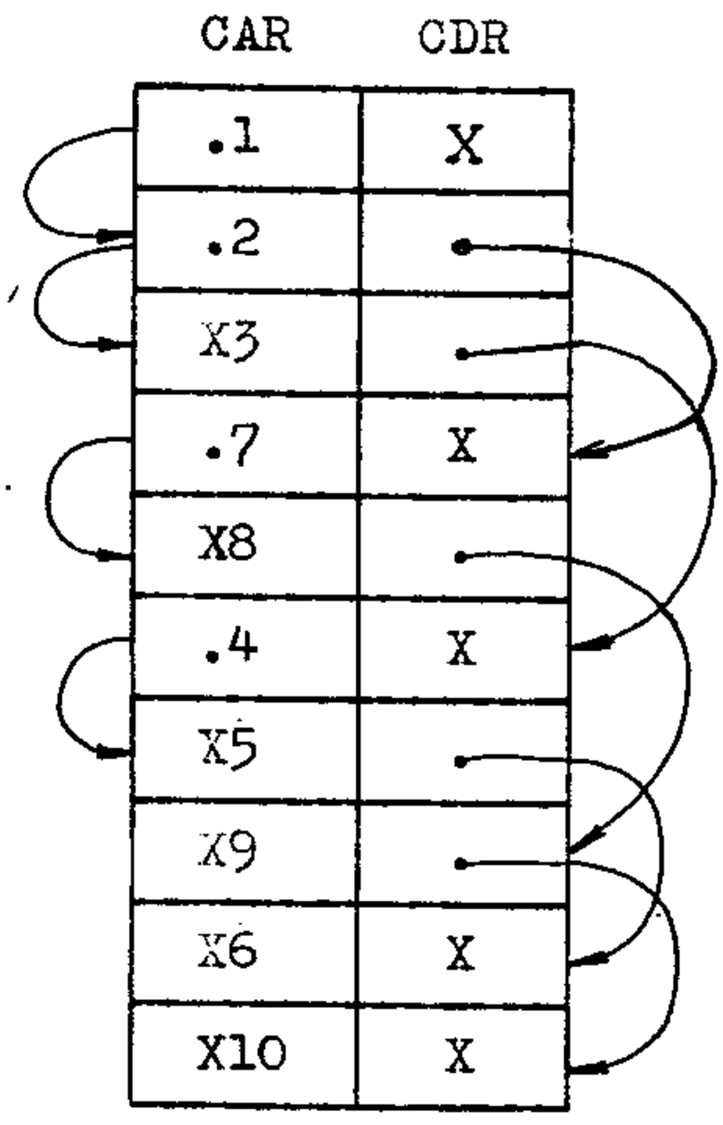
Primer

Drvetu T odgovara lista prikazana u obliku matrice.

---

* Oznake CAR i CDR su skraćenice naziva "Contents of the Address Register" i "Contents of the Decrement Register". Ovi registri postoje na računaru IBM 7090 na kojem je realizovana prva verzija LISP-interpretatora. Osnake su danas u upotrebi iz tradicionalnih razloga.





RB	CAR	CDR
1	2	0
2	3	7
3	0	4
4	5	0
5	0	6
6	0	0
7	8	0
8	0	9
9	0	10
10	0	0

Liste se u programskom jeziku LISP realizuju kao [71] niz slogova. Svaki slog se sastoji iz četiri polja. Prva dva polja sadrže informaciju o sadržaju CAR i CDR, a poslednja dva o vrsti sadržaja prva dva polja tj. da li sadrže atome ili pokazivače. Najnovije realizacije programskog jezika LISP sadrže i peto polje koje čuva informaciju o upotrebljivosti celog sloga. Ova poslednja informacija je od izuzetne koristi jer omogućava efikasno korišćenje memorijskog prostora. Za vreme izvršavanja LISP-programa dolazi do stalnih promena lista.

Novi članovi se ubacuju, a neki stari se izostavljaju iz lista. Ako se nebi vodilo računa o slobodnim(neiskorišćenim) članovima lista brzo bi došlo do prekoračenja memorijskog prostora. LISP - interpretator sadrži poseban blok koji služi za upravljanje memorijom tzv. skupljač otpadaka. Znači, po izvršavanju svake LISP-naredbe na scenu stupa skupljač odpadaka koji kontroliše sve slogove lista i od onih koji nisu u upotrebi obrazuje slobodnu listu.

1.3.4.3.4. Stogovi

Stog je specifična linearna lista. Specifičnost se ogleda u isključivom operisanju sa početkom (vrhom) liste. U suštini postoje dve osnovne operacije sa stogovima:

- upis u stog i
- čitanje stoga.

Primer

Neka se u stogu nalaze elementi A,B i C. Upisom elementa D, Stog postaje D,A,B,C. Čitanjem stoga dobija se polazno stanje.

U programskom jeziku PL/I postoji mogućnost definisanja stoga. Operacije ALLOCATE upis i FREE čitanje služe za rad sa stogovima.

1.3.5. Operatori

Bogatstvo programskog jezika ogleda se brojem različitih operatora.

1.3.5.1. Aritmetički operatori

Uobičajeni aritmetički operatori su:
+(čuvanje znaka, sabiranje),
-(promena znaka, oduzimanje),
*(množenje),
/(deljenje),

** ili ↑ (stepenovanje),
div (količnik celobrojnog deljenja) i
mod (ostatak celobrojnog deljenja).
Operatori:
- čuvanje znaka i
- promena znaka
su unarni operatori.

Ostali navedeni operatori su binarni operatori.

Unarni operator može:

- prethoditi operandu i onda se naziva prefiks operator ili

- slediti iza operanda i onda se naziva podfiks operator.

Primer

U programskom jeziku C izrazi: A=++B i A=B++ redom znače:

B=B+1
A=B

odnosno

A=B
B=B+1.

Binarni operator može:

- prethoditi operandima i onda se naziva prefiks operator,

- biti izmedju operanada i onda se naziva infiks operator i

- slediti iza operanada i onda se naziva postfiks operator.

U principu binarni operatori, koji se susreću u programskim jezicima, su infiks operatori.

1.3.5.2. Aritmetički izrazi

Uobičajeno pravilo obrazovanja izraza iz operanada i operatora glasi:

1. Zapisan podatak je ime osnovnog podatka, na primer identifikator, konstanta itd.

Pojedinačno zapisan podatak je izraz.

2. Ako je o binarni infiks operator i ako su I_1 i I_2 izrazi onda je I_1 o I_2 izraz.

3. Ako je o unarni prefiks operator i I izraz onda je oI izraz.

4. Ako je I izraz onda je (I) izraz.

Primeri

Neka je A promenljiva i 3.2 konstanta. Na osnovu pravila 1. A i 3.2. su izrazi.

Na osnovu pravila 2.: A+3.2 je izraz.

Na osnovu pravila 3.: -A, -3.2 su izrazi.

Na osnovu pravila 4.: (A+3.2) je izraz itd.

1.3.5.3. Relacijski operatori

Uobičajeni relacijski operatori su:

< ili .LT. (manje);

<=,=< ili .LE. (manje ili jednako);

= ili .EQ. (jednako);

>=,=> ili .GE. (veće ili jednako);

> ili .GT. (veće);

<> , >< ili .NE. (različito).

Navedeni relacijski operatori su binarni.

1.3.5.4. Relacijski izrazi

Neka je r relacijski operator i I_1 i I_2 izrazi. $I_1 r I_2$ je relacijski izraz.

Primeri

A<3.2,

A+3.2 .EQ. A itd.

su relacijski izrazi.

Vrednost relacijskog izraza je logička konstanta:
- istina (true) ili
- laž (false).

1.3.5.5. Logički operatori

Uobičajeni logički operatori su:
and ili .AND. (i),
or ili .OR. (ili) i
not ili .NOT. (ne).
Ne je unaran operator, a i i ili su binarni operatori.

1.3.5.6. Logički izrazi

1. Pojedinačno zapisan podatak koji ima logičku vrednost je logički izraz.

2. Relacijski izrazi su logički izrazi.

3. Ako je l binarni logički operator i ako su L_1 i L_2 logički izrazi onda je L_1 l L_2 je logički izraz.

4. Ako je l unarni logički operator i L logički izraz, onda je lL logički izraz.

5. Ako je L logički izraz onda je (L) logički izraz.

Primeri

Neka su X i Y logičke promenljive, a A i 3.2 aritmetički izrazi.

Na osnovu 1. X i Y su logički izrazi.
Na osnovu 2. A<3.2 je logički izraz.
Na osnovu 3. X .AND. A<3.2 je logički izraz itd.

1.3.5.7. Razni drugi operatori i izrazi

Pojedini programski jezici imaju specifične operatore kao što su na primer:

- azbučni operatori (tj. operatori za rad sa niskama znakova) (BASIC, SNOBOL, PL/I),

- skupovni operatori (PASCAL) itd.

Navedeni operatori definisani su nad odgovarajućim:

- azbučnim izrazima,
- skupovnim izrazima itd.

Primer

U BASIC-jeziku postoji azbučni operator dopisivanja (+ ili &). Operator dopisivanja & preslikava uredjen par niski (a,b) u nisku c

$$\& : (a,b) \rightarrow c$$

Niska c dobija se dopisivanjem na nisku a niske b.

Na primer neka je: a = "BEO" i b = "GRAD". Onda je:

c = a+b ="BEO" + "GRAD" = "BEOGRAD".

Simbol || se koristi u PL/I-jeziku kao operator dopisivanja.

1.3.5.8. Asocijativnost i prioritet operatora

Da bi se izračunala vrednost izraza, redosled izvršavanja operatora mora da bude strogo odredjen. U suprotnom, za iste podatke, dobile bi se različite vrednosti izraza. Na primer, izraz a+b+c može se interpretirati na dva načina:

a) ((a+b)+c),

b) (a+(b+c)).

U prvom slučaju argumenti su grupisani sleva udesno i za operator + se kaže da je levo-asocijativan.

U drugom slučaju argumenti su grupisani s desna ulevo i za operator + se kaže da je desno-asocijativan.

Medjutim, bez obzira na način grupisanja argumenata, vrednost prethodnog izraza ostaje neporomenjena. Ovaj zaključak, ne važi u opštem slučaju. Navedimo kontra primer. Izraz a+b*c, analogno rezonojući, može se interpretirati na sledeća dva načina:

a) ((a+b)*c),

b) (a+(b*c)).

Očigledno vrednost izraza zavisi od interpretacije. Pitanje je sad: kojoj od interpretacija tj. kojem od načina grupisanja argumenata dati prednost. Nastali problem se rešava uvodjenjem prioriteta operatora. Prioritet operatora odredjuje način grupisanja argumenata, tj. najpre se grupišu argumenti operatora višeg prioriteta i tako dalje redom.

Uobičajeno je da je operator * višeg prioriteta od operatora +, pa je tačna druga interpretacija izraza.

U svakom programskom jeziku strogo je propisana asocijativnost i prioritet izmedju operatora.

Primer

U programskom jeziku APL svi operatori su desno-asocijativni i istog su prioriteta. [30]

Izraz a+b*c+d se interpretira kao izraz

(a+(b*(c+d)))

Prioriteti raznih operatora najpoznatijih programskih jezika dati su u tabeli 1. Operatori su navedeni u opadajućem redosledu prioriteta, pri čemu su u istom redu grupisani operatori istog prioriteta.

FORTRAN

unarno +, unarno -, **

*,/

+,-

.LT., .EQ., .GT., .LE., .NE., .GE.

.NOT.

.AND.

.OR.

ALGOL

↑

×,/,÷

+,-

<,=, >, ≤, ≠, ≥

¬

∧

∨

⊃

≡

PASCAL

NOT
*,/, DIV, MOD, AND
+,-,OR
=, <, >, <>, <=, >=

BASIC

unarno +, unarno -, ↑
*,/
+,-,&
=, <, >, <>, ><, <=, =<, >=, =>

COBOL

**, EXPONENTIATED BY
*, MULTIPLED BY, TIMES, /, DIVIDED BY
+, PLUS, -,MINUS
IS [NOT] GREATER THAN, IS [NOT] >
IS [NOT] LESS THAN, IS [NOT] <
IS [NOT] EQUAL TO, IS [NOT] =, IS UNEQUAL TO
EQUALS
EXCEEDS
AND
OR
NOT

PL/I

**, unarni +, unarni -
*,/
+,-
||
< ,=, >, <=, >=, ¬<, ¬=, ¬>
&
|

Tabela 1.

Prioritet operatora je uglavnom isti ali postoji razlika u asocijativnosti operatora.

U FORTRAN-jeziku, konstruktorima kompilatora je ostavljeno da se sami odluče za levu odnosno desnu asocijativnost operatora.

U ALGOL-jeziku svi binarni operatori su levo-asocijativni.

U PL/I-jeziku svi binarni operatori su levo-asocijativni izuzev operatora stepenovanja koji je desno asocijativan, tj. a ** b ** c se interpretira kao (a ** (b ** c)) itd.

1.3.5.9. Algebarske osobine operatora

Mnogi operatori zadovoljavaju izvesne algebarske zakone. Ovim je omogućeno kompilatoru da izvrši izvesna pojednostavljenja izraza i na taj način generiše efikasniji prevod.

Aritmetičke operacije sabiranje i množenje su:

- komutativne i
- asocijativne.

Takodje je množenje distributivno u odnosu na sabiranje. Medjutim, u slučaju računarske aritmetike, zakon komutacije obično važi, ali zakon asocijativnosti i distributivnosti najčešće

Množenje je distributivno u odnosu na sabiranje. Zakon komutacije obično važi, dok zakoni asocijativnosti i distributivnosti najčešće ne važe.

Na primer, u opštem slučaju, zbog prekoračenja ne važi

$$(a+b)+c=a+(b+c).$$

Slično, zbog gubljenja značajnih cifara moguće je dobiti sasvim drugi rezultat.

1.3.5.10. Usaglašavanje vrsta argumenata operatora

Većina savremenih programskih jezika dozvoljava mogućnost da argumenti operatora budu različiti ali srodnih vrsta.

Primer

U FORTRAN-jeziku argumenti brojnog izraza mogu da budu:

- celobrojni,
- realni obične i dvostruke tačnosti,
- kompleksni obične i dvostruke tačnosti.

Sa navedenom mogućnošću javlja se problem vrste rezultata mešovitih izraza. Problem se rešava definisanjem vrste rezultata svakog operatora za svaku moguću kombinaciju vrsta argumenata.

Primer

Vrsta aritmetičkog izraza FORTRAN-jezika u slučaju operatora +,-,* i / data je u tabeli 2.

	Celi	Real ob t	Real ob t	Kompl ob t	Kompl ob t
Celi	Celi	Real ob t	Real dv t	Kompl ob t	Kompl dv t
Real ob t	Real ob t	Real ob t	Real dv t	Kompl ob t	Kompl dv t
Real dv t	Real dv t	Real dv t	Real dv t	Kompl dv t	Kompl dv t
Kompl ob t	Kompl ob t	Kompl ob t	Kompl dv t	Kompl ob t	Kompl dv t
Kompl dv t	Kompl dv t	Kompl dv t	Kompl dv t	Kompl dv t	Kompl dv t

Tabela 2

1.3.5.11. Realizacija operatora

Za većinu operatora definisanih nad skupom primitivnih vrsta podataka postoji odgovarajuća mašinska naredba. Aritmetički i logički operatori se relativno jednostavno realizuju.

Relacijski operatori se realizuju mašinskim naredbama uslova i skoka.

Ostali operatori se obično realizuju nizom mašinskih naredbi organizovanim u obliku potprograma.

U slučaju mini-računara koji imaju relativno skroman repertoar naredbi, uobičajeno je da se naredbe množenja i deljenja realizuju u obliku potprograma.

Šarolikost repertoara mašinskih naredbi otežava postavljanje generalne strategije generisanja kôda.

1.3.5.12. Dodeljivanje

Dodeljivanje je najčešće korišćena operacija u programima. Sintaksa operatora dodeljivanja varira od programskog jezika do programskog jezika. U tabeli 3. na primeru aritmetičke naredbe a=b navedeni su zapisi operatora dodeljivanja za najpoznatije programske jezike.

programski jezik	a=b
FORTRAN	A=B
ALGOL	A:=B
BASIC	LET A=B ili samo A=B
PASCAL	A:=B
COBOL	MOVE B TO A
PL/I	A=B
APL	A←B
LISP	(SETQ A B)
SNOBOL	A=B

Tabela 3

Zbog jednoznačnosti, poželjno je da se operator dodeljivanja razlikuje od relacijskog operatora jednakosti.

1.3.5.13. Leva i desna vrednost

Lokacija i vrednost identifikatora su dva različita pojma. Prethodno rečeno je očigledno u slučaju jednostavne naredbe dodele A:=B čije je značenje:"stavi vrednost od B na lokaciju A". A i B imaju različita značenja i ako su na isti način napisani. Pozicija od B, tj. činjenica da se B nalazi sa desne strane operatora dodele, označava da je u pitanju vrednost. Slično, pozicija od A, tj. činjenica da se A nalazi sa leve strane operatora dodele, ukazuje da je u pitanju lokacija.

Vrednost identifikatora naziva se d-vrednost identifikatora. Lokacija identifikatora naziva se l-vrednost identifikatora. Oznake d i l su prva slova reči desno i levo.

d i l vrednost izraza definišu se na sledeći način:

1) d i l vrednost jednočlanog izraza jednaka je d i l vrednosti člana izraza.

2) Ako se izraz sastoji iz više članova, d-vrednost izraza jednaka je vrednosti izraza izračunatoj na odgovarajući način. l-vrednost izraza postoji isključivo ako izraz označava lokaciju. Retki su programski jezici čiji izrazi imaju l-vrednost.

Primeri

Promenljiva ima l i d vrednost. d-vrednost odgovara tekućoj vrednosti promenljive. l-vrednost je adresa memorijske lokacije u kojoj se čuva vrednost promenljive.

Konstanta ima d-vrednost, a nema l-vrednost.

U programskim jezicima BLISS i ALGOL-68 ne važi prethodna definicija.

U programskom jeziku BLISS ime promenljive uvek označava l-vrednost bez obzira sa koje se strane znaka dodele nalazi promenljiva. Unarni operator tačka (.) koristi se za označavanje d-vrednosti promenljive. Tako na primer izraz A←.B+.C znači "sabrati vrednosti promenljivih B i C i dobijeni zbir dodeliti promenljivoj A, tj. upisati u memorijsku lokaciju rezervisanu za čuvanje vrednosti promenljive A". Izraz A←B znači "d-vrednost A jednaka je l-vrednosti B, tj. A postaje pokazivač B".

U programskom jeziku ALGOL-68 pravi se razlika izmedju l-vrednosti i d-vrednosti promenljive. Po konvenciji promenljiva predstavlja l-vrednost. Ako se želi da promenljiva predstavlja d-vrednost - opisnom naredbom ref to se mora posebno naglasiti.

1.3.5.14. Realizacija dodeljivanja

Postoji više načina realizacije naredbe dodeljivanja:

a:=τ

gde je a - promenljiva, a

τ - izraz.

Moguća realizacija je:

1. Generiše se kôd za izračunavanje d vrednosti izraza Υ. d-vrednost izraza Υ smešta se u registar r.

2. Ako su promenljiva a i izraz Υ različite vrste generiše se kôd za transformaciju d vrednosti izraza Υ u vrstu promenljive a.

3. Ako promenljiva a nije proste vrste, generiše se kôd za izračunavanje 1 vrednosti promenljive a.

4. Generiše se kod za upis sadržaja registra r na memorijsku lokaciju odredjenu 1-vrednošću promenljive a.

1.3.5.15. Dodeljivanje kao operator

Programski jezici ALGOL-68, BLISS, C, neke verzije BASIC-a tretiraju operator dodeljivanja kao binaran infiks operator. Prioritet operatora dodeljivanja je niži od prioriteta aritmetičkih, relacijskih i logičkih operatora.

Primer

U programskom jeziku C dozvoljeni su izrazi čiji su članovi oblika:

$$a=\Upsilon$$

gde je a promenljiva, a Υ izraz.

Recimo

$$A=(B=C+D)-(E=F*G)$$

se interpretira kao niz:

$$B=C+D$$

$$E=F*G$$

$$A=B-E.$$

1.3.5.16. Opšte dodeljivanje

Programski jezici ALGOL-68, PL/1, APL itd. dozvoljavaju dodeljivanje oblika:

$$a:=b$$

gde su a i b složene strukture podataka.

Zahteva se da se složene strukture podataka a i b slažu u veličini, vrsti, strukturi itd.

Primeri

Neka je A niz. U programskom jeziku PL/I naredbom A=Ø postavljaju se svi članovi niza A na nulu. Ako je B niz iste dimenzije i vrste kao niz A naredbom A = B svi članovi niza A izjednačavaju se sa odgovarajućim članovima niza B itd.

1.3.6. Naredbe

Osnovne radnje kao što su na primer izračunavanje vrednosti, dodeljivanje, upravljanje itd. specificirane su naredbama programskog jezika. Svaki programski jezik sadrži veći broj raznih naredbi. Naredbe se medjusobno razlikuju po formi i sadržaju.

1.3.6.1. Proste i složene naredbe

Naredba je prosta ako se ne sastoji iz više naredbi. Naredbe ulaza, izlaza, skoka, dodeljivanja su primeri prostih naredbi.

Programski jezici SNOBOL, APL sadrže isključivo proste naredbe. Program na jednom od spomenuta dva jezika je niz prostih naredbi.

Naredba je složena ako se sastoji iz više naredbi.

Logička naredba IF programskog jezika FORTRAN

IF (uslov) naredba

je primer složene naredbe.

Složene naredbe omogućavaju grupisanje logički povezanih radnji u jednu celinu. Složene naredbe povećavaju čitljivost programa.

Programski jezik PASCAL je izuzetno bogat složenim naredbama:

1. begin naredba; naredba;...; naredba end;
2. while uslov do naredba;
3. for indeks := poč-vred to kraj-vred do naredba;
4. if uslov then naredba else naredba;

su primeri složenih naredbi PASCAL-jezika.

Programski jezik ALGOL takodje sadrži niz složenih naredbi.

1.3.6.2. Vrste naredbi

U principu, naredbe programskih jezika mogu se razvrstati u 5 grupa:

1. Izračunavanja,
2. Upravljanja (kontrole),
3. Strukturisanja,
4. Deklarisanja i
5. Ulaza i izlaza.

Naredba dodeljivanja je tipična naredba izračunavanja. U naredbi izračunavanja izvršavaju se operatori nad odgovarajućim operandima u cilju dobijanja novih podataka.

Kod većine programskih jezika, upravljanje izvršavanjem programa automatski se prenosi sa jedne naredbe na sledeću u nizu. U cilju promene redosleda izvršavanja naredbi programa uvedene su naredbe upravljanja. Tipične naredbe upravljanja su naredbe:

- bezuslovnog skoka,
- uslovnog skoka,
- prelaska na potprogram i
- povratka u program.

Zadnjih 20 godina vodjeno je bezbroj diskusija i napisano je niz radova na temu: kako treba da izgledaju naredbe upravljanja izvršavanjem programa. Naredba bezuslovnog skoka (tj. goto naredba) bila je u središtu svih ovih razmatranja. Upotreba naredbe bezuslovnog skoka bez ograničenja može da stvori niz teškoća.

Naredbe logičkog početka i kraja su tipični primeri naredbi strukturisanja. Naredbe strukturisanja služe za grupisanje prostih naredbi u složenu naredbu.

Naredbe strukturisanja i deklasiranja su tzv. opisne

naredbe. Prilikom njihovog prevodjenja ne generiše se kôd već isključivo informacije o atributima promenljivih, nizova itd. koje se čuvaju u tabelama simbola i koriste se pri prevodjenju i izvršavanju programa. Naredbe deklarisanja vrste promenljivih, dimenzije nizova itd. su tipične naredbe deklarisanja.

Naredbe ulaza-izlaza služe, kao što im samo ime kaže, za unošenje i izdavanje podataka programa. Kod većine programskih jezika naredbe ulaza-izlaza realizuju se kao potprogrami. Ovi potprogrami su dostupni operativnom sistemu računara i imaju zadatak da:

- upravljaju i kontrolišu rad ulazno-izlaznih organa računara,
- transformišu podatke iz jednog oblika u drugi.

Kod većine programskih jezika postoji mogućnost formatiranja podataka. Format podatka je informacija o načinu interpretiranja (tj. prihvatanja i izvršavanja) podataka.

Kod nekih programskih jezika (BASIC, PASCAL, ALGOL) format podataka je odredjen unutrašnjom konvencijom jezika, a kod drugih (FORTRAN) mora se uvek eksplicitno definisati.Format podataka se realizuje pomoću naredbi ulaza-izlaza (PASCAL) ili kao posebna format naredba (FORTRAN).

1.3.7. Programi

Najviši nivo - vrh hijerarhijske strukture konstrukcija programskih jezika čine:

- blokovi,
- procedure i potprogrami,
- programi.

Na primerima najpoznatijih programskih jezika razmotrimo njihovu strukturu programa.

1.3.7.1. FORTRAN-programi

FORTRAN-program se sastoji iz glavnog programa i proizvoljnog broja potprograma.

Potprogrami mogu da budu:
- opšti potprogrami,
- funkcijski potprogrami,
- funkcijske naredbe i
- blokovi podataka.

Glavni program i potprogrami, ako postoje, mogu se zajedno ili posebno kompilirati. Nezavisnost potprograma od programa omogućava modularnu konstrukciju programa.

Glavni program i potprogrami sastoje se iz niza:
- opisnih naredbi i
- izvršnih naredbi.

Glavni program i potprogrami medjusobno komuniciraju - razmenjuju podatke preko imenovanih i neimenovanih zajedničkih (common) zona.

Podaci koji se koriste u bar dve programske jedinice nazivaju se globalni podaci.

Podaci koji se isključivo koriste u jednoj programskoj jedinici, nazivaju se lokalni podaci. Globalni podaci se moraju navesti u COMMON-zoni.

1.3.7.2. ALGOL-programi

ALGOL-program sastoji se iz glavnog programa i proizvoljnog broja potprograma sastavljenih iz blokova. Blok se sastoji iz niza opisa i operatora. Svaka konstrukcija počinje sa begin, a završava se sa end.

Glavni program je poseban blok. Potprogram se sastoji iz zaglavlja i tela. U zaglavlju se navodi ime potprograma, imena i vrsta formalnih parametara. Telo je u opštem slučaju blok.

1.3.7.3. COBOL-programi

COBOL-program sastoji se iz četiri odeljka. Ovakvom organizacijom programa postignuta su dva cilja:

1) Razdvojeni su mašinski-zavisni i mašinski-nezavisni elementi programa.

2) Razdvojeni su zapisi algoritama i podataka.

Odeljak identifikacije programa sadrži ime programa, autora i druge informacije od značaja za dokumentaciju.

Odeljak opreme sadrži mašinski zavisne specifikacije programa.

Odeljak podataka sadrži opise podataka.

Odeljak procedure sadrži opis algoritma.

1.3.7.4. PL/I-programi

PL/I-program sastoji se iz niza potprograma od koji je svaki blokovske strukture. Potprogram se sastoji iz niza opisnih i izvršnih naredbi. U opštem slučaju, svaka vrsta naredbe ima osnovni oblik i veči broj dodatnih mogućnosti koje programer po potrebi koristi. Zbog velikog broja raznih mogućnosti, sintaksa i semantika PL/I-jezika je složena što otežava njegovu uptrebu.

1.3.7.5. LISP-programi

LISP-program je jednostavne strukture. Sastoji se iz niza definicija potprograma - funkcija za kojima sledi niz poziva istih. Blokovske strukture ne postoje.

Uzajamno dejstvo raznih funkcija moguće je isključivo za vreme izvršavanja.

Funkcije LISP-jezika definišu se kao izrazi. Svaki operator je funkcija koja izračunava odredjenu vrednost.

1.3.7.6. BASIC-programi

BASIC-program se sastoji iz glavnog programa i proizvoljnog broja programskih segmenata. Glavni program i programski segmeni povezuju se ili preklapaju. Postoje potprogrami ali nisu opšti. Glavni program i programski segmenti sastoje se iz programskih redova. Programski red sastoji se iz više naredbi.

1.3.7.7. PASCAL-programi

PASCAL-program se sastoji iz glavnog programa i proizvoljnog broja procedura. Procedure moraju biti definisane pre upotrebe.

PASCAL-program se sastoji iz tri dela:

- zaglavlja programa,
- deklaracija i
- izvršnih naredbi.

U zaglavlju programa navodi se ime programa i imena eksternih datoteka.

U deklarativnom delu definišu se sve promenljive, konstante i strukture podataka koje se koriste u izvršnom delu programa.

1.3.7.8. APL-programi

U APL-jeziku ne postoji glavni program. Najveća programska jedinica je potprogram. Potprogram se sastoji iz niza programskih redova. Programski red sadrži jednu ili više naredbi. Svakom programskom redu interpretator obavezno dodeljuje broj reda, a programer po želji obeležje reda. Izvršavanje potprograma inicira se:

a) pozivom iz drugog potprograma,
b) neposrednim zahtevom programera.

APL-potprogrami sadrže najviše dva argumenta. Prvim argumentom se zadaju podaci, a drugim argumentom se uzimaju rezultati. Argumenti u opštem slučaju mogu da budu nizovi tako da broj argumenata - dva ne predstavlja neko posebno ograničenje.

2. PREVODIOCI

2.1. Klasifikacija prevodilaca

U najopštijem slučaju prevodilac je organ za prevodjenje tekstova sa jednog jezika na drugi. U računarstvu se pod prevodiocem podrazumeva program za prevodjenje programa sa jednog programskog jezika na drugi programski jezik. Potreba za prevodjenjem javlja se istovremeno sa željom da se izvrši program napisan na programskom jeziku za koji ne postoji tehnički realizovan izvršni organ. Programi, isključivo napisani, na mašinskom jeziku datog računara mogu se neposredno (bez prevodjenja) izvršavati na istom računaru. Programski jezik koji se prevodi naziva se ulazni ili izvorni jezik. Program napisan na ulaznom (izvornom) jeziku naziva se ulazni (izvorni) program. Programski jezik na koji se vrši prevodjenje naziva se izlazni, objekt ili ciljni jezik. Program na izlaznom ciljnom/ jeziku naziva se prevedeni objekt program

Zavisno od vrste ulaznog jezika prevodioci se dele na:
- asemblere,
- kompilatore i
- generatore.

Ulazni jezik:
- asemblera je mnemokôd (simbolički ili niži programski jezik),
- kompilatora - proceduralno-orijentisan (viši programski) jezik i
- generatora - problemski-orijentisan jezik.

Izlazni jezik prevodioca obično nije mašinski jezik već tzv. medjujezik (medjukôd). Na primer medjujezik programskog jezika SNOBOL je poljski zapis.

Medjujezik može da bude:

a) programski jezik nižeg nivoa koji se dalje pomoću odgovarajućeg prevodioca prevodi;

b) programski jezik iste vrste samo s jednostavnijim naredbama. Takvi prevodioci nazivaju se pretprocesori. Na primer, postoji FORTRAN-pretprocesor koji prevodi strukturiranu verziju programskog jezika FORTRAN na standardni FORTRAN.

Realizacija programa na računaru sastoji se u principu iz dva procesa:

- prevodjenja izvornog programa,
- izvršavanja prevedenog programa.

Proces prevodjenja izvornog programa i proces izvršavanja prevedenog programa mogu vremenski da budu:

a) razdvojeni,

b) povezani.

Sistemi automatizacije programiranja (programski sistemi) kod kojih su procesi prevodjenja izvornog programa i izvršavanja prevedenog programa vremenski razdvojeni nazivaju se prevodilački (translatorski) sistemi. Sistemi kod kojih su procesi prevodjenja izvornog programa i izvršavanja prevedenog programa vremenski povezani nazivaju se interpretatorski sistemi.

Na slici 2.1. grafički je prikazano prevodjenje i izvršavanje programa u prevodilačkom sistemu.

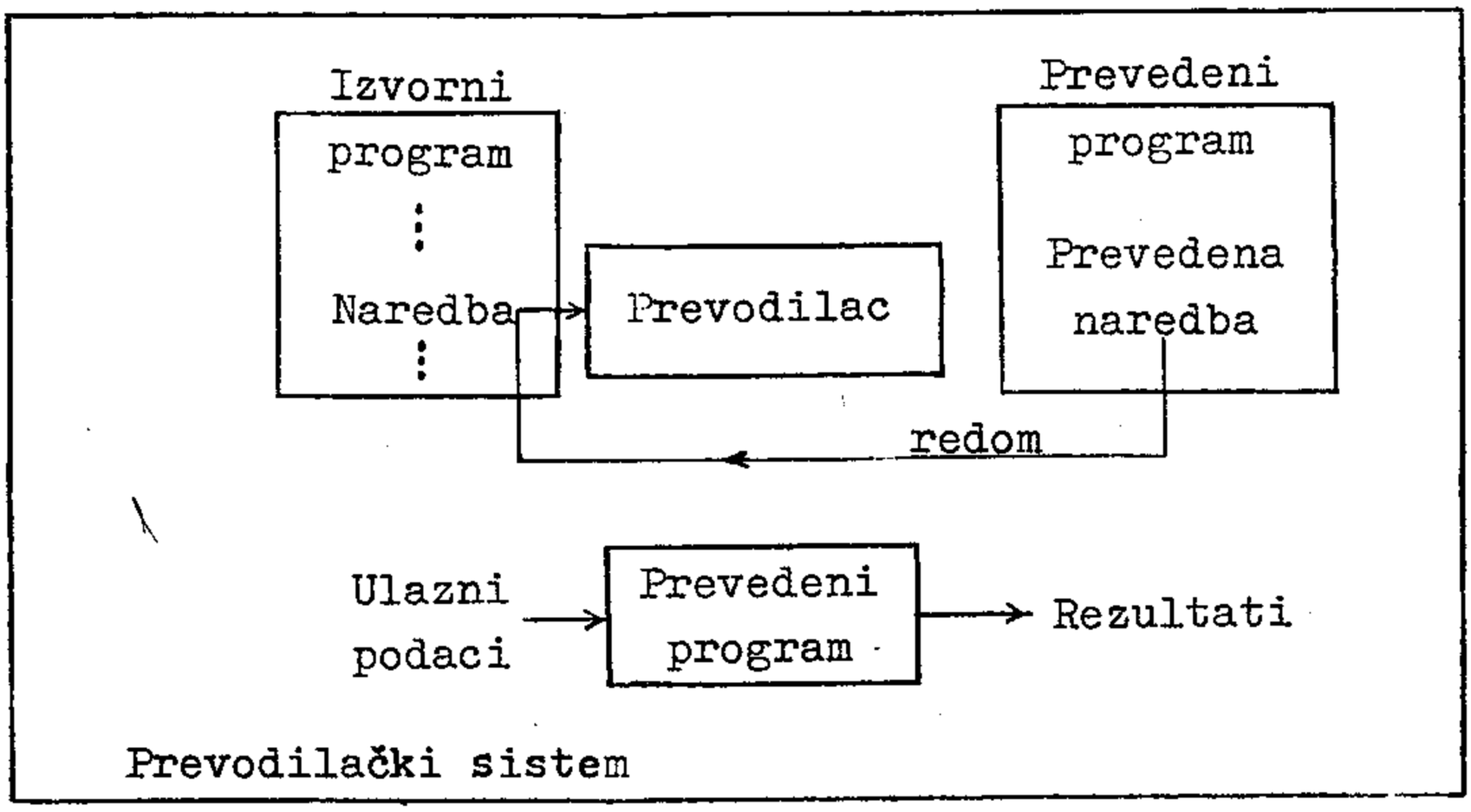
Na slici 2.2. grafički je prikazano prevodjenje i izvršavanje programa u interpretatorskom sistemu.

2.2. Struktura kompilatora

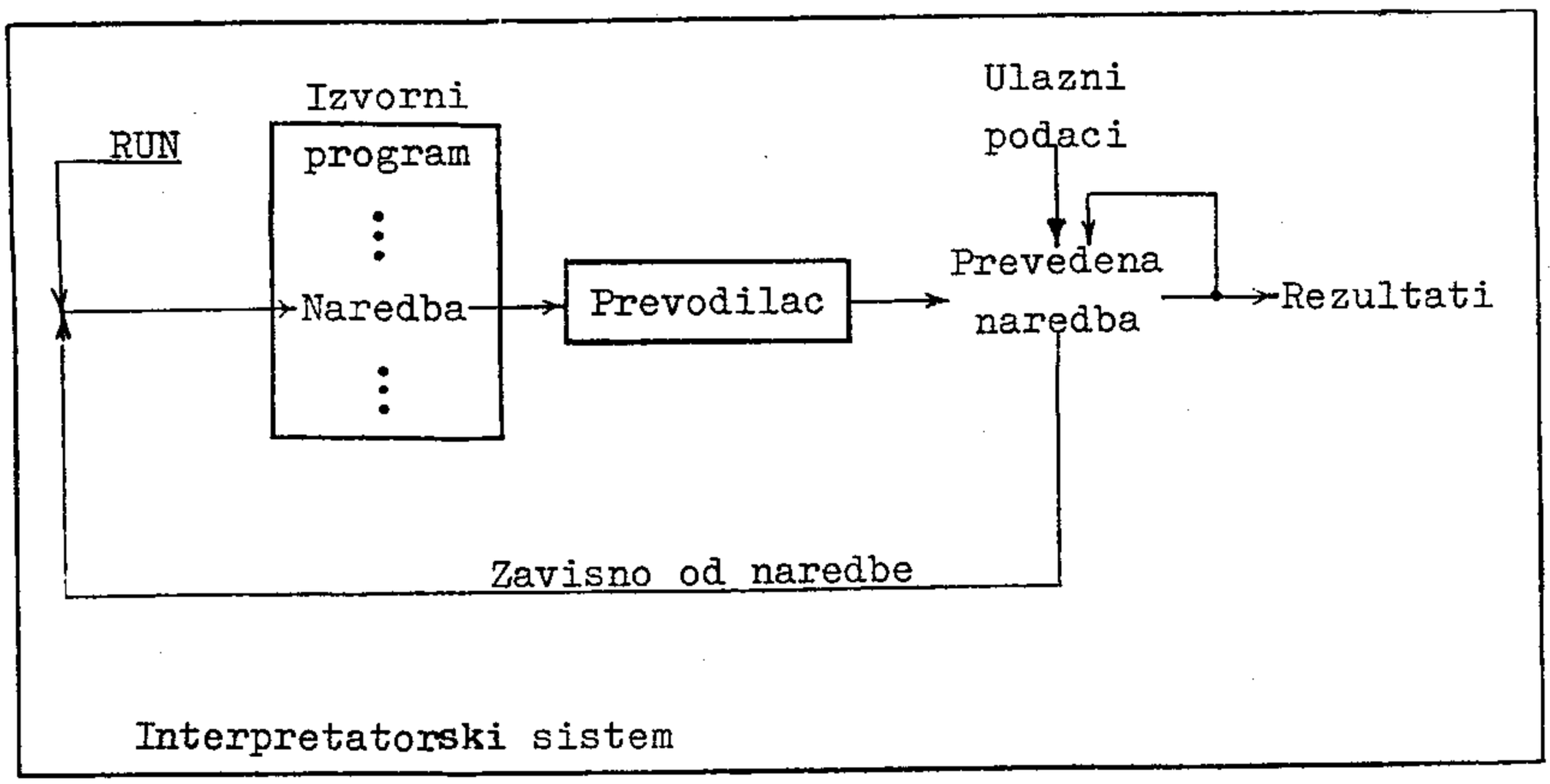
U praksi se najviše koriste proceduralno-orijentisani (viši programski) jezici. Zato se u Tezi prvenstveno razmatra kompilacija proceduralno-orijentisanih jezika i konstruisanje kompilatora.

Proces kompilacije je veoma složen tako da nije opravdano bilo sa logičke, bilo sa realizatorske strane proučavati

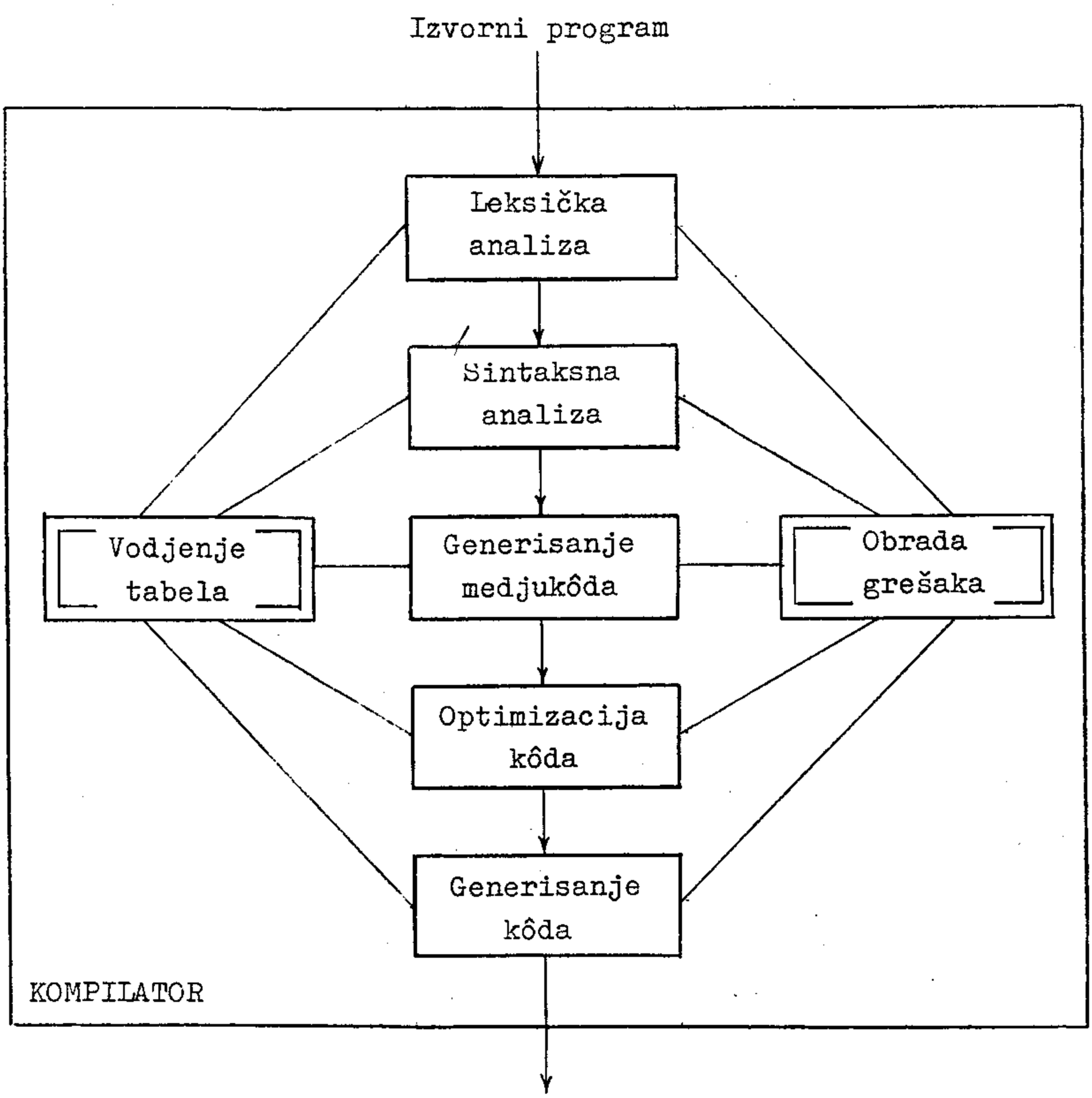
proces kompilacije kao jedan jedinstven nedeljiv proces. Naprotiv, uobičajeno je deljenje procesa kompilacije na niz potprocesa tzv. faza kao što je prikazano na slici 2.3. [40]



Slika 2.1.



Slika 2.2.



Slika 2.3.

Faza je logička celina čije su ulazne i izlazne veličine odredjene reprezentacije programa. Ulazne veličine faze uzimaju se neposredno iz programa koji se kompilira ili su dobijene izvršavanjem prethodnih faza.

Prva faza je leksička analiza. Zadatak leksičke analize je obrazovanje niski simbola izvornog jezika koje predstavljaju logičku celinu. Ove niske nazivaju se leksičke konstrukcije. Najčešće leksičke konstrukcije su: [3]

- službene reči kao što su na primer: DO, IF itd.,
- identifikatori (imena) kao što su: X, IPS1 itd.,

- znaci operacija kao što su: <=, +, ↑ itd.,
- znaci interpunkcije kao što su: (,), ; itd.

Ulazni podaci leksičkog analizatora su naredbe izvornog programa, a izlazni podaci su nizovi leksičkih konstrukcija koji se predaju sledećoj fazi na dalju obradu. Leksičke konstrukcije se najčešće iz praktičnih razloga kodiraju celim brojevima i predstavljaju uredjenim parovima:

(i,j)

gde je:

- i kôd leksičke konstrukcije, a
- j kôd vrste konstrukcije.

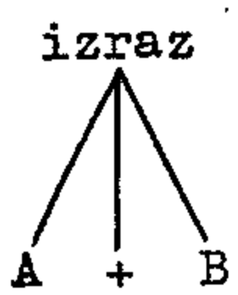
Na primer, službena reč DO se može predstaviti kao

(7,1)

gde je 7 kôd službene reči DO, a 1 kod vrste, tj. službene reči.

Druga faza je sintaksna analiza. Zadatak sintaksne analize je grupisanje leksičkih konstrukcija u sintaksne konstrukcije. Na primer, sledeće tri leksičke konstrukcije A+B grupisane su u sintaksnu konstrukciju koja se naziva izraz. Izrazi se mogu dalje grupisati u naredbe itd.

Sintaksne konstrukcije se najčešće predstavljaju u obliku drveta čiji su listovi leksičke konstrukcije. Na primer, sintaksno drvo izraza A+B je:



Unutrašnji čvorovi drveta odgovaraju niskama leksičkih konstrukcija koje su na višem nivou logički povezane.

Treća faza je generisanje medjukôda. Generator medjukôda koristi konstrukcije dobijene od sintaksnog analizatora za obrazovanje mnoštva jednostavnih naredbi. Danas postoji čita niz različitih zapisa, notacija za predstavljanje medjukôda. U principu, svi oni se svode na naredbe oblika:

operator $operand_1$,..., $operand_i$; $i \geq 1$.

Ove naredbe mogu se smatrati makro-naredbama nekog makro--jezika. Znači, medjukôd je makro-jezik.

Četvrta faza je optimizacija kôda ili tačnije rečeno optimizacija medjukôda. U slučaju jednostavnijih kompilatora ova faza obično ne postoji (izostavlja se). Zadatak optimizacije kôda je poboljšanje medjukôda u smislu povećanja brzine izvršavanja programa, odnosno smanjivanja potrebnog memorijskog prostora.

Peta faza je generisanje kôda ili još preciznije rečeno generisanje objekt-kôda. Osnovna razlika izmedju medjukôda i objekt-kôda je ta što u slučaju medjukôda nisu odredjeni registri koji se koriste pri računanju, odnosno nije izvršena raspodela medjumemorijskog prostora i ostalih resursa računara.

Svaki kompilator sadrži izvestan broj tabela. U tabelama se čuvaju razne značajne informacije o programu i njegovim sastavnim delovima. Tako, na primer, postoji tabela promenljivih koje se koriste u programu. Ime promenljive, vrsta promenljive, adresa memorijskog registra koji sadrži vrednost promenljive itd. su informadije koje sadrži tabela promenljivih.

Za vreme prevodjenja programa kompilator treba da otkrije sve leksičke i sintaksne greške koje postoje u programu i o njima da obavesti korisnika - autora programa. Po otkrivanju prve greške u programu nije dobro da kompilator prekine dalju kompilaciju programa, već je treba nastaviti sa ciljem da otkrije što je moguće više grešaka. Faze generisanje medjukoda, objekt-koda i optimizacije po otkrivanju bar jedne greške nemaju dalje smisla pa se preskaću.

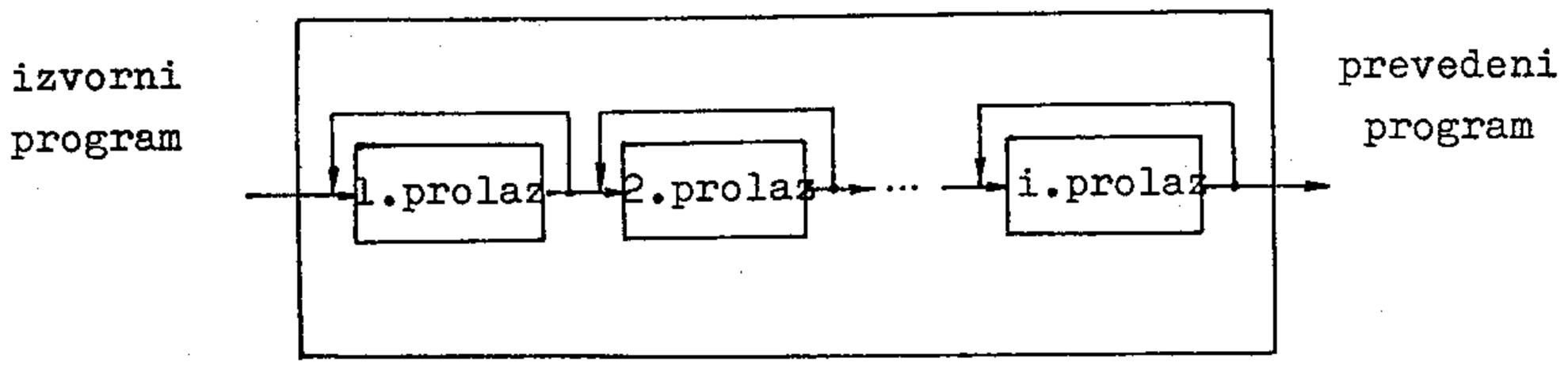
Prevedeni program koji sadrži bar jednu grešku ne može se izvršiti. Greške treba popraviti i iz početka ponoviti kompilaciju programa.

Blokovi: Rad sa tabelama i Obrada grešaka koriste se u svim navedenim fazama kompilacije.

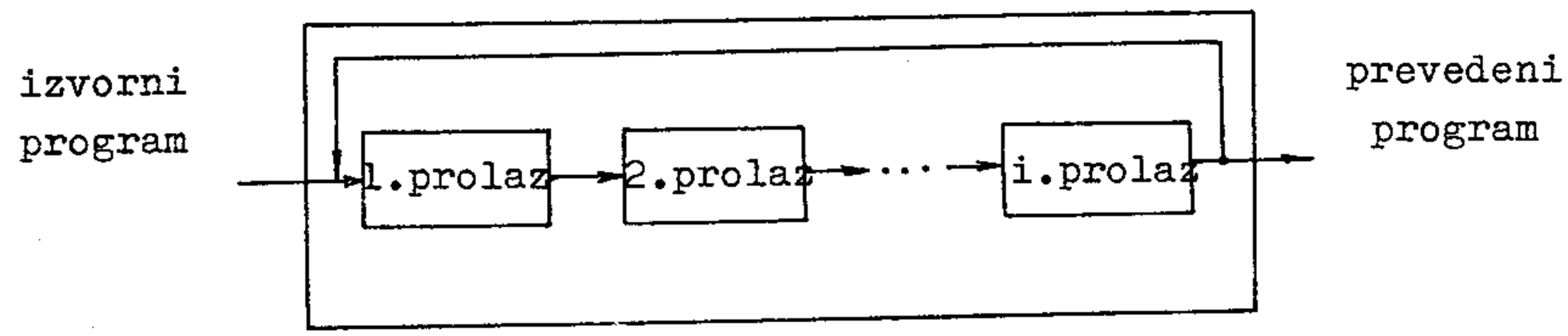
2.2.1. Prolazi

Pri realizaciji kompilatora iz raznih razloga uobičajeno je da se delovi jedne ili više faza spajaju i obrazuju module (blokove) koji se nazivaju prolazi. [38]

Skoro svi kompilatori su više prolazni. Ulazni podatak prvog prolaza je naredba ili ceo izvorni program. Ulazni podaci narednih prolaza su izlazni podaci prethodnih prolaza. Izlazni podatak poslednjeg prolaza je rezultat prevodjenja, odnosno prevedena naredba ili program. Prevodjenje se u principu može organizovati na dva načina. Prvi način je da svaki prolaz u potpunosti obavi posao nad celim programom pre nego što preda upravljanje sledećem prolazu. Drugi način je da svaki prolaz obavi posao nad samo jednom naredbom programa, a zatim preda upravljanje narednom prolazu. Prevodjenje programa odvija se višestrukim izvršavanjem niza prolaza. Navedene organizacije prikazane su redom na slici 2.4. i 2.5.



Slika 2.4.



Slika 2.5.

Prolazi najčešće medjusobno komuniciraju i razmenjuju podatke preko teka koje se čuvaju na nekom od spoljnjih memorijskih medijuma (obično je to magnetni disk).

Broj prolaza kompilatora prvenstveno je uslovljen tehničkim karakteristikama računarskog sistema i osobinama izvornog jezika. Tako, na primer, više prolazni kompilator zahteva manje

operativnog memorijskog prostora od jednoprolaznog kompilatora. Za vreme izvršavanja višeprolaznog kompilatora potrebno je samo da se tekući prolaz nalazi u operativnoj memoriji računara. Prolazi koji se trenutno ne izvršavaju i teke nalaze se na spoljnim nosiocima informacija. Višeprolazni kompilator je znatno sporiji od jednoprolaznog kompilatora, upravo za vreme učitavanja naredbi prolaza, upisa i čitanja podataka na/sa spoljne memorije.

Struktura izvornog jezika utiče na broj prolaza. Programski jezik ALGOL-68 dozvoljava naknadno deklarisanje promenljivih, tj. deklarisanje promenljivih posle njihove upotrebe. Očigledno izrazi koji sadrže nedeklarisane promenljive ne mogu se neposredno prevesti. Spomenuta osobina zahteva bar dva prolaza odgovarajućeg kompilatora.

2.2.2. Smanjenje broja prolaza

Svaka faza predstavlja transformaciju programa iz jednog oblika u drugi oblik. Samo na početku prve i na kraju poslednje faze program se nalazi u eksplicitnom obliku kao uredjen niz naredbi dok se u ostalim fazama čuva u implicitnom obliku najčešće u vidu tabela. Opšta težnja je da se smanji broj faza, a da pritom one ostanu i dalje relativno jednostavne. U nekim slučajevima je to moguće. Na primer, u slučaju leksičke analize bez obzira na veličinu izvornog programa potrebna je mala memorijska zona za smeštaj ulaznih podataka i fiksna za čuvanje izlaznih podataka. Zato često faze leksičke i sintaksne analize čine jedan prolaz. Više faza se može spojiti u jedan prolaz. Ova programska tehnika naziva se metoda naknadnog prevodjenja. Izložimo ukratno metodu naknadnog prevodjenja. Ako se izlazni podatak faze ne može sasvim utvrditi bez novih ulaznih podataka za koje se pretpostavlja da kasnije slede - izdavanje rezultata se ne mora odložiti za kasnije, već se rezultati mogu odmah izdati onakvi kakvi su, a da se praznine naknadno (kasnije) popune. Na ovaj način izbegava se još jedan prolaz mada povratak ipak pos-

toji. Metoda naknadnog prevodjenja veoma se često koristi pri prevodjenju simboličkih jezika. Na primer, neka je data naredba

SKOK a

ali pre naredbe sa obeležjem a.

Dvoprolazni asembler će u prvom prolazu upisati u tabelu simbola sve identifikatore (obeležja naredbi, konstante i promenljive) zajedno sa odgovarajućim memorijskim adresama relativnim u odnosu na početak programa. U drugom prolazu mnemokod, tj. službena reč SKOK zameniće se odgovarajućim kodom operacije, a identifikator, tj. obeležje a zameniće se odgovarajućom memorijskom adresom odredjenom za vreme prvog prolaza.

Jednoprolazni asembler će zadatak rešiti, recimo, na sledeći način: Odmah će generisati odgovarajući kod operacije, dok će adresni deo ostaviti ne popunjen jer to trenutno i ne može da uradi. U tabeli poternica upisaće adresu naredbe relativno u odnosu na početak programa i traženi adresni deo naredbe (obeležje) i čekaće da se tokom dalje obrade pojavi naredba sa traženim obeležjem a. Svaki put kad naidje naredba sa obeležjem proveriće u tabeli poternica da li su za dotično obeležje raspisane poternice. Ako poternice postoje nema problema da se odgovarajuće naredbe dopune, a poternice obrišu. Takodje se nadjeno obeležje naredbe upisuje u tabelu identifikatora (obeležja naredbi) zajedno sa vrednošću brojača naredbi i priznakom korišćenja obeležja. Po obradi svih naredbi, asembleru preostaje prekontrolisati tabelu poternica i obeležja naredbi. Ako u tabeli poternica postoje poternice znači da postoje naredbe koje nisu prevedene do kraja. Tačno se zna i koje su to naredbe. Ako u tabeli obeležja naredbi postoje priznaci neiskorišćenih obeležja znači da su ista obeležja suvišna. I u jednom i u drugom slučaju o pronadjenim greškama treba na prigodan način obavestiti korisnika programa. Prva vrsta grešaka je fatalne prirode i program se ne može izvršiti. Druga vrsta grešaka je upozoravajuće prirode i program se može izvršiti uz bojaznost da se negde potkrala semantička greška.

Može se navesti niz sličnih primera za slučaj proceduralno-orijentisanih jezika koje kompilatori na analogan način rešavaju.

Primetimo da je rastojanje izmedju "zakrpa" bitno jer je poželjno da se naredbe koje se moraju naknadno prevesti sve vreme nalaze u operativnoj memoriji i budu "pri ruci" prevodiocu. U slučaju većih programa, a relativno male operativne memorije može se javiti problem nedostatka memorijskog prostora koji se dalje teško rešava.

Vredno je na kraju primetiti da ALGOL-iki programski jezici nisu podesni za jednoprolaznu kompilaciju jer na primer skokovi unapred mogu bitno uticati na dužinu prevedenog programa. Nasuprot, u slučaju FORTRAN-olikih jezika poželjno je velike programe razbijati na delove i pisati u obliku manjih potprograma. Tu se naknadno prevodjenje vrši za vreme povezivanje programa i potprograma. Zato se ne treba čuditi što prevodjenje FORTRAN-skih programa relativno kratko traje, a povezivanje može da se oduži.

2.3. Leksička analiza

Prva faza kompilacije je leksička analiza. Ulazni podatak leksičkog analizatora je izvorni program. Leksički analizator čita izvorni program karakter po karakter i od njih obrazuje niz leksičkih konstrukcija (token-a). Leksička konstrukcija je niz susednih karaktera izvornog programa koji čine osnovnu logičku celinu. Identifikatori, službene reči, konstante, operatori, znaci interpretacije su primeri leksičkih konstrukcija.

Primer

Naredba:

IF(IKS.GT.7.3) IPS=-5

programskog jezika FORTRAN sadrži sledećih 9

IF,(,IKS,.GT.,7.3,),IPS,=,-5

leksičkih konstrukcija.

Šta će se proglasiti za leksičku konstrukciju prvenstveno zavisi od programskog jezika i projektanta prevodioca. U opštem slučaju, leksička konstrukcija je niska izvornog programa koja predstavlja nezavisnu jedinicu i kao takva se dalje koristi.

Primer

Nije prirodno u prethodnom primeru, recimo, za leksičke konstrukcije uzeti delove I, IK promenljive IKS ili celobrojni deo 7, odnosno razlomljeni deo .3 realne konstante 7.3.

Izlazni podatak leksičkog analizatora je niz leksičkih konstrukcija.

Mogu se uočiti dve vrste leksičkih konstrukcija. Prvu vrstu leksičkih konstrukcija čine tzv. specifične niske kao što su na primer službene reči (IF) ili interpunkcijiski znaci (zagrade).Drugu vrstu leksičkih konstrukcija čine opcione niske kao što su promenljive konstante, obeležja itd. Da bi se lakše radilo sa leksičkim konstrukcijama, bez obzira na vrstu, logičke konstrukcije se predstavljaju u obliku uredjenog para čiji je prvi elemenat vrsta leksičke konstrukcije.

leksička konstrukcija = [vrsta, vrednost] .

Primer

Leksička konstrukcija IKS iz prethodnog primera je tipa "identifikator (ime promenljive)" i ima vrednost "IKS"

IKS = [identifikator, IKS] .

Leksička konstrukcija IF, takodje iz prethodnog primera, je tipa "službena reč" i ima vrednost "IF"

IF = [službena reč, IF] .

Jednostavnosti radi leksičke konstrukcije prve vrste ne pišu se u obliku uredjenog para.

Leksička analiza i njoj naredna faza sintaksna analiza najčešće čine jedan prolaz. U tako realizovanom prolazu leksički analizator radi pod kontrolom sintaksnog analizatora. Leksički analizator po potrebi snabdeva leksičkim konstrukcijama sintaksni analizator. Pri tom leksički analizator blisko saradjuje sa blokom zaduženim za rad sa tabelama. Obavezni podaci koje leksički analizator daje sintaksnom analizatoru su:

- kôd vrste leksičke konstrukcije i
- vrednost (indeks mesta tabele simbola na kojem se čuva vrednost) leksičke konstrukcije.

2.3.1. Nalaženje leksičkih konstrukcija

Da bi našao sledeću leksičku konstrukciju, leksički analizator mora da ispita karaktere izvornog programa koji neposredno slede iza prethodno odredjene leksičke konstrukcije. Često nije dovoljno da leksički analizator ispita samo one karaktere koji pripadaju narednoj leksičkoj konstrukciji već i više drugih da bi utvrdio šta je zaista leksička konstrukcija. Postupak se ponavlja.

Primer

Da bi leksički analizator našao prvu po redu leksičku konstrukciju naredbe ciklusa:

DO1ØI=-5,+5

programskog jezika FORTRAN: DO mora redom da ispituje karaktere dotične naredbe zaključno sa zarezom. Tek onda je u stanju da tvrdi da je niska DO službena reč, a ne prefiks promenljive DO1ØI, tj. da je upitanju naredba ciklusa, a ne aritmetička naredba. Znači, leksički analizator je morao da bi utvrdio službenu reč DO dužine 2 slova da ispita još 7 narednih karaktera: 1ØI=-5,. Dalje, leksičkom analizatoru neće biti teško da utvrdi preostale leksičke konstrukcije: 1Ø,I,=,-5,=,-5,zarez,+5.

Po završetku leksičke analize dobija se sledeći niz leksičkih konstrukcija:

DO[obeležje,1Ø][promenljiva,I] = [konstanta,-5],[konstanta,+5]

Vrednosti leksičkih konstrukcija mogu se zadati indeksima tabele simbola u kojoj se čuvaju informacije o promenljivim, konstantama, obeležjima itd. U tom slučaju, prethodan niz leksičkih konstrukcija, može recimo izgledati:

DO[obeležje,217][promenljiva,15Ø] =[konstanta,5],[konstanta,13].

217,150,5,13 su odgovarajući indeksi tabele simbola prikazane na slici 2.6.

Tabela simbola

1	
⋮	
5	celobrojna, -5
⋮	
13	celobrojna, +5
⋮	
100	
101	
⋮	
150	celobrojna, I
⋮	
200	
201	
⋮	
217	1Ø
⋮	
300	

deo tabele simbola
rezervisan za konstante

deo tabele simbola
rezervisan za promenljive

deo tabele simbola
rezervisan za obeležje

Slika 2.6.

Ako se vrste leksičkih konstrukcija kôdiraju, recimo na sledeći način:

vrsta	kôd
konstanta	1
promenljiva	2
obeležje	3

prethodan niz leksičkih konstrukcija biće:

DO[3,217][2,150] = [1,5] , [1,13] .

Uobičajeno je da se leksičke konstrukcije prve vrste kôdiraju negativnim celim brojevima recimo na sledeći način:

leksička konstrukcija prve vrste	kôd	
⋮		Službene reči
DO	-4	
⋮		
=	- 36	Operatori
⋮		
,	- 42	Znaci interpunkcije itd.
⋮		

Konačno se dobija sledeći čisto brojevni zapis razmatranog niza leksičkih konstrukcija:

-4, 3, 217, 2, 150, -36, 1, 5, -42, 1, 13.

Ovaj niz je moguća leksička reprezentacija naredbe

DO 1Ø I = -5, +5

i predstavlja izlaz leksičkog analizatora, odnosno ulaz sintaksnog analizatora.

2.4. Sintaksna analiza

Sintaksni analizator izvršava, u principu, sledeća dva zadatka. Prvi zadatak mu je da kontroliše usaglašenost niza lek-

sičkih konstrukcija sa sintaksnom specifikacijom izvornog jezika. Drugi zadatak mu je da obrazuje drvoliku strukturu nad nizom leksičkih konstrukcija. Listovi drveta su leksičke konstrukcije, a koren drveta je početni simbol gramatike kojom je definisana sintaksa izvornog jezika. Drvolika struktura je izlazni podatak sintaksnog analizatora.

Primer

U programskom jeziku FORTRAN nije dozvoljen izraz oblika:

A*+C

gde su A i C promenljive, a * i + odgovarajući znaci aritmetičkih operacija. [4]

Leksičkom analizom dobija se sledeći niz leksičkih konstrukcija:

promenljiva * + promenljiva

koji je ulazni podatak sintaksnog analizatora.

Došavši do znaka +, sintaksni analizator otkriva grešku jer prema sintaksnim pravilima nije dozvoljeno postojanje dva susedna binarna operatora u jednom izrazu.

Naredni zadatak sintaksne analize je obrazovanje hijerarhijske strukture od nastupajućeg niza leksičkih konstrukcija sa utvrdjivanjem podnizova leksičkih konstrukcija od kojih se dalje obrazuju složenije konstrukcije jezika.

Primer

Izraz:

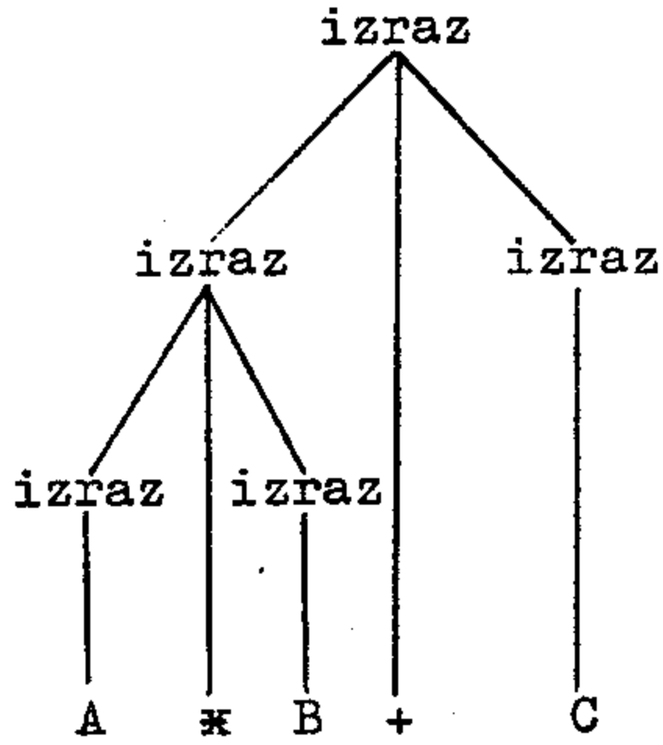
A*B+C

može se interpretirati na dva razna načina:

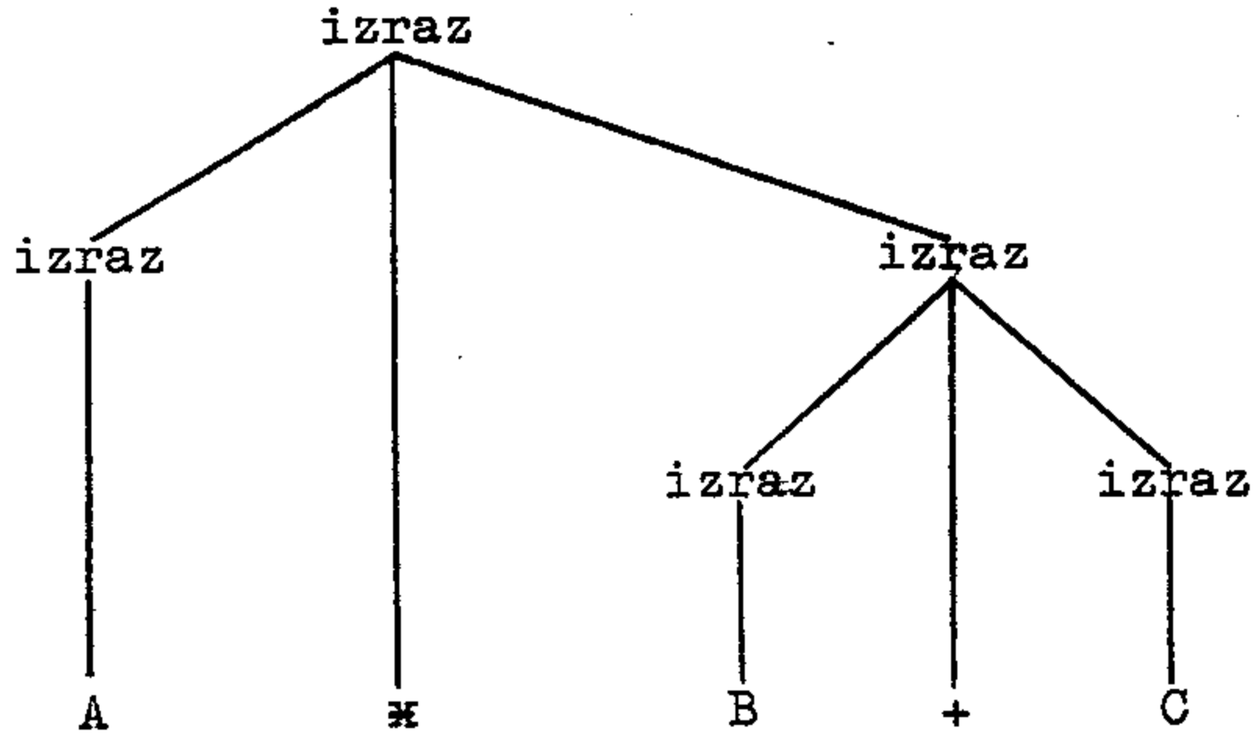
- pomnožiti A sa B pa dodati C (tako se na primer radi u programskom jeziku FORTRAN) ili
- sabrati B sa C pa množiti A sa prethodno dobijenom sumom (tako se na primer radi u programskom jeziku APL).

Svaka od navedenih interpretacija može se predstaviti

sintaksnim drvetom - dijagramom za predstavljanje sintaksne strukture izraza. Odgovarajuća sintaksna drveća prikazana su na slici 2.7. i 2.8.



Slika 2.7.



Slika 2.8.

Sintaksnom definicijom programskog jezika odredjena je interpretacija i hijerarhijska struktura izvornog programa.

2.5. Generisanje medjukôda *

Rezultat rada sintaksnog analizatora je reprezentacija izvorne naredbe ili programa u obliku sintaksnog drveta. Zadatak faze generisanja medjukôda je transformacija sintaksnog drveta u medjujezičku reprezentaciju izvorne naredbe odnosno programa.

2.5.1. Troadresni kôd

Najviše korišćen u praksi medjujezik je troadresni kôd. Tipične naredbe troadresnog kôda su oblika:

$$A := B \ op \ C.$$

A,B i C su operandi, a op je binarni operator.

Navedene naredbe troadresnog kôda služe za prevodjenje naredbi obrade izvornog programa.

Primer

Sintaksnom drvetu prikazanom na slici 2.7. odgovara sledeći niz naredbi troadresnog kôda:

$$p_1 := A * B$$
$$p_2 := p_1 + C.$$

Sintaksnom drvetu prikazanom na slici 2.8. odgovara sledeći niz naredbi troadresnog kôda:

$$p_1 := B + C$$
$$p_2 := A * p_1.$$

p_1 i p_2 su pomoćne promenljive.

Troadresni kôd takodje sadrži naredbe bezuslovnog i uslovnog skoka.

Naredba bezuslovnog skoka je oblika:

skok o

gde je o obeležje naredbe na koju se prenosi izvršavanje programa.

---

* Uobičajeni su još nazivi unutrašnji kôd, pomoćni kôd ili prelazni kôd.

Naredbe uslovnog skoka su oblika:

ako A re B skok o_1

skok o_2 .

A i B su operandi, re je realcija ($>$, $\geqslant$, $=$, $\neq$, $<$, $\leqslant$), a o_1 i o_2 su obeležja naredbi na koje se prenosi izvršavanje programa u zavisnosti da li je relacija ispunjena ili nije ispunjena.

Naredbe skoka se koriste za prevodjenje kontrolnih naredbi izvornog programa kao što su na primer:
goto, while-do, repeat-until, if-then-else itd.

Primer

```
while  N>Ø   do
begin
       N:=N-1;
       Y:=X*Y
end    ;
```

je kontrolna naredba PASCAL jezika.

Leksička reprezentacija ove naredbe je sledeći niz leksičkih konstrukcija:

while [promenljiva, n_1] > [konstanta, n_2] do

begin

[promenljiva, n_1] := [promenljiva, n_1] - [konstanta, n_3];

[promenljiva, n_4] := [promenljiva, n_5] * [promenljiva, n_4]

end;

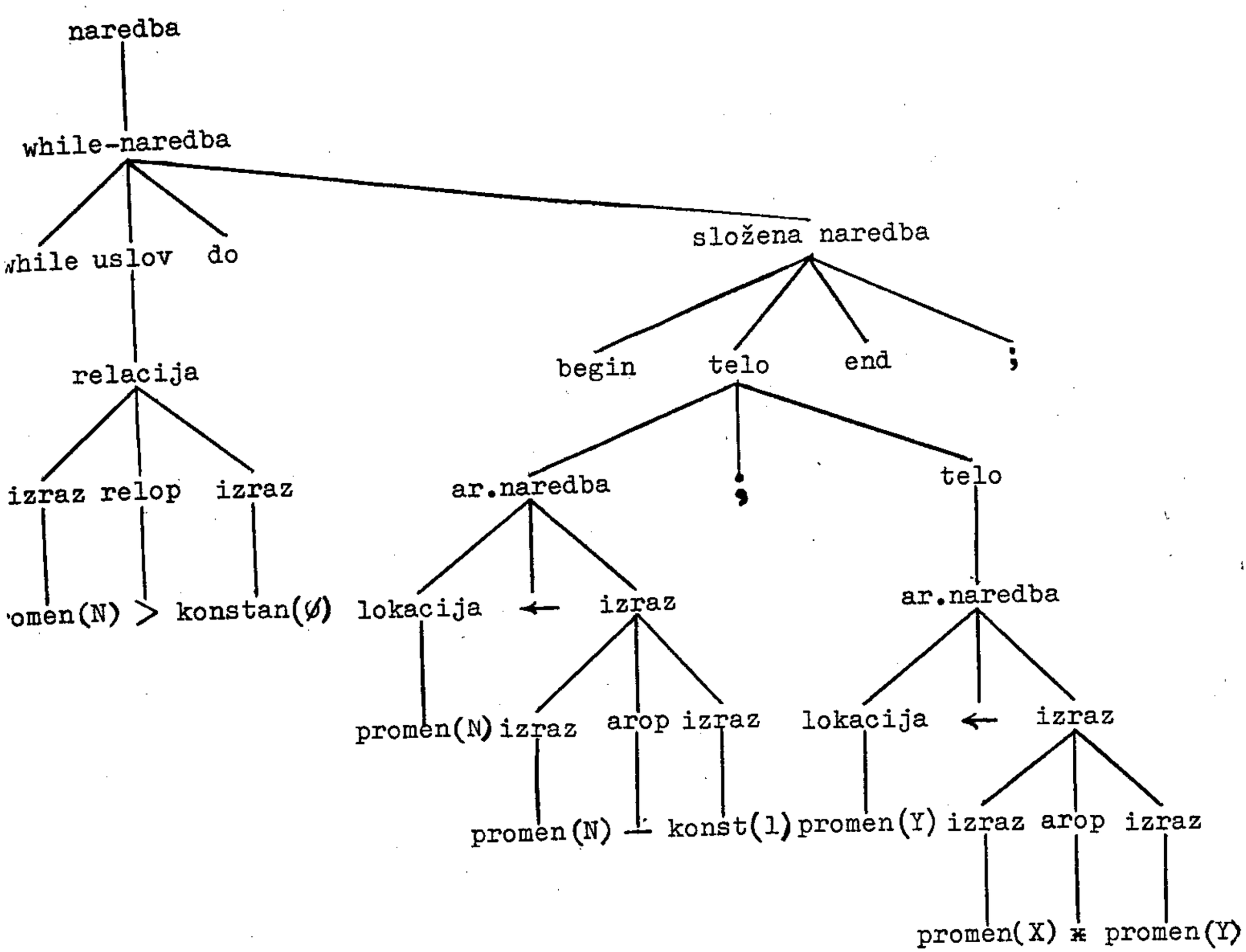
n_1, n_2, n_3, n_4, i n_5 su indeksi redova tabele simbola u kojima se čuvaju vrednosti za N, Ø, 1, Y i X .

Moguća sintaksna reprezentacija prethodne naredbe je sintaksno drvo prikazano na slici 2.9.

Zbog veličine sintaksnog drveta korišćene su skraćenice.

- arop za aritmetički operator i
- relop za relacijski operator.

Vrednosti promenljivih i konstanti stavljene su u zagrade. Sintaksno drvo može se transformisati u sledeći niz naredbi troadresnog koda:



Slika 2.9.

```
o1 : ako  N > Ø    skok o2
     skok  o3
o2 : N:= N-1
     Y:= X*Y
     skok o1
o3: ———————    produžetak
```

Dobijeni prevod nije optimalan jer postoje skokovi jedan preko drugog.

Većina kompilatora ne generiše u potpunosti sintaksno drvo već čim je to moguće medjukôd. Na taj način se povećava brzina kompilacije,a i manji su zahtevi za operativnim memorijskim prostorom računara.

2.6. Optimizacija

Opšta je želja da prevedeni (objekt) program bude što brži i da zauzima što manji memorijski prostor. Zato bolji kompilatori sadrže fazu opitimizacije medjukôda. Zadatak optimizatora je da izvrši transformaciju medjukôda na oblik iz kojeg se dobija brži i kraći prevedeni program.

Naziv optimizacija je u izvesnom smislu pogrešan jer je algoritamski ne rešiv problem optimalnog programa. Optimizovan program je bolji od njemu semantički ekvivalentnog neoptimizovanog programa u nekim karakteristikama ali to još uvek ne znači da je on apsolutno najbolji. Optimizaciju programa treba shvatiti kao pokušaj poboljšanja programa u pogledu brzine izvršavanja, zauzeća memorijskog prostora i drugih karakteristika. Optimizatori, koji su danas u upotrebi, mogu u principu 2 do 3 puta da poboljšaju neoptimizovane prevedene programe i daju kôd istog kvaliteta kao kad ga piše iskusan programer. Navedeni faktor nije za zanemarivanje specijalno u slučaju programa koji se često izvršavaju. Uštede mogu da budu velike.

2.6.1. Lokalna optimizacija

Lokalna optimizacija je najjednostavnija i često se koristi. Naziv lokalna potiče jer se koristi za optimizaciju delova naredbi programa, a ne celog programa. Lokalnu optimizaciju objasnimo na primerima:

Primer 1

Skok preko skoka kvari strukturiranost programa i nepotrebno ga produžava. Odstranjivanjem suvišnog skoka se umesto dve naredbe:

```
        ako  N > Ø   skok o2
        skok o3
o2:  ____________
```

dobija jedna naredba:

```
        ako  N ⩽ Ø   skok  o3
```

Primer 2

Izdvajanje podizraza, koji se javljaju u bar dva izraza, u poseban izraz je lep primer lokalne optimizacije.

Neka su date sledeće dve naredbe dodele:

```
A:= B+C+D+E;
F:= B+G+D+H;
```

programskog jezika PASCAL. Nije teško uočiti da se izraz:

```
B+D
```

javlja u oba izraza sdesne strane znaka dodele. Jednostavnim transformacijama:

```
A:= B+D+C+E;
F:= B+D+G+H;
```

tj.

```
P:= B+D;
A:= P+C+E;
F:= P+G+H;
```

umesto dve naredbe prividno se dobijaju tri naredbe i nova promenljiva P ali je prevedeni program kraći. Zaista, u prvom slučaju prevod je:

```
p1:= B+C
p2:= p1+D
A:=  p2+E
p3:= B+G
p4:= p3+D
F:=  p4+H
```

tj. 6 naredbi + 4 pomoćne promenljive.

u drugom slučaju prevod je:

```
P:= B+D
p1:= P+C
A:= p1+E
p2:= P+G
F:= p2+H
```

tj. 5 naredbi + 3 pomoćne promenljive.

Neko može primetiti da iskusan programer neće nikad nešto slično gore navedenom napisati. To je tačno, ali iz toga još ne sledi da lokalna optimizacija nema smisla, odnosno da se retko može primeniti. Zaista, kompilator je često u situaciji da

sam generiše podizraze. Na primer, naredba dodele programskog jezika PASCAL:

A[I]:= B[I]+C[I]

A[I], B[I] i C[I] su realne indeksne promenljive u slučaju računara čija je dužina registra 2 bajta, zahteva od kompilatora da tri puta generiše izraz

2*I

i na taj način svakoj realnoj indeksnoj promenljivoj dodeli po 2 registra,odnosno 4 bajta. Optimizator tu može da pomogne da se izraz 2*I generiše samo jedanput. Primetimo da programer nema mogućnosti da na nivou izvornog programa saopšti kompilatoru da izraz 2*I jedanput generiše, a ne svaki put kad naidje indeks I.

2.6.2. Optimizacija petlji

Optimizacija petlji je složenija od lokalne optimizacije, ali kad se uspe izvršiti puno doprinosi povećanju brzine izvršavanja programa. Ideja optimizacije petlji sastoji se u sledećem:

- ako postoji, naći invarijantu petlje,
- invarijantu petlje staviti ispred, odnosno iza petlje.

Invarijantu petlje čine naredbe tela petlje čijim se iznošenjem iz petlje ne menja semantika programske celine.

Primer

Naredba AS:=S/N je invarijanta petlje:

```
for  I:=1 to N do
begin
     S:=S+X[I];
    AS:=S/N
end;
```

Iznošenjem invarijante iz petlje i stavljanjem iza petlje ne menja se semantika programske celine:

```
for  I:=1 to N
do   S:=S+X[I];
    AS:=S/N;
```

a izvršava se optimizacija petlje.

2.7. Generisanje kôda

U fazi generisanja kôda izvršava se prevodjenje (transformacija) medjukôda u niz naredbi mašinskog jezika.

Primer

Generator koda, obično na sledeći način, prevodi model aritmetičke naredbe:

A:=B op C

troadresnog kôda u niz mašinski-orijentisanih (simboličkih) naredbi jednoadresnog računara:

MUA B
OP C
AUM A .

op je binarni operator, odnosno znak aritmetičke operacije,
OP je mnemokôd odgovarajuće aritmetičke operacije.

Uobičajena je sledeća veza izmedju op i OP:

op	OP
+	SAB
-	ODU
*	MNO
/	DEL

Navedeno prevodjenje medjukôda u mašinski kôd podseća na razvijanje makro-naredbe. Dobijeni prevod sadrži niz suvišnih naredbi upisa u akumulator (MUA) i uzimanja iz akumulatora (AUM). Ove naredbe nepotrebno zauzimaju operativnu memoriju računara i usporavaju izvršavanje programa.

Primer

Direktan prevod naredbi:

A:= B+C
E:= A-D

je:

```
MUA   B
SAB   C
-----------
AUM   A
MUA   A
-----------
ODU   D
AUM   E
```

Očigledno je da su naredbe:

```
AUM   A   i
MUA   A
```

suvišne i mogu se izostaviti.

Prevod u tom slučaju je:

```
MUA   B
SAB   C
ODU   D
AUM   E
```

Problem suvišnih naredbi u principu se može rešiti na sledeća dva načina:

Prvi način je uvesti dodatnu fazu optimizacije mašinskog kôda. Zadatak novouvedenog optimizatora je da pronalazi grupe naredbi bez dejstva i izostavlja ih iz programa.

Drugi način je da generator koda vodi evidenciju o tekućem sadržaju registara. Imajući informacije o sadržajima registara, generator kôda može da spreči generisanje suvišnih naredbi.

U praksi se drugi način češće koristi.

Mnogi računari imaju izvestan broj tzv. "brzih" registara sa kojima je računanje daleko brže. Dobar generator kôda treba ove registre što je moguće efikasnije iskoristiti. Navedeni problem poznat je u literaturi kao problem raspodele registara i izuzetno teško se optimalno rešava.

2.8. Vodjenje tabela

Kompilator skuplja informacije o svim podacima koji se nalaze u izvornom programu. Tako, na primer, kompilator treba da zna koje je vrste promenljiva, kolika je dimenzija niza, koliko argumenata sadrži funkcija itd.

U principu, informacije se zadaju na tri načina:
- unutrašnjom konvencijom, tj. podrazumevaju se,
- implicitno i
- eksplicitno.

Primer

U FORTRAN-jeziku vrsta promenljive odredjena je:

a) unutrašnjom konvencijom, tj. ako ime promenljive počinje slovom I, J, K, L, M ili N promenljiva je celobrojna, u suprotnom promenljiva je realna,

b) implicitnom deklaracijom po početnom slovu imena promenljive i

c) eksplicitnom deklaracijom po konkretnom imenu promenljive.

Primer

U BASIC-jeziku niz koji sadrži manje od deset članova ne mora se dimenzionisati. Niz koji sadrži deset i više članova mora se dimenzionisati.

Informacije se skupljaju u fazi leksičke i sintaksne analize izvornog programa i upisuju u tabelu simbola. Na primer, kada leksički analizator otkrije (prepozna) identifikator-promenljivu IKS treba da proveri da li ime promenljive IKS već postoji u tabeli simbola i u slučaju da ne postoji, unese ga. Vrednost imena promenljive odredjena je rednim brojem (indeksom) reda tabele simbola koji sadrži dato ime promenljive. Neka je dalje vrsta promenljive IKS odredjena eksplicitnom deklaracijom REAL IKS. Kada sintaksni analizator prepozna deklaraciju REAL upisaće na odgovarajućem mestu tabele simbola da je promenljiva IKS realna. Sakupljene informacije koriste se u narednim fazama.

Primer

Neka je dat izraz:

A+B

A je celobrojna promenljiva, a B je realna promenljiva.

Da bi se izračunala vrednost izraza A+B, neophodno je da su vrednosti obe promenljive iste vrste. Unutrašnjom konvencijom propisan je prioritet izmedju različitih vrsta promenljivih (realno je starije od celobrojnog) pa je neophodno generisati kôd za prevodjenje vrednosti promenljive A iz fiksnog u pokretni zarez. Sabiranje će se izvršiti u pokretnom zarezu, a dobijeni rezultat biće realan broj.

U slučaju programskih jezika kod kojih nisu dozvoljeni mešoviti izrazi, kompilator mora da ih otkriva i šalje prigodan izveštaj o grešci.

Semantička analiza koristi se za:

- odredjivanje vrste medjurezultata,
- kontrolisanje usaglašenosti argumenata i operatora i
- odredjivanje odgovarajuće operacije operatora.

Semantička analiza proteže se na fazu sintaksne analize, generisanja medjukôda i generisanje kôda.

Primer

U programskom jeziku BASIC znakom + istovremeno se označavaju operacije:

- sabiranje u fiksnom zarezu,
- sabiranje u pokretnom zarezu i
- dopisivanje znakovnih podataka.

Slično znak = koristi se kao:

- operator dodeljivanja,
- relacijski operator.

2.9. Obrada grešaka

Jedan od najvažnijih zadataka kompilatora je:

- blagovremeno otkrivanje grešaka i
- prigodno izveštavanje o otkrivenim greškama.

Izveštaj o grešci treba eksplicitno da ukaže na:

- mesto nastanka greške,
- vrstu greške i
- načine otklanjanja greške.

Greške se mogu otkriti u svim fazama kompilacije.

Na primer:

- leksički analizator otkriva simbole koji nisu iz azbuke jezika ili uvidja da je promenjen očekivan redosled simbola;
- sintaksni analizator nije u stanju da konstruiše drvo izvodjenja za naredbu izvornog programa jer je na nekom mestu izostavljena odgovarajuća leksička konstrukcija;
- generator medjukôda može da otkrije da su argumenti datog operatora neodgovarajućeg tipa;
- optimizator kôda može da otkrije suvišne naredbe tj. naredbe koje se za vreme izvršavanja programa nikad ne izvršavaju;
- generator kôda pronalazi konstante koje po vrednosti prelaze ograničenja konkretnog računara;
- potprogrami zaduženi za vodjenje tabela otkrivaju višestruko deklarisane identifikatore, izostavljena obeležja naredbi itd.

Kad god se u nekoj fazi otkrije greška o tome se obaveštava potprogram zadužen za obradu grešaka koji izdaje odgovarajuće izveštaje. Po otkrivanju greške u naredbi, bolji kompilatori nastavljaju sa daljom analizom naredbe težeći da otkriju moguće preostale greške. Ova analiza je izuzetno teška i vezana je sa opasnošću otkrivanja nepostojećih grešaka koje mogu samo zbuniti korisnika. Na primer, ako se zaboravi dimenzionisati niz kompilator će javiti greške u svim naredbama u kojima se niz koristi iako naredbe mogu biti bez greške!

2.10. Struktura interpretatora

Interpretator se obično sastoji iz sledeća tri bloka:

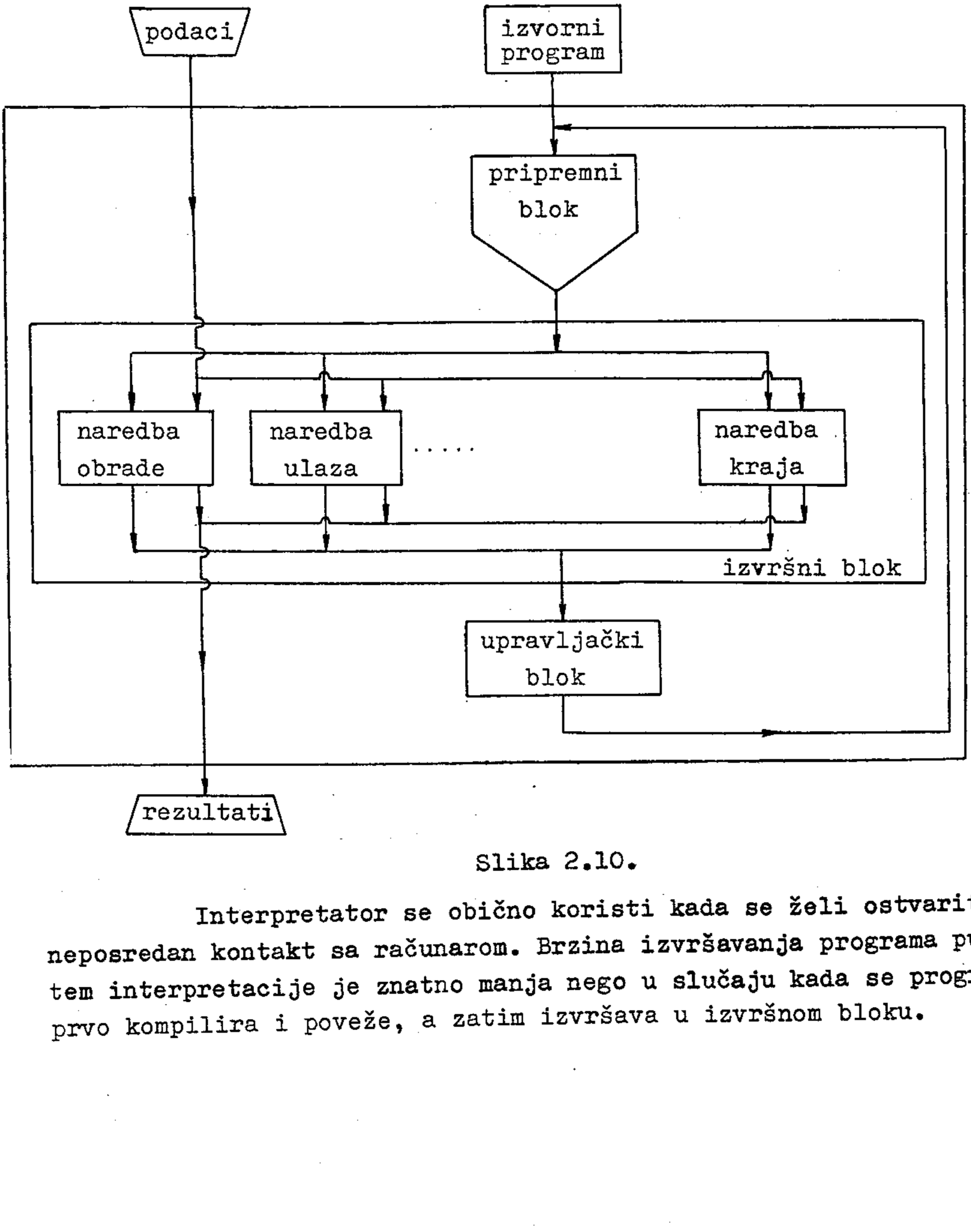
- pripremnog,
- izvršnog i
- upravljačkog.

Pripremni blok služi za prepoznavanje naredbi izvršnog programa.

Izvršni blok se sastoji iz niza podblokova, pri čemu je svaki podblok zadužen za odredjenu vrstu naredbi izvornog jezika.

Upravljački blok utvrdjuje redosled izvršavanja naredbi izvornog programa i uopšte rukovodi celokupnim radom interpretatora.

Struktura interpretatora prikazana je na slici 2.10.



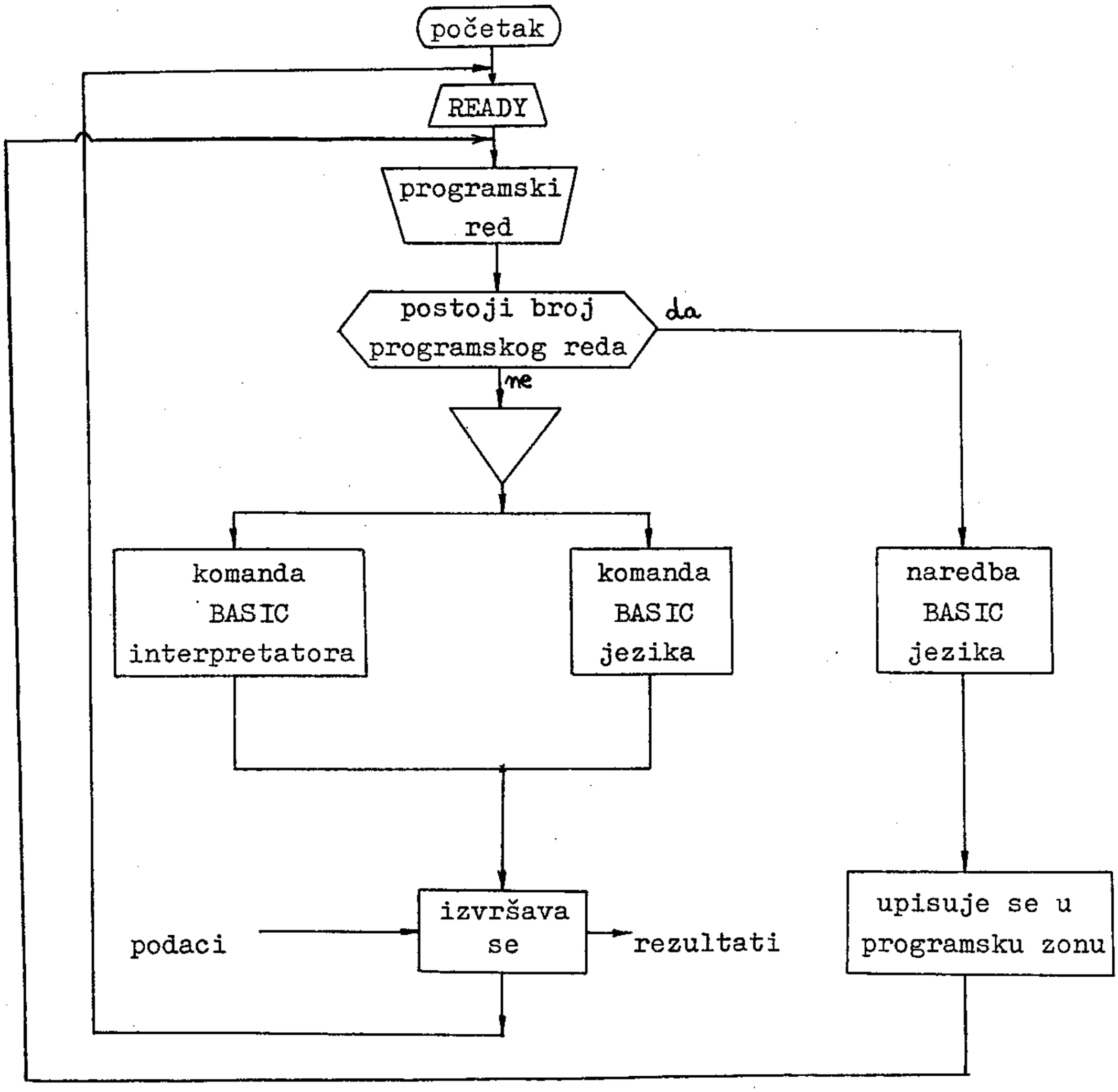
Slika 2.10.

Interpretator se obično koristi kada se želi ostvariti neposredan kontakt sa računarom. Brzina izvršavanja programa putem interpretacije je znatno manja nego u slučaju kada se program prvo kompilira i poveže, a zatim izvršava u izvršnom bloku.

Primer

Programski jezik BASIC najčešće se realizuje pomoću interpretatora. Ovo je uslovljenom strukturom i namenom BASIC-jezika.

Struktura BASIC-interpretatora prikazana je na slici 2.11. [129]



Slika 2.11.

2.11. Pribor za pisanje prevodilaca

Danas u svetu postoji bogata kolekcija pribora za pisanje prevodilaca. Svaki pribor je izmišljen sa ciljem da pomogne čoveku u konstruisanju prevodilaca. Zavisno od širine primena, pribori se mogu razvrstati u više grupa počev od relativno jednostavnih generatora, leksičkih i sintaksnih analizatora do veoma složenih sistema kao što su kompilatori-kompilatora, generatori-kompilatora i sistemi za pisanje prevodilaca koji omogućavaju pravljenje kompilatora na osnovu zadatih specifikacija izvornog i ciljnog jezika i postupka kompilacije. Ulazna specifikacija mora da sadrži:

1) opis leksičke i sintaksne strukture izvornog jezika;

2) opis kompilacije i rezultata kompilacije svake konstrukcije izvornog jezika;

3) opis računara na kojem će se dalje izvršavati (interpretirati) dobijeni rezultati kompilacije.

U većini slučajeva, specifikacija se sastoji iz kolekcije programa ugradjenih u skelet kompilatora-kompilatora.

Medjutim, najsavremeniji kompilatori-kompilatora dozvoljavaju da deo specifikacije bude zadat na neproceduralan način. Na primer, umesto da napiše program za sintaksnu analizu konstrukcije nekog jezika, korisnik će napisati gramatiku željenog jezika, a kompilator-kompilatora će umesto njega na osnovu zadate gramatike generisati program za sintaksnu analizu. Prednosti drugog načina rada - pristupa su više no očigledne.

Iz činjenice da danas u svetu postoji veći broj raznih kompilatora-kompilatora može se zaključiti da svi oni imaju izvesne nedostatke, odnosno ograničenja. Po mišljenju većeg broja stručnjaka glavni problemi masovne primene kompilatora-kompilatora su:

1) koji deo ukupnog posla kompilator-kompilatora može izvršiti automatski i

2) koliko je celokupan sistem fleksibilan, tj. za koju klasu ulaznih i izlaznih podataka je kompilator-kompilatora predvedjen.

Primer

Problem leksičke analize programskih jezika je u potpunosti rešen. Leksički analizatori, bez obzira o kojem se programskom jeziku radi, u principu su isti. Razlike su u specifičnosti službenih reči i simbola konkretnog programskog jezika. Da bi se razlike prevazišle specifičnosti se uzimaju za ulazne podatke.

Osnovni elementi pribora za pisanje kompilatora su:

1. Generator leksičkog analizatora,
2. Generator sintaksnog analizatora i
3. Generator kôda.

Prednost primene navedenih generatora je velika pouzdanost dobijenih programa.

2.12. Primeri poznatih prevodilaca

1. Jedan od prvih sovjetskih prevodilaca bio je Alfa--prevodilac. Konstruisan je u periodu od 1961. do 1964. godine u Računarskom centru Sibirskog odeljenja Akademije nauka SSSR pod rukovodstvom A.P.Eršova. Na slici 2.12. grafički je prikazana struktura Alfa-prevodioca. [29]

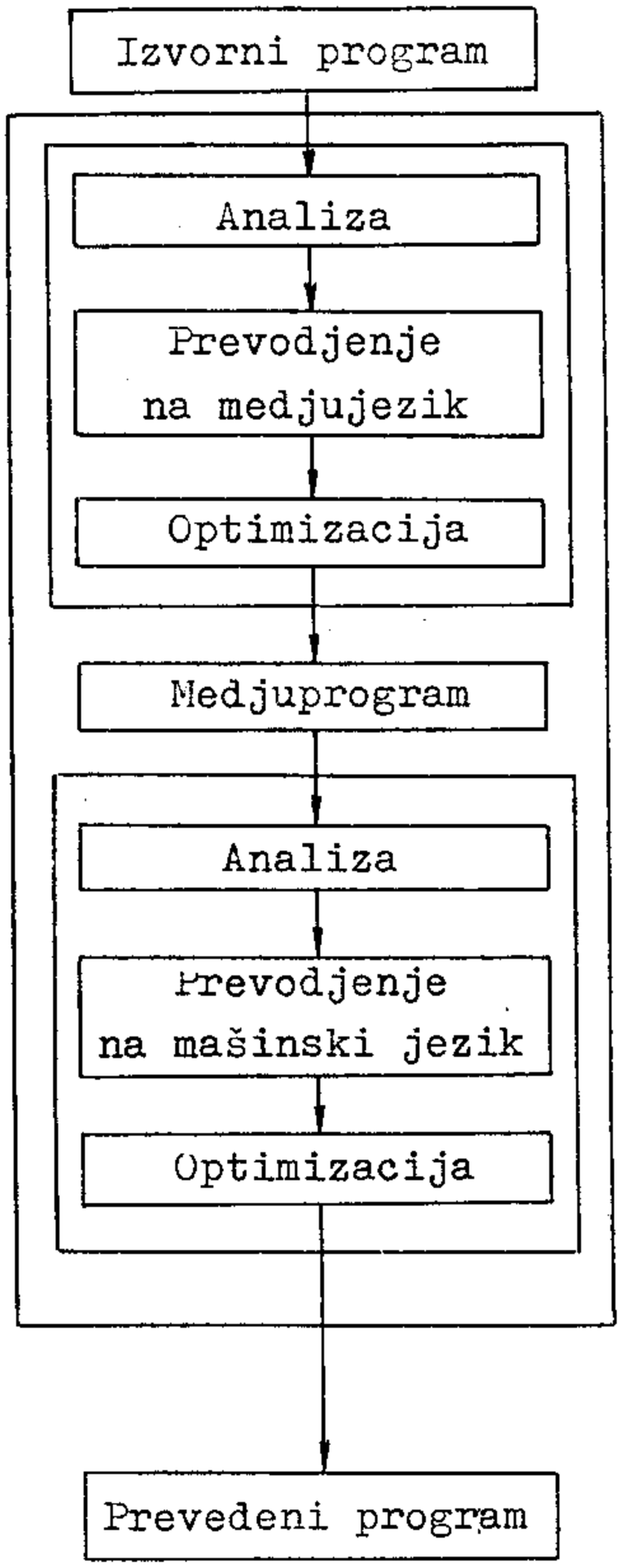
Proces prevodjenja odvija se u dva prolaza.

Prvi prolaz je prevodjenje programa sa izvornog jezika na medjujezik. Drugi prilaz je prevodjenje programa sa medjujezika na mašinski jezik.

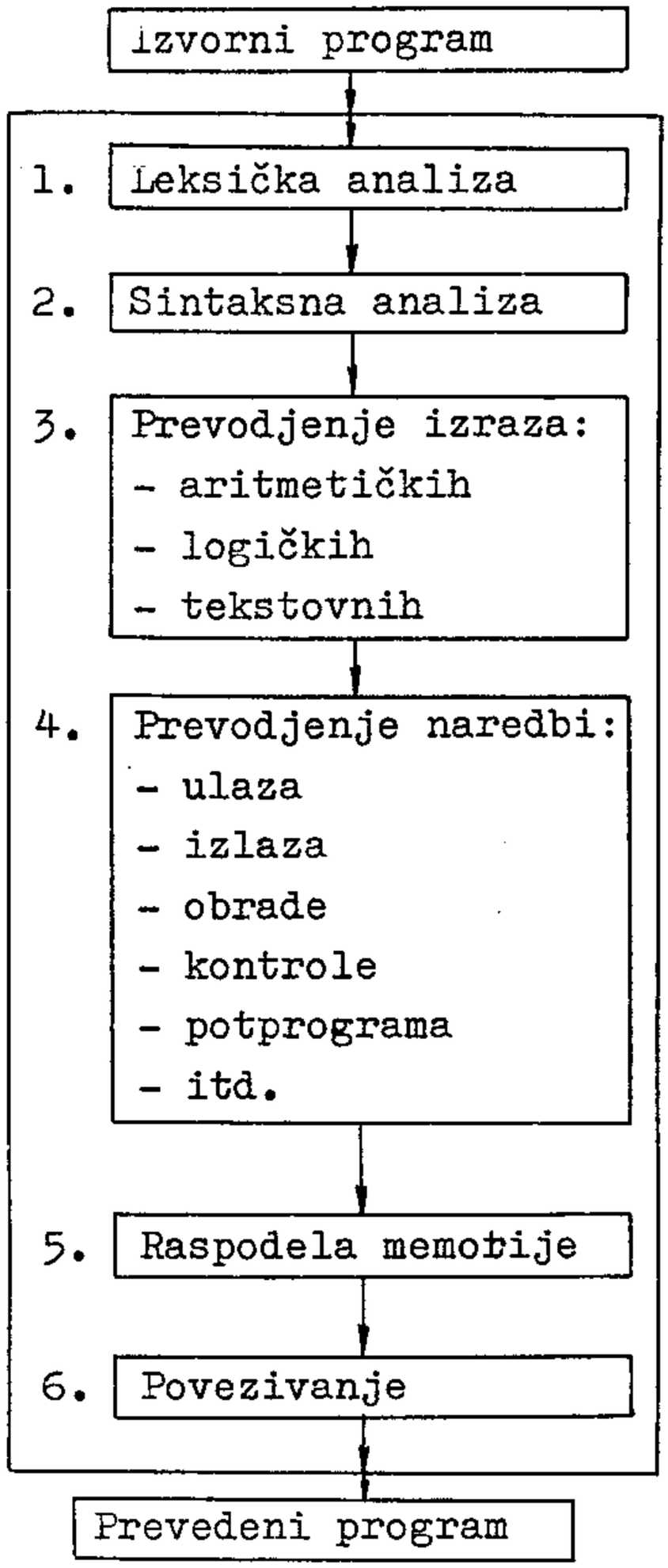
Svaki prolaz sastoji se iz tri etape. U prvoj etapi izvršava se leksička i sintaksna analiza, u drugoj prevodjenje, a u trećoj optimizacija.

Izvorni jezik je ALGOL.

2. ALGOL-60 prevodilac TA-2 za računar M-20 konstruisan je u Institutu za primenjenu matematiku ANSSSR. Projektom je rukovodio Šura-Bura, a izgradnja je trajala od 1961. do 1963. godine. Prevodilac je višeprolazan. Svaki prolaz čini jedan blok. Blokovi se nalaze na magnetnoj traci i redom se izvršavaju. Svaki blok se izvršava samo jedanput i tom prilikom obradi ceo program. Struktura prevodioca TA-2 prikazana je na slici 2.13.



Slika 2.12.



Slika 2.13.

Objasnimo, ukratko, navedene blokove. Izvorni program napisan je na ALGOL-u i nalazi se na karticama. Blok 1 ima zadatak da pročita program sa čitača kartica, prevede ga na medjujezik, smesti na magnetni doboš i izda program sa izveštajima o postojećim greškama na štampaču.

Blok 2 proverava sintaksnu ispravnost programa i konstruiše strukture pogodne za dalju obradu.

Blokovi 3 i 4 prevode na mašinski jezik odredjene konstrukcije izvornog programa.

Blok 5 izvršava sintaksnu raspodelu memorijskog prostora. Statička raspodela memorijskog prostora zahteva dimenzionisanje i deklarisanje svih nizova i promenljivih. Ako u operativnoj memoriji računara nema dovoljno mesta za smeštaj nizova, deo nizova se smešta na spoljne memorijske medijume.

Blok 6 prevodi program iz relativnih adresa u apsolutne adrese. Od posebnih delova obrazuje se radni program spreman za izvršavanje.

Prevodilac zauzima oko 20000 memorijskih registara od čega 3000 otpada na razne tabele. Brzina prevodjenja je 40 - 50 naredbi u minutu.

Struktura prevodioca TA-2 je uslovljena veličinom operativne memorije računara M-20. Operativna memorija računara M-20 sastoji se iz svega 4K = 4096 memorijskih registara, što je 5 puta manje od ukupne veličine programa.

3. C-kompilator. C je univerzalan programski jezik. Izmislio ga je D.M.Ritchie, a koristi se kao osnovni programski jezik UNIX operativnog sistema. C-kompilator se koristi na raznim računarima, a najčešće kod serija: PDP-11, Honeywell-6070 i IBM-370. [81]

Struktura C-kompilatora prikazana je na slici 2.14.

C-kompilator je troprolazni kompilator. I prolaz se sastoji iz:

- leksičke analize,
- sintaksne analize i
- generisanja medjukòda.

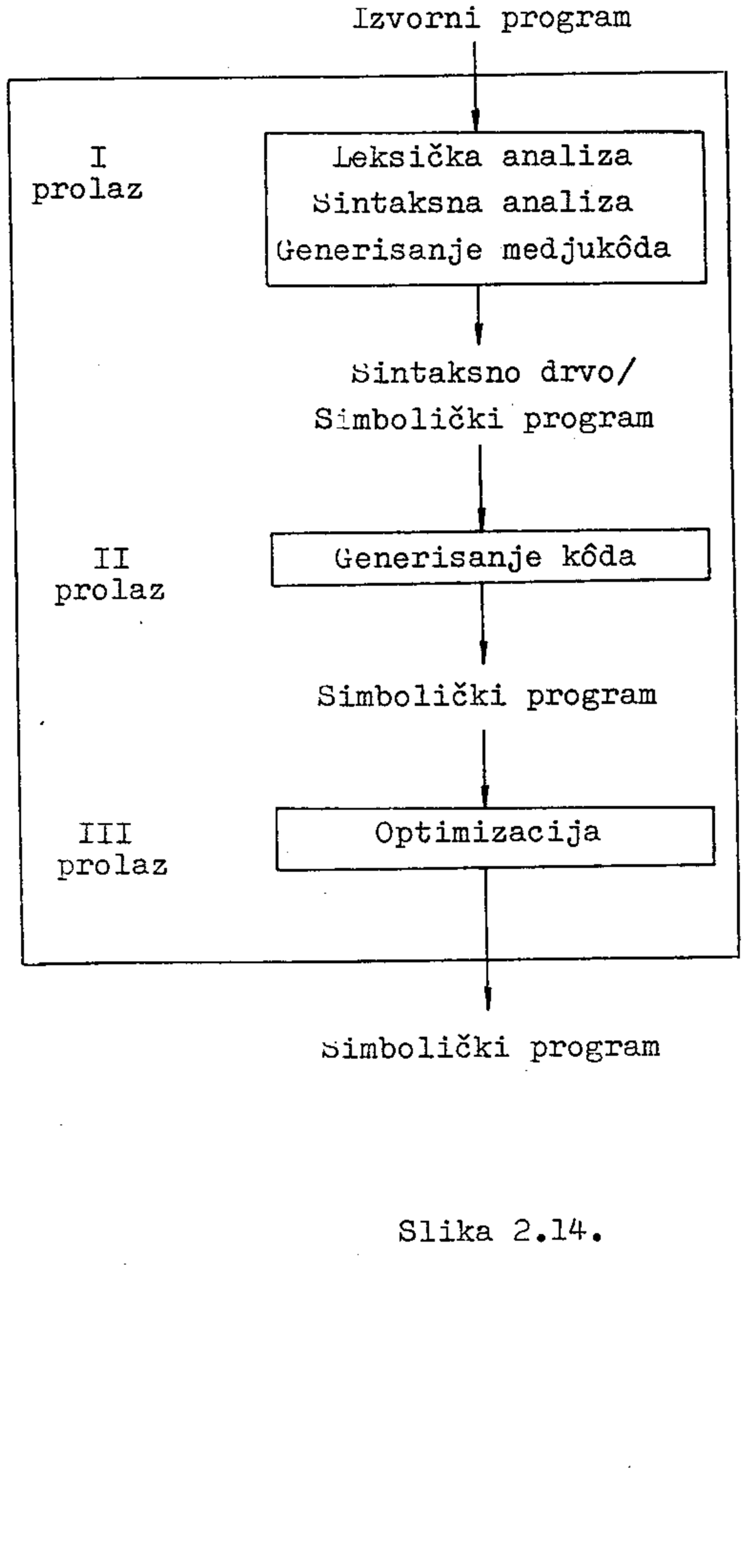
Analiza izraza vrši se metodom prioriteta operatora, a naredbi silaznom rekurzijom.

Medjukôd izraza je sintaksno drvo, a naredbi simbolički kôd.

II prolaz je generisanje kôda. Kôd se generiše iz sintaksnog drveta metodom obeležavanja.

III prolaz je optimizacija generisanog kôda. Optimizacija se sastoji iz niza transformacija koje imaju za cilj da odstrane:

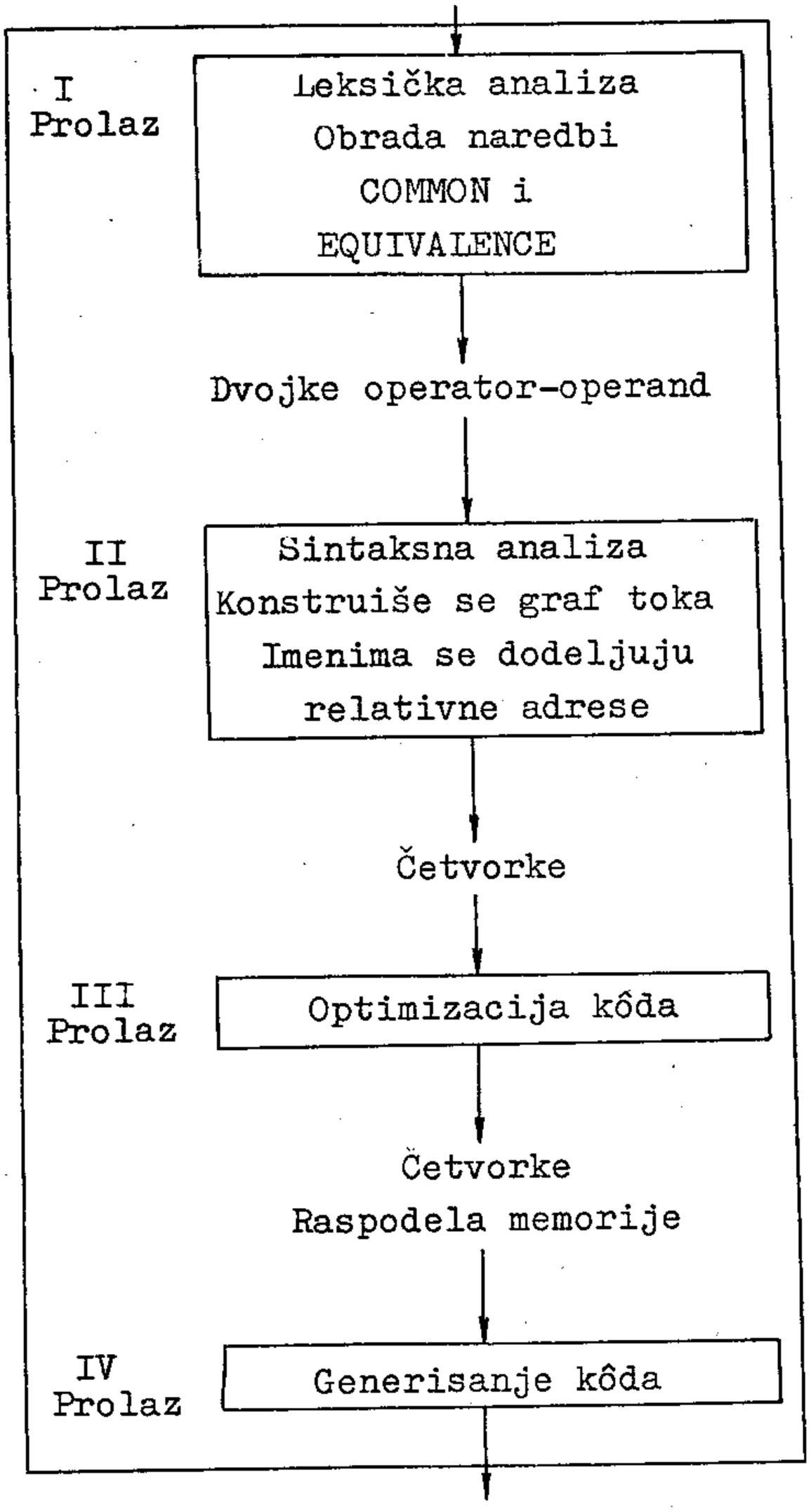
- suvišne naredbe skoka,
- naredbe koje se nikad neće izvršiti.



Slika 2.14.

4. FORTRAN H-kompilator. FORTRAN H-kompilator projektovan je za računare serije IBM-360. [49]. Kompilator se sastoji iz trinaest potprograma. Jedan od potprograma je stalno prisutan u operativnoj memoriji i upravlja preklapanjem i uopšte izvršavanjem drugih potprograma. Potprogrami su grupisani u 4 prolaza pri čemu neki potprogrami ulaze u sastav više prolaza.

Struktura H-kompilatora prikazana je na slici 2.15.



U I prolazu izvršava se leksička analiza. Rezultat leksičke analize je niz dvojaka oblika (operator, operand). Pod operatorima se podrazumevaju ne samo uobičajeni operatori već i znaci interpunkcije. Na primer, aritmetička naredba A=B(I)+C transformiše se u niz:

"aritmetička naredba"	A
=	B
(i	I
)	-
+	C

Leksički analizator pravi razliku izmedju leve zagrade uvedene zbog korišćenja indeksa i leve zagrade uvedene zbog grupisanja operanada. Sa (i je obeležena leva zagrada koja odgovara indeksu. Desna zagrada nema odgovarajućeg operanda. Ne pravi se razlika izmedju desnih zagrada različite vrste.

Slika 2.15.

U I prolazu takodje se obradjuju naredbe COMMON i EQUIVALENCE. Njihova obrada se sastoji u upisivanju odgovarajućih informacija u tabelu simbola.

U II prolazu izvršava se sintaksna analiza i generisanje troadresnog kôda. Naredbe troadresnog kôda prikazuju se u obliku četvorki. Na primer, naredbi A:=B+C odgovara četvorka (+,B, C,A). Kako FORTRAN-jezik ne sadrži strukturne kontrolne naredbe kao što su na primer while-do ili repeat-until sintaksna analiza je relativno jednostavna. Sintaksni analizator zasnovan je na metodi prioriteta operatora.

Takodje, u II prolazu konstruiše se graf toka. Četvorke se grupišu u osnovne blokove i utvrdjuju se imena koja se koriste u bloku. Imenima se dodeljuju relativne adrese.

U III prolazu izvršava se optimizacija kôda. Koriste se metode lokalne optimizacije. Pronalaze se i izbacuju iz programa zajednički podizrazi i invarijantni delovi ciklusa. Umesto elementarnih funkcija stavljaju se odgovarajući kôdovi. Stepenovanje celim brojem zamenjuje se nizom množenja, a množenje i deljenje zamenjuje se šiftovanjem.

U IV prolazu izvršava se generisanje kôda. Iz četvorki i prethodno izvršene raspodele memorije dobija se izvršni program u relativnim adresama.

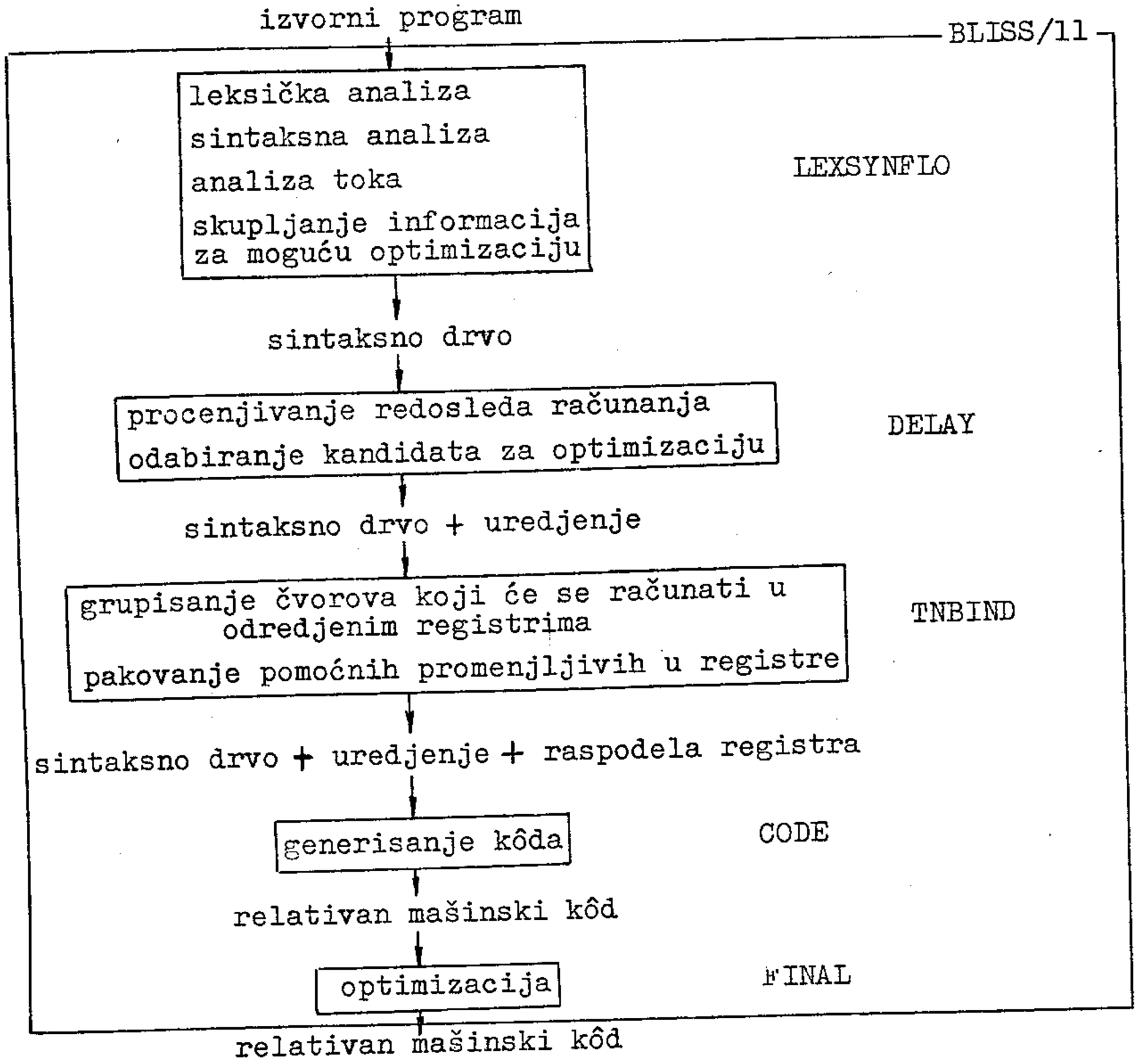
5. BLISS/11-kompilator. BLISS je sistemski programski jezik. Wulf, W. [97] je izmislio BLISS-jezik i konstruisao kompilator za seriju računara PDP-11. Pri projektovanju BLISS-jezika i kompilatora posebno se vodilo računa da se dobije što je moguće kraći prevod izvornog programa pri čemu on ne mora da bude i najbrži pri kasnijem izvršavanju. BLISS-kompilator je namenjen računarima sa malom operativnom memorijom i primenama kod kojih se ne zahteva posebna brzina izvršavanja prevedenih programa.

Kompilator je jednoprolazan i sastoji se iz 5 faza.

Struktura BLISS/11-kompilatora prikazana je na slici 2.16.

U fazi LEXSYNFLO izvršava se leksička i sintaksna analiza. Sintaksni analizator zasniva se na metodi rekursivnog silaženja. BLISS ne dozvoljava GOTO naredbu pa su sve algoritamske

šeme BLISS-procedura rastavljive. Sledeći važan zadatak LEXSYNFLO je otkrivanje grupa sličnih izraza. Ove grupe su kandidati za zamenu sa jednim posebnim potprogramom.



Slika 2.16.

Zamenom će se učiniti da se program nešto sporije izvršava ali i uštedeti memorijski prostor računara.

U fazi DELAY ispitivanjem sintaksnog drveta dolazi se do zaključka koju metodu optimizacije treba primeniti. Takodje se utvrdjuje redosled izvršavanja izraza. On se zasniva na metodi strategija obeležavanja.

U fazi TNBIND povezuju se privremena imena i registri. Pri tome se koristi strategija da se prva grupa čvorova sintaksnog drveta dodeli istom registru. Dokazana je prednost dodeljivanja potomcima registra koji su nekad koristili preci.

U fazi CODE vrši se transformacija drveta izvodjenja na osnovu uredjenja i dodeljenih registara u relativan mašinski kôd.

U fazi FINAL izvršava se dodatno ispitivanje relativnog mašinskog koda u cilju poboljšanja. Poboljšanja mogu da budu:

- prepravljanje skoka na skok,
- komplementiranje uslova koje povlači gubljenje skoka,
- eliminisanje suvišnih ili nikad izvršivih naredbi.

U slučaju računara PDP-11 vrši se kad god je moguće zamena naredbe "jump" naredbom "branch" i na taj način se štedi 1 bajt.

3. BELEŽENJE SINTAKSE PROGRAMSKIH JEZIKA

3.1. Nedostaci prirodnih jezika koji su uzrokovali nastanak veštačkih jezika

Vekovima su se u naukama isključivo koristili prirodni jezici. Medjutim, postepeno se uvidelo da je korišćenje prirodnih jezika u naukama vezano za niz teškoća. Navodimo neke od tih teškoća.

Pri analizi rečenice razlikuje se:
- forma (struktura, oblik) i
- sadržaj (značenje, smisao).

U slučaju prirodnih jezika dogadja se da jednoj formi odgovara više sadržaja. Na primer rečenica: "Sve je jasno o ubistvu profesora" nije jednoznačna. Iz nje se ne vidi da li je profesor ubijen ili je on izvršio ubistvo.

Mnoge rečenice prirodnog jezika nisu višeznačne ali **nisu ni sasvim jednoznačne. Na primer rečenica: "Školska tabla je zelena" samo približno odredjuje boju table, jer kao što se zna postoji niz nijansi zelene boje.**

U nauci se obično koristi pisani, a ne govorni oblik jezika. Pisani oblik prirodnog jezika lišen je nekih mogućnosti govornog jezika što ide na račun povećanja nejednoznačnosti. Na primer rečenica: "Tamo gore, gore, gore" bez označavanja naglaska reči gore skoro da nema smisla. Naravno postoje pravila za označavanje naglaska ali sad to ide na račun povećanja složenosti pisanja prirodnog jezika.

Gramatička pravila mogu da zavise od značenja reči odnosno rečenica. Na primer pravilno se kaže "Vidim automobil" ili

"Vidim konja", a ne "Vidim konj".

Zavisnost forme gramatičkih pravila od sadržaja rečenice je neželjena osobina prirodnih jezika, posebno onda kada se želi vršiti formalno prevodjenje prirodnih jezika.

Na kraju, navedimo poznati paradoks Beri-a: "Neka je n najmanji prirodan broj koji se ne može definisati na srpsko-hrvatskom jeziku sa manje od 30 reči".

Ovom rečenicom od 20 reči okarakterisan je na srpsko-hrvatskom jeziku prirodan broj n, koji se ne može definisati na srpsko-hrvatskom jeziku sa manje od 20 reči. Paradoks je očigledan.

Znači, prirodni jezici su nejednoznačni i neprecizni. Razumevanje nekih rečenica iskazanih na prirodnom jeziku, zahteva (podrazumeva) poznavanje situacije u odgovarajućoj oblasti u vreme koje je rečenica izrečena.

Načini savladjivanja navedenih nedostataka prirodnih jezika otkrivaju se praćenjem razvoja prirodnih jezika. U nedrima prirodnih jezika, na njihovoj osnovi, korišćenjem rečničkog fonda i gramatika, u cilju obezbedjenja potreba egzaktnih nauka počeli su nastajati podjezici, čije su rečenice jednoznačne, a sadržaji ne zavise od spoljnjih uslova. Sadržaji rečenica takvog podjezika isključivo su odredjeni njihovom formom.

Primer takvog podjezika je brojni sistem.

3.2. Objekt-jezik i metajezik

Kada su shvaćeni suština i prednost ovakvih podjezika, rodila se ideja obrazovanja (konstruisanja) veštačkih jezika oslobodjenih prethodno navedenih nedostataka, a istovremeno sličnih prirodnim jezicima i prilagodjenih naučnim oblastima u kojima će se koristiti (primenjivati).

Ako bi se u rečenici navedenoj malo pre kao paradoks Beri-a, govorilo ne o srpsko-hrvatskom već o nekom drugom jeziku na primer ruskom, ničeg paradoksalnog nebi bilo u njoj. Odavde proizilazi mogućnost otklanjanja (razrešenja) paradoksa Beri-a uvodjenjem formalnih jezika sa ograničenjem da je na njima nemoguće govoriti o njima samim. Za opis jednog formalnog jezika neophodan je drugi formalan jezik.

Jezik koji se opisuje naziva se objekt-jezik. Jezik koji se koristi za opisivanje objekt-jezika naziva se metajezik.

Primer

Jezik koji se koristi za opisivanje metajezika naziva se metametajezik.

U ovom radu srpsko-hrvatski jezik se koristi kao metajezik.

Govoreći o jeziku neophodno je jasno razlikovati metajezik i objekt-jezik. U suprotnom su moguće razne nejasnoće.

Metajezik mora posedovati mogućnosti opisa forme i sadržaja rečenica objekt-jezika. Često se metajezik sastoji iz dva jezika. Prvi jezik je namenjen za opisivanje forme rečenica objekt -jezika i naziva se metasintaksni jezik. Drugi jezik se koristi za opisivanje sadržaja rečenica objekt-jezika i naziva se metasemantički jezik.

Skup pravila za definisanje forme rečenica objekt-jezika naziva se sintaksa objekt-jezika. Slično, skup pravila za definisanje sadržaja rečenica objekt-jezika naziva se semantika objekt--jezika.

3.3. Sintaksa jezika

U prirodnim jezicima postoje složene konstrukcije. Tako se reči obrazuju iz slova pomoću veznika prvog reda. Od reči se pomoću veznika drugog reda obrazuju fraze. Na sličan način se od fraza pomoću veznika trećeg reda obrazuju rečenice [104].

Primer

U svakoj rečenici napisanoj na nekom prirodnom jeziku unutar reči postoje veznici prvog reda odredjeni rasporedom slova. Reči su okružene i spojene u frazu veznicima drugog reda. Veznici drugog reda su razmaci. Fraze su okružene i povezane u rečenicu veznicima trećeg reda. Veznici trećeg reda su znaci interpunkcije: , ; itd. **Početni veznik trećeg reda odredjen je velikim**

slovom prve reči rečenice, a završni veznik trećeg reda odredjen je tačkom na kraju rečenice.

Ako bi veznike drugog i trećeg reda proglasili slovima rečenica bi postala reč. Medjutim, ovim pojednostavljenjem gubi se preglednost forme rečenice.

Nešto kasnije razmatraće se formalni (programski) jezici čije rečenice (naredbe) mogu da budu složenije konstrukcije od onih koje se obično javljaju u prirodnim jezicima i obratno sasvim jednostavne konstrukcije koje se sastoje iz jedne jedine reči.

Pri opisu forme rečenica objekt-jezika prvo se navodi azbuka, a zatim pravila za obrazovanje (konstrukciju) rečenica. Svako pravilo odredjuje operaciju, koja se koristi za konstruisanje rečenica objekt-jezika. Pri tom je, naravno, definisan skup podataka na koje se operacija može primeniti.

Pod sintaksom formalnog jezika podrazumeva se skup (spisak) operacija sa poznatom oblašću definisanosti.

3.4. Jezici

Definicija: Azbuka Σ je konačan skup simbola [6].

Primer

Azbuka programskog jezika FORTRAN sastoji se iz tri grupe simbola:

1. Velikih slova engleske azbuke*: A B C D E F ...Z
2. Dekadnih cifara: Ø 1 2 3 4 5 6 7 8 9
3. Specijalnih simbola: + - * / = , . ( ) $

Poredak navodjenja simbola nije od značaja. Navedeni simboli i samo oni koriste se za pisanje FORTRAN-programa.

Definicija: Niz $u=a_1a_2...a_m$ simbola azbuke Σ je niska ili reč nad azbukom Σ. Dužina niske je broj simbola niske (u ovom slučaju m). Operacija dopisivanja na nisku $u=a_1a_2...a_m$ niske $v=b_1b_2...b_n$ kojom se dobija niska $uv=a_1a_2...a_mb_1b_2...b_n$ naziva se dopisivanje (konkatenacija) niski u i v.

Definicija: Korisno je pretpostaviti postojanje niske (reči) dužine Ø - tzv. prazne reči. Prazna reč se označava sa e. Prazna reč ima sledeće osobine:

* Kod nekih realizacija FORTRAN-jezika, skup slova je dopunjen slovima latinične i/ili ćirilične azbuke koja nisu sadržana u engleskoj azbuci. Obično se ova slova koriste za pisanje literala i komentara.

ue=u=eu

uev=uv

u i v su proizvoljne reči.

Definicija: Σ^* je skup svih niski nad azbukom Σ uključujući i praznu reč.

Definicija: Jezik L (nad azbukom Σ) je podskup skupa Σ^*.

Primer

Neka je $\Sigma = \{a,b\}$.

a) $L_1=\Sigma^*= \{e,a,b,aa,ab,ba,bb,aaa,...\}$ je jezik.

b) $L_2= \{b,ab,aab,abb\}$ je jezik.

c) $L_3= \{a^p | p$ je prost broj$\}$ je jezik.

d) $L_4= \{a^m | Z \ni m \geqslant \emptyset\}$ je jezik.

a^m, gde je m nenegativan ceo broj, je niska aa...a dužine m.

a^0 je prazna reč.

Primer

Neka je Σ azbuka programskog jezika FORTRAN. Skup sintaksno ispravnih FORTRAN-programa je jezik.

Važno je uočiti da definicija jezika ne sadrži pojam značenja (semantike) jezika. Jezik je prosto skup niski pa je teorija jezika teorija osobina (svojstva) skupova niski.

Definicija: Proizvod $L \circ L'$ jezika L i L′ je skup reči dobijenih dopisivanjem na reči jezika L reči jezika L′.

Primer

$$L_2 \circ \{a,b\} = \{b, ab, aab, abb\} \circ \{a,b\} =$$
$$= \{ba, aba, aaba, abba, bb, abb, aabb, abbb\} .$$

Primer

$$L_3 \circ L_4 = \{a^p | p \text{ je prost broj}\} \circ \{a^m | Z \ni m \geqslant \emptyset\} =$$
$$= \{a^{p+m} | p \text{ je prost broj i } Z \ni m \geqslant \emptyset\} .$$

Veoma često se izostavlja $\circ$ pa se umesto $L \circ L'$ piše LL'.

Proizvod jezika nije komutativna operacija tj.

$$L \circ L' \neq L' \circ L$$

Definicija: Stepen jezika L se induktivno definiše na sledeći način:

$$L^0 = \{e\}$$
$$L^n = L \cdot L^{n-1}, \quad n \geq 1.$$

$L^1 = L \cdot L^0 = L \cdot \{e\} = L.$

Za $n \geq 1$, L^n je skup niski dobijenih dopisivanjem n niski jezika L.

Definicija: Zatvaranje jezika L u oznaci L^* je:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots$$

Definicija: Tranzitivno zatvaranje jezika L u oznaci L^+ je:

$$L^+ = L^1 \cup L^2 \cup \ldots \cup L^n \cup \ldots$$

3.5. Gramatike

Jezici koji se sastoje iz konačnog broja niski mogu se jednostavno definisati navodjenjem svih niski. Medjutim, jezici koji se koriste u programiranju i razmatraju se u ovom radu su beskonačni i ne mogu se definisati na prethodan način. Jedan od načina definisanja beskonačnih jezika je pomoću gramatika [3].

Definicija: Gramatika G je četvorka:

$$(N, \Sigma, P, S)$$

gde je:

1. N - konačan skup nezavršnih (neterminalnih, pomoćnih) simbola,
2. Σ - konačan skup završnih (terminalnih) simbola.

$$N \cap \Sigma = \emptyset$$

3. P - konačan skup pravila izvodjenja (smena) oblika:

$$\alpha \rightarrow \beta$$

gde je:

$$\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^* \quad , \text{ a}$$
$$\beta \in (N \cup \Sigma)^* \quad .$$

4. S - početni (rečenički) simbol.

$$S \in N \quad .$$

Primer

Neka je:
{A} - skup nezavršnih simbola,
{a,b,c} - skup završnih simbola,
{A→aAb,A→c} - skup pravila izvodjenja i
A - početni simbol.
({A},{a,b,c},{A→aAb,A→c},A)
je gramatika.

Skup pravila izvodjenja P se često definiše na neformalan način. Jednostavno se kaže: "pravila izvodjenja su:" i sledi spisak pravila izvodjenja.

Primer

Neka je:
$G_1=(\{A,b,c\},\{a,b,c\},P,A)$
gramatika. **Pravila izvodjenja su:**

A → abc
Bb → bB
bC → Cb
aC → aa
A → aBbc
Bc → Cbcc
aC → aaB

Očigledno je:
- skup nezavršnih simbola $N=\{A,B,C\}$,
- skup završnih simbola $\Sigma=\{a,b,c\}$ i
- početni simbol A.

Da bi se razumeo način na koji gramatika definiše jezik treba shvatiti simbol "→" kao skraćenicu "može se zameniti". Krenuvši od početnog simbola S primenjujući pravila izvodjenja na niske nezavršnih i završnih simbola dobijaju se nove i nove niske. Proces generisanja niski se nastavlja sve dok se ne dobije niska završnih simbola (tj. niska koja se sastoji isključivo iz završnih simbola) ili prazna reč. Završna niska je reč (rečenica) jezika definisanog gramatikom.

Proces generisanja jezika razmotrimo na primeru gramatike G. Početni simbol je A. Na njega se mogu primeniti oba pravila izvodjenja: A→aAb i A→c jer je leva strana pravila izvodjenja upravo početni simbol A.

Primenom prvog pravila izvodjenja dobija se niska aAb, a primenom drugog pravila izvodjenja dobija se niska C. Niska C je dužine jedan i sastoji se iz završnog simbola C pa je reč (pripada) jeziku definisanom gramatikom G. Na nisku C, tj. na reč jezika ne mogu se dalje primenjivati pravila izvodjenja gramatike. Niska aAb sadrži nezavršni simbol A i nije reč (ne pripada) jeziku definisanom gramatikom G. Proces generisanja jezika se dalje nastavlja. Primenom prvog pravila izvodjenja dobija se niska aaAbb koja sadrži nezavršni simbol A - nije reč jezika - proces generisanja se nastavlja. Primenom drugog pravila izvodjenja dobija se niska acb koja se isključivo sastoji iz završnih simbola pa pripada jeziku definisanom gramatikom G. Nastavljajući proces generisanja jezika dobijaju se reči: aacbb, aaacbbb, ..., aa...acbb ...b itd. Kraće napisano jezik definisan gramatikom G je skup reči oblika:

$$\{ a^m c b^m \mid m \geqslant \emptyset \}.$$

Razmatrani proces generisanja jezika može se kraće napisati:

$A \Rightarrow c.$ $\quad c \in L(G).$

$A \Rightarrow aAb \Rightarrow acb.$ $\quad acb \in L(G).$

$A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aacbb.$ $\quad aacbb \in L(G).$

$\vdots$

Sa L(G) se obeležava jezik L definisan (izveden, generisan) gramatikom G. Znači $L(G)=\{a^m c b^m \mid m>\emptyset\}$.

Gramatika G_1 je znatno složenija od gramtike G. Nije baš jednostavno odmah reći šta je jezik $L_1=L(G_1)$. U cilju otkrivanja jezika L_1 definisanog gramatikom G_1 potrebno je napisati više izvodjenja reči jezika L_1.

1) $A \Rightarrow abc$; $\quad abc \in L_1$

2) $A \Rightarrow aBbc$

$\Rightarrow abBc$

$\Rightarrow abCbcc$

$\Rightarrow aCbbcc$

$\Rightarrow aabbcc$; $\quad aabbcc \in L_1$

3) $A \Rightarrow aBbc$

$\Rightarrow abBc$

$\Rightarrow abCbcc$

$\Rightarrow aCbbcc$

$\Rightarrow aaBbbcc$

$\Rightarrow aabBbcc$

$\Rightarrow aabbBcc$

$\Rightarrow aabbCbccc$

$\Rightarrow aabCbbccc$

$\Rightarrow aaCbbbccc$

$\Rightarrow aaabbbccc$; $aaabbbccc \in L_1$.

Svaki red dobijen je iz prethodnog reda primenom odgovarajućih pravila izvodjenja. Tako je na primer niska abBc dobijena iz niske aBbc primenom pravila izvodjenja $Bb \rightarrow bB$.

Posmatrajući dobijene reči može se pretpostaviti da je jezik:

$$L_1 = L(G) = \{ a^n b^n c^n \mid n \geqslant 1 \} .$$

<u>Definicija</u>: Neka je $G=(N, \Sigma, P, S)$ gramatika. Niska ν' neposredno je izvedena iz niske ν ako i samo ako je $\nu = \gamma\alpha\delta$, $\nu' = \gamma\beta\delta$ i $\alpha \rightarrow \beta$. γ i δ su proizvoljne niske, a $\alpha \rightarrow \beta$ je pravilo izvodjenja gramatike G.

<u>Definicija:</u> Niska ν' je izvedena iz niske ν akko je $\nu' = \nu$ ili postoji niz niski $\nu_0, \nu_1, \ldots, \nu_n$ takvih da je $\nu = \nu_0$, $\nu' = \nu_n$ i ν_{i+1} je neposredno izvedeno iz ν_i, $\emptyset \leqslant i \leqslant n-1$. Niz:

$$\nu = \nu_0 \Rightarrow \nu_1 \Rightarrow \nu_2 \Rightarrow \ldots \Rightarrow \nu_n = \nu'$$

je izvodjenje dužine n. Takodje se kaže da je ν izvodjenje dužine $\emptyset$.

Piše se: $\nu \overset{*}{\underset{G}{\Rightarrow}} \nu'$

ako je ν' izvedena iz ν.

Piše se: $\nu \overset{+}{\underset{G}{\Rightarrow}} \nu'$

ako je ν' izvedena iz ν pomoću izvodjenja dužine strogo veće od $\emptyset$.

U slučajevima gde nema opasnosti da se pobrka gramatika G, slovo G se izostavlja i piše se:

$\nu \Rightarrow \nu'$

$\nu \overset{*}{\Rightarrow} \nu'$ ili $\nu \Rightarrow * \nu'$

$\nu \overset{+}{\Rightarrow} \nu'$ ili $\nu \Rightarrow + \nu'$.

Definicija: Jezik L generisan gramatikom G označava se sa L(G) je skup svih niski $w \in \Sigma^*$ za koje važi $S \Rightarrow^* w$ (tj. izvedene su iz početnog simbola S).

$$L(G) = \{ w \mid w \in \Sigma^* \wedge S \Rightarrow^* w \} \quad .$$

Definicija: Reč (niska) $w \in (N \cup \Sigma)^*$ je rečenična forma (fraza) gramatike G akko je $S \Rightarrow^* w$. Reč w je rečenica gramatike G akko je $S \Rightarrow^* w$ i $w \in \Sigma^*$.

3.6. Klasifikacija gramatika

N. Chomsky je izvršio klasifikaciju gramatika zavisno od oblika pravila izvodjenja na četiri tipa:

- gramatike tipa Ø ili neograničene gramatike.*
- gramatike tipa 1 ili kontekstno-zavisne gramatike.
- gramatike tipa 2 ili kontekstno-slobodne gramatike.
- gramatike tipa 3 ili regularne gramatike.

Pravila izvodjenja gramatika tipa Ø su bez ograničenja.

Pravila izvodjenja gramatika tipa 1 su oblika:

$$\alpha \rightarrow \beta$$

pri čemu je dužina $(\alpha) \leqslant$ dužina (β). α i $\beta \in (N \cup \Sigma)^+$.

Pravila izvodjenja gramatika tipa 2 su oblika:

$$A \rightarrow \alpha$$

A je nezavršni simbol, tj. $A \in N$, a α je niska nezavršnih ili završnih simbola ili prazna reč, tj. $\alpha \in (N \cup \Sigma)^*$.

Pravila izvodjenja gramatika tipa 3 su jednog od sledeća dva oblika:

a) $A \rightarrow xB$ ili
$A \rightarrow x$

x je niska završnih simbola, tj. $x \in \Sigma^*$, a A i B su nezavršni simboli, tj. $A, B \in N$.

Ovakve regularne gramatike nazivaju se desno-linearne gramatike.

b) $A \rightarrow Bx$ ili
$A \rightarrow x$

x je niska završnih simbola, tj. $x \in \Sigma^*$, A i B su nezavršni simboli, tj. $A, B \in N$.

* Navedeni nazivi gramatika su uobičajeni u literaturi ali nisu baš najsrećnije odabrani. Recimo, umesto neograničene gramatike bolji naziv bi bio gramatike bez ograničenja ili umesto kontekstno-slobodne kontekstno-oslobodjene itd. Poštujući tradiciju pridržavaćemo se navedenih naziva.

Regularne gramatike ovakvog oblika nazivaju se levo--linearne gramatike.

Definicija: Jezik je regularan ako se može generisati regularnom gramatikom.

Definicija: Jezik je kontekstno-slobodan ako se može generisati kontekstno-slobodnom gramatikom.

Definicija: Jezik je kontekstno-zavisan ako se može generisati kontekstno-zavisnom gramatikom.

Primer

a) Gramatika G je kontekstno-slobodna gramatika pa je jezik $L(G)=\{a^n c b^n \mid n \geqslant \emptyset\}$ kontekstno-slobodan jezik.

b) Gramatika G_1 je kontekstno-zavisna gramatika pa je jezik $L_1=L(G_1)=\{a^n b^n c^n \mid n \geqslant 1\}$ kontekstno-zavisan jezik.

Napomena

1. Regularne gramatike zadovoljavaju uslove kontekstno--slobodnih gramatika - obrnuto ne važi. Znači, regularni jezici su kontekstno-slobodni jezici, dok kontekstno-slobodni jezici nisu u opštem slučaju regularni jezici.

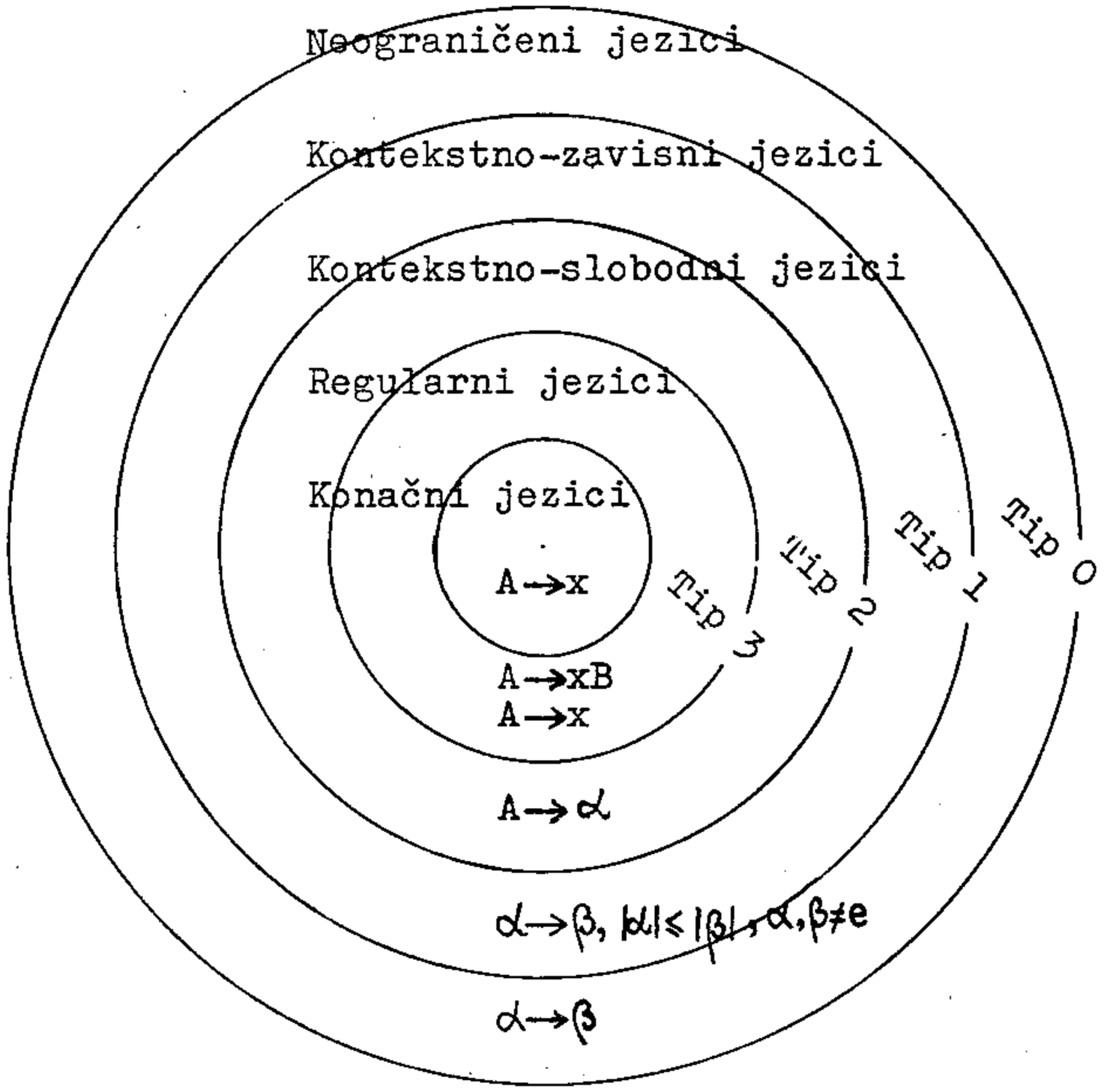
2. Zbog uslova: dužina $(\alpha) \leqslant$ dužina (β), $\alpha, \beta \in (N \cup \Sigma)^+$ niska β nemože da bude prazna reč, tj. $\beta \neq e$. Znači, prazna reč nemože da bude desna strana pravila izvodjenja $\alpha \to \beta$ kontekstno-zavisne gramatike. Medjutim, prazna reč može da bude desna strana pravila izvodjenja $A \to \alpha$ kontekstno-slobodne gramatike. Ova razlika nije bitna. Može se dokazati da za svaku kontekstno-slobodnu gramatiku G postoji kontekstno-slobodna gramatika G' takva da ne sadrži pravila izvodjenja oblika $A \to e$, a pritom se izvedeni jezici isključivo razlikuju za praznu reč, tj.

$$L(G')=L(G)/\{e\} .$$

Gramatika G' je kontekstno-zavisna. Znači, kontekstno--slobodni jezici sa izuzetkom onih koji sadrže praznu reč su kontekstno-zavisni jezici.

Još uvek nije dat jednostavan primer jezika tipa Ø koji nije jezik tipa 1. Hopcroft i Ullman su naveli do sada najjednostavniji primer ali koji je opet veoma složen.

Na slici 3.1. su prikazani odnosi izmedju gramatika i jezika različitih tipova i pravila izvodjenja.



Slika 3.1.

3.7. Kontekstno-slobodne gramatike i beleženje sintakse programskih jezika

Kontekstno-slobodne gramatike se najčešće koriste za beleženje sintakse programskih jezika. Osnovni razlog zato je sličnost izmedju pravila izvodjenja:

$$A \rightarrow \alpha \quad ; \quad A \in N \ , \ \alpha \in (N \cup \Sigma)^*$$

i pravila prevodjenja reči jednog rečnika:

reč ⟶ prevod.

Ljudi se još od osnovne škole uče i navikavaju na korišćenje rečnika, rečničkih definicija tj. pravila prevodjenja (izvodjenja) i prevodjenje (izvodjenje) pomoću rečnika. Posle 10-tak godina takvog rada ljudi su se skoro sasvim privikli na rad sa rečničkim definicijama tako da su im onda, kad čuju za formalne gramatike i klasifikaciju Chomsky-og, kontekstno-slobodne gramatike nekako najbliže i najprirodnije. Pogrešno bi bilo reći da su kontekstno-zavisne gramatike nejasne i neprihvatljive za čoveka ali činjenica je da mu manje odgovaraju. Primer koji ide u prilog rečenog bi bio: neuporedivo je lakše bukvalno prevoditi neki tekst (druga je stvar što će se verovatno dobiti loš prevod) nego ga prevoditi vodeći računa o kontekstu.

Pravila izvodjenja kontekstno-slobodnih gramatika uvek se mogu raščlaniti na sledeće delove:

- nezavršni simbol ($A \in N$),
- simbol izvodjenja (⟶) i
- niska nezavršnih i/ili završnih simbola ($\alpha \in (N \cup \Sigma)^*$).

Drugačije rečeno, nezavršni simbol se definiše pomoću niza nezavršnih i/ili završnih simbola. Zar se sličan slučaj ne javlja pri prevodjenju pomoću rečnika? U rečniku je jedna reč objašnjena (prevedena) pomoću jedne ili više drugih reči. Ako ne znamo (ne razumemo) značenje nekih reči prevoda - objašnjenja za njih opet možemo da potražimo u rečniku.

U narednim poglavljima biće ukratko izloženi razni načini pisanja pravila izvodjenja kontekstno-slobodnih gramatika ili još konkretnije rečeno načini beleženja (notacije) sintakse programskih jezika.

3.8. Notacija Chomsky-og

Notacija, koja je do sada korišćena u vezi sa gramatikama, naziva se notacija Chomsky-og.

Najbitniji deo gramatike je konačan skup P pravila izvodjenja koja definišu proces generisanja (izvodjenja) niski (reči, rečenica) jezika. Pravilo izvodjenja je uredjen par niski ili tačnije rečeno element skupa $(N \cup \Sigma)^{*} N (N \cup \Sigma)^{*} \times (N \cup \Sigma)^{*}$. N je skup nezavršnih simbola, a Σ je skup završnih simbola. Prva niska je proizvoljna niska simbola koja obavezno sadrži bar jedan nezavršni simbol. Druga niska je proizvoljna niska nezavršnih i/ili završnih simbola.

Chomsky je predložio (koristio) sledeću notaciju:

Ako se drugačije ne kaže:

(1) Malim slovima s početka engleske azbuke a,b,c,d,..., ciframa Ø,1,...,9 i specijalnim znacima +,-,*,/,... označavaju se završni simboli.

(2) Velikim slovima s početka enegleske azbuke A,B,C, D,... i S označavaju se nezavršni simboli.

Sa S se označava početni simbol.

(3) Velikim slovima s kraja engleske azbuke U,V,...,Z označavaju se simboli za koje nije bitno da li su završni ili nezavršni simboli.

(4) Malim slovima s kraja engleske azbuke u,v,...,z označavaju se niske završnih simbola.

(5) Slovima grčke azbuke $\alpha, \beta, \gamma, \ldots$ označavaju se niske završnih i/ili nezavršnih simbola.

(6) Dozvoljena je upotreba indeksa.

Pravilo izvodjenja se umesto uredjenog para (α,β) piše u obliku: $\alpha \rightarrow \beta$. $\longrightarrow$ je metasimbol.

Niz pravila izvodjenja:

$$\begin{aligned} \alpha &\rightarrow \beta_1 \\ \alpha &\rightarrow \beta_2 \\ &\vdots \quad \vdots \\ \alpha &\rightarrow \beta_n \end{aligned}$$

se piše u obliku: $\alpha \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$. | je metasimbol.

Primer

Neka je gramatika $G_a=(N,\Sigma,P,S)$ definisana na sledeći način:

Skup nezavršnih simbola N= {S,V,E,T,F,I,L,D}.
Skup završnih simbola Σ= {=,+,*,(,),a,...,z,ø,...,9} .
Početni simbol je S.
Pravila izvodjenja P su:

S → V=E
E → T|E+T
T → F|T*F
F → V|I|(E)
V → L|VL|VD
I → D|ID
L → a|b|c|d|e|f|g|h| ... |x|y|z
D → ø|1|2|3|4|5|6|7|8|9

Gramatika G_a definiše jezik jednostavnih aritmetičkih naredbi.
Primeri rečenica ovog jezika su niske:

a=b
iksipsilon=x*y
ul=(u-l) itd.

Na prethodnom primeru se vide neki nedostaci notacije Chomsky-og.

Notacija Chomsky-og je podesna za primenu u teorijskim razmatranjima formalnih gramatika ali pri konkretnom radu sa gramatikama je nepodesna.

Recimo, prethodna pravila izvodjenja bi bila mnogo jasnija ako bi se umesto skraćenice S pisala cela engleska reč Statement*(naredba) slično V - Variable (promenljiva), E - Expression (izraz) itd. Da je gramatika G_a kojim slučajem bila duža javio bi se problem nedostatka slova. Dalje, teško je poštovati dogovor o raspodeli slova. Notacija Chomsky-og je neupotrebljiva ako su simboli objekt-jezika velika slova engleske azbuke itd.

Rešavajući konkretne probleme (zadatke) - projektujući prevodioce, operativne sisteme itd. niz stručnjaka došlo je do raznih ideja praktičnijeg (pogodnijeg) zapisa pravila izvodjenja kontekstno-slobodnih gramatika nego što je notacija Chomsky-og.

---

* U lingvistici se koristi termin sentence (rečenica).

U radu će ukratko biti izložene notacije:
- Backus-ova,
- modifikovane Backus-ove,
- Ingerman-ova,
- Iverson-ova,
- obrnuta,
- Berckhardt-ova,
- Lukas-ova
- Van Wijngaarden-ova i
- grafička.

Navedene notacije su ilustrovane primerima.

3.9. Backus-ova notacija

Počev od 6Ø-tih godina ovog veka često se koristi Backus-ova notacija kao sredstvo opisa metasintaksnih jezika. U slučaju Backus-ove notacije metasintaksni jezik sastoji se iz konačnog broja rečenica, tzv. metalingvističkih formula. Za konstruisanje metalingvističkih formula koriste se univerzalni metasimboli: ::= i |.

Simbol ::= se čita "po definiciji je", a simbol | se čita "ili". Univerzalni metasimboli nisu završni ni nezavršni simboli gramatike. Metasimbole* po potrebi, uvodi čovek - konstruktor objekt-jezika koji je obično i konstruktor metajezika. Metasimboli su proizvoljne fraze (rečenice) prirodnog jezika stavljene u uglaste zagrade. Za označavanje uglastih zagrada koriste se simboli < > . Prethodne fraze su obično nazivi gramatičkih klasa objekt-jezika.

Pored metasimbola u metalingvističkim formulama koriste se simboli objekt-jezika.**

---

* U literaturi su još uobičajeni termini:
  - metalingvistički simbol,
  - metalingvistička promenljiva,
  - metapromenljiva.

** U literaturi se susreću nazivi:
  - metalingvistička konstanta,
  - metakonstanta,
  - objekt-simbol.

Simboli objekt-jezika se lako prepoznaju jer nisu univerzalni metasimboli i nalaze se van uglastih zagrada.

Metalingvistička formula sastoji se iz leve i desne strane razdvojene univerzalnim metasimbolom ::=. Leva strana metalingvističke formule je metasimbol. Desna strana metalingvističke formule može biti:

- prazna niska,
- konačna niska metasimbola i/ili simbola objekt-jezika i
- konačan niz prethodnih niski medjusobno razdvojenih univerzalnim metasimbolom |.

Po definiciji dve metalingvističke formule sa jednakim levim, a različitim desnim stranama označavaju isto što i metalingvistička formula dobijena dopisivanjem na kraj desne strane jedne od metalingvističkih formula univerzalnog metasimbola | i desne strane druge različite metalingvističke formule.

Važi i obratno tj. ako na desnoj strani metalingvističke formule postoji univerzalan metasimbol | onda su metalingvističke formule sa levim stranama jednakim levoj strani date metalingvističke formule, a sdesnim stranama redom jednakim delovima date metalingvističe formule do i od univerzalnog metasimbola | ekvivalentne datoj metalingvističkoj formuli.

Metalingvističke formule su lako razumljive. Čitaju se na prirodnom jeziku. Uglaste zagrade se ne izgovaraju.

Primer

Formalan jezik za predstavljanje prirodnih brojeva pomoću Backus-ove notacije je:

(1) <nenulta cifra> ::=1|2|3|4|5|6|7|8|9

(2) <cifra> ::=Ø|<nenulta cifra>

(3) <prirodan broj> ::=<nenulta cifra>|
<prirodan broj> <cifra>

Metasimbol <prirodan broj> označava rečenicu objekt-jezika. Navedeni metajezik definiše: 3, 124 itd. kao prirodne brojeve tj. kao rečenice objekt-jezika. Zaista, na osnovu metalingvističke formule (1) jedna od vrednosti metasimbola <ne nulta cifra> je i 3. Na osnovu metalingvističke formule (3) sledi da je

jedna od vrednosti metasimbola <prirodan broj> nenulta cifra 3.
Slično se dokazuje da je 124 takodje prirodan broj.

Primer

Jezik jednostavnih aritmetičkih naredbi definisan je sledećim metalingvističkim formulama Backus-a:

<aritmetička naredba> ::= <promenljiva> = <izraz>
<izraz> ::= <sabirak> | <izraz> + <sabirak>
<sabirak> ::= <činilac> | <sabirak> * <činilac>
<činilac> ::= <promenljiva> | <ceo broj bez znaka> | (<izraz>)
<promenljiva> ::= <slovo> | <promenljiva><slovo> | <promenljiva><cifra>
<ceo broj bez znaka> ::= <cifra> | <ceo broj bez znaka><cifra>
<slovo> ::= A|B|...|X|Y|Z
<cifra> ::= Ø|1|2|3|4|5|6|7|8|9

Već na prvi pogled se može uočiti da su:

<aritmetička naredba>
<promenljiva>
<izraz>
<sabirak>
<činilac>
<promenljiva>
<ceo broj bez znaka>
<slovo>
<cifra>

metasimboli, a:

=
+
*
(
)
A
B
:
Z
Ø
:
9

simboli objekt-jezika.

Jedino ostaje nejasno šta je početni simbol? Ovaj problem se može rešiti:

- uvodjenjem konvencije da je prvi po redu metasimbol početni simbol,
- posebnim naglašavanjem. Recimo, poslednji red sadrži isključivo početni simbol.

Metasimboli koriste se u metalingvističkim formulama kao jedinstvene celine pa ih treba shvatiti kao slova metajezika.

Uslov koji mora zadovoljiti svaka rečenica objekt-jezika glasi: "rečenica objekt-jezika ne sadrži metasimbole i dobijena je kao vrednost jedne od niski koje se nalaze na desnoj strani metalingvističke formule, čija je leva strana metasimbol, koji označava rečenicu".

Kod metajezika formulacija ovog uslova se ne navodi računajući da se uvek podrazumeva kada se koristi Backus-ova notacija.

Pomoću Modifikovane Backus-ove notacije - Backus-ova notacija se može opisati na sledeći način:

<metalingvistička formula> ::=
<metasimbol><univ. metasimbol izvodj.><niz altern.>
<niz altern.>::=<altern.>|
<altern.><univ. metasimbol altern.><niz altern.>
<altern.>::=<član>|<altern.><član>
<član>::=<metasimbol>|<simbol>
* <metasimbol>::=≪niz simbola≫
<niz simbola>::=<simbol>|<niz simbola><simbol>
** <univ. metasimbol izvodj.>::=::=
*** <univ. metasimbol altern.>::=|
<simbol> ::= <slovo>|<cifra>|<ostali tipografski znaci>
<slovo> ::= A|B|...|Z
<cifra> ::= Ø|1|...|9
<ostali tipografski znaci> ::= +|-|*|/|↑|%|ß| ...

Problematične metalingvističke formule obeležene su su sa ⁎.

Metalingvističku formulu:

<metasimbol>::=≪niz simbola≫

treba shvatiti na sledeći način:

metasimbol je po definiciji:

leva uglasta zagrada za kojom sledi

niz simbola za kojim sledi

desna uglasta zagrada.

Znači prva od zagrada u slučaju udvojenih levih uglastih zagrada shvaća se kao simbol objekt-jezika, a druga kao sastavni deo definicije metasimbola.

Slično, prvu od zagrada u slučaju udvojenih desnih uglastih zagrada treba shvatiti kao sastavni deo definicije metasimbola, a drugu kao simbol objekt-jezika.

Metalingvistička formula:

<univ. metasimbol izvodj>::=::=

ima sledeće značenje:

univerzalni metasimbol izvodjenja je po definiciji ::=.

Metalingvistička formula:

<univ. metasimbol altern>::=|

ima sledeće značenje:

univerzalni metasimbol alternative je po definiciji |.

Pri programskoj realizaciji Backus-ove notacije tri prethodno navedene metalingvističke formule u datom obliku prave problema jer nisu sasvim korektno definisane. Ovaj problem se rešava uvodjenjem sledeće konvencije:

'<simbol>

označava pripadnost <simbol>-a objekt-jeziku. Ništa ne smeta ako <simbol> već pripada objekt-jeziku, tj. važi:

'<simbol>≡<simbol>

U skladu s predloženom konvencijom problematične metalingvističke formule postaju oblika:

<metasimbol>::='≪niz simbola≫'

<univ. metasimbol izvodj.>::='::=
<univ. metasimbol altern.>::='|

i sasvim su jednoznačne.

Napomena

Kako | nije tipografski znak u programskoj realizaciji Backus-ove notacije koristi se znak uzvika (uzvičnik)!

3.10. Modifikovane Backus-ove notacije

U ovom odeljku navodimo neke modifikacije Backus-ove notacije.

Modifikovana Backus-ova notacija, slično Backus-ovoj notaciji, koristi se za zapisivanje pravila izvodjenja kontekstno--slobodnih gramatika.

Modifikovana Backus-ova notacija prikazuje konstrukcije programskog jezika u obliku metalingvističkih formula.

Metalingvistička formula piše se u obliku:

$$A ::= \alpha$$

A je metalingvistička promenljiva,

::= je znak operacije dodeljivanja,

α je metalingvistički izraz.

Metalingvistička formula ima sledeće značenje: vrednost metalingvističkog izraza α dodeljuje se metalingvističkoj promenljivoj A.

Metalingvistički izraz je sastavljen iz metalingvističkih konstanti i metalingvističkih promenljivih medjusobno razdvojenih metalingvističkim operatorima.

Metalingvistički operatori su:

- operator spajanja,
- operator razdvajanja i
- operator ponavljanja.

Metalingvistička operacija spajanja piše se u obliku:

$$\alpha \cdot \beta \text{ ili } \alpha\beta$$

α i β su metalingvistički izrazi, a

$\circ$ je znak operatora spajanja.

Znak operatora spajanja najčešće se izostavlja.

Metalingvistička operacija razdvajanja piše se u obliku

$$\alpha|\beta \text{ ili } \left\{\begin{matrix}\alpha\\\beta\end{matrix}\right\}$$

α i β su metalingvistički izrazi, a | odnosno $\{\}$ je znak operatora razdvajanja.

Metalingvističke operacije spajanja i razdvajanja su binarne operacije.

Metalingvistička operacija ponavljanja je unarna operacija sa sledećim osobinama:

1. $[\alpha]_i^j = \underbrace{\alpha\alpha\ldots\alpha}_{i}|\underbrace{\alpha\alpha\ldots\alpha\alpha}_{i+1}|\ldots|\underbrace{\alpha\alpha\ldots\alpha\alpha\ldots\alpha}_{j}$

$$i \leqslant j \in N \cup \{\emptyset\}$$

2. $[\alpha]_1^1 = \alpha$
3. $[\alpha]_i^i = \underbrace{\alpha\alpha\ldots\alpha}_{i}$
4. $[\alpha]_\emptyset^1 = [\alpha] = e|\alpha$
5. $[\alpha]_\emptyset^j = e|\alpha|\alpha\alpha|\ldots|\underbrace{\alpha\alpha\ldots\alpha}_{j}$
6. $[\alpha]_\emptyset^\infty = [\alpha]^* = e|\alpha|\alpha\alpha|\ldots|\alpha\alpha\ldots\alpha\ldots$
7. $[\alpha]_1^\infty = [\alpha]^+ = \alpha|\alpha\alpha|\ldots|\alpha\alpha\ldots\alpha\ldots$

α je metalingvistički izraz,

e je prazna reč,

[ ] je znak operatora ponavljanja.

Modifikovana Backus-ova notacija predstavlja poboljšanje Backus-ove notacije. Poboljšanje se sastoji u uvodjenju novih univerzalnih metasimbola:

- { }
- [ ] sa graničnicima: i,j,-,*

odnosno povećanju izražajnih mogućnosti metalingvističkog jezika.

Nedostatak Modifikovane Backus-ove notacije je višenivoski zapis, tj. zapis metalingvističkih formula u dva, tri ili **više redova.**

Modifikovana Backus-ova notacija je dobra za neformalni zapis ali se ne može neposredno primeniti na računaru.

Primer

Pomoću Modifikovane Backus-ove notacije ceo broj se može opisati na sledeći način:

$$\langle \text{ceo broj} \rangle ::= \left[\langle \text{znak} \rangle\right]\left[\langle \text{cifra} \rangle\right]^{+}$$

$$\langle \text{znak} \rangle ::= \left\{ \begin{matrix} + \\ - \end{matrix} \right\}$$

$$\langle \text{cifra} \rangle ::= \emptyset \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Primer

Gramatika G_a zapisana Modifikovanom Backus-ovom notacijom je:

$$\langle \text{aritmetička naredba} \rangle ::= \langle \text{promenljiva} \rangle = \langle \text{aritmetički izraz} \rangle$$

$$\langle \text{aritmetički izraz} \rangle ::= \langle \text{sabirak} \rangle \left[+ \langle \text{sabirak} \rangle \right]^{*}$$

$$\langle \text{sabirak} \rangle ::= \langle \text{činilac} \rangle \left[* \langle \text{činilac} \rangle \right]^{*}$$

$$\langle \text{činilac} \rangle ::= \langle \text{promenljiva} \rangle \mid \langle \text{ceo broj} \rangle \mid (\langle \text{aritmetički izraz} \rangle)$$

$$\langle \text{promenljiva} \rangle ::= \langle \text{slovo} \rangle \left[\left\{ \begin{matrix} \langle \text{slovo} \rangle \\ \langle \text{cifra} \rangle \end{matrix} \right\} \right]^{*}$$

$$\langle \text{ceo broj} \rangle ::= \left[\langle \text{cifra} \rangle \right]^{+}$$

$$\langle \text{slovo} \rangle ::= A \mid B \mid C \mid \ldots \mid X \mid Y \mid Z$$

$$\langle \text{cifra} \rangle ::= \emptyset \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Predlažemo dalju modifikaciju (poboljšanje) Backus-ove notacije kojim zapis postaje jednoredni (jednonivoski).

U tom cilju uvedimo sledeće univerzalne metasimbole:

- $\leqslant$ ne više od,
- $\geqslant$ ne manje od .

Iza novouvedenih univerzalnih metasimbola mora obavezno da sledi ceo neoznačen broj - graničnik.

Primer

<ceo broj> ::= ⩽1 <znak>⩾1 <cifra>
<znak> ::= + | -
<cifra> ::= ø | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Primer

Gramatika G_a zapisana Poboljšanom Backus-ovom notacijom je:

<aritmetička naredba> ::= <promenljiva> = <aritmetički izraz>
<aritmetički izraz> ::= <sabirak>⩾ø[+ <sabirak>]
<sabirak> ::= <činilac>⩾ø [* <činilac>]
<činilac> ::= <promenljiva> | <ceo broj> | (<aritmetički izraz>)
<promenljiva> ::= <slovo>⩾ø [<slovo> | <cifra>]
<ceo broj> ::= ⩾1 <cifra>
<slovo> ::= A | B | C | ... | X | Y | Z
<cifra> ::= ø | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Univerzalni metasimboli { } nemaju smisla i ne upotrebljavaju se.

Univerzalni metasimboli [ ] koriste se kao metazagrade.

$$[\alpha]_i^j \Leftrightarrow \geqslant i \leqslant j\alpha$$

Uopštavanjem metalingvističkih formula dobijaju se univerzalne metalingvističke formule.

Uopštavanje se vrši na sledeći način. Neka je formalan jezik L opisan pomoću metajezika M. Dalje, neka su metalingvističke formule metajezika M napisane u Backus-ovoj notaciji. Transformišimo metalingvističke formule metajezika M tako da nijedna formula više ne sadrži na desnoj strani univerzalni metasimbol | . Novo dobijeni sistem formula označimo sa M'. Desna strana svake formule je niska simbola objekt-jezika i/ili metajezika.

Operaciju povezivanja (konkatenaciju) simbola $\alpha_1\alpha_2$,.. .. α_i u reč $\alpha_1\alpha_2...\alpha_i$ označimo sa $S_i(\alpha_1,\alpha_2,...,\alpha_i)$.

Pri tom je $S_1(\alpha_1)=\alpha_1$, tj. S_1 je operacija identičnosti.

Zamenivši u formulama metajezika M'desne strane (koje su oblika $\alpha_1\alpha_2...\alpha_i$) odgovarajućim zapisima oblika $S_i(\alpha_1,\alpha_2,...,\alpha_i)$ dobija se metajezik M'' čije su formule oblika:

$$\alpha ::= S_i(\alpha_1,\alpha_2,\ldots,\alpha_i);$$

(Kraj formule označen je sa;)

Metajezik M´´je kod metajezika M´. Metajezik M´´opisuje isti objekt-jezik L kao metajezik M´. Dopunski simboli koji se koriste u metajeziku za zapis desnih strana formula ne smeju biti simboli objekt-jezika. U slučaju Backus-ove notacije ovaj zahtev je važio isključivo za univerzalne metasimbole (::=i |) i uglaste zagrade (< i >). Argumenti desnih strana formula su simboli objekt-jezika i/ili metajezika. Leve strane formula su isključivo metasimboli. Konstrukcije objekt-jezika koje se nalaze na desnoj strani formula nazivaju se morfemi objekt-jezika. Skup morfema čini bazu objekt-jezika.

Uopštenje Backus-ove notacije sastoji se u sledećem:

1) Morfemi objekt-jezika su proizvoljne konstrukcije klasa (A,B).

2) Operacije odredjene desnim stranama formula su operacije čiji su polazni podaci skup konstrukcija klasa (A,B), a rezultati - konstrukcije iste klase.

3) Odustaje se od obaveznog zagradjivanja metasimbola. I dalje se navodi spisak (azbuka) metasimbola. Jedan izdvojeni metasimbol odgovara pojmu "rečenica objekt-jezika". Simboli koji se koriste za označavanje operacija male zagrade, zapete i tačka--zapete ubrajaju se u metasimbole.

4) U metajeziku se koriste veznici (za označavanje početka, produžetka i kraja) različiti od veznika objekt-jezika.

<u>Napomena</u>

Sva slova objekt-jezika različita su od metasimbola. Formule dobijene kao rezultat (gore opisanog uopštavanja nazivaju se univerzalne metaformule. Metajezik, koji se sastoji iz konačnog broja univerzalnih metaformula, naziva se induktivno-generativan.

Uslov da konstrukcija dobijena pomoću univerzalne metaformule pripada objekt-jeziku je: "konstrukcija se dobija pomoću formule čija je leva strana izdvojeni metasimbol".

Metajezik M'' koji je kôd metajezika M' je specijalan slučaj induktivno-generativnog metajezika. Sledi da je metajezik M (zadan Backus-ovom notacijom) specijalan slučaj induktivno--generativnog metajezika.

Primer

Induktivno-generativni metajezik, ekvivalentan metajeziku M:

<nenulta cifra> ::= 1|2|3|4|5|6|7|8|9
<cifra> ::= ∅ |<nenulta cifra>
<prirodan broj> ::=<nenulta cifra>|
<prirodan broj> <cifra>

dobija se sledećim transformacijama:

M': <nenulta cifra> ::= 1
<nenulta cifra> ::= 2
⋮
<nenulta cifra> ::= 9
<cifra> ::= ∅
<cifra> ::=<nenulta cifra>
<prirodan broj> ::= <nenulta cifra>
<prirodan broj> ::= <prirodan broj><cifra>

i sastoji se iz sledećih univerzalnih metaformula:

M'': <nenulta cifra> ::= S_1(1);
<nenulta cifra> ::= S_1(2);
⋮
<nenulta cifra> ::= S_1(9);
<cifra> ::= S_1(∅);
<cifra> ::= S_1(<nenulta cifra>);
<prirodan broj> ::= S_1(<nenulta cifra>);
<prirodan broj> ::= S_2(<prirodan broj>,<cifra>);

Da bi razumeli ovaj metajezik neophodno je znati:

- da je induktivno-generativan,
- azbuku objekt-jezika:
  Ø 1 2 3 4 5 6 7 8 9 i
- azbuku meta-jezika:
 < nenulta cifra><cifra> <prirodan broj> ::=S_1S_2( ),;

3.11. Ingerman-ova notacija

Ingerman-ova notacija opisuje konstrukcije (pojmove) programskih jezika kao niz pravila. Pravila su reči čija su slova simboli objekt-jezika i/ili metajezika. Poslednje slovo reči je metasimbol.

Ingerman-ova notacija je pojednostavljena Backus-ova notacija.

Backus-ovoj metalingvističkoj formuli

$$A::=\alpha$$

odgovara

Ingerman-ovo pravilo

αA

α je prost metalingvistički izraz,

A je metalingvistička promenljiva.

Prost metalingvistički izraz sastoji se iz metalingvističkih konstanti i/ili metalingvističkih promenljivih medjusobno razdvojenih metalingvističkim operatorima spajanja.

Ingerman-ova pravila ne sadrže univerzalni metasimbol dodeljivanja već se on podrazumeva.

Prednost Ingerman-ove notacije u odnosu na sve druge notacije je krajnja jednostavnost, a nedostatak veći broj pravila tj. male izražajne mogućnosti.

U slučaju Ingerman-ove notacije broj univerzalnih metasimbola je zaista minimalan i iznosi 2, tj. jedini univerzalni metasimboli su < i > i koriste se za obeležavanje metapromenljivih.

Primer

Jezik jednostavnih aritmetičkih naredbi može se opisati sledećim nizom Ingerman-ovih pravila izvodjenja:

<promenljiva> = <aritmetički izraz> <aritmetička naredba>
<sabirak><aritmetički izraz>
<aritmetički izraz> + <sabirak> <aritmetički izraz>
<činilac> <sabirak>
<sabirak> * <činilac> <sabirak>
<promenljiva> <činilac>
<ceo broj> <činilac>
(<aritmetički izraz>) <činilac>
<slovo><promenljiva>
<promenljiva> <slovo><promenljiva>
<promenljiva><cifra><promenljiva>
<cifra> <ceo broj>
<ceo broj><cifra><ceo broj>
A <slovo>
B <slovo>
C <slovo>
⋮
X <slovo>
Y <slovo>
Z <slovo>
Ø <cifra>
1 <cifra>
⋮
9 <cifra>

Primedba

Na prvi pogled izgleda prirodnije da su Ingerman-ova pravila oblika:

$$A\alpha$$

ali praksa je pokazala da je oblik:

$$\alpha A$$

podesniji.

3.12. Iverson-ova notacija

Iverson je 1964 godine predložio notaciju kojom je želeo da izbegne "brbljivost" Backus-ove notacije i informativnu bezličnost notacije Chomsky-og.

U Iverson-ovoj notaciji gramatika se zadaje u obliku tabele. Tabela se u principu sastoji iz tri kolone. U prvoj koloni se upisuje redni broj pravila (tj. broj reda). U drugoj koloni se upisuje leva strana Backus-ove metalingvističke formule. U trećoj koloni se upisuje desna strana Backus-ove metalingvističke formule. Novina je da se umesto metalingvističke promenljive upisuje broj reda koji sadrži definiciju metalingvističke promenljive.

Primer

Gramatika G_a zapisana Iversón-ovom notacijom je:

1	aritmetička naredba	5 = 2
2	aritmetički izraz	3 | 2 + 3
3	sabirak	4 | 3 * 4
4	činilac	5 | 6 | (2)
5	promenljiva	7|57|58
6	ceo broj	8 | 68
7	slovo	A|B|C|D|... | X|Y|Z
8	cifra	ø| 1| 2|3|4| 5|6|7|8|9

Primetimo da se više ne koriste univerzalni metasimboli <i>.

3.13. Obrnuta notacija

Obrnuta notacija je pokušaj spajanja valjanih osobina prethodno izloženih notacija.

Nedostatak Backus-ove i Ingerman-ove notacije je relativno veliki broj metalingvističkih formula u slučaju malo složenijeg objekt-jezika. Tako kod programskih jezika broj metalingvističkih formula se obično kreće od 200 naviše.

Obrnuta notacija je izmišljena sa ciljem da se smanji broj pravila.

Navedeni simboli imaju sledeće značenje:

[ ]	opcija
{ }	izbor
...	ponavljanje
.n.	n ponavljanja
=	po definiciji je
|	ili
' '	oznaka završnog simbola .

Naziv Obrnuta notacija dolazi od činjenice da se nezavršni simboli ne zagradjuju tj. nestavljaju se u uglaste zagrade već se obrnuto.završni simboli zagradjuju,tj. stavljaju se pod apostrofe.

Obrnuta notacija se često koristi za opisivanje jezika operativnih sistema.

Primer

Sledećim nizom metalingvističkih formula:

aritmetička naredba = promenljiva '=' aritmetički izraz

aritmetički izraz = {sabirak | aritmetički izraz '+' sabirak}

sabirak = {činilac | sabirak '*' činilac}

činilac = {promenljiva | ceo broj | '(' aritmetički izraz ')'}

promenljiva = slovo [{slovo | cifra} ...]

ceo broj = cifra [cifra ...]

slovo = {A|B|C|...|X|Y|Z}

cifra = {Ø|1|2|3|4|5|6|7|8|9}

opisan je jezik jednostavnih aritmetičkih naredbi.

Modifikovanu obrnutu notaciju koristi firma Digital za opis programskih jezika. Tako se na primer u priručniku BASIC-11 Language Reference Manual - Order No. DEC-11-LIBBB-A-D susreću opisi:

- funkcijske naredbe

DEF FN letter [{Ø / %}] (var1 [, var2,...,var5])= expr

- dimenzionisanja rasute datoteke

DIM #integer1, variable (integer2 [, integer3]) [=integer4]

- programskog ciklusa

FOR var = expr1 TO expr2 [STEP expr3]

- uslovnog prelaska

IF relational expression { THEN statement / THEN line number / GOTO line number }

itd.

Očigledno, izmena je u tome što su završni simboli umesto pod znacima navoda napisani velikim slovima.

Obrnuta notacija podesna je za obradu na računaru što je od posebnog značaja.

3.14. Berkhardt-ova notacija

Berkhardt je 1965.godine izmislio "savršen" način razlikovanja simbola objekt-jezika od simbola metajezika. On se setio da metalingvističke promenljive piše u drugoj boji (recimo polumasna slova) ili još jednostavnije da ih podvlači.

Glavni nedostatak Berkhardt-ove notacije je nemogućnost primene na računaru.

Primer

Berkhardt-ova ili Modifikovana Berkhardt-ova notacija se najčešće koriste za opisivanje konstrukcija COBOL-jezika. Recimo definicija odeljka za identifikaciju je:

IDENTIFICATION DIVISION.
PROGRAM-ID. naziv - programa.
[AUTHOR. rečenica . ...]

itd.

Modifikacija je u tome što su službene reči (simboli objekt-jezika) podvučeni umesto metalingvističkih promenljivih.

3.15. Lukas-ova notacija

Za opis gramatika programskih jezika korisno je sledeće proširenje metalingvističkih formula Backus-a koje je predložio Lukas (1969. godine).

(1) Ako je $A \in N$, $Z_i \in N \cup \Sigma (i=1,\ldots,n)$ i $\alpha,\beta \in (N \cup \Sigma)^*$ to se niz pravila izvodjenja:

$$A \rightarrow \alpha\beta$$
$$A \rightarrow \alpha Z_1 \beta$$
$$A \rightarrow \alpha Z_2 \beta$$
$$\vdots$$
$$A \rightarrow \alpha Z_n \beta$$

može napisati pomoću novih univerzalnih simbola [[i]] u obliku:

$$A ::= \alpha [[Z_1 | Z_2 | \cdots | Z_n]] \beta$$

(2) Slično se niz pravila izvodjenja:

$$A \rightarrow \alpha Z_1 \beta$$
$$A \rightarrow \alpha Z_2 \beta$$
$$\vdots$$
$$A \rightarrow \alpha Z_n \beta$$

može napisati pomoću novih univerzalnih simbola { i } u obliku:

$$A ::= \alpha \{ Z_1 | Z_2 | \cdots | Z_n \} \beta$$

(3) Ako je $A \in N$, $\alpha \in (N \cup \Sigma)^*$, to se sledeći par pravila izvodjenja:

$$A \rightarrow \alpha$$
$$A \rightarrow A\alpha$$

može napisati pomoću novog univerzalnog simbola ... u obliku:

$$A ::= \alpha \ldots$$

(4) Ako su A,B i $C \in N$ i $\alpha \in (N \cup \Sigma)^*$ to se pravila izvodjenja:

$$A \rightarrow B$$
$$A \rightarrow BC$$
$$C \rightarrow \alpha B$$
$$C \rightarrow C\alpha B$$

mogu napisati pomoću novog univerzalnog simbola . u obliku:

$$A ::= \{ \alpha . B \ldots \}$$

Zaista, prva dva pravila se mogu napisati kao:

$A \rightarrow B | BC$ tj.

$A \rightarrow B(e | C)$ tj.

$A ::= B [[C]]$

Poslednja dva pravila se mogu napisati kao:

$C \rightarrow C\alpha B \mid \alpha B$ tj.

$C ::= \alpha B \ldots$ tj.

$C ::= \{\alpha B\} \ldots$ **

Spojivši * i ** dobija se

$A ::= B [\![\{\alpha B\} \ldots]\!]$

ili što se skraćeno može napisati

$A ::= \{\alpha . B \ldots\}$

Pomoću Backus-ove notacije može se zadati sintaksa proizvoljnog ALGOL-ikog jezika. Sintaksa programskog jezika ALGOL-60 po pravilu se tako definiše. Sintakse drugih programskih jezika kao što su na primer PL/1 i FORTRAN mogu se takodje definisati pomoću kontekstno-slobodnih gramatika.

Primer

Sledeći niz pravila definiše fragment programskog jezika sličnog PL/I.

<program> ::= <blok>

<blok> ::= početak [<deklaracija>]; <spisak naredbi> kraj

<deklaracija> ::= {;.<dekl> ...}

<dekl> ::= <dekl prom> | <dekl proc> <ostal. dekl>

<dekl prom> ::= {integer | logical <ostale dekl prom>} <prom>

<prom> ::= <ident>

<dekl proc> ::= procedura <ident> [<lista param>]; <nared>

<lista param> ::= ({,. <ident> ...})

<lista nared> ::= {;. <nared> ...}

<nared> ::= <aritm nared> | <blok> | <poziv proc> | <ostale nared>

<poziv proc> ::= call <ident> [<lista argum>]

<lista argum> ::= ({,. <ident> ...})

<aritm nared> ::= <ident> = <izraz>

<izraz> ::= <konst> | <prom> | <bin izraz> | <unarni izraz>

<konst> ::= <log konst> | <integer konst>

<bin izraz> ::= (<izraz> <bin oper> <izraz>)

<unarni izraz> ::= <unar oper> <izraz>

Lukas-ova notacija je bolja od Backus-ove notacije jer je kraća i preglednija.

3.16. Van Wijngaarden-ova notacija

Van Wijngaarden-ovom notacijom izvršena je modifikacija metalingvističkih formula Backus-a sa ciljem da se učine čitljivijim, a pritom zadrže iste izražajne mogućnosti.

Pravila Van Wijngaarden-ove notacije su oblika: "član" za kojim slede "dve tačke" i "niz alternativa" medjusobno razdvojenih "tačka zarezom". Niz alternativa završava se "tačkom". Alternativa je niz - moguće i prazan "članova" medjusobno razdvojenih "zarezom". Član je niz "slova". Član je završni simbol ako se završava rečju "simbol". U suprotnom član je nezavršni simbol. Razmaci su bez značaja i mogu se po želji umetati.

Pravila Van Wijngaarden-ove notacije mogu se na sledeći način zapisati Backus-ovom notacijom:

<pravilo izvodjenja> ::= <član>:<niz alternativa> .
<niz alternativa> ::= <alternativa> |
<niz alternativa> ; <alternativa>
<alternativa> ::= <prazan> | <niz članova>
<niz članova> ::= <član> | <niz članova> , <član>
<član> ::= <slovo> | <član><slovo>
<slovo> ::= a | b | c | d | e | f | g | h | ... | x | y | z
<prazan> ::= -

Navedeni niz metalingvističkih formula može se prevesti u sledeći niz Van Wijngaarden-ovih pravila

(1) pravilo: član,
dve tačke simbol,
niz alternativa,
tačka simbol.

(2) niz alternativa:
alternativa;
niz alternativa,
tačka zarez simbol,
alternativa.

(3) alternativa: prazan;
niz članova.

(4) niz članova: član;
niz članova,
zarez simbol,
član.

(5) član: slovo;
član,
slovo.

(6) slovo: slovo a simbol;
slovo b simbol;
⋮
slovo z simbol;
desna uglasta zagrada simbol;
leva uglasta zagrada simbol.

(7) prazan: .

Dobijeni niz Van Wijngaarden-ovih pravila je potpun tek kada mu se pridruži tabela završnih simbola i njihovih reprezentacija u jeziku.

Završni simbol	Tipografski simbol
dve tačke simbol	:
tačka simbol	.
tačka zarez simbol	;
zarez simbol	,
slovo a simbol	a
slovo b simbol	b
⋮	⋮
slovo z simbol	z
desna uglasta zagrada simbol	>
leva uglasta zagrada simbol	<

Primer

Gramatika G_a zapisana Van Wijngaarden-ovom notacijom je:

(1) aritmetička naredba: promenljiva,
jednakost simbol,
aritmetički izraz.

(2) aritmetički izraz: sabirak;
aritmetički izraz,
plus simbol,
sabirak.

(3) sabirak: činilac;
sabirak,
puta simbol,
činilac.

(4) činilac: promenljiva;
ceo broj;
leva mala zagrada simbol,
aritmetički izraz,
desna mala zagrada simbol.

(5) promenljiva: slovo;
promenljiva,
slovo;
promenljiva,
cifra.

(6) ceo broj: cifra;
ceo broj,
cifra.

(7) slovo: slovo a simbol;
⋮
slovo z simbol.

(8) cifra: cifra nula simbol;
⋮
cira devet simbol.

Tabela završnih simbola je:

Završni simbol	Tipografski simbol
jednakost simbol	=
plus simbol	+
puta simbol	×
leva mala zagrada simbol	(
desna mala zagrada simbol	)
slovo a simbol	a
⋮	⋮
slovo z simbol	z
cifra nula simbol	ø
⋮	⋮
cifra devet simbol	9

3.17. Grafička notacija

Grafička notacija koristi se za neformalno ispitivanje gramatika. Svakoj metalingvističkoj formuli odgovara usmeren graf. Skupu metalingvističkih formula odgovara skup grafova koji kad se povežu čine graf jezika.

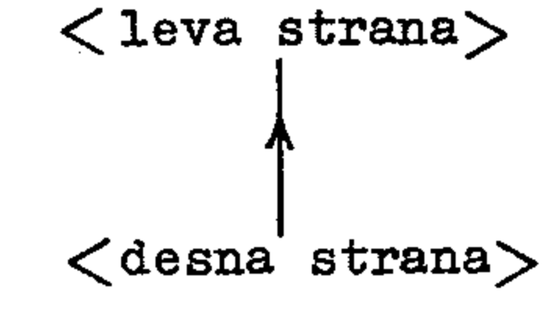
Univerzalni metasimboli Backus-ove notacije grafički se prikazuju na sledeći način:

1) ::= grafički se prikazuje kao vertikalna strelica označavajući smer izvodjenja

Na primer metalingvistička formula

<leva strana> ::= <desna strana>

grafički se prikazuje:

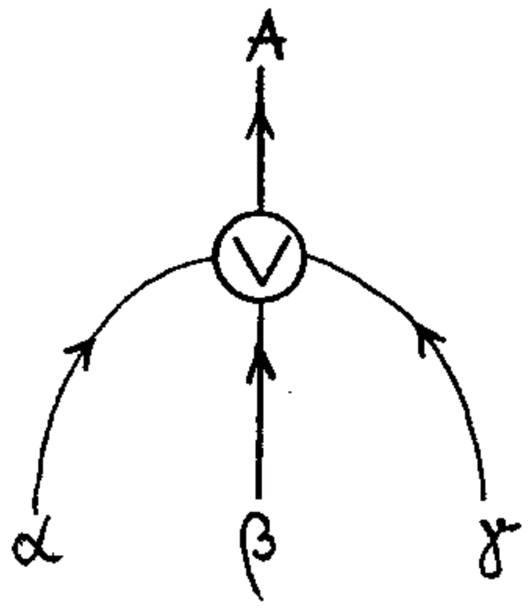


2) | grafički se prikazuje kao krug u koji je upisan simbol $\vee$. Dve ili više strelica dolazi u krug, a samo jedna strelica izlazi iz kruga. Redosled dolazaka strelica u krug je proizvoljan.

Na primer metalingvistička formula

$$A::=\alpha|\beta|\gamma$$

grafički se prikazuje:

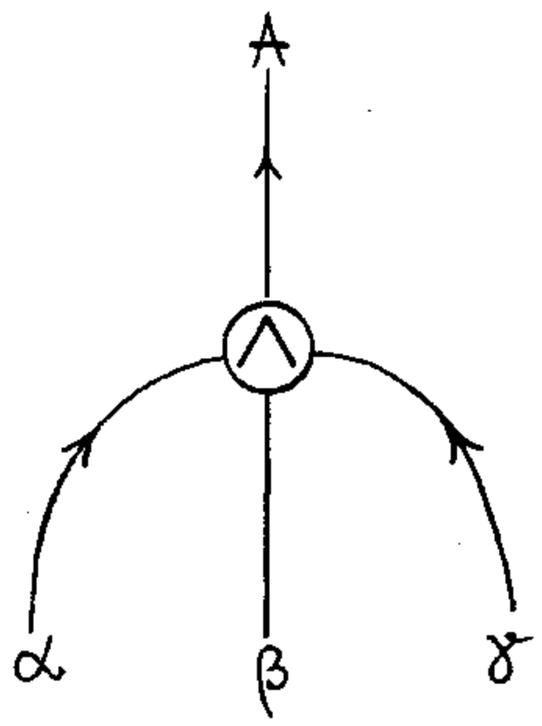


3) Dopisivanje se grafički prikazuje kao krug u koji je upisan simbol $\wedge$. Dve ili više strelica dolazi u krug, a samo jedna strelica izlazi iz kruga. Redosled dolazaka strelica u krug je od značaja i odgovara redosled simbola na desnoj strani odgovarajuće metalingvističke formule.

Na primer metalingvistička formula

$$A::=\alpha\beta\gamma$$

grafički se prikazuje:

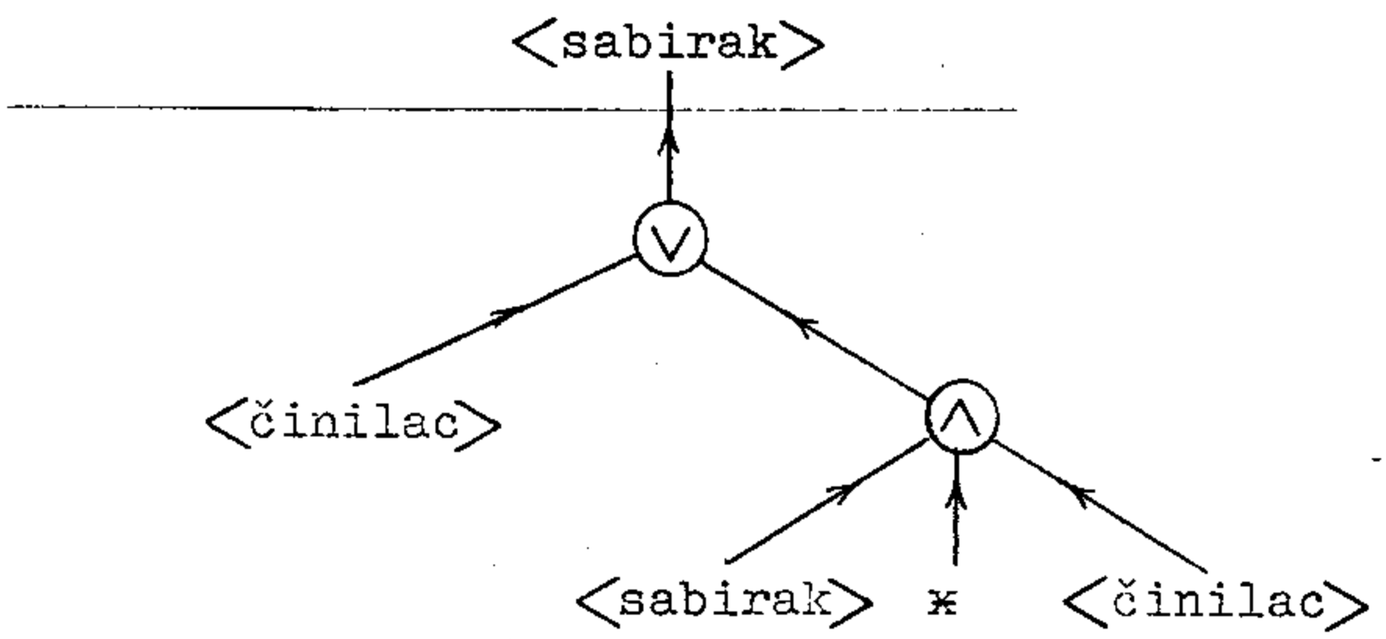


Primer

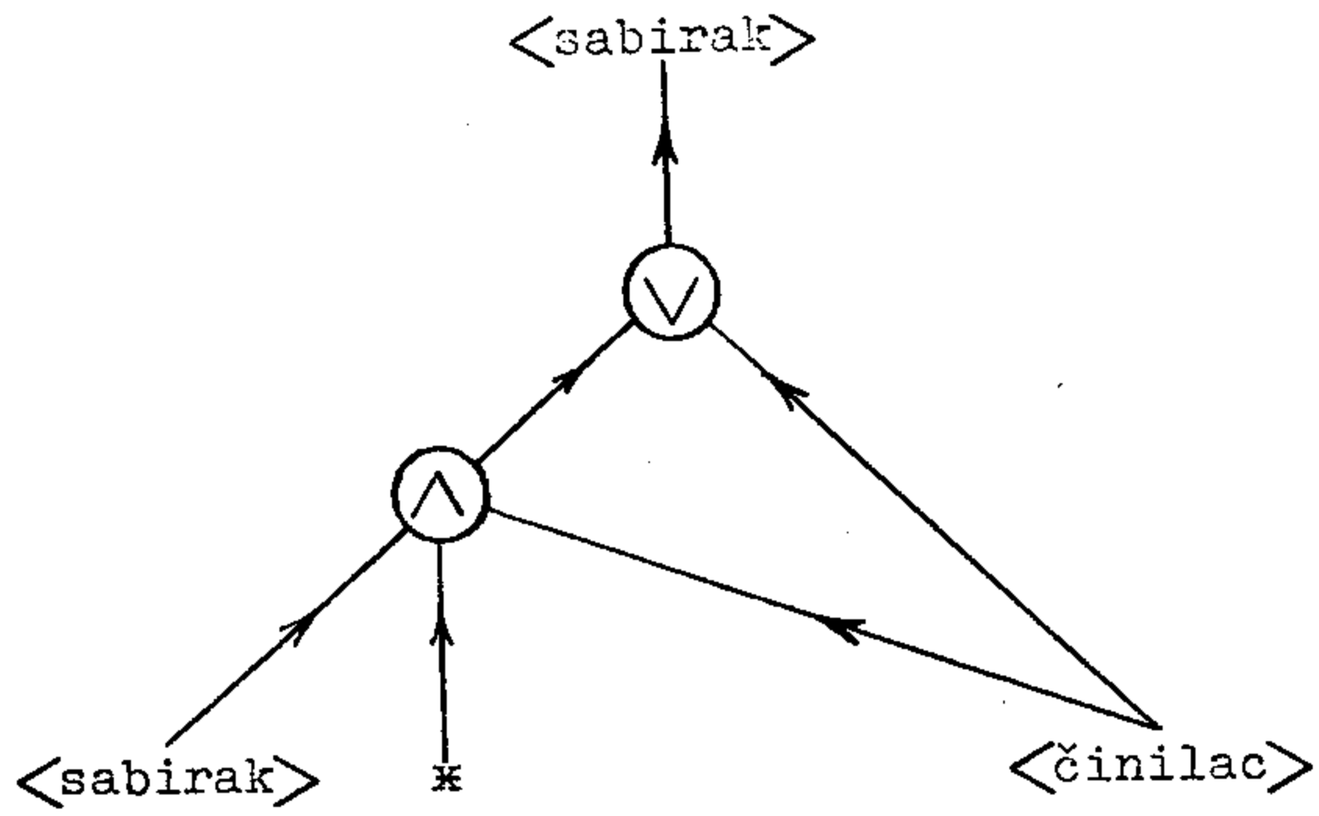
Metalingvistička formula:

<sabirak> ::= <činilac> | <sabirak> * <činilac>

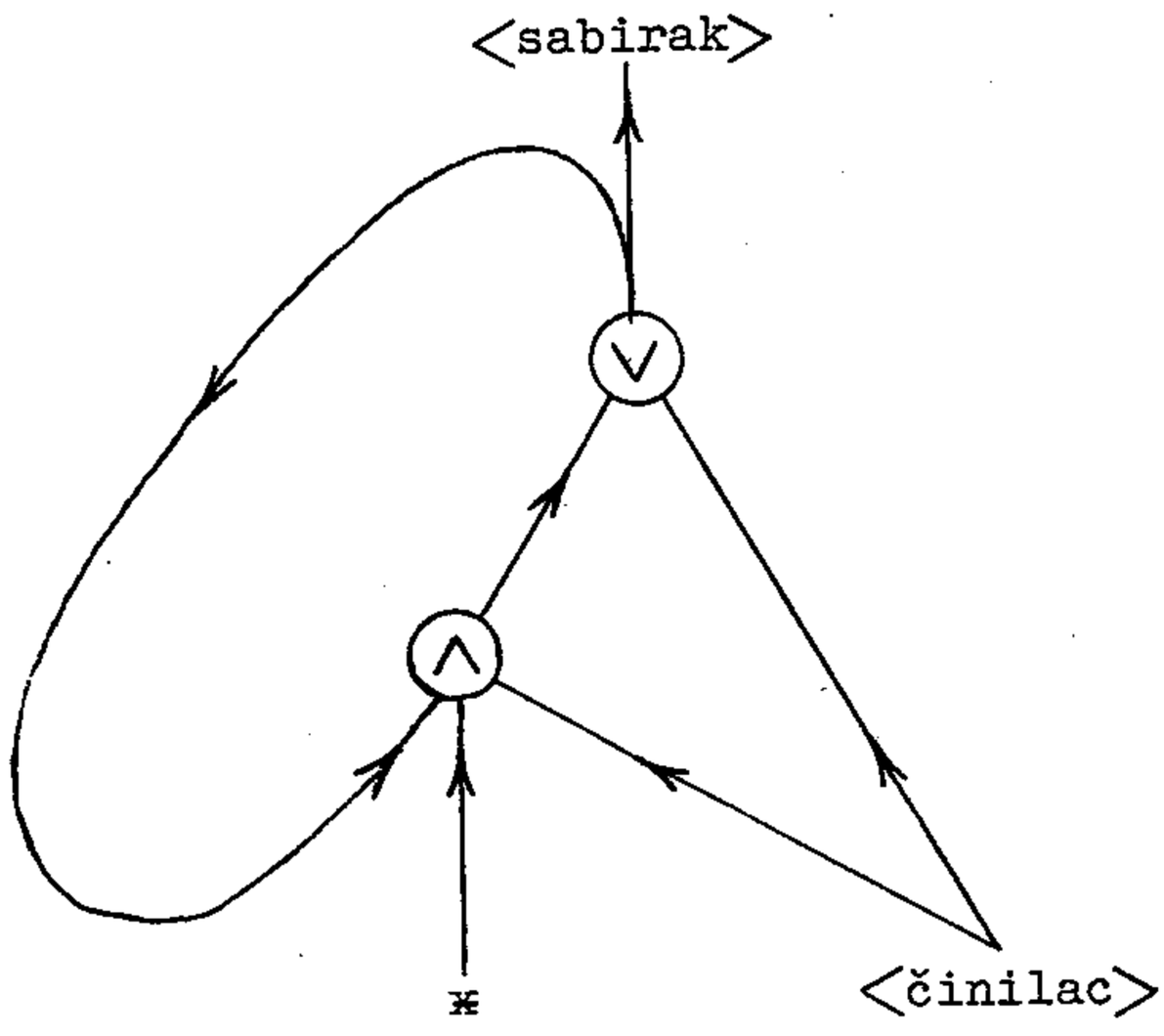
grafički se prikazuje kao usmeren graf:



Transformacijom navedenog grafa dobija se sažetiji graf:



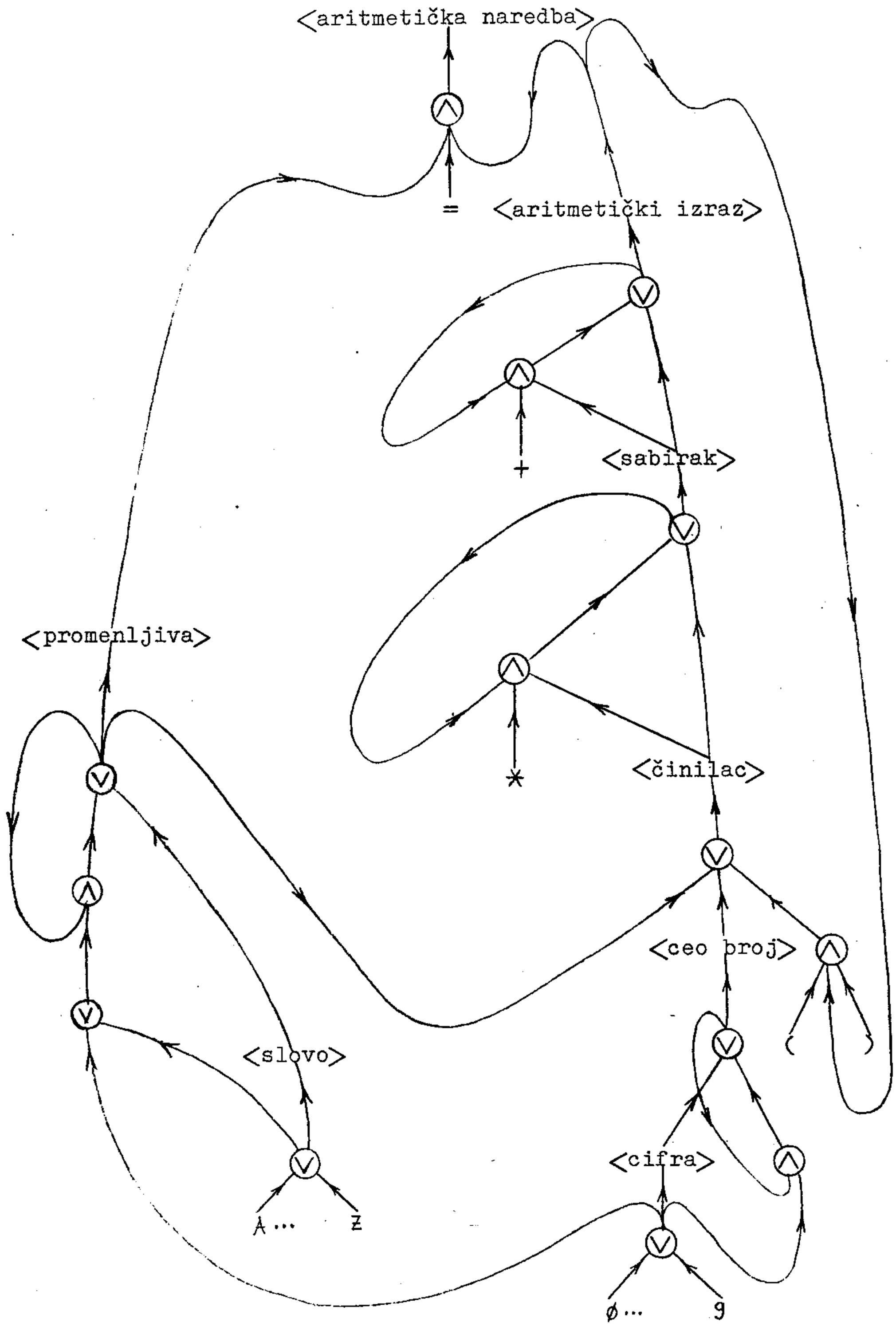
ili još sažetije:



Primer

Skupu metalingvističkih formula kojim je opisan jezik jednostavnih aritmetičkih izraza odgovara sledeći graf:

Dobijeni graf sadrži petlje. Svako pravilo javlja se jedanput.

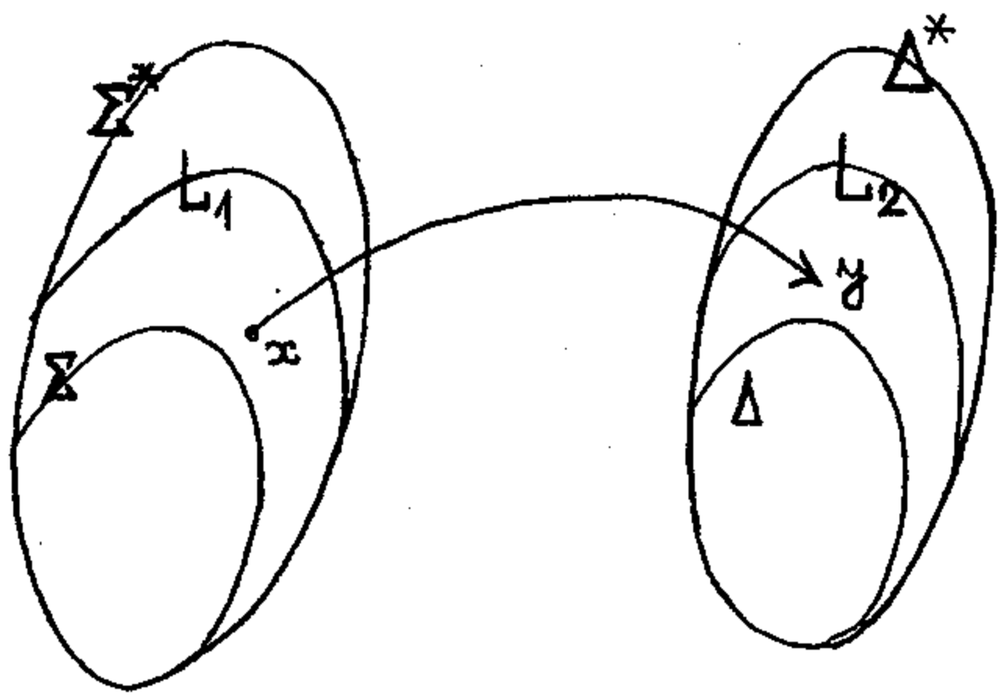

<aritmetička naredba>
=
<aritmetički izraz>
<sabirak>
+
<činilac>
*
<promenljiva>
<slovo>
A ... Z
<ceo broj>
( )
<cifra>
0 ... 9

4. SINTAKSNO-ORIJENTISANO PREVODJENJE

4.1. Formalna definicija prevodjenja

U principu postoje dva načina definisanja prevodjenja. Prvi način je pomoću šeme prevodjenja. Drugi način je pomoću prepoznavača.

Definicija 1: Neka je Σ - ulazna azbuka, a Δ- izlazna azbuka. Prevodjenje jezika $L_1 \subseteq \Sigma^*$ na jezik $L_2 \subseteq \Delta^*$ je preslikavanje τ skupa Σ^* na skup Δ^*, tj. $\tau: \Sigma^* \to \Delta^*$. Jezik L_1 je oblast definisanosti, a jezik L_2 je skup vrednosti preslikavanja τ .



Drugačije rečeno, prevodjenje τ jezika L_1 na jezik L_2 je podskup skupa $\Sigma^* \times \Delta^*$.

$$\tau \subseteq \Sigma^* \times \Delta^*$$

Oblast definisanosti prevodjenja τ je jezik L_1:

$L_1 = \{x \mid$ za neko y, $(x,y) \in \tau\}$.

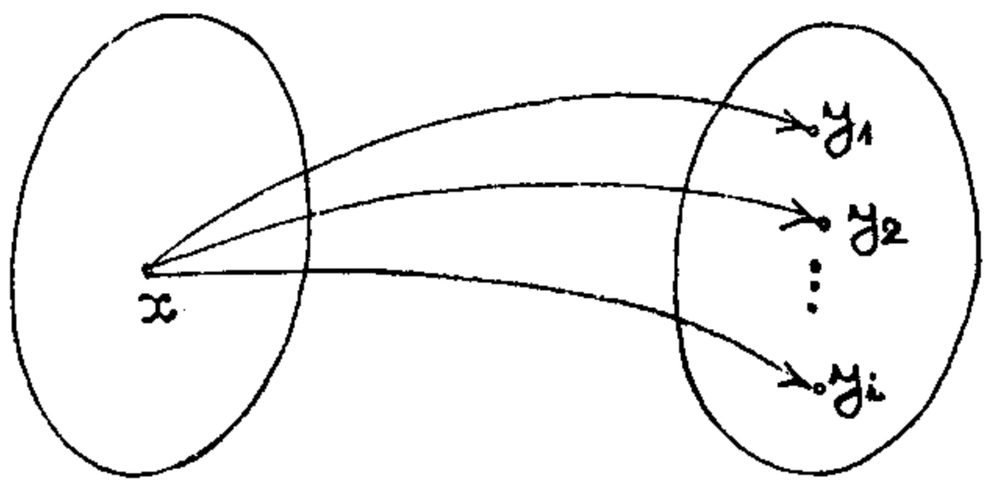
Skup vrednosti prevodjenja τ je jezik L_2:

$L_2 = \{y \mid$ za neko x, $(x,y) \in \tau\}$.

Jezik L_1 naziva se ulazni (izvorni) jezik, a jezik L_2 izlazni (ciljni) jezik.

U slučaju da je $(x,y)\in\tau$ niska x naziva se ulazna niska, a niska y izlazna niska. Niska y je prevod niske x.

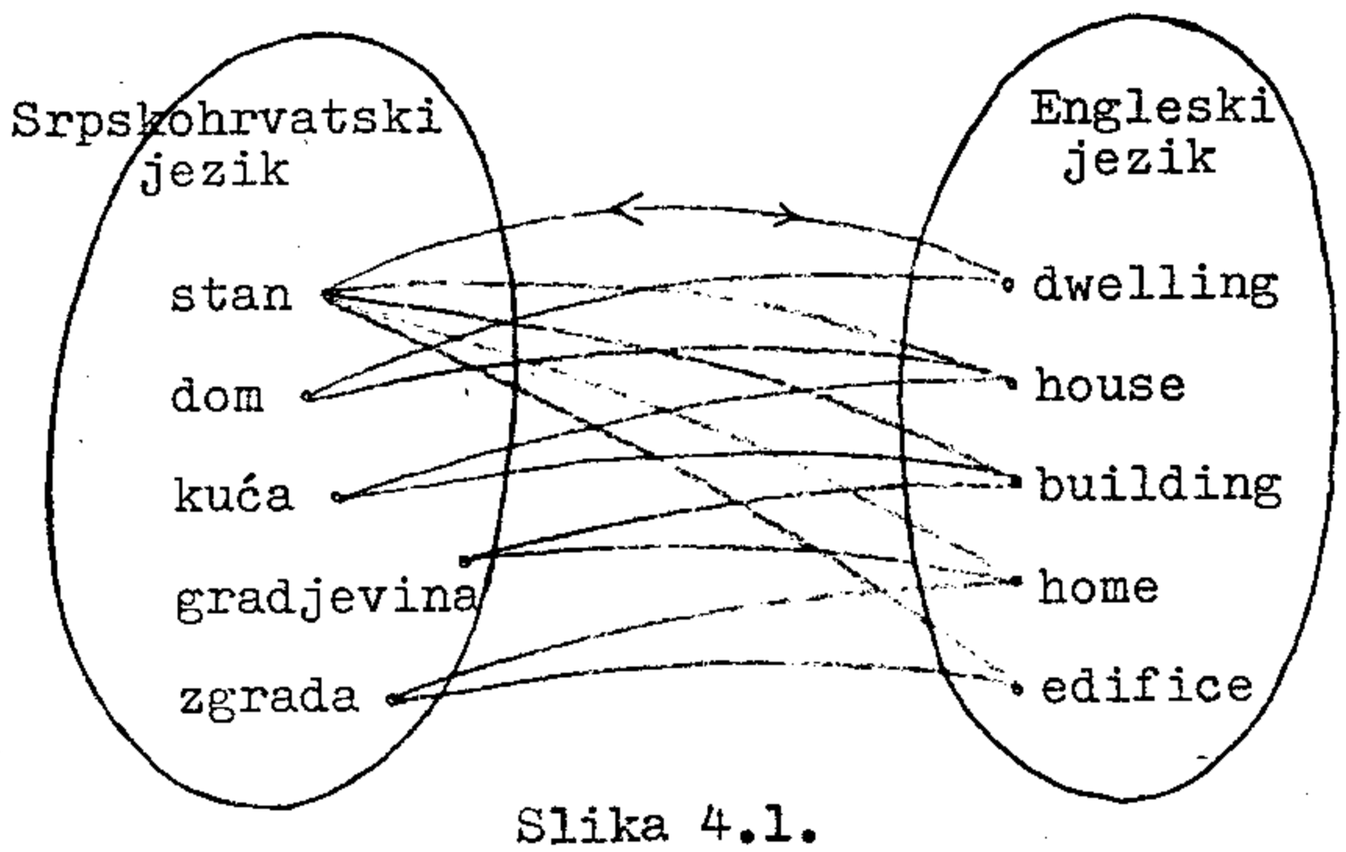
U opštem slučaju preslikavanje τ ne mora da bude jednoznačno, tj. za datu ulaznu nisku x mogu da postoje više izlaznih niski y_i; i≥1.



To je čest slučaj u prirodnim jezicima.

Primer

Slika 4.1. ilustruje nejednoznačnost prevodjenja srpskohrvatskog jezika na engleski jezik i obratno.

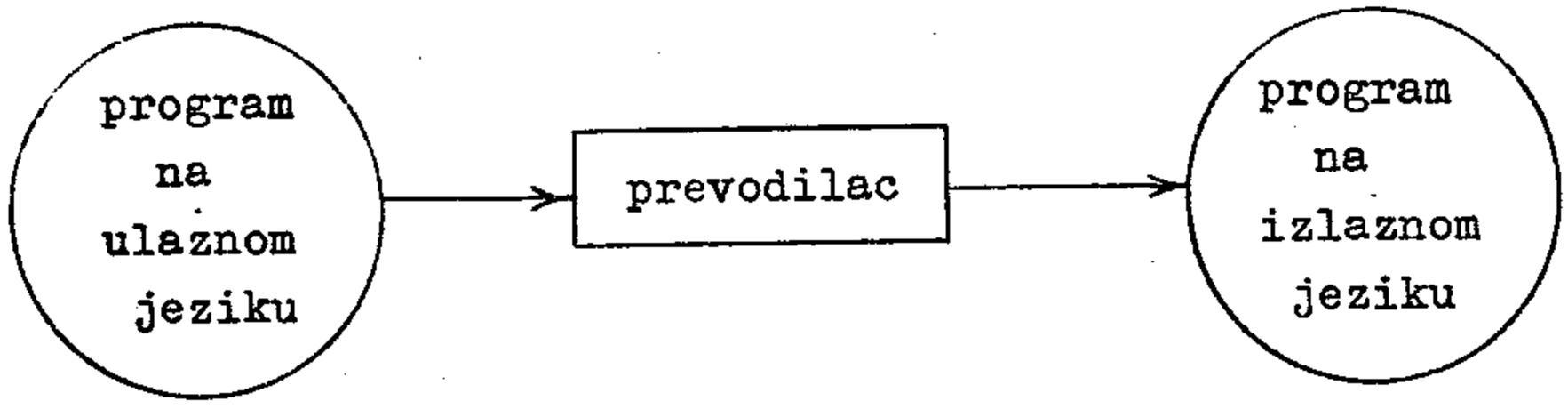


Slika 4.1.

Prevodjenje jednog programskog jezika na drugi programski jezik mora da bude jednoznačno preslikavanje tj. funkcija.

Primer

Prevodilac izvršava prevodjenje programa sa ulaznog jezika* na izlazni jezik**.



Izražavajući se jezikom matematike možemo reći da je prevodilac funkcija jedne nezavisne promenljive. Vrednost nezavisne promenljive je program na ulaznom jeziku. Vrednost funkcije prevodilac je program na izlaznom jeziku. Ulazni i izlazni jezik mogu da budu isti ali i različiti jezici. U literaturi se najčešće sreće sledeća klasifikacija:

ulazni jezik	izlazni jezik	prevodilac
mašinski-orijentisan, simbolički, niži	mašinski	asembler
proceduralno-orijentisan, viši	mašinski, medju,	kompilator
problemski-orijentisan	mašinski, medju	generator

Kompilator se najčešće sastoji iz četiri dela:

- leksičkog analizatora,
- sintaksnog analizatora,
- generatora kôda i
- optimizatora kôda.

Svaki od navedenih delova izvršava odredjenu funkciju (vrstu prevodjenja).

---

* Niske raspoloživih simbola

** Prevod takodje može da bude izveštaj o otkrivenim greškama.

Leksički analizator izvršava leksičku analizu.

Sintaksni analizator izvršava sintaksnu analizu.

Generator kôda izvršava generisanje kôda.

Optimizator kôda izvršava optimizaciju kôda.

Leksička analiza je prevodjenje programa na ulaznom jeziku u niz leksičkih konstrukcija.

Sintaksna analiza je prevodjenje niza leksičkih konstrukcija u niz sintaksnih konstrukcija predstavljenih u obliku drveta.

Generisanje kôda je prevodjenje niza sintaksnih konstrukcija u program na izlaznom (medju) jeziku.

Optimizacija kôda je prevodjenje programa sa izlaznog jezika u optimizovan program na izlaznom jeziku.

Znači, kompilator izvršava redom funkcije:

1. l - leksička analiza,
2. s - sintaksna analiza,
3. g - generisanje kôda i
4. o - optimizacija kôda.

$y=K(x)=o(g(s(l(x))))$

x je program na ulaznom jeziku,

y je program na izlaznom jeziku.

U slučaju otkrivanja greške (leksičke, sintaksne itd.) u programu vrednost funkcije kompilacije je odgovarajući izveštaj o grešci.

Osnovni i najznačajniji problem teorije prevodjenja je zadavanje funkcije provodjenja.

Definicija 2: Neka je Σ ulazna azbuka i Δ izlazna azbuka. Homomorfizam je preslikavanje h ulazne azbuke Σ u skup Δ^* svih reči izlazne azbuke Δ tj. $h:\Sigma \rightarrow \Delta^*$.

Oblast definisanosti homomorfizma h proširuje se na skup Σ^* svih reči ulazne azbuke Σ uzimanjem da je:

$h(e)=e$, e je prazna reč i

$h(xa)=h(x)h(a)$, $x\in\Sigma^*$ i $a\in\Sigma$

Primer

Neka je $\Sigma=\{a,b\}$ i $\Delta=\{\emptyset,1\}$. Definišimo homomorfizam h na sledeći način:

$h(a)=\emptyset$

$h(b)=1$.

Ako je jezik $L=\{a,b\}^*=\Sigma^*$

onda je $h(L)=\{\emptyset,1\}^*=\Delta^*$

Teorema 1: U opštem slučaju funkcija prevodjenja nije homomorfizam.

Dokaz: Teoremu ćemo dokazati navodjenjem kontraprimera*.

Kontraprimer

Funkcija prevodjenja $\tau : x \to x^R$, $x \in \{a,b\}^*$ ne može se zadati pomoću homomorfizma.

Zaista, po definiciji je na primer za reč x=aab

$$\tau(aab)=baa$$

Ako bi funkcija prevodjenja τ bila homomorfizam važilo bi:

$$\tau(aab)= \tau(aa)\,\tau(b)= \tau(a)\,\tau(a)\,\tau(b),$$

odnosno:

$$\tau(a)=b$$
$$\tau(a)=a$$
$$\tau(b)=a\ .$$

Prevod slova a je slovo a ili slovo b što je suprotno definiciji funkcije prevodjenja τ .

4.2. Šema sintaksno-orijentisanog prevodjenja

Definicija 3 Prevodjenje konačnog skupa niski tj. konačnog jezika naziva se konačno prevodjenje. Slično, prevodjenje beskonačnog skupa niski tj. beskonačnog jezika naziva se beskonačno prevodjenje.

Problem definisanja beskonačnog prevodjenja analogan je problemu definisanja beskonačnog jezika.

Jedan od formalizama koji se koristi za definisanje beskonačnog prevodjenja je šema sintaksno-orijentisanog prevodjenja.

---

* Direktan dokaz teoreme sledi na osnovu sledećih činjenica:
- Neka su Σ i Δ konačne azbuke i h homomorfizam $h:\Sigma \to \Delta^*$.
  Postoji najviše prebrojivo mnogo homomorfizama h.
- Neka su $L_1 \subseteq \Sigma^*$ i $L_2 \subseteq \Delta^*$ beskonačni jezici i τ prevodjenje jezika L_1 u jezik L_2 tj. $\tau : L_1 \to L_2$.
  Postoji najmanje $2^{\aleph_0}$ prevodjenja τ .

Šema sintaksno-orijentisanog prevodjenja je gramatika ulaznog jezika čijem je svakom pravilu izvodjenja pridruženo pravilo prevodjenja. Kad god se primenjuje pravilo izvodjenja primenjuje se odgovarajuće pravilo prevodjenja. Znači, proces izvodjenja i proces prevodjenja izvršavaju se uporedo.

Primer

Data je gramatika $G=(\{S\},\{a,b,e\},P,S)$
gde je P skup sledećih pravila izvodjenja:

$$S \to aS$$
$$S \to bS$$
$$S \to e$$

Gramatika G generiše jezik $L=\{a,b\}^*$.

Dalje, neka je τ simetrično prevodjenje:

$$\tau : x \to x^R \quad ; \quad x \in \{a,b\}^*$$

Šema sintaksno-orijentisanog prevodjenja τ je:

redni broj	pravilo izvodjenja	pravilo prevodjenja
1	$S \to aS$	$S \to Sa$
2	$S \to bS$	$S \to Sb$
3	$S \to e$	$S \to e$

Izvodjenje i prevodjenje niske aab je:

$S \overset{1}{\Rightarrow} aS$
$\overset{1}{\Rightarrow} aaS$
$\overset{2}{\Rightarrow} aabS$
$\overset{3}{\Rightarrow} aabe = aab$

$S \overset{1}{\Rightarrow} Sa$
$\overset{1}{\Rightarrow} Saa$
$\overset{2}{\Rightarrow} Sbaa$
$\overset{3}{\Rightarrow} ebaa = baa$

Zaista, $\tau(aab)=baa$.

Umesto razdvojenog uobičajeno je spojeno pisanje:

$(S,S) \overset{1}{\Rightarrow} (aS,Sa)$
$\overset{1}{\Rightarrow} (aaS,Saa)$
$\overset{2}{\Rightarrow} (aabS,Sbaa)$
$\overset{3}{\Rightarrow} (aabe,ebaa)=(aab,baa)$.

Članovi niza izvedenih parova niski su oblika:

(ulazna niska, izlazna niska)

Izlazna niska dobija se zamenom odgovarajućeg nezavršnog simbola desnom stranom pravila prevodjenja, pridruženog pravilu izvodjenja upotrebljenog za izvodjenje ulazne niske.

Definicija 4:Prevodilac izvršava prevodjenje τ ako za svaku ulaznu nisku x odredjuje izlaznu nisku y, takvu da je $(x,y)\in\tau$.

Šema sintaksno orijentisanog prevodjenja T definiše prevodjenje $\tau(T)$. Na osnovu šeme sintaksno-orijentisanog prevodjenja T može se konstruisati prevodilac koji će izvršavati prevodjenje $\tau(T)$. Za datu ulaznu nisku x, pomoću pravila izvodjenja šeme sintaksno-orijentisanog prevodjenja T, prevodilac odredjuje (ako je to moguće) izvodjenje niske x iz početnog simbola S:

$$S=\alpha_0=\alpha_1=\alpha_2=\ \ldots=\ \alpha_n=x$$

Zatim prevodilac, koristeći pravila prevodjenja šeme sintaksno-orijentisanog prevodjenja T i prethodno izvodjenje niske x izvršava prevodjenje:

$$S=\beta_0=\beta_1=\beta_2=\ \ldots=\ \beta_n=y$$

Niska y je prevod niske x.

Niska α_i dobija se iz niske α_{i-1} (i=1,...,n) primenom odgovarajućeg pravila izvodjenja. Slično, niska β_i dobija se iz niske β_{i-1} (i=1,...,n) primenom odgovarajućeg pravila prevodjenja.

Izvodjenje i prevodjenje niske x može da se izvršava:

- odvojeno
- istovremeno.

Pri kompilaciji programskih jezika susreće se prevodjenje aritmetičkih izraza iz uobičajenog (infiksnog) zapisa u tzv. poljski zapis.

Zapis aritmetičkih izraza bez upotrebe zagrada naziva se bezzagradni zapis. Poljski zapis je najvažnija vrsta bezzagradnog zapisa.

Definicija 5.* Označimo redom sa:

in(I),
pre(I) i
post (I)

infiksni, prefiksni i postfiksni zapis aritmetičkog izraza I.**

in(I) po def. I.

Neka je dalje:

O - skup znakova binarnih operacija i

A - skup operanada.

Prefiksni i postfiksni poljski zapis rekursivno se definišu na sledeći način:

(1) Ako je izraz I=a, gde je a∈A, onda je:

a) pre(I)=pre(a)=a

b) post(I)=post(a)=a

(2) Ako je izraz $I=I_1 o I_2$, gde je o znak binarne operacije, a I_1 i I_2 izrazi, onda je:

a) $pre(I)=pre(I_1 o I_2)=o\ pre(I_1)pre(I_2)$

b) $post(I)=post(I_1 o I_2)=post(I_1)post(I_2)o$

(3) Ako je izraz I=(I), onda je:

a) pre(I)=pre((I))

b) post(I)=post((I))

Primer

Prefiksni poljski zapis izraza a⁎(b+c) je ⁎a+bc.

Dokaz: pre(a⁎(b+c)) $\overset{2a}{=}$ ⁎pre(a)pre((b+c))
$\overset{1a}{=}$ ⁎a pre((b+c))
$\overset{3a}{=}$ ⁎a pre(b+c)
$\overset{2a}{=}$ ⁎a+pre(b)pre(c)
$\overset{1a}{=}$ ⁎a+b pre(c)
$\overset{1a}{=}$ ⁎a+bc .

Postfiksni poljski zapis izraza a⁎(b+c) je abc+⁎

Dokaz: post (a⁎(b+c)) $\overset{2b}{=}$ post(a)post((b+c))⁎
$\overset{1a}{=}$ a post((b+c))⁎
$\overset{3b}{=}$ a post(b+c)⁎
$\overset{2b}{=}$ a post(b)post(c)+⁎
$\overset{1a}{=}$ ab post(c)+⁎
$\overset{1a}{=}$ abc+⁎ .

⁎ Postoji i tzv. direktna definicija poljskog zapisa

⁎⁎ Pretpostavlja se da je aritmetički izraz I potpuno zagradjen, odnosno da je poštovana konvencija o brisanju zagrada

Primer

Data je gramatika G=({E,T,F} , {+,*,(,),a,b,c} ,P,E)
gde je P skup pravila izvodjenja:

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

Gramatika G generiše jezik aritmetičkih izraza u infiks zapisu.

Skup znakova binarnih operacija o={+,*}

Skup operanada A= {a,b,c}.

Neka je τ prevodjenje aritmetičkih izraza iz infiks zapisa u prefiksni poljski zapis:

τ: aritmetički izraz u infiksnom zapisu $\longrightarrow$ aritmetički izraz u prefiksnom poljskom zapisu

Šema sintaksno-orijentisanog prevodjenja aritmetičkog izraza iz infiks zapisa u prefiksni poljski zapis je:

redni broj	pravilo izvodjenja	pravilo prevodjenja
1	E → T	E → T
2	E → E+T	E → +ET
3	T → F	T → F
4	T → T*F	T → *TF
5	F → (E)	F → E
6	F → a	F → a
7	F → b	F → b
8	F → c	F → c

Izvodjenje i prevodjenje niske a*(b+c) je:

$$(E,E) \overset{1}{\Rightarrow} (T,T) \overset{4}{\Rightarrow} (T*F,\ *TF) \overset{3}{\Rightarrow} (F*F,\ *FF)$$

$\overset{6}{\Rightarrow}$ (a*F, *aF)
$\overset{5}{\Rightarrow}$ (a*(E), *aE)
$\overset{2}{\Rightarrow}$ (a*(E+T), *a+ET)
$\overset{1}{\Rightarrow}$ (a*(T+T), *a+TT)
$\overset{3}{\Rightarrow}$ (a*(F+T), *a+FT)
$\overset{7}{\Rightarrow}$ (a*(b+T), *a+bT)
$\overset{3}{\Rightarrow}$ (a*(b+F), *a+bF)
$\overset{8}{\Rightarrow}$ (a*(b+c), *a+bc)

Znači, prevod niske a*(b+c) je *a+bc

Na sličan način može se definisati šema sintaksno-orijentisanog prevodjenja aritmetičkih izraza iz postfiksnog poljskog zapisa u infiksni zapis.

redni broj	pravilo izvodjenja	pravilo prevodjenja
1	E → T	E → T
2	E → ET+	E → E+T
3	T → F	T → F
4	T → TF*	T → T*F
5	F → E	F → (E)
6	F → a	F → a
7	F → b	F → b
8	F → c	F → c

Neka je Θ - skup operacija, tj.

$\Theta = \{o(i,j,k) \mid i=1...m, j=1...n, k=1...1\}$

Indeks i označava prioritet operacije.
Indeks j označava arnost operacije.
Indeks k označava vrstu operacije.
Dalje, neka je Σ skup argumenata, tj.

$\Sigma = \{a_s \mid s=1...t\}$

Gramatika $G_{pre}=(\{T_i | i=o...m\}, \theta \cup \Sigma, P_{pre}, T_o)$

gde je P_{pre} skup pravila izvodjenja:

$$T_{i-1} \rightarrow T_i \bigg|_{j=1}^{n} \bigg|_{k=1}^{l} o(i,j,k)\ T_{i-1}^{j-1}\ T_i;\ i=1,2,...,m-1$$

$$T_m \rightarrow T_o \bigg|_{s=1}^{t} a_s$$

generiše jezik aritmetičkih izraza sastavljenih iz operacija skupa θ i argumenata skupa Σ napisanih u prefiksnom poljskom zapisu.

Članovi za koje nije definisana operacija izostavljaju se.

Gramatika $G_{post}=(\{T_i | i=o...n\}, \theta \cup \Sigma, P_{post}, T_o)$

gde je P_{post} skup pravila izvodjenja:

$$T_{i-1} \rightarrow T_i \bigg|_{j=1}^{n} \bigg|_{k=1}^{l} T_{i-1}^{j-1}\ T_i\ o(i,j,k);\ i=1,2,...,m-1$$

$$T_m \rightarrow T_o \bigg|_{s=1}^{t} a_s$$

generiše jezik aritmetičkih izraza sastavljenih iz operacija skupa θ i argumenata skupa Σ napisanih u postfiksnom poljskom zapisu.

Članovi za koje nije definisana operacija izostavljaju se.

Skupovi pravila izvodjenja P_{pre} i P_{post} napisani su veoma sažeto, koristeći sledeću notaciju:

1) $\bigg|_{i=1}^{m} X_i = X_1 | X_2 | X_3 | ... | X_m$

2) $X^j = \underbrace{XX...X}_{j\text{-puta}}$, $X^o=e$, $X^1=X$

U razvijenom obliku skup pravila izvodjenja P_{pre} je:

$T_o \rightarrow T_1 | o(1,1,1)T_1 | o(1,1,2)T_1 | ... | o(1,1,1)T_1 |$

$o(1,2,1)T_oT_1 | o(1,2,2)T_oT_1 | ... | o(1,2,1)T_oT_1 | ...$

$\vdots$

$o(1,n,1)T_o^{n-1}T_1 | o(1,n,2)T_o^{n-1}T_1 | ... | o(1,n,1)T_o^{n-1}T_1$

$$T_1 \rightarrow T_2 \,|\, o(2,1,1)T_2 \,|\, o(2,1,2)T_2 \,|\ldots \quad \text{itd.}$$

Primer

$$\text{Gramatika } G = \left\{ \{T_i \,|\, i=0\ldots3\},\ \Theta \cup \Sigma, P, T_o \right\}$$

gde je $\Theta = \{$ o(1,2,1)≡+(binarno),
o(1,2,2)≡-(binarno),
o(2,2,1)≡×,
o(2,2,2)≡/,
o(3,1,1)≡⊕ (unarno),
o(3,1,2)≡⊖ (unarno) i
o(3,2,1)≡↑$\}$

$\Sigma = \{a\}$ i

$P = \{ T_o \rightarrow T_1 \,|\, +T_oT_1 \,|\, -T_oT_1$
$T_1 \rightarrow T_2 \,|\, \times T_1T_2 \,|\, /T_1T_2$
$T_2 \rightarrow T_3 \,|\, +\,T_3 \,|\, -\,T_3 \,|\, \uparrow T_2T_3$
$T_3 \rightarrow T_o \,|\, a \}$

generiše jezik aritmetičkih izraza sastavljenih iz uobičajenih aritmetičkih operacija i argumenta a napisanih u prefiksnom poljskom zapisu.

Gramatike G_{pre} i G_{post} dobijene su uopštavanjem konkretnih gramatika.

Gramatike G_{pre} i G_{post} su kontekstno-slobodne i imaju veliku praktičnu i teorijsku vrednost. Na primer ako su članovi skupa Θ - logičke operacije, a članovi skupa Σ - logičke konstante i/ili promenljive onda gramatike G_{pre} i G_{post} redom definišu jezike logičkih izraza napisanih u prefiksnom odnosno postfiksnom poljskom zapisu.

Gramatika G_{pre} može se iskoristiti za definisanje $\mathcal{L}$-izraza programskog jezika LISP.

Teorema 2*: Prevodjenje izraza iz prefiksnog poljskog zapisa u postfiksni poljski zapis je jednoznačno. Važi i obratno.

Dokaz: Gramatika G_{pre} jednoznačno definiše jezik izraza napisanih u prefiksnom poljskom zapisu. Slično, gramatika G_{post}

---

* Ovu teoremu je već dokazao Lukašijevič samo na drugačiji način

na jednoznačan način definiše jezik izraza napisanih u postfiksnom poljskom zapisu. Izmedju pravila izvodjenja gramatike G_{pre} i gramatike G_{post} postoji sledeća obostrano jednoznačna korespodencija:

$$T_{i-1} \rightarrow T_i \prod_{j=1}^{n} \coprod_{k=1}^{l} o(i,j,k) T_{i-1}^{j-1} T_i \leftrightarrow T_{i-1} \rightarrow T_i \prod_{j=1}^{n} \coprod_{k=1}^{l} T_{i-1}^{j-1} T_i o(i,j,k)$$

$$i=1,2,\ldots,m-1$$

$$T_m \rightarrow T_o \coprod_{s=1}^{t} a_s \leftrightarrow T_m \rightarrow T_o \coprod_{s=1}^{t} a_s$$

koja ujedno predstavlja dokaz tvrdjenja teoreme.

Primer

Sledeća šema sintaksno-orijentisanog prevodjenja definiše prevodjenje aritmetičkih izraza iz prefiksnog poljskog zapisa u postfiksni poljski zapis.

redni broj	pravilo izvodjenja	pravilo prevodjenja
1	E → T	E → T
2	E → +ET	E → ET+
3	T → F	T → F
4	T → *TF	T → TF*
5	F → E	F → E
6	F → a	F → a
7	F → b	F → b
8	F → c	F → c

Ako pravilo izvodjenja i pravila prevodjenja promene mesto dobija se šema provodjenja izraza iz postfiksnog poljskog zapisa u prefiksni poljski zapis.

Definicija 6: Šema sintaksno-orijentisanog prevodjenja T je petorka:

$$T = (N, \Sigma, \Delta, R, S)$$

gde je:

(1) N - konačan skup nezavršnih simbola,

(2) Σ - konačan skup završnih simbola - ulazna azbuka,

(3) Δ - konačan skup prevedenih simbola - izlazna azbuka.

$N \cap (\Sigma \cup \Delta) = \emptyset$

(4) R - konačan skup pravila izvodjenja i prevodjenja oblika:

$A \longrightarrow \alpha, \beta, \Pi$

gde je:

$A \in N$,

$\alpha \in (N \cup \Sigma)^{*}$

$\beta \in (N \cup \Delta)^{*}$ i

Π - permutacija* indeksa nezavršnih simbola niski α i β,

j-ti nezavršni simbol niske α je $\Pi(j)$-ti nezavršni simbol niske β.

(5) S - početni simbol

$S \in N$.

Ako su svi nezavršni simboli niski α i β medjusobno različiti očigledna je njihova uzajamna korespodencija. Navodjenje permutacije Π indeksa nezavršnih simbola niski α i β je u takvom slučaju suvišno. Pravila izvodjenja i prevodjenja su tada oblika:

$$A \longrightarrow \alpha, \beta$$

Definicija 7: Forma šeme sintaksno-orijentisanog prevodjenja $T=(N, \Sigma, \Delta, R, S)$ je trojka:

$$F=(\alpha, \beta, \Pi)$$

gde je:

(1) $\alpha \in (N \cup \Sigma)^{*}$

(2) $\beta \in (N \cup \Delta)^{*}$ i

(3) Π - permutacija indeksa nezavršnih simbola niski α i β.

---

* Permutacija Π k elemenata označava se:

$$[i_1, i_2, \ldots i_k]$$

gde je: $1 \leq i_j \in N \leq k$ i

$i_m \neq {}^{j} i_n$ za $m \neq n$.

Po definiciji je $\Pi(j) = i_j$.

Niske α i β sadrže jednak broj nezavršnih simbola. Za svako i, $1 \leq i \leq k$, k je broj nezavršnih simbola, i-ti nezavršni simbol niske α je identičan Π(i)-tom nezavršnom simbolu niske β. i-ti nezavršni simbol niske α i Π(i)-ti nezavršni simbol niske β nazivaju se korespodentnim. Ako su svi nezavršni simboli niske α i β medjusobno različiti očigledna je njihova uzajamna korespodencija, pa je navodjenje permutacije Π indeksa nezavršnih simbola niski α i β suvišno. Forma je oblika:

$$F=(\alpha,\beta)$$

Primer

Neka je:

α = FOR <prom> := <ar.izraz> TO <ar. izraz>
STEP <ar.izraz> DO <naredba>; ,

β = L1: <prom> := <ar.izraz> ;
<naredba> ;
IF <prom> <= <ar.izraz> GOTO L2;
<prom> := <prom> + <ar.izraz> ;
GOTO L1;
L2: CONTINUE; i

$\Pi = [1,2,5,8,3]$.

$F=(\alpha,\beta,\Pi)$ nije forma!

Zaista, niske α i β sadrže različit broj nezavršnih simbola.

Neka je:

$T=(N,\Sigma,\Delta,R,S)$- šema sintaksno-orijentisanog prevodjenja,

(α_1,β_1,Π_1) i (α_2,β_2,Π_2) - forme

A - i-ti nezavršni simbol niske α_1 i

$A \rightarrow \gamma;\delta,\Pi$ - pravilo izvodjenja i prevodjenja.

Dalje, neka niska γ sadrži $m \geq o$ nezavršnih simbola.

Definicija 8: Forma (α_2,β_2,Π_2) neposredno se izvodi iz forme (α_1,β_1,Π_1) zamenom:

- i-tog nezavršnog simbola niske α_1 niskom γ ,
- Π_1(i)-tog nezavršnog simbola niske β_1 niskom δ .

Permutacija Π_2 obrazuje se na sledeći način:

(1) Za svako $j<i$
ako je $\Pi_1(j)<\Pi_1(i)$
onda je $\Pi_2(j)=\Pi_1(j)$
inače $\Pi_2(j)=\Pi_1(j)+m-1$

(2) Za svako $j>i$
ako je $\Pi_1(j)<\Pi_1(i)$
onda je $\Pi_2(j+m-1)=\Pi_1(j)$
inače $\Pi_2(j+m-1)=\Pi_1(j)+m-1$

(3) Za svako d, $1 \leq d \leq m$
$\Pi_2(i+d-1)=\Pi_1(i)+\Pi(d)+1$

Primer

Neka je (α_1,β_1,Π_1) forma, pri čemu je:
α_1 = sum <izraz> with <promenljiva> := <izraz> to <izraz>
β_1 = begin
t:=∅;
for <promenljiva> := <izraz> to <izraz> do
t:= t+ <izraz>
end
$\Pi_1 = [4,1,2,3]$
<izraz> i <promenljiva> su nezavršni simboli,
sum, with, to, := su ulazni simboli.
begin, for, to, do, end, t, :=, +, ∅ su izlazni simboli.
Dalje, neka je $A \rightarrow \gamma, \delta, \Pi$ pravilo izvodjenja i prevodjenja, pri čemu je:
A - prvi nezavršni simbol niske α_1 tj. <izraz>
γ - <identifikator> ,
δ - <identifikator> i
$\Pi = [1]$
Primenom pravila izvodjenja i prevodjenja $A \rightarrow \gamma, \delta, \Pi$ na formu (α_1,β_1,Π_1) neposredno se izvodi forma (α_2,β_2,Π_2) gde je:
α_2 = sum <identifikator> with <promenljiva> := <izraz>
to <izraz>
β_2 = begin
t:=∅

```
for <promenljiva> := <izraz> to <izraz> do
t:= t+ <identifikator>
end
```

$\Pi_2 = [4,1,2,3]$

Definicija 9:

(1) (S,S) je izvedena forma.

(2) Ako je $(\alpha A\beta, \alpha' A \beta')$ izvedena forma i
$A \rightarrow \gamma, \gamma'$ pravilo izvodjenja i prevodjenja
onda je $(\alpha\gamma\beta, \alpha'\gamma'\beta')$ neposredno izvedena forma.

Primer

Neka je (aBbA,aAbB) izvedena forma i
A→ bA,aA pravilo izvodjenja i prevodjenja
onda je (aBbbA,aaAbB) neposredno izvedena forma.
Neposredno izvodjenje formi označava se $\overset{}{\underset{T}{\Rightarrow}}$. $\overset{}{\underset{T}{\Rightarrow}}$ je relacija definisana na skupu $(N \cup \Sigma)^{*} \times (N \cup \Delta)^{*} \times \Pi$ gde je Π skup permutacija.

Teorema 3 Relacija izvodjenja formi je:
- refleksivna i
- tranzitivna,

a nije
- simetrična.

Dokaz
Zaista, neka je $(\alpha A \beta, \alpha' A \beta')$ izvedena forma. Postoji pravilo izvodjenja i prevodjenja A→ A,A. Zato važi:
$(\alpha A \beta, \alpha' A \beta') \overset{}{\underset{T}{\Rightarrow}} (\alpha A \beta, \alpha' A \beta')$
Slično se dokazuju preostale osobine.
Tranzitivno zatvaranje, refleksivno-tranzitivno zatvaranje i k-ti stepen relacije $\overset{}{\underset{T}{\Rightarrow}}$ označava se redom $\overset{+}{\underset{T}{\Rightarrow}}$, $\overset{*}{\underset{T}{\Rightarrow}}$ i $\overset{k}{\underset{T}{\Rightarrow}}$.
U slučaju kad nemože doći do zabune, izostavlja se priznak T šeme sintaksno-orijentisanog prevodjenja.

4.3. Osobine sintaksno-orijentisanog prevodjenja

Definicija 10: Sintaksno-orijentisano prevodjenje $\tau(T)$ je prevodjenje definisano šemom T sintaksno-orijentisanog prevodjenja.

$\tau(T)$ je skup parova (x,y):

$$\tau=\tau(T)=\{(x,y)\,|\,(S,S)\overset{*}{\underset{T}{\Rightarrow}}(x,y),x\in\Sigma^{*}\ i\ y\in\Delta^{*}\}$$

Definicija 11: Neka je $T=(N,\Sigma,\Delta,R,S)$ šema sintaksno-orijentisanog prevodjenja. Gramatika $G_u=(N,\Sigma,P_u,S)$ gde je $P_u=\{A\to\alpha\,|\,A\to\alpha,\beta\in R\}$ je ulazna gramatika šeme sintaksno-orijentisanog prevodjenja T. Gramatika $G_i=(N,\Delta,P_i,S)$ gde je $P_i=\{A\to\beta\,|\,A\to\alpha,\beta\in R\}$ je izlazna gramatika šeme sintaksno-orijentisanog prevodjenja T.

Teorema 4: Oblast definisanosti $L_u\subseteq\Sigma^{*}$ i skup vrednosti $L_i\subseteq\Delta^{*}$ sintaksno-orijentisanog prevodjenja τ su kontekstno-slobodni jezici.

Dokaz: Neka je $T=(N,\Sigma,\Delta,R,S)$ šema sintaksno-orijentisanog prevodjenja, $G_u=(N,\Sigma,P_u,S)$ ulazna gramatika šeme sintaksno-orijentisanog prevodjenja T i $G_i=(N,\Delta,P_i,S)$ izlazna gramatika šeme sintaksno-orijentisanog prevodjenja T. Šema sintaksno-orijentisanog prevodjenja T definiše sintaksno-orijentisano prevodjenje $\tau(T)$:

$$\tau(T)=\{(x,y)\,|\,(S,S)\overset{*}{\underset{T}{\Rightarrow}}(x,y),x\in\Sigma^{*}\ i\ y\in\Delta^{*}\}$$

Oblast definisanosti sintaksno-orijentisanog prevodjenja $\tau(T)$ je jezik $L_u=\{x\mid S\overset{*}{\underset{G_u}{\Rightarrow}}x,\ x\in\Sigma^{*}\}$

Skup vrednosti sintaksno-orijentisanog prevodjenja $\tau(T)$ je jezik $L_i=\{y\mid S\overset{*}{\underset{G_i}{\Rightarrow}}y,\ y\in\Delta^{*}\}$

Pravila izvodjenja gramatike $G_u=(N,\Sigma,P_u,S)$ su oblika:

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

pa je gramatika G_u kontekstno-slobodna.

Pravila izvodjenja gramatike $G_i=(N,\Delta,P_i,S)$ su oblika:

$A\to\beta$ $\quad A\in N\quad\beta\in(N\cup\Delta)^{*}$

pa je gramatika G_i kontekstno-slobodna.

Neposredno sledi da je jezik L_i generisan gramatikom G_i kontekstno-slobodan.

Slično, jezik L_u generisan gramatikom G_u je kontekstno-slobodan što je i tvrdjenje teoreme.

Teorema 5: Neka su τ_1 i τ_2 SO-prevodjenja redom definisana šemama T_1 i T_2:

$T_1 = (N_1, \Sigma, \Delta, R_1, S_1)$

$T_2 = (N_2, \Sigma, \Delta, R_2, S_2)$.

Ako je

$$\tau_1(x) = \tau_2(x)$$

za svako x iz oblasti definisanosti Σ^*,

onda je:

$$L_i^{(1)} = L_i^{(2)}$$

$$\tau_1 = \tau_2.$$

Dokaz: Po definiciji je:

$$\tau_1 = \tau(T_1) = \{(x,y) \mid (S_1,S_1) \overset{*}{\underset{T_1}{\Rightarrow}} (x,y), x \in \Sigma^*, y \in \Delta^*\} \text{ i}$$

$$\tau_2 = \tau(T_2) = \{(x,z) \mid (S_2,S_2) \overset{*}{\underset{T_2}{\Rightarrow}} (x,z), x \in \Sigma^*, z \in \Delta^*\}.$$

Neka je $\tau_1(x) = y$

i $\tau_2(x) = z$.

Kako je $\tau_1(x) = \tau_2(x)$

sledi da je $y = z$

a odatle tvrdjenje teoreme.

Teorema 6: Problem ekvivalentnosti SO-prevodjenja je nerešiv.

Dokaz:

Neka su τ_1 i τ_2 dva proizvoljna SO-prevodjenja, redom definisana šemama T_1 i T_2:

$T_1 = (N_1, \Sigma, \Delta, R_1, S_1)$ i

$T_2 = (N_2, \Sigma, \Delta, R_2, S_2)$.

Neka je:

$G_u^1 = \overline{(N_1, \Sigma, P_u^1, S_1)}$

$G_i^1 = (N_1, \Delta, P_i^1, S_1)$

$G_u^2 = (N_2, \Sigma, P_u^2, S_2)$

$G_i^2 = (N_2, \Delta, P_i^2, S_2)$

gde je:

$P_u^1 = \{A \to \alpha \mid A \to \alpha, \beta, \Pi \in R_1\}$

$P_i^1 = \{A \to \beta \mid A \to \alpha, \beta, \Pi \in R_1\}$

$P_u^2 = \{A \to \alpha \mid A \to \alpha, \beta, \Pi \in R_2\}$

$P_i^2 = \{A \to \beta \mid A \to \alpha, \beta, \Pi \in R_2\}$

Navedene gramatike su kontekstno-slobodne i redom definišu oblast definisanosti i skup vrednosti SO-prevodjenja τ_1 i τ_2:

$$L_u^1 , L_i^1, L_u^2, L_i^2$$

Znači, problem ekvivalentnosti

$$\tau_1 = \tau_2$$

dva SO-prevodjenja svodi se na problem ekvivalentnosti

$$L_u^1 = L_u^2 \quad \text{i}$$

$$L_i^1 = L_i^2$$

dva para kontekstno-slobodna jezika, koji je kao što se zna - nerešiv.

Definicija 12: Neka je τ SO-prevodjenje.

τ je obostrano-jednoznačno (biunivoko) prevodjenje akko važi ekvivalencija:

$$\forall(x_1,x_2)(x_1 \in L_u, x_2 \in L_u)\ \tau(x_1) = \tau(x_2) \Leftrightarrow x_1 = x_2 \ .$$

Teorema 7: Neka je τ SO-prevodjenje.

Neka su

$$A \rightarrow \alpha_1, \beta_1, \Pi_1 \quad \text{i}$$

$$A \rightarrow \alpha_2, \beta_2, \Pi_2$$

pravila izvodjenja i prevodjenja.

τ je biunivoko prevodjenje akko važi ekvivalencija:

$$\alpha_1 = \alpha_2 \Leftrightarrow (\beta_1 = \beta_2) \wedge (\Pi_1 = \Pi_2)$$

Dokaz: Neka je τ SO-biunivoko prevodjenje. Uočimo izvodjenje i prevodjenje reči x_1:

$$(S,S)=(a_o,b_o) \Rightarrow (a_1,b_1) \Rightarrow \ldots \Rightarrow (a_u,b_u)=(x_1,\tau(x_1))$$

Pretpostavimo da se primenom pravila izvodjenja $A \rightarrow \alpha_1$ od a_i prelazi na a_{i+1}, $a_i \Rightarrow a_{i+1}$

a da se primenom pravila prevodjenja $A \rightarrow \beta_1$ od b_i prelazi na b_{i+1}, $b_i \Rightarrow b_{i+1}$.

Ako je $\alpha_1 = \alpha_2$, a $\beta_1 \neq \beta_2$

primenom pravila prevodjenja $A \rightarrow \beta_2$

od b_i prelazi se na $b'_{i+1} \neq b_{i+1}$.

pa se na kraju izvodjenja i prevodjenja dobija $(x_1, \tau(x_1'))$ pri čemu je $\tau(x_1) \neq \tau(x_1')$ što je nemoguće ako je τ SO-biunivoko prevodjenje. Odatle sledi: ako je $\alpha_1 = \alpha_2$ onda je $\beta_1 = \beta_2$.

Na sličan način pri pretpostavci $\alpha_1 \neq \alpha_2$, $\beta_1 = \beta_2$ dolazi se do protivurečnosti pa se ista odbacuje.

Neka važe ekvivalencije:

$$\alpha_1 = \alpha_2 \iff \beta_1 = \beta_2 \iff \Pi_1 = \Pi_2$$

Sva pravila izvodjenja i prevodjenja su medjusobno različita pa je jednoznačno definisan prelaz

$$(a_i, b_i) \Rightarrow (a_{i+1}, b_{i+1}) \quad (\forall i)\ i=1,2\ldots$$

odakle sledi da je τ SO-biunivoko prevodjenje.

Neka je $T_1 = (N_1, \Sigma_1, \Delta_1, R_1, S_1)$ - šema sintaksno-orijentisanog prevodjenja

$$\tau_1 = \tau(T_1) = \left\{ (x_1, y_1) \,\middle|\, (S_1, S_1) \overset{*}{\underset{T_1}{\Rightarrow}} (x_1, y_1),\ x_1 \in \Sigma_1^*\ \text{i}\ y_1 \in \Delta_1^* \right\}$$

Slično, neka je $T_2 = (N_2, \Sigma_2, \Delta_2, R_2, S_2)$ šema sintaksno-orijentisanog prevodjenja

$$\tau_2 = \tau(T_2) = \left\{ (x_2, y_2) \,\middle|\, (S_2, S_2) \overset{*}{\underset{T_2}{\Rightarrow}} (x_2, y_2),\ x_2 \in \Sigma_2^*\ \text{i}\ y_2 \in \Delta_2^* \right\}$$

Da bi postojala kompozicija τ sintaksno-orijentisanih prevodjenja τ_1 i τ_2 neophodno je da su ispunjeni sledeći uslovi:

a) $N_1 = N_2$

b) $S_1 = S_2$

c) $\Delta_1 = \Sigma_2$

Za svako pravilo izvodjenja i prevodjenja

$A \to \alpha, \beta, \Pi_1 \in R_1$

postoji pravilo izvodjenja i prevodjenja

$A \to \beta, \gamma, \Pi_2 \in R_2$

gde je

$$A \in N_1,\ \alpha \in (N_1 \cup \Sigma_1)^*,\ \beta \in (N_1 \cup \Delta_1)^*$$

$$A \in N_2,\ \beta \in (N_2 \cup \Sigma_2)^*,\ \gamma \in (N_2 \cup \Delta_2)^*$$

Navedeni uslovi nazivaju se uslovi kompozicije.

Teorema 8: Neka sintaksno-orijentisana prevodjenja τ_1 i τ_2 zadovoljavaju uslove kompozicije.

Kompozicija τ sintaksno-orijentisanih prevodjenja τ_1 i τ_2 je sintaksno-orijentisano prevodjenje.

Dokaz: Na osnovu definicija:
- šeme SO-prevodjenja,
- SO-prevodjenja i
- uslova kompozicije sledi:

$T_1=(N,\Sigma_1,\Delta_1,R_1,S)$

$T_2=(N,\Delta_1,\Delta_2,R_2,S)$

$$\tau_1=\tau(T_1)=\left\{(x,y)\ \middle|\ (S,S)\underset{T_1}{\overset{*}{\Rightarrow}}(x,y),x\in\Sigma_1^*\ i\ y\in\Delta_1^*\right\}$$

$$\tau_2=\tau(T_2)=\left\{(y,z)\ \middle|\ (S,S)\underset{T_2}{\overset{*}{\Rightarrow}}(y,z),y\in\Delta_1^*\ i\ z\in\Delta_2^*\right\}$$

Neka je

$$T=(N,\Sigma_1,\Delta_2,R,S)$$

gde je $R=\left\{A\to\alpha,\gamma,\Pi\ \middle|\ A\to\alpha,\beta,\Pi_1\in R_1, A\to\beta,\gamma,\Pi_2\in R_2, \Pi=\Pi_1\circ\Pi_2\right\}$

i $\tau=\tau(T)=\left\{(x,z)\ \middle|\ (S,S)\underset{T}{\overset{*}{\Rightarrow}}(x,z),x\in\Sigma_1^*\ i\ z\in\Delta_2^*\right\}$

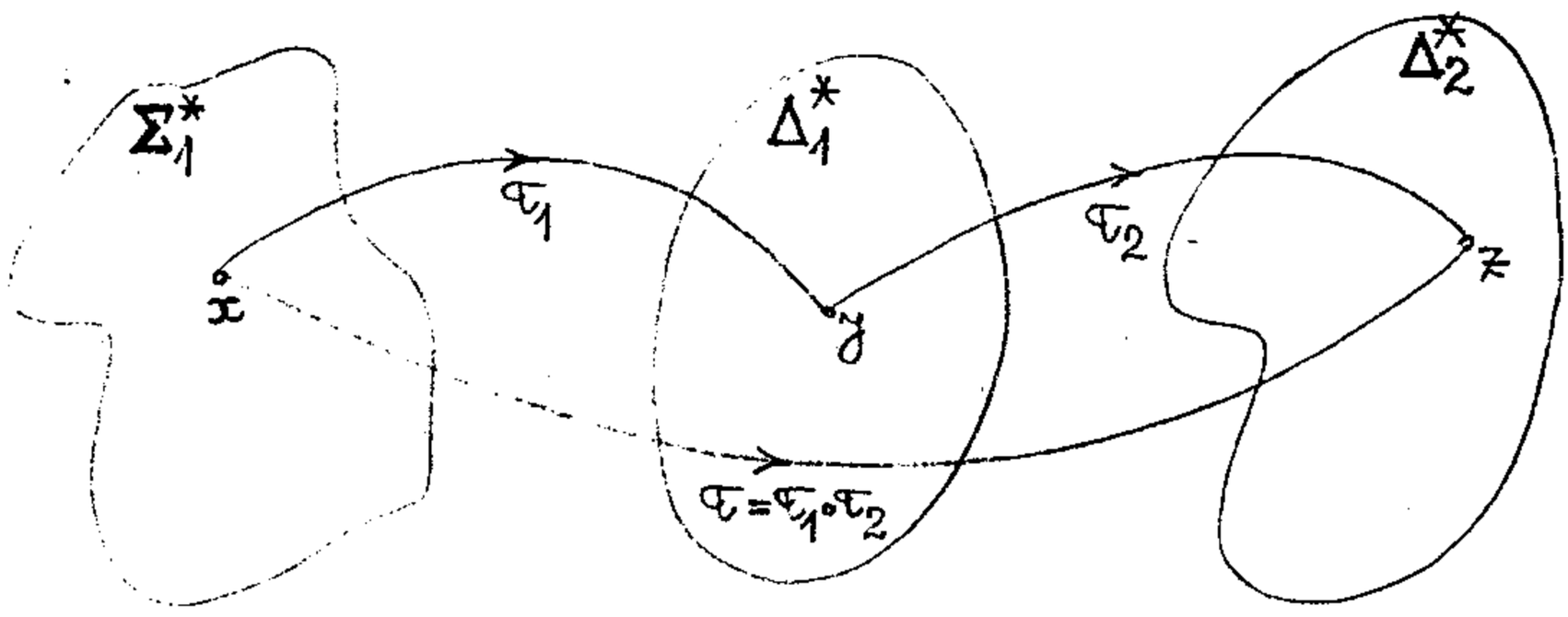
T je šema SO-prevodjenja τ koje je kompozicija SO-prevodjenja τ_1 i τ_2.

Zaista, neka je:

$x\in\Sigma_1^*$

$y=\tau_1(x)\in\Delta_1^*$

$z=\tau_2(y)\in\Delta_2^*$



$y=\tau_1(x)$ znači

$(x,y)\in\tau_1$ odnosno da postoji izvodjenje i prevodjenje

$(S,S)=(\alpha_0,\beta_0)\Rightarrow(\alpha_1,\beta_1)\Rightarrow\ldots\Rightarrow(\alpha_i,\beta_i)\Rightarrow\ldots\Rightarrow(\alpha_n,\beta_n)=(x,y)$

slično, $z=\tau_2(y)$ znači

$(y,z)\in\tau_2$ odnosno da postoji izvodjenje i prevodjenje

$(S,S)=(\beta_0,\gamma_0)\Rightarrow(\beta_1,\gamma_1)\Rightarrow\ldots\Rightarrow(\beta_i,\gamma_i)\Rightarrow\ldots\Rightarrow(\beta_n,\gamma_n)=(y,z)$

odakle sledi da postoji izvodjenje i prevodjenje

$(S,S)=(\alpha_0,\gamma_0)\Rightarrow(\alpha_1,\gamma_1)\Rightarrow\ldots\Rightarrow(\alpha_i,\gamma_i)\Rightarrow\ldots\Rightarrow(\alpha_n,\gamma_n)=(x,z)$

što dalje znači da je

$(x,z)\in\tau=\tau_1\circ\tau_2$ odnosno da je

$z=\tau(x)=\tau_2(\tau_1(x))=\tau_1\circ\tau_2(x)$

τ je SO-prevodjenje .

Definicija 13: Neka je $\tau_i=\tau(T_i)$, $i=1,2,\ldots,n$ SO-prevodjenje definisano šemom SO-prevodjenja

$T_i=(N,\Sigma_i,\Delta_i,R_i,S)$, $i=1,2,\ldots,n$.

Da bi postojala kompozicija τ SO-prevodjenja τ_i $i=1,2,\ldots,n$. neophodno je da su ispunjeni sledeći uslovi:

a) $\Delta_i=\Sigma_{i+1}$, $i=1,2,\ldots,n-1$

b) Za svako pravilo izvodjenja i prevodjenja

$A\to\beta_{i-1},\beta_i,\Pi_i\in R_i$

postoji pravilo izvodjenja i prevodjenja

$A\to\beta_i,\beta_{i+1},\Pi_{i+1}\in R_{i+1}$ $\quad i=2,3,\ldots,n-1.$

Navedeni uslovi nazivaju se uslovi kompozicije.

Teorema 9: Neka SO-prevodjenja τ_i, $i=1,2,\ldots,n$ zadovoljavaju uslov kompozicije.

Kompozicija τ n SO-prevodjenja τ_i, $i=1,2,\ldots,n$ je SO-prevodjenje.

Dokaz: Neposredno sledi na osnovu matematičke indukcije i dokaza prethodne teoreme.

Definicija 14: Neka je $\tau_i=\tau(T_i)$, $i=1,2,\ldots,n$ SO-prevodjenje definisano šemom SO-prevodjenja $T_i=(N,\Sigma_i,\Delta_i,R_i,S)$, $i=1,2,\ldots n$.

Uslovi: $\Sigma_1=\Sigma_2=\ldots=\Sigma_n=\Delta_1=\Delta_2=\ldots=\Delta_n$

nazivaju se uslovi jednakosti.

Šeme T_i SO-prevodjenja koja zadovoljavaju uslove jednakosti su oblika (N,Σ,Δ,R_i,S), $i=1,2,\ldots,n$.

Teorema 10: Neka SO-prevodjenja τ_1 i τ_2 zadovoljavaju uslove kompozicije i jednakosti.

Ne važi zakon komutacije, tj.

$\tau_1 \circ \tau_2 \neq \tau_2 \circ \tau_1$

Dokaz Pretpostavimo suprotno, pa navodjenjem kontra-primera dokažimo tvrdjenje teoreme.

Neka je $T_1 = (\{\langle bb\rangle, \langle bc\rangle\}, \{0,1\}, \{0,1\}, R_1, \langle bb\rangle)$

gde je $R_1 = \{ \langle bb\rangle \rightarrow \langle bc\rangle, \langle bc\rangle$
$\langle bb\rangle \rightarrow \langle bb\rangle \langle bc\rangle, \langle bb\rangle \langle bc\rangle$
$\langle bc\rangle \rightarrow 0,0$
$\langle bc\rangle \rightarrow 1,0\}$

$\tau_1 = \tau(T_1) = \{(x,y) \mid x \in \{0,1\}^+, \ y \in \{0\}^+, \ |x| = |y|\}$

Dalje, neka je $T_2 = (\{\langle bb\rangle, \langle bc\rangle\}, \{0,1\}, \{0,1\}, R_2, \langle bb\rangle)$

gde je $R_2 = \{\langle bb\rangle \rightarrow \langle bc\rangle, \langle bc\rangle$
$\langle bb\rangle \rightarrow \langle bb\rangle \langle bc\rangle, \langle bb\rangle \langle bc\rangle$
$\langle bc\rangle \rightarrow 0,1$
$\langle bc\rangle \rightarrow 1,1\}$

$\tau_2 = \tau(T_2) = \{(x,y) \mid x \in \{0,1\}^+, \ y \in \{1\}^+, \ |x| = |y|\}$

$\forall x, \ x \in \{0,1\}^+$

$\tau_1 \circ \tau_2(x) = \tau_2(\tau_1(x)) = \tau_2(y) = z$

$y \in \{0\}^+, \ |x| = |y|$

$z \in \{1\}^+, \ |y| = |z|$

$\tau_2 \circ \tau_1(x) = \tau_1(\tau_2(x)) = \tau_2(z) = y$

Kako je $z \neq y$ sledi da je

$\tau_1 \circ \tau_2 \neq \tau_2 \circ \tau_1$

Teorema 11: Neka SO-prevodjenja τ_1, τ_2 i τ_3 zadovoljavaju uslove kompozicije.

Važi zakon asocijacije tj.

Dokaz: Neka je

$T_1 = (N, \Sigma_1, \Delta_1, R_1, S)$,

$T_2 = (N, \Delta_1, \Delta_2, R_2, S)$ i

$T_3 = (N, \Delta_2, \Delta_3, R_3, S)$.

Za svako pravilo izvodjenja i prevodjenja

$A \rightarrow \alpha, \beta, \Pi_1 \in R_1$

postoje pravila izvodjenja i prevodjenja

$A \rightarrow \beta, \gamma, \Pi_2 \in R_2$ i

$A \rightarrow \gamma, \delta, \Pi_3 \in R_3$.

Dalje, neka je:

$\tau_1(x)=y$ (1)

$\tau_2(y)=z$ (2)

$\tau_3(z)=u.$ (3)

Iz (1) sledi da postoji izvodjenje i prevodjenje:

$$(S,S)\Rightarrow(\alpha_0,\beta_0)\Rightarrow(\alpha_1,\beta_1)\Rightarrow\ldots\Rightarrow(\alpha_n,\beta_n)=(x,y) \quad (1')$$

Analogno na osnovu (2) i (3) sledi:

$$(S,S)\Rightarrow(\beta_0,\gamma_0)\Rightarrow(\beta_1,\gamma_1)\Rightarrow\ldots\Rightarrow(\beta_n,\gamma_n)=(y,z) \quad (2')$$

$$(S,S)\Rightarrow(\gamma_0,\delta_0)\Rightarrow(\gamma_1,\delta_1)\Rightarrow\ldots\Rightarrow(\gamma_n,\delta_n)=(z,u) \quad (3')$$

Na osnovu (1') i (2') sledi da je:

$$(\tau_1\circ\tau_2)(x)=z$$

a na osnovu (2') i (3') da je:

$$(\tau_2\circ\tau_3(y))=u$$

pa je

$$((\tau_1\circ\tau_2)\circ\tau_3)(x)=\tau_3((\tau_1\circ\tau_2)(x))=\tau_3(z)=u$$

$$(\tau_1\circ(\tau_2\circ\tau_3))(x)=(\tau_2\circ\tau_3)(\tau_1(x))=(\tau_2\circ\tau_3)(y)=u$$

što predstavlja tvrdjenje teoreme.

Definicija 15: SO-prevodjenje τ_I

$$\tau_I=\tau(T_I)=\{(x,x)\mid(S,S)\overset{*}{\underset{T_I}{\Rightarrow}}(x,x),\ x\in\Sigma^*\}$$

definisano šemom

$$T_I=(N,\Sigma,\Sigma,R_I,S)$$

gde je:

$$R_I=\{A\rightarrow\alpha,\alpha,\Pi_I\}$$

$A\in N$

$\alpha\in(N\cup\Sigma)^*$

$\Pi_I=[1,2,\ldots]$

je identičko prevodjenje jezika Σ^*.

Definicija 16: Neka je τ SO-prevodjenje.

Ako postoji prevodjenje τ^{-1} takvo da je:

$-\tau\cdot\tau^{-1}=\tau_I$

$-\tau^{-1}\cdot\tau=\tau_I$

onda je prevodjenje τ^{-1} inverzno prevodjenje, prevodjenja τ .

$\tau\circ\tau^{-1}$ je identičko prevodjenje jezika Σ^*

$\tau^{-1}\circ\tau$ je identičko prevodjenje jezika Δ^*

Teorema 12: Neka je $T=(N,\Sigma,\Delta,R,S)$ šema SO-prevodjenja $\Phi(T)$,

$$\Phi(T)=\left\{(x,y)\mid (S,S)\overset{*}{\underset{T}{\Rightarrow}}(x,y),\ x\in\Sigma^*,\ y\in\Delta^*\right\}$$

Ako je $\Phi(T)$ obostrano jednoznačno (biunivoko) prevodjenje onda je

$T^{-1}=(N,\Delta,\Sigma,R^{-1},S)$ šema SO-inverznog prevodjenja $\Phi^{-1}(T)$,

$$\Phi^{-1}(T)=\left\{(y,x)\mid (S,S)\overset{*}{\underset{T^{-1}}{\Rightarrow}}(y,x),\ y\in\Delta^*,\ x\in\Sigma^*\right\}$$

gde je

$$R^{-1}=\left\{A\to\beta,\alpha,\Pi^{-1}\mid A\to\alpha,\beta,\Pi\in R \text{ i } \Pi\cdot\Pi^{-1}=\Pi_I\right\}$$

Dokaz: Prevodjenje τ^{-1} je inverzno prevodjenje, prevodjenja τ ako su ispunjeni sledeći uslovi:

$-\tau\circ\tau^{-1}=\tau_I$ i

$-\tau^{-1}\cdot\tau=\tau_I$

Kako je:

$T=(N,\Sigma,\Delta,R,S)$ i

$$\Phi(T)=\left\{(x,y)\mid (S,S)\overset{*}{\underset{T}{\Rightarrow}}(x,y),\ x\in\Sigma^*,\ y\in\Delta^*\right\}$$

odnosno

$T^{-1}=(N,\Delta,\Sigma,R^{-1},S)$ i

$$\Phi(T^{-1}=\left\{(y,x)\mid (S,S)\overset{*}{\underset{T^{-1}}{\Rightarrow}}(y,x),\ y\in\Delta^*,\ x\in\Sigma^*\right\}$$

neposredno sledi da je:

$T\cdot T^{-1}=(N,\Sigma,\Sigma,R\cdot R^{-1},S)$ i

$$\tau(T)\circ\tau(T^{-1})=\left\{(x,x)\mid (S,S)\underset{T^{-1}\circ T}{\overset{*}{\Longrightarrow}}(x,x),\ x\in\Sigma^*\right\}=\tau(T\circ T^{-1})$$

odnosno

$$T^{-1}\circ T=(N,\Delta,\Delta,R^{-1}\circ R,S) \text{ i}$$

$$\tau(T^{-1})\circ\tau(T)=\left\{(y,y)\mid (S,S)\underset{T\circ T^{-1}}{\overset{*}{\Longrightarrow}}(y,y),\ y\in\Delta^*\right\}=\tau(T^{-1}\circ T)$$

Kako je

$$R=\left\{A\to\alpha,\beta,\Pi \mid A\in N,\alpha\in(N\cup\Sigma)^*,\ \beta\in(N\cup\Delta)^*\right\} \text{ i}$$

$$R^{-1}=\left\{A\to\beta,\alpha,\Pi^{-1}\mid A\to\alpha,\beta,\Pi\in R \text{ i } \Pi\circ\Pi^{-1}=\Pi_I\right\}$$

neposredno sledi da je:

$$R\circ R^{-1}=\left\{A\to\alpha,\alpha,\Pi_I\right\}=R_I$$

odnosno

$$R^{-1}\circ R=\left\{A\to\beta,\beta,\Pi_I\right\}=R_I$$

pa je

$$T\circ T^{-1}=(N,\Sigma,\Sigma,R_I,S)=T_I$$

odnosno

$$T^{-1}\circ T=(N,\Sigma,\Sigma,R_I,S)=T_I$$

pa je: $\tau\circ\tau^{-1}=\tau(T)\circ\tau(T^{-1})=\tau(T\circ T^{-1})=\tau(T_I)=\tau_I$

odnosno

$$\tau^{-1}\circ\tau=\tau(T^{-1})\circ\tau(T)=\tau(T^{-1}\circ T)=\tau(T_I)=\tau_I$$

što je i trebalo dokazati .

Teorema 13: Neka je τ SO-obostrano-jednoznačno prevodjenje.

Inverzno prevodjenje τ^{-1} je jedinstveno.*

Dokaz: Pretpostavimo suprotno, tj. neka su τ_1 i τ_2 dva medjusobno različita inverzna prevodjenja, prevodjenja τ . Tada su $\tau_1\tau$ i $\tau_2\tau$ identička prevodjenja jezika Δ^* tj.

$\tau_1\tau(y)=\tau_I(y)=y$ -II-

$\tau_2\tau(y)=\tau_I(y)=y$ za svako y

$\tau_1\tau(y)=\tau_2\tau(y)$ -II-

---

* Prema prethodnim teoremama skup svih SO-prevodjenja je semigrupa sa jedinicom, dakle inverzni element ako postoji je jedinstven.

pa je:

$$\tau(\tau_1(y)) = \tau(\tau_2(y))$$

Kako je τ - obostrano jednoznačno prevodjenje sledi da je:

$$\tau_1(y) = \tau_2(y) \qquad \text{za svako } y \in \Delta^*$$

pa je

$$\tau_1 = \tau_2$$

što je suprotno polaznoj pretpostavci. Sledi da je:

$$\tau_1 = \tau_2 = \tau^{-1}$$

Teorema 14: Neka je τ SO-prevodjenje koje nije obostrano jednoznačno. Inverzno prevodjenje τ^{-1} je višeznačno.

Dokaz: Teoremu ćemo dokazati navodjenjem kontra-primera. Zaista, neka je:

$$T = (\{\langle bb\rangle, \langle bc\rangle\}, \{\emptyset, 1\}, \{\emptyset, 1\}, R, \langle bb\rangle)$$

gde je

$$R = \{\langle bb\rangle \to \langle bc\rangle, \langle bc\rangle$$
$$\langle bb\rangle \to \langle bb\rangle\ \langle bc\rangle, \langle bb\rangle\ \langle bc\rangle$$
$$\langle bc\rangle \to \emptyset, \emptyset$$
$$\langle bc\rangle \to 1, \emptyset\}$$

$$\tau = \tau(T) = \{(x,y) \mid x \in \{\emptyset, 1\}^+, y \in \{\emptyset\}^+, |x| = |y|\}$$

$$T^{-1} = (\{\langle bb\rangle, \langle bc\rangle\}, \{\emptyset, 1\}, \{\emptyset, 1\}, R^{-1}, \langle bb\rangle)$$

gde je

$$R^{-1} = \{\langle bb\rangle \to \langle bc\rangle, \langle bc\rangle$$
$$\langle bb\rangle \to \langle bb\rangle\ \langle bc\rangle, \langle bb\rangle\ \langle bc\rangle$$
$$\langle bc\rangle \to \emptyset, \emptyset$$
$$\langle bc\rangle \to \emptyset, 1\}$$

Pravila izvodjenja i prevodjenja:

$$\langle bc\rangle \to \emptyset, \emptyset$$
$$\langle bc\rangle \to \emptyset, 1$$

skupa R^{-1} uzrokuju višeznačnost pa je inverzno prevodjenje τ^{-1} višeznačno.

Na primer:

$$\tau(\emptyset 1) = \emptyset\emptyset \quad , \text{ a}$$

$$\tau^{-1}(\emptyset\emptyset) = \begin{cases} \emptyset\emptyset \\ \emptyset 1 \\ 1\emptyset \\ 11 \end{cases} .$$

4.3.1. Prosta SO-prevodjenja

Definicija 17: Šema sintaksno-orijentisanog prevodjenja T=(N,Σ,Δ,R,S) je prosta ako se nezavršni simboli niski α i β svih pravila A→α,β skupa R javljaju u istom redosledu.

Drugačije rečeno, šema sintaksno-orijentisanog prevodjenja T=(N,Σ,Δ,R,S) je prosta ako sve forme (α,β,Π) zadovoljavaju uslov: Π je identička permutacija (tj. $\forall i\ \Pi(i)=i$).

Definicija 18: Prosto sintaksno-orijentisano prevodjenje τ je prevodjenje definisano prostom šemom T(τ) sintaksno-orijentisanog prevodjenja.

Sve prethodno navedene šeme sintaksno-orijentisanog prevodjenja su proste.

Prosta sintaksno-orijentisana prevodjenja predstavljaju važnu klasu prevodjenja. Relativno jednostavno je konstruisati prevodilac koji izvršava prosto sintaksno-orijentisano prevodjenje.

Primer

Data je šema sintaksno-orijentisanog prevodjenja

$$T=(\{E,T,F\},\{a,+,*,(,)\},\{a,+,*,(,)\},R,E)$$

gde je R skup sledećih pravila:

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

Data šema sintaksno-orijentisanog prevodjenja je složena.

$\tau(T)=\{(x,y) \mid$ x i y su aritmetički izrazi sastavljeni iz simbola a,+,*,(i)$\}$.

Vrednosti aritmetičkih izraza x i y su jednake, medjutim aritmetički izraz x je levo-asocijativan dok je aritmetički izraz y desno asocijativan.

Teorema 15: Kompozicija
- prostog i složenog,
- složenog i prostog

sintaksno-orijentisanog prevodjenja je složeno sintaksno-orijenti-

sano prevodjenje.

Kompozicija konačno mnogo:

- prostih

sintaksno-orijentisanih prevodjenja je prosto sintaksno-orijentisano prevodjenje.

Kompozicija

- složenog i složenog

sintaksno-orijentisanog prevodjenja je složeno sintaksno-orijentisano prevodjenje ako prevodjenja nisu medjusobno inverzna - u suprotnom je prosto sintaksno-orijentisano prevodjenje.

Dokaz:

Neka je $T = (N,\Sigma,\Delta,R,S)$ šema sintaksno-orijentisanog prevodjenja $\tau = \tau(T)$.

Ako su sva pravila izvodjenja i prevodjenja šeme T oblika:

$$A \rightarrow \alpha,\beta,\Pi_I$$

šema T i prevodjenje τ su prosti.

Dovoljno je da postoji jedno pravilo izvodjenja i prevodjenja šeme T oblika:

$$A \rightarrow \alpha,\beta,\Pi \neq \Pi_I$$

pa da šema T i prevodjenje τ budu složeni.

Neka su:

$$A \rightarrow \alpha,\beta,\Pi_1 \qquad \text{i}$$

$$A \rightarrow \beta,\gamma,\Pi_2$$

odgovarajuća pravila izvodjenja i prevodjenja šeme T_1 odnosno T_2. Njihovom kompozicijom dobija se:

$$A \rightarrow \alpha,\gamma,\Pi = \Pi_1 \circ \Pi_2$$

pravilo izvodjenja i prevodjenja šeme $T = T_1 \circ T_2$.

Kako je:

$$\Pi = \Pi_I \circ \Pi = \Pi \circ \Pi_I, \quad \Pi \neq \Pi_I$$
$$\Pi_I = \Pi_I \circ \Pi_I$$
$$\Pi_I \neq \Pi = \Pi_1 \circ \Pi_2, \quad \Pi_1 \neq \Pi_I, \quad \Pi_2 \neq \Pi_I$$
$$\Pi_I = \Pi = \Pi_1 \circ \Pi_2, \quad \Pi_1 \neq \Pi_I, \quad \Pi_2 \neq \Pi_I, \quad \Pi_1 = \Pi_2^{-1}, \quad \Pi_2 = \Pi_1^{-1}$$

sledi istinitost tvrdjenja teoreme.

Teorema 16: Neka je τ prosto SO-prevodjenje.

Inverzno prevodjenje τ^{-1} prostog SO-prevodjenja je prosto SO-prevodjenje.

Dokaz: Neka je T = (N,Σ,Δ,R,S) šema prostog SO-prevodjenja $\tau = \tau(T)$. Pravila izvodjenja i prevodjenja šeme T su oblika:

$$A \rightarrow \alpha, \beta, \Pi_I$$

Šema T^{-1} inverznog SO-prevodjenja $\tau^{-1} = \tau(T^{-1})$ je oblika:

$$T^{-1} = (N, \Delta, \Sigma, R^{-1}, S)$$

gde je: $R^{-1} = \{A \rightarrow \beta, \alpha, \Pi^{-1} \mid A \rightarrow \alpha, \beta, \Pi_I \in R \text{ i } \Pi_I \circ \Pi^{-1} = \Pi_I\}$

Kako je:

$\Pi = \Pi_I \circ \Pi$ za svaku permutaciju Π

sledi da je: $\Pi_I \circ \Pi^{-1} = \Pi^{-1} = \Pi_I$

Znači, pravila skupa R^{-1} su oblika:

$$A \rightarrow \beta, \alpha, \Pi_I$$

pa su šema T^{-1} i prevodjenje τ^{-1} prosti.

4.3.2. Desno-linearna SO-prevodjenja

Definicija 19: Šema SO-prevodjenja T=(N,Σ,Δ,R,S) je desno-linearna ako su sva pravila izvodjenja i prevodjenja šeme T oblika:

$$A \rightarrow xB, yB \quad \text{ili}$$
$$A \rightarrow x, y$$

gde su: $A, B \in N$,

$x \in \Sigma^*$ i

$y \in \Delta^*$.

Definicija 20: Desno-linearno SO-prevodjenje τ je prevodjenje definisano desno-linearnom šemom $T(\tau)$ SO-prevodjenja.

Teorema 17: Desno-linearno SO-prevodjenje τ je prosto SO-prevodjenje.

Dokaz: Neposredno sledi na osnovu definicije desno-linearnog odnosno prostog SO-prevodjenja.

Teorema 18: Oblast definisanosti i skup vrednosti desno-linearnog SO-prevodjenja τ su desno-linearni jezici.

Dokaz: Analogan dokazu opštijeg slučaja tj. kada je prevodjenje τ SO, a nije desno-linearno.

Teorema 19: Neka je τ desno-linearno SO-prevodjenje.

Neka su:

$$A \rightarrow x_1 B, y_1 B \quad i$$

$$A \rightarrow x_2 B, y_2 B$$

odnosno

$$A \rightarrow x_1, y_1$$

$$A \rightarrow x_2, y_2$$

pravila izvodjenja i prevodjenja.

τ je biunivoka prevodjenje akko važi ekvivalencija:

$$x_1 = x_2 \Leftrightarrow y_1 = y_2$$

Dokaz: Analogan dokazu opštijeg slučaja.

Teorema 20: Kompozicija konačno mnogo desno-linearnih SO-prevodjenja je desno-linearno SO-prevodjenje.

Dokaz: Neka su:

$$A \rightarrow xB, yB \quad i$$

$$A \rightarrow yB, zB$$

pravila izvodjenja i prevodjenje desno-linearnih SO-šema T_1 i T_2.

Njihova kompozicija je pravilo izvodjenja i prevodjenja:

$$A \rightarrow xB, zB$$

odakle dalje primenom matematičke indukcije sledi tvrdjenje teoreme.

Teorema 21: Neka je τ desno-linearno SO-prevodjenje

Inverzno prevodjenje τ^{-1} desno-linearnog SO-prevodjenja je desno-linearno SO-prevodjenje.

Dokaz: Neka je $T=(N,\Sigma,\Delta,R,S)$ šema desno-lenearnog SO-prevodjenja $\tau = \tau(T)$. Pravila izvodjenja i prevodjenja šeme T su oblika:

$$A \rightarrow xB, yB \quad ili$$

$$A \rightarrow x, y \quad .$$

Šema T^{-1} inverznog SO-prevodjenja $\tau^{-1} = \tau(T^{-1})$ je oblika:

$$T^{-1} = (N, \Delta, \Sigma, R^{-1}, S)$$

gde je: $R^{-1} = \{ A \rightarrow yB, xB \mid A \rightarrow xB, yB \in R \} \cup$

$$\{ A \rightarrow y, x \mid A \rightarrow x, y \in R \} .$$

Odavde sledi da su šema T^{-1} i prevodjenje τ^{-1} desno-linearni.

4.4. Sintaksno-orijentisano prevodjenje proceduralno-orijentisanog jezika na mašinski-orijentisan jezik

Ulazni jezik je proceduralno-orijentisan BASIC-oliki jezik. Sintaksa ulaznog jezika definisana je nizom metalingvističkih formula datih u spisku 1.

Izlazni jezik je mašinski-orijentisan jezik nastavnog računara. Sintaksa izlaznog jezika definisana je nizom metalingvističkih formula datih u spisku 2.

T1, T2 i T3 su negrafički simboli i redom služe za pozicioniranje obeležja, mnemokôda i adrese.

Sintaksa jezika prevodjenja definisana je nizom metalingvističkih formula datih u spisku 3.

L je priznak obeležja (simboličke adrese).

S je priznak stoga (magazinske memorije).

Ulazni i izlazni jezik i jezik prevodjenja su kontekstno-slobodni jezici.

Šema sintaksno-orijentisanog prevodjenja ulaznog jezika na izlazni jezik definisana* je gramatikom ulaznog jezika i gramatikom jezika prevodjenja.

Sintaksno-orijentisano prevodjenje proceduralno-orijentisanog jezika na mašinski-orijentisan jezik ilustrovaćemo na primeru naredbe:

```
1Ø LET X = - 2.13;
```

čiji je prevod niz naredbi:

```
T1 L1Ø T2 MUA T3 2.13
       T2 AUM T3 S1
       T2 MUA T3 Ø
       T2 ODU T3 S1
       T2 AUM T3 X
```

---

* Iz praktičnih razloga izvršeno je razdvajanje jedinstvenih pravila izvodjenja i prevodjenja na posebna pravila izvodjenja i pravila prevodjenja.

```
<A-IZRAZ>::=<SABIRAK>!
           +<SABIRAK>!
           -<SABIRAK>!
           <A-IZRAZ>+<SABIRAK>!
           <A-IZRAZ>-<SABIRAK>
<SABIRAK>::=<CINILAC>!
             <SABIRAK>*<CINILAC>!
             <SABIRAK>/<CINILAC>
<CINILAC>::=<OSNOVA>!
             <CINILAC>^<OSNOVA>
<OSNOVA>::=<IDENTIFIKATOR>!
           (<A-IZRAZ>)!
           <FUNKCIJA>
<IDENTIFIKATOR>::=<PROMENLJIVA>!
                  <KONSTANTA>
<FUNKCIJA>::=<IME-FUNKCIJE>(<A-IZRAZ>)
<IME-FUNKCIJE>::=ABS!ATN!COS!EXP!LOG!SIN!SQR
<PROMENLJIVA>::=<SLOVO>!
                <SLOVO><CIFRA>
<KONSTANTA>::=<BROJ>
<BROJ>::=<CEO>!
         <DECIMALAN>!
         <CEO><EKSPONENT>!
         <DECIMALAN><EKSPONENT>
<CEO>::=<CIFRA>!
       <CEO><CIFRA>
<DECIMALAN>::=<CEO><RAZLOMLJEN>
<RAZLOMLJEN>::=.<CEO>
<EKSPONENT>::=E<CEO>!
              E<ZNAK><CEO>
<ZNAK>::=+!-
<SLOVO>::=A!B!C!D!E!F!G!H!I!J!K!L!M!N!O!P!Q!R!S!T!U!V!W!X!Y!Z
<CIFRA>::=0!1!2!3!4!5!6!7!8!9
<LET-NAREDBA>::=<PROMENJIVA>=<A-IZRAZ>!
                LET<PROMENLJIVA>=<A-IZRAZ>
<INPUT-NAREDBA>::=INPUT<INPUT-LISTA>
<INPUT-LISTA>::=<PROMENLJIVA>!
                <INPUT-LISTA>,<PROMENLJIVA>
<PRINT-NAREDBA>::=PRINT<PRINT-LISTA>
<PRINT-LISTA>::=<A-IZRAZ>!
                <PRINT-LISTA>,<A-IZRAZ>
<GOTO-NAREDBA>::=GOTO<BROJ-PROGRAMSKOG-REDA>
<IF-NAREDBA>::=IF<R-IZRAZ>THEN<BROJ-PROGRAMSKOG-REDA>!
              IF<R-IZRAZ>GOTO<BROJ-PROGRAMSKOG-REDA>!
              IF<R-IZRAZ>THEN<BASIC-NAREDBA>
<R-IZRAZ>::=<A-IZRAZ><R-OPERATOR><A-IZRAZ>
<R-OPERATOR>::='>=!'>!'<'>!=!'<!'<=
<STOP-NAREDBA>::=STOP
<END-NAREDBA>::=END
<BASIC-NAREDBA>::=<LET-NAREDBA>!
                  <INPUT-NAREDBA>!
                  <PRINT-NAREDBA>!
                  <GOTO-NAREDBA>!
                  <IF-NAREDBA>!
                  <STOP-NAREDBA>!
                  <END-NAREDBA>
<NAREDBA>::=<BASIC-NAREDBA>!
           <NAREDBA>\<BASIC-NAREDBA>
<PROGRAMSKI-RED>::=<BROJ-PROGRAMSKOG-REDA><NAREDBA>
<BROJ-PROGRAMSKOG-REDA>::=<CEO>
<RED>::=<PROGRAMSKI-RED>;
```

Spisak 1

```
<A-IZRAZ>::=<SABIRAK>!
            +<SABIRAK>!
            <SABIRAK>T2AUMT3S1T2MUAT30T20DUT3S1!
            <SABIRAK>T2AUMT3S1<A-IZRAZ>T2SABT3S1!
            <SABIRAK>T2AUMT3S1<A-IZRAZ>T20DUT3S1
<SABIRAK>::=<CINILAC>!
            <CINILAC>T2AUMT3S1<SABIRAK>T2MNOT3S1!
            <CINILAC>T2AUMT3S1<SABIRAK>T2DELT3S1
<CINILAC>::=<OSNOVA>!
            <OSNOVA>T2AUMT3S1<CINILAC>T2STEPENT3S1
<OSNOVA>::=T2MUAT3<IDENTIFIKATOR>!
           <A-IZRAZ>!
           <FUNKCIJA>
<IDENTIFIKATOR>::=<PROMENLJIVA>!
                  <KONSTANTA>
<FUNKCIJA>::=<A-IZRAZ>T2PPRT3<IME-FUNKCIJE>
<IME-FUNKCIJE>::=ABS!ATN!COS!EXP!LOG!SIN!SQR
<PROMENLJIVA>::=<SLOVO>!
                <SLOVO><CIFRA>
<KONSTANTA>::=<BROJ>
<BROJ>::=<CEO>!
         <DECIMALAN>!
         <CEO><EKSPONENT>!
         <DECIMALAN><EKSPONENT>
<CEO>::=<CIFRA>!
        <CEO><CIFRA>
<DECIMALAN>::=<CEO><RAZLOMLJEN>
<RAZLOMLJEN>::=.<CEO>
<EKSPONENT>::=E<CEO>!
              E<ZNAK><CEO>
<ZNAK>::=+!-
<SLOVO>::=A!B!C!D!E!F!G!H!I!J!K!L!M!N!O!P!Q!R!S!T!U!V!W!X!Y!Z
<CIFRA>::=0!1!2!3!4!5!6!7!8!9
<LET-NAREDBA>::=<A-IZRAZ>T2AUMT3<PROMENLJIVA>!
                <A-IZRAZ>T2AUMT3<PROMENLJIVA>
<INPUT-NAREDBA>::=<INPUT-LISTA>
<INPUT-LISTA>::=T2ULAZT3<PROMENLJIVA>!
               <INPUT-LISTA>T2ULAZT3<PROMENLJIVA>
<PRINT-NAREDBA>::=<PRINT-LISTA>
<PRINT-LISTA>::=<A-IZRAZ>T2AUMT3S1T2IZLAZT3S1!
               <PRINT-LISTA><A-IZRAZ>T2AUMT3S1T2IZLAZT3S1
<GOTO-NAREDBA>::=T2BEST3L<BROJ-PROGRAMSKOG-REDA>
<IF-NAREDBA>::=<R-IZRAZ>T3L<BROJ-PROGRAMSKOG-REDA>!
              <R-IZRAZ>T3L<BROJ-PROGRAMSKOG-REDA>!
              <R-IZRAZ>T3S2T2BEST3S3T1S2<BASIC-NAREDBA>T1S3T2BEZDT3BEZD
<R-IZRAZ>::=<A-IZRAZ>T2AUMT3S1<A-IZRAZ>T20DUT3S1<R-OPERATOR>
<R-OPERATOR>::=T2NNES!T2POS!T2NNUS!T2NUS!T2NES!T2NPOS
<STOP-NAREDBA>::=T2STOPT3STOP
<END-NAREDBA>::=T2ENDT3END
<BASIC-NAREDBA>::=<LET-NAREDBA>!
                  <INPUT-NAREDBA>!
                  <PRINT-NAREDBA>!
                  <GOTO-NAREDBA>!
                  <IF-NAREDBA>!
                  <STOP-NAREDBA>!
                  <END-NAREDBA>
<NAREDBA>::=<BASIC-NAREDBA>!
           <NAREDBA>\<BASIC-NAREDBA>
<PROGRAMSKI-RED>::=T1L<BROJ-PROGRAMSKOG-REDA><NAREDBA>
<BROJ-PROGRAMSKOG-REDA>::=<CEO>
<RED>::=<PROGRAMSKI-RED>;
```

Spisak 2

```
<RED>::=T1<OBELEZJE>T2<NAREDBA>!
        T2<NAREDBA>
<OBELEZJE>::=L<CEO>!
             S<CEO>
<NAREDBA>::=<TIP-I><IDENTIFIKATOR>!
            <TIP-P><PROMENLJIVA>!
            <TIP-O><OBELEZJE>!
            <TIP-F><FUNKCIJA>!
            <TIP-B><BEZ-ZNACAJA>
<TIP-I>::=MUA!
          SAB!ODU!MNO!DEL!STEPEN!
          IZLAZ
<TIP-P>::=AUM!
          ULAZ
<TIP-O>::=NNES!POS!NNUS!NUS!NES!NPOS!
          BES
<TIP-F>::=PPR
<TIP-B>::=BEZD!STOP!END
<IDENTIFIKATOR>::=<PROMENLJIVA>!
                  <KONSTANTA>
<PROMENLJIVA>::=<SLOVO>!
                <SLOVO><CIFRA>
<SLOVO>::=A!B!C!D!E!F!G!H!I!J!K!L!M!N!O!P!Q!R!S!T!U!V!W!X!Y!Z
<CIFRA>::=0!1!2!3!4!5!6!7!8!9
<KONSTANTA>::=<BROJ>
<BROJ>::=<CEO>!
         <DECIMALAN>!
         <CEO><EKSPONENT>!
         <DECIMALAN><EKSPONENT>
<CEO>::=<CIFRA>!
        <CEO><CIFRA>
<DECIMALAN>::=<CEO><RAZLOMLJEN>
<RAZLOMLJEN>::=<CEO>
<EKSPONENT>::=E<CEO>!
              E<ZNAK><CEO>
<ZNAK>::=+!-
<FUNKCIJA>::=ABS!ATN!COS!EXP!LOG!SIN!SQR
<BEZ-ZNACAJA>::=BEZD!STOP!END
```

Spisak 3

Izvodjenje naredbe 1Ø LET X = - 2.13; je:

```
<RED> => <PROGRAMSKI-RED>;
      => <BROJ-PROGRAMSKOG-REDA> <NAREDBA>;
      => <CEO> <NAREDBA>;
      => <CEO> <CIFRA> <NAREDBA>;
      => <CIFRA> <CIFRA> <NAREDBA>;
      => 1 <CIFRA> <NAREDBA>;
      => 10 <NAREDBA>;
      => 10 <BASIC-NAREDBA>;
      => 10 <LET-NAREDBA>;
      => 10 LET <PROMENLJIVA> = <A-IZRAZ>;
      => 10 LET <SLOVO> = <A-IZRAZ>;
      => 10 LET X = <A-IZRAZ>;
      => 10 LET X = - <SABIRAK>;
      => 10 LET X = - <CINILAC>;
      => 10 LET X = - <OSNOVA>;
      => 10 LET X = - <IDENTIFIKATOR>;
      => 10 LET X = - <KONSTANTA>;
      => 10 LET X = - <BROJ>;
      => 10 LET X = - <DECIMALAN>;
      => 10 LET X = - <CEO> <RAZLOMLJEN>;
      => 10 LET X = - <CIFRA> <RAZLOMLJEN>;
      => 10 LET X = - 2 <RAZLOMLJEN>;
      => 10 LET X = - 2. <CEO>;
      => 10 LET X = - 2. <CEO> <CIFRA>;
      => 10 LET X = - 2. <CIFRA> <CIFRA>;
      => 10 LET X = - 2.1 <CIFRA>;
      => 10 LET X = - 2.13;
```

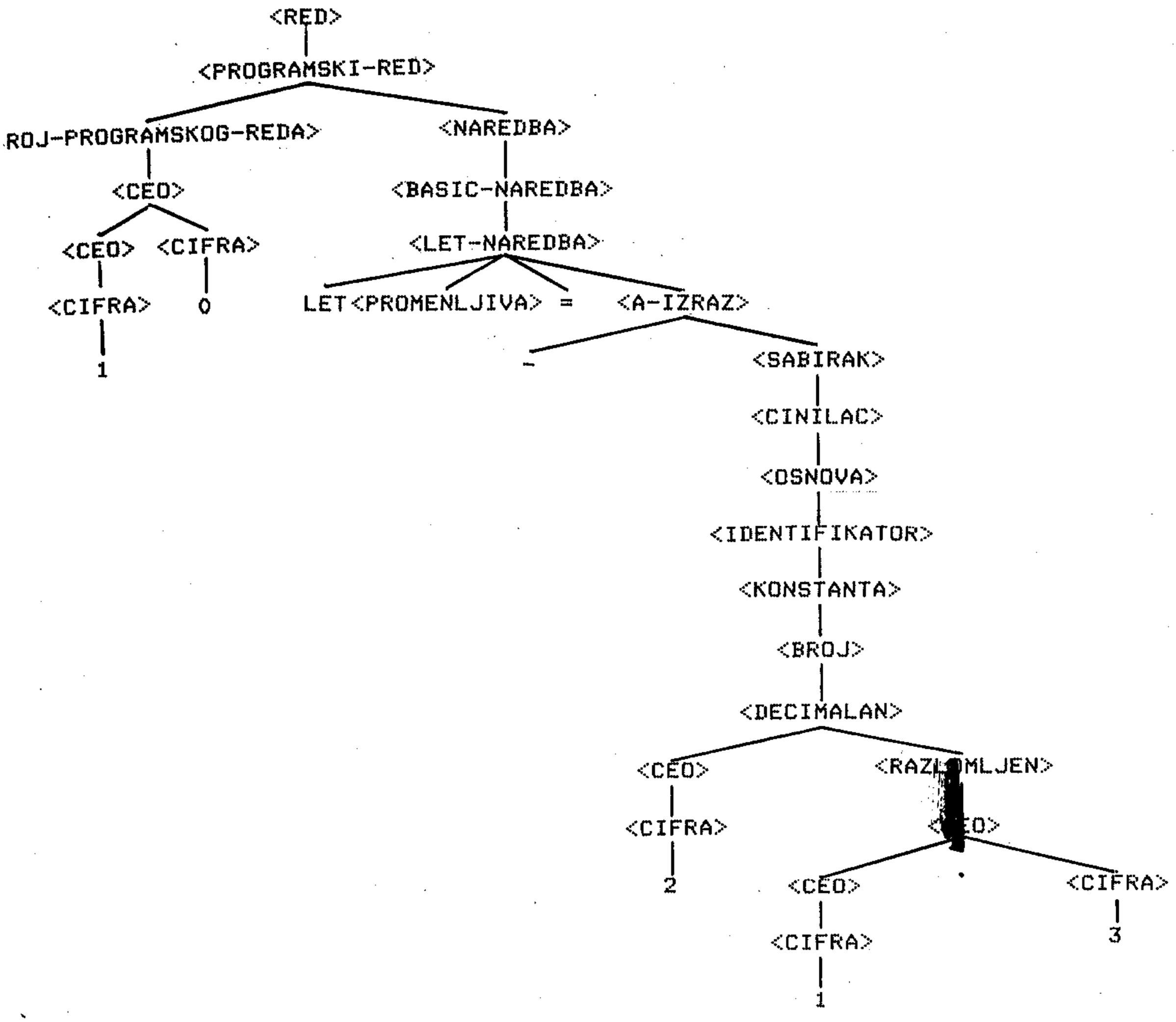
Prevodjenje naredbe 1Ø LET X = - 2.13; na mašinski-orijentisan jezik je:

```
<RED> => <PROGRAMSKI-RED>
      => T1 L <BROJ-PROGRAMSKOG-REDA> <NAREDBA>
      => T1 L <CEO> <NAREDBA>
      => T1 L <CEO> <CIFRA> <NAREDBA>
      => T1 L <CIFRA> <CIFRA> <NAREDBA>
      => T1 L1 <CEO> <NAREDBA>
      => T1 L10 <NAREDBA>
      => T1 L10 <BASIC-NAREDBA>
      => T1 L10 <LET-NAREDBA>
      => T1 L10 <IZRAZ> T2 AUM T3 <PROMENLJIVA>
      => T1 L10 <IZRAZ> T2 AUM T3 <SLOVO>
      => T1 L10 <IZRAZ> T2 AUM T3 X
      => T1 L10 <SABIRAK> T2 AUM T3 S1
                          T2 MUA T3 0
                          T2 ODU T3 S1
                          T2 AUM T3 X
      => T1 L10 <CINILAC> T2 AUM T3 S1
                          T2 MUA T3 0
                          T2 ODU T3 S1
                          T2 AUM T3 X
```

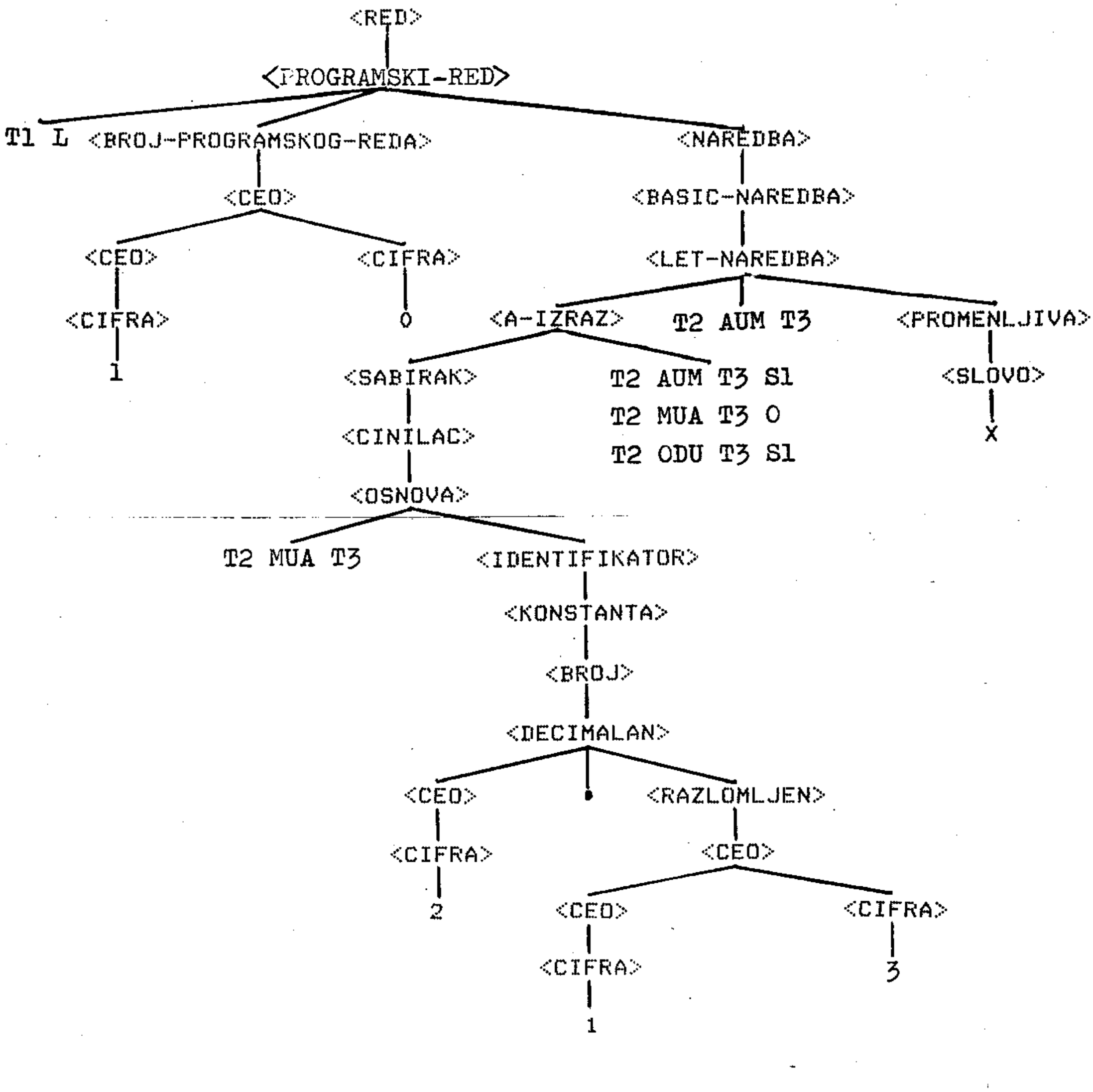
```
=> T1 L10 <OSNOVA> T2 AUM T3 S1
                   T2 MUA T3 0
                   T2 ODU T3 S1
                   T2 AUM T3 X
=> T1 L10 T2 MUA T3 <IDENTIFIKATOR>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 <KONSTANTA>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 <BROJ>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 <DECIMALAN>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 <CEO> <RAZLOMLJEN>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 <CIFRA> <RAZLOMLJEN>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 2 <RAZLOMLJEN>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 2. <CEO>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 2. <CEO> <CIFRA>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 2. <CIFRA> <CIFRA>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
```

```
=> T1 L10 T2 MUA T3 2.1 <CIFRA>
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 2.13
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
```

Drvo izvodjenja naredbe 1Ø LET X = - 2.13; je:



Drvo prevodjenja naredbe 1Ø LET X = - 2.13; je:



4.5. Jedna interpretacija šeme sintaksno-orijentisanog prevodjenja

Šema sintaksno-orijentisanog prevodjenja može se interpretirati* kao metoda transformisanja drveta izvodjenja ulazne gramatike u drvo izvodjenja izlazne gramatike.

Reč x ulaznog jezika je krošnja drveta izvodjenja ulazne gramatike. Neka je prevod reči x - reč y izlaznog jezika. Reč y je krošnja drveta izvodjenja izlazne gramatike.

Prevod reči x - reč y može se dobiti na sledeći način:

1) Konstruiše se drvo izvodjenja D ulazne gramatike G_u čija je krošnja reč x.

2) Pomoću šeme sintaksno-orijentisanog prevodjenja transformiše se drvo izvodjenja D u drvo izvodjenja D' izlazne gramatike G_i.

3) Krošnja drveta izvodjenja D' je reč y.

Algoritam: Transformisanje drveta pomoću šeme sintaksno-orijentisanog prevodjenja.

Ulazni podaci:

- Šema sintaksno-orijentisanog prevodjenja $T=(N,\Sigma,\Delta,R,S)$.
- Ulazna gramatika $G_u=(N,\Sigma,P_u,S)$.
- Izlazna gramatika $G_i=(N,\Delta,P_i,S)$.
- Drvo izvodjenja D ulazne gramatike G_u čija je krošnja reč $x\in\Sigma^*$

Izlazni podaci:

- Drvo izvodjenja D' izlazne gramatike G_i čija je krošnja reč $y\in\Delta^*$
- prevod reči x tj. $(x,y)\in\tau(T)$.

Postupak:

(1) Primeniti korak (2) rekursivno, počevši od korena drveta D.

(2) Neka je n unutrašnje teme drveta D i neka su temena $n_1,n_2,\ldots,n_k$ direktni potomci temena n .

(a) Iz spiska temena $n_1,n_2,\ldots,n_k$ izostaviti temena obeležena završnim simbolima i/ili praznim slovom.

(b) Neka je $A\to\alpha$ - pravilo izvodjenja ulazne gramatike G_u koje odgovara temenu n i njegovim direktnim potomcima temenima $n_1,n_2,\ldots,n_k$. A je obeležje temena n dok je niska α konkatenacija obeležja temena $n_1,n_2,\ldots,n_k$.

---

* Druga moguća interpretacija su transformacione gramatike Chomsky-og.

Izabrati iz skupa R pravilo izvodjenja i prevodjenja $A \rightarrow \alpha, \beta, \Pi$ i izvršiti zamenu temena $n_1, n_2, \ldots, n_k$, temenima $n_1', n_2', \ldots n_k'$. Niska β je konkatenacija obeležja temena $n_1', n_2', \ldots, n_k'$, a Π je permutacija indeksa nezavršnih simbola.

Uporedom sa zamenom temena izvršiti zamenu odgovarajućih poddrveta.

(c) Primeniti korak (2) na direktne potomke temena n, koji su nezavršni simboli, redom s leva udesno.

Primer:

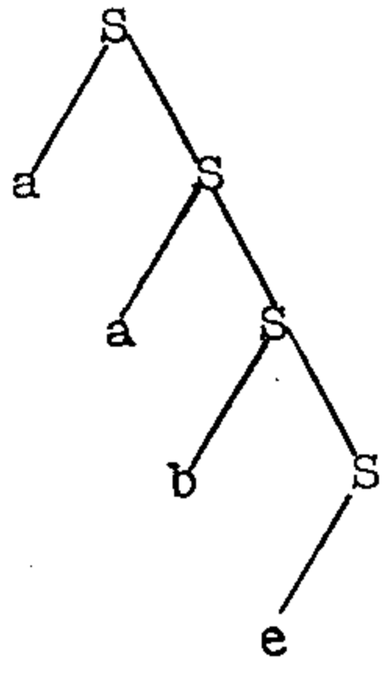
Neka je $T=(\{S\}, \{a,b,e\}, \{a,b,e\}, R, S)$

gde je $R=\{ S \rightarrow aS, Sa$

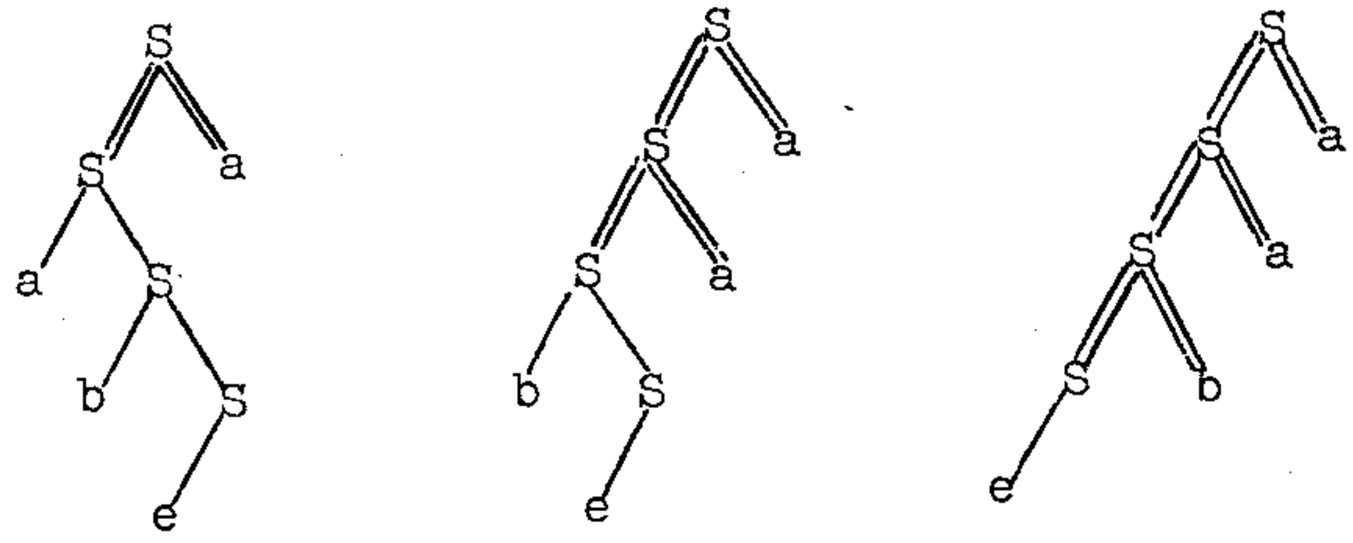
$S \rightarrow bS, Sb$

$S \rightarrow e, e \}$

Drvo izvodjenja D niske aab je:



Primenom prethodnog algoritma dobija se niz drveta od kojih je poslednje drvo prevodjenja D′:



U cilju bolje preglednosti potezi drveta prevodjenja nacrtani su dvostruko.

Prevod reči aab je reč baa .

Teorema 22:

(1) Ako su x i y redom krošnje drveta D i D' iz prethodnog algoritma onda je $(x,y) \in \tau(T)$

(2) Ako je $(x,y) \in \tau(T)$ onda postoji drvo izvodjenja D čija je krošnja x i niz drveta konstruisani na osnovu prethodnog algoritma od kojih je poslednje drvo prevodjenja D' čija je krošnja y.

Dokaz: Ovu teoremu dokazali su Aho i Ullman [3] i ona predstavlja dokaz korektnosti prethodnog algoritma.

Definicija 21: Ako se izostave svi potezi drveta izvodjenja koji se završavaju završnim simbolom i/ili praznim slovom dobija se potkresano drvo izvodjenja.

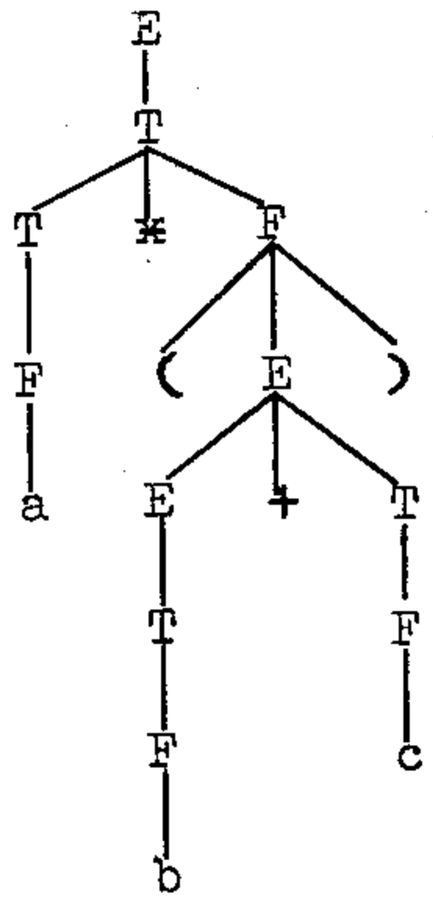
Primer

Data je gramatika G=({E,T,F} , {+,*,(,),a,b,c} ,P,E)

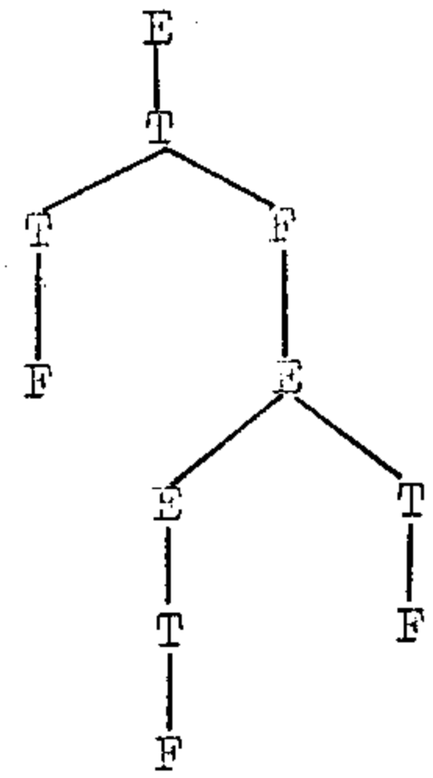
gde je P skup pravila izvodjenja:

$$P = \{ E \to T \mid E+T$$
$$T \to F \mid T*F$$
$$F \to (E) \mid a \mid b \mid c \}$$

Drvo izvodjenja niske a*(b+c) je:



Potkresano drvo je:



Teorema 23: Neka je $\mathcal{D}$ potkresano drvo izvodjenja, a $\mathcal{D}'$ potkresano drvo prevodjenja šeme sintaksno-orijentisanog prevodjenja $T=(N,\Sigma,\Delta,R,S)$.

a) Drvo $\mathcal{D}$ jednako je (izomorfno je) drvetu $\mathcal{D}'$.

b) Temena drveta $\mathcal{D}$ i $\mathcal{D}'$ su jednako obeležena.

Dokaz: Drvo $\mathcal{D}$ je potkresano drvo izvodjenja kontekstno-slobodne gramatike $G_u = (N,\Sigma,P_u,S)$, tj. $\mathcal{D}$ je obeleženo uredjeno drvo za koje važi:

a) koren drveta $\mathcal{D}$ obeležen je sa S

b) $\mathcal{D}_1,\ldots \mathcal{D}_k$ su potkresana poddrveta čiji su koreni redom $X_1,\ldots,X_k$ - nezavršni simboli-direktni potomci korena S, tj. postoji pravilo izvodjenja*:

$$S \rightarrow x_1X_1x_2 \ldots x_kX_kx_{k+1}$$

skupa P_u.

Slično, drvo $\mathcal{D}'$ je potkresano drvo izvodjenja kontekstno-slobodne gramatike $G_i=(N,\Sigma,P_i,S)$, tj. $\mathcal{D}$ je obeleženo uredjeno drvo za koje važi:

a) koren drveta $\mathcal{D}$ obeležen je sa S;

b) $\mathcal{D}'_1,\ldots,\mathcal{D}'_k$ su potkresana poddrveta čiji su koreni redom $X'_1,\ldots,X'_k$ - nezavršni simboli - direktni potomci korena S, tj. postoji pravilo izvodjenja:

$$S \rightarrow x'_1X'_1x'_2\ldots x'_kX'_kx'_{k+1}$$

skupa P_i.

---

* Reči x_i odnosno x_i $i=1,2,\ldots,k+1$ mogu da budu i prazne reči.

Niz nezavršnih simbola $X'_1 \ldots X'_k$ je permutacija niza nezavršnih simbola $X_1 \ldots X_k$ tj. postoji pravilo izvodjenja i prevodjenja.

$$S \rightarrow x_1 X_1 x_2 \ldots x_k X_k x_{k+1}, x'_1 X'_1 x'_2 \ldots x'_k X'_k x'_{k+1}, \Pi \qquad (*)$$

skupa R.

Postoji parcijalno bijektivno preslikavanje f_S

$$f_S: \{X_1, \ldots, X_k\} \rightarrow \{X'_1, \ldots, X'_k\}$$

definisano na sledeći način: $f_S(X_i) = X'_{\Pi(i)}$; i=1,...,k.

Na osnovu (*) sledi daje: SRX_i akko $SRf_S(X_i)$.

Konstruišimo totalno bijektivno preslikavanje f

$$f : \{X \mid X \text{ je obeležje temena drveta } \mathcal{D}\} \rightarrow \{X' \mid X' \text{ je obeležje temena drveta } \mathcal{D}'\}$$

na sledeći način:

$$(1) \quad f(X) = \bigcup_X f_X(X_i) = \bigcup_X X'_{\Pi(i)} \qquad i=1,2,\ldots$$

gde je: $X \rightarrow \ldots X_i \ldots, \ldots X'_{\Pi(i)} \ldots, \Pi$ (**)

Na osnovu (**) sledi da je:

(2) XRX_i akko $f(X) R f(X_i)$ i=1,2,...

(1) i (2) predstavljaju tvrdjenje teoreme .

Teorema 24: Neka je D drvo izvodjenja, a D' drvo prevodjenja šeme SO-prevodjenja T.

Drvo D različito je od drveta D'.

Dokaz: Neposredno sledi na osnovu činjenice da SO-prevodjenje nije homomorfizam.

4.6. Modifikacije šeme sintaksno-orijentisanog prevodjenja

Pravila izvodjenja i prevodjenja šeme SO-prevodjenja

$$T=(N,\Sigma,\Delta,R,S)$$

su oblika:

$$A \rightarrow \alpha, \beta, \Pi$$

gde je: $A \in N$

$\alpha \in (N \cup \Sigma)^*$

$\beta \in (N \cup \Delta)^*$

i uz to nezavršni simboli niske β su Π-permutacija nezavršnih simbola niske α.

Znači, nezavršni simboli niske β su upotpunosti odredjeni nezavršnim simbolima niske α i permutacijom Π. Izvršimo zamenu nezavršnih simbola niske β odgovarajućim indeksima permutacije Π, a permutaciju Π izostavimo. Tako modifikovano pravila izvodjenja i prevodjenja su oblika:

$$A \rightarrow \alpha, \beta'$$

gde je $\beta \in (\mathcal{N} \cup \Delta)^*$; $\mathcal{N} = \{1,2,3,\ldots\}$

Da bi se izbegao slučaj $\mathcal{N} \cap \Delta \neq \emptyset$ - umesto skupa uzima se skup $\mathcal{N}_G = \{G1,G2,G3,\ldots\}$.

Niska α vodi nisku β' pa otuda i naziv sintaksno-vodjena (orijentisano) prevodjenje.

Primer

Modifikovana pravila prevodjenja proceduralno-orijentisanog jezika na mašinski-orijentisan jezik nastavnog računara data su u spisku 4.

Prevodjenje naredbe 1Ø LET X=-2.13; na mašinski-orijentisan jezik je:

```
<RED> => G1
      => T1 L G1 G2
      => T1 L G1 G2
      => T1 L G1 G2 G2
      => T1 L G1 G2 G2
      => T1 L1 G2 G2
      => T1 L10 G2
      => T1 L10 G1
      => T1 L10 G1
      => T1 L10 G2 T2 AUM T3 G1
      => T1 L10 G2 T2 AUM T3 G1
      => T1 L10 G2 T2 AUM T3 X
      => T1 L10 G1 T2 AUM T3 S1
                   T2 MUA T3 0
                   T2 ODU T3 S1
                   T2 AUM T3 X
      => T1 L10 G1 T2 AUM T3 S1
                   T2 MUA T3 0
                   T2 ODU T3 S1
                   T2 AUM T3 X
   => T1 L10 G1 T2 AUM T3 S1
                T2 MUA T3 0
                T2 ODU T3 S1
                T2 AUM T3 X
=> T1 L10 T2 MUA T3 G1
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
```

```
G1!
G1!
G1T2AUMT3S1T2MUAT3OT2ODUT3S1!
G2T2AUMT3S1G1T2SABT3S1!
G2T2AUMT3S1G1T2ODUT3S1!
G1!
G2T2AUMT3S1G1T2MNOT3S1!
G2T2AUMT3S1G1T2DELT3S1!
G1!
G2T2AUMT3S1G1T2STEPENT3S1!
T2MUAT3G1!
G1!
G1!
G1!
G2T2PPRT3G1!
ABS!ATN!COS!EXP!LOG!SIN!SQR!
G1!
G1G2!
G1!
G1!
G1!
G1G2!
G1G2!
G1!
G1G2!
G1G2!
.G1!
EG1!
EG1G2!
+!-!
<SLOVO>::=A!B!C!D!E!F!G!H!I!J!K!L!M!N!O!P!Q!R!S!T!U!V!W!X!Y!Z!
<CIFRA>::=0!1!2!3!4!5!6!7!8!9!
G2T2AUMT3G1!
G2T2AUMT3G1!
G1!
T2ULAZT3G1!
G1T2ULAZT3G2!
G1!
G1T2AUMT3S1T2IZLAZT3S1!
G1G2T2AUMT3S1T2IZLAZT3S1!
T2BEST3LG1!
G1T3LG2!
G1T3LG2!
G1T3S2T2BEST3S3T1S2G2T1S3T2BEZDT3BEZD!
G1T2AUMT3S1G3T2ODUT3S1G2!
T2NNES!T2POS!T2NNUS!T2NUS!T2NES!T2NPOS!
T2STOPT3STOP!
T2ENDT3END!
G1!
G1!
G1!
G1!
G1!
G1!
G1!
G1!
G1!
G1G2!
T1LG1G2!
G1!
G1!
```

Spisak 4

```
=> T1 L10 T2 MUA T3 G1
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 G1
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 G1
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
=> T1 L10 T2 MUA T3 G1 G2
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
```

•

•

•

•

dok se konačno ne dobije prevod:

```
=> T1 L10 T2 MUA T3 2.13
          T2 AUM T3 S1
          T2 MUA T3 0
          T2 ODU T3 S1
          T2 AUM T3 X
```

G1 i G2 mogu se shvatiti kao saobraćajci koji pokazuju: "produži ulevo odnosno udesno po drvetu".

4.7. Uopštenje šeme sintaksno-orijentisanog prevodjenja

Po definiciji je šema SO-prevodjenja T petorka:

$$T=(N,\Sigma,\Delta,R,S)$$

gde je:

N - skup nezavršnih simbola,

Σ - ulazna azbuka,

Δ - izlazna azbuka,

R - skup pravila izvodjenja i prevodjenja

S - početni simbol.

Pravila izvodjenja i prevodjenja su oblika:

$$A \rightarrow \alpha, \beta, \Pi$$

gde je:

$A \in N$

$\alpha \in (N \cup \Sigma)^*$

$\beta \in (N \cup \Delta)^*$ i

Π - permutacija indeksa nezavršnih simbola niski α i β.

Permutacija Π indeksa nezavršnih simbola niski α i β navodi se u slučaju višestrukog pojavljivanja jednog ili više nezavršnih simbola u niski α odnosno niski β.

Sada ćemo dokazati da se svako pravilo:

$$A \rightarrow \alpha, \beta, \Pi$$

može svesti na niz pravila:

$$A \rightarrow \alpha', \beta', \Pi'$$

takvih da su svi nezavršni simboli niske α' odnosno niske β' medjusobno različiti pa je permutacija Π' očigledna i ne mora se navoditi.

Definicija 22: Kontekstno-slobodna gramatika $G=(N,\Sigma,P,S)$ zadata je u normalnom obliku ako su za sva pravila izvodjenja ispunjeni uslovi:

- isti nezavršni simbol javlja se najviše jedanput na desnoj strani pravila.
- S se ne javlja na desnoj strani pravila.

Postoji algoritam svodjenja proizvoljne kontekstno-slobodne gramatike na normalni oblik.

Definicija 23: Šema SO-prevodjenja $T=(N,\Sigma,\Delta,R,S)$ zadata je u normalnom obliku ako su za sva pravila izvodjenja i prevodjenja ispunjeni uslovi:

- isti nezavršni simbol javlja se najviše jedanput u niski α odnosno u niski β
- S se ne javlja u niski α i niski β.

Algoritam: Svodjenje šeme SO-prevodjenja na normalan oblik.

Ulaz: Šema SO-prevodjenja $T=(N,\Sigma,\Delta,R,S)$

Izlaz: Šema SO-prevodjenja $T'=(N',\Sigma',\Delta',R',S')$ u normalnom obliku .

Postupak: Šema T′dobija se iz šeme T na sledeći način:

(1) Ako se početni simbol S javlja u niski α odnosno u niski β nekog pravila izvodjenja i prevodjenja skupa R, uvodjenjem novog* početnog simbola S′i pravila izvodjenja i prevodjenja S′→S,S početni simbol S′više se neće javljati u niski α odnosno niski β nijednog pravila izvodjenja i prevodjenja skupa R′.

(2) U skup pravila R' neposredno se unose pravila skupa R oblika:

$$A \rightarrow \alpha, \beta, \Pi^{**}$$

kod kojih se isti nezavršni simbol javlja najviše jedanput u niski α odnosno u niski β.

(3) Umesto pravila iz R oblika:

$$A \rightarrow \alpha, \beta, \Pi$$

kod kojih se jedan ili više nezavršnih simbola višestruko pojavljuju u niski α odnosno u niski β unosi se niz pravila oblika:

$$A \rightarrow \alpha', \beta', \Pi' = \Pi^{**}$$
$$X^{(1)} \rightarrow X_i, X_i$$
$$X^{(2)} \rightarrow X_i, X_i$$
$$\vdots$$
$$Y^{(k)} \rightarrow X_j, X_j$$

- niske α' i β' dobijene su odgovarajućom zamenom nezavršnih simbola
- $X_i, \ldots, X_j$ su nezavršni simboli koji su se višestruko pojavljivali,
- $X^{(1)}, X^{(2)}, \ldots, Y^{(k)}$ su novouvedeni nezavršni simboli.

Teorema 25: Neka je T=(N,Σ,Δ,R,S) šema proizvoljnog SO-prevodjenja $\tau(T)$. Ako je T′= (N′,Σ,Δ,R′,S′) šema SO-prevodjenja $\tau(T')$ dobijena svodjenjem šeme T na normalni oblik onda je $\tau(T) = \tau(T')$

Dokaz: Neka je G=(N,Σ,P,S) - kontekstno-slobodna gramatika i $A \rightarrow \alpha B \beta$ - pravilo izvodjenja.

Dalje neka su:

$$B \rightarrow \gamma_1 \mid \gamma_2 \mid \cdots \mid \gamma_n$$

sva moguća pravila izvodjenja čiji je koren B.

---

* $S' \notin N$, $S' \in N' = N \cup \{S'\}$

** Permutacija Π se može izostaviti.

Gramatika $G'=(N,\Sigma,P',S)$

gde je $P'=(P-\{A\to\alpha B\beta\})\cup\{A\to\alpha\gamma_1\beta\,|\,\alpha\gamma_2\beta\,|\cdots|\,\alpha\gamma_n\beta\}$

je: - kontekstno-slobodna i

- $L(G')=L(G)$.

Drugačije rečeno izvršena zamena pravila izvodjenja $A\to\alpha B\beta$ odgovarajućim nizom pravila izvodjenja $A\to\alpha\gamma_1\beta\,|\,\alpha\gamma_2\beta\,|\cdots|\,\alpha\gamma_n\beta$ ne utiče na promenu jezika L(G). Očigledno je da važi i obratno. Izbacivanje višestrukih nezavršnih simbola iz niske α odnosno β pravila izvodjenja i prevodjenja $A\to\alpha,\beta,\Pi$ je poseban slučaj gore opisanih zamena. Odavde sledi da se svodjenjem gramatika G_u odnosno G_i šeme T na normalni oblik ne menja jezik L_u odnosno L_i pa je $\tau(T)=\tau(T')$ što je i trebalo dokazati.

Često se pri prevodjenju jednog prirodnog jezika na drugi prirodan jezik dobijaju više sintaksno različita, a semantički ekvivalentna (ili skoro ekvivalentna) prevoda. Onda prevodilac, na osnovu razmih kriterijuma, odlučuje šta je konačan prevod.

Šema SO-prevodjenja T definisana na dosadašnji način, definiše jednoznačno SO-prevodjenje $\tau(T)$. Drugačije rečeno:

$$(\forall x)x\in L_u(\exists_1 y)y\in L_i\quad \tau(x)=y$$

tj.

$$(\neg\exists\, y_1,y_2)y_1,y_2\in L_i\quad \tau(x)=y_1\wedge\tau(x)=y_2$$

Neka je data šema SO-prevodjenja T. Odgovarajućim transformacijama šema T može se svesti na normalni oblik, koji se dalje može definisati kao trojka:

$$T=(G_u,G_i,Q)$$

gde je:

(1) $G_u=(N,\Sigma,P_u,S)$ - ulazna gramatika

a) N - konačan skup nezavršnih simbola

b) Σ - ulazna azbuka

c) P_u - konačan skup pravila izvodjenja oblika: $A\to\alpha$; $A\in N$, $\alpha\in(N\cup\Sigma)^*$

d) S - početni simbol

(2) $G_i=(N,\Delta,P_i,S)$ - izlazna gramatika

a) Δ - izlazna azbuka

b) P_i - konačan skup pravila izvodjenja $B\to\beta$; $B\in N$, $\beta\in(N\cup\Delta)^*$

Gramatike G_u i G_i su kontekstno-slobodne i nalaze se u normalnom obliku

(3) Q je obostrano-jednoznačno preslikavanje skupa P_u u skup P_i: $Q : P_u \rightarrow P_i$
koje svakom pravilu izvodjenja $A \rightarrow \alpha$ skupa P_u, pridružuje pravilo izvodjenja i prevodjenja.

Definicija 24: Uopštena šema SO-prevodjenja je četvorka:

$$U = (G_u, G_i, H, Q)$$

gde je:

(1) $G_u=(N,\Sigma,P_u,S)$ - ulazna gramatika.
   a) N - konačan skup ulaznih nezavršnih simbola
   b) Σ - ulazna azbuka
   c) P_u- konačan skup pravila izvodjenja oblika:
      $A \rightarrow \alpha$; $A \in N, \alpha \in (N \cup \Sigma)^*$
   d) S - početni simbol.

(2) $G_i=(\Gamma,\Delta,P_i,S_1)$ - izlazna gramatika.
   a) Γ - konačan skup izlaznih nezavršnih simbola
   b) Δ - izlazna azbuka
   c) P_i- konačan skup pravila izvodjenja oblika:
      $B \rightarrow \beta$; $B \in \Gamma, \beta \in (\Gamma \cup \Delta)^*$
   d) S_1- početni simbol.

Gramatike G_u i G_i su kontekstno-slobodne i date su u normalnom obliku.

(3) H je funkcija koja svakom ulaznom nezavršnom simbolu $A \neq S$ pridružuje skup izlaznih nezavršnih simbola $\{A_1, A_2, \ldots, A_m\}$.
$A \neq B \Rightarrow H(A) \cap H(B) = \emptyset$. $H(S)=S_1$.

(4) Q je funkcija koja svakom pravilu izvodjenja $A \rightarrow \alpha$; $A \neq S$ skupa P_u pridružuje skup pravila izvodjenja

$$\{A_1 \rightarrow \beta_1, A_2 \rightarrow \beta_2, \ldots, A_m \rightarrow \beta_m\}$$

skupa P_i

$$Q(S \rightarrow \alpha) = \{S_1 \rightarrow \beta_1, S_1 \rightarrow \beta_2, \ldots, S_1 \rightarrow \beta_m\}$$

Izlazni nezavršni simboli niske β_i su i-ti i predstavljaju permutaciju ulaznih nezavršnih simbola niske α.

Primer

Neka je:

(1) $G_u=(\{S',S\},\{a,b,e\},P_u,S')$

gde je: $P_u = \{ S' \to S,$
$S \to aS,$
$S \to bS,$
$S \to e \}$

(2) $G_i = (\{S_1', S_1, S_2, S_3\}, \{p,k,a,b,x,y,z,e\}, P_i, S_1')$

gde je: $P_i = \{S_1' \to S_1,$
$S_1' \to pS_2$
$S_1' \to S_3$
$S_1 \to xS_1,$
$S_2 \to aS_2,$
$S_3 \to S_3 z,$
$S_1 \to yS_1,$
$S_2 \to bS_2,$
$S_1 \to e,$
$S_2 \to k,$
$S_3 \to e\}$

(3) $H(S') = S_1'$
$H(S) = \{S_1, S_2, S_3\}$

(4) $Q(S' \to S) = \{ S_1' \to S_1,$
$S_1' \to pS_2,$
$S_1' \to S_3\}$
$Q(S \to aS) = \{ S_1 \to xS_1,$
$S_2 \to aS_2,$
$S_3 \to S_3 z \}$
$Q(S \to bS) = \{S_1 \to yS_1,$
$S_2 \to bS_2,$
$S_3 \to S_3 z\}$
$Q(S \to e) = \{S_1 \to e,$
$S_2 \to k,$
$S_3 \to e\}$

$U = (G_u, G_i, H, B)$ je uopštena šema SO-prevodjenja.

Za svako $x \in \Sigma^*$ definišimo $U(x)$ - skup prevoda od x na sledeći način:

(1) Ako $x \notin L(G_u)$ onda je $U(x)=\emptyset$

(2) Ako je $x \in L(G_u)$ onda u gramatici G_u postoji izvodjenje reči x iz početnog simbola S:

$$\begin{aligned} S = \alpha_0 &\Rightarrow \alpha_1 \\ &\Rightarrow \alpha_2 \\ &\vdots \\ &\Rightarrow \alpha_{i-1} \\ &\Rightarrow \alpha_i \\ &\Rightarrow \alpha_{i+1} \\ &\vdots \\ &\Rightarrow \alpha_\eta = x. \end{aligned}$$

Izvodjenje reči x definiše drvo izvodjenja D čija je krošnja reč x.

Prevodi reči x rekursivno se definišu na sledeći način:

(a) Svakom unutrašnjem temenu n drveta izvodjenja D obeleženog pravilom izvodjenja $A \to \alpha$; $A \in N, \alpha \in (N \cup \Sigma)^*$ pridružuje se skup $H(A)=\{A_1, A_2, \ldots, A_m\}$; $A_i \in \Gamma$, $i=1,2,\ldots,m$.

Vrednosti izlaznih nezavršnih simbola A_i, $i=1,2,\ldots,m$ izračunavaju se pomoću skupa pravila izvodjenja $R(A \to \alpha)=\{A_i \to \beta_i \mid i=1,2,\ldots,m\}$ i vrednosti izlaznih nezavršnih simbola koji figurišu u niski β_i $i=1,\ldots,m$.

(b) Neka je $\alpha = x_0 B_1 x_1 B_2 x_2 \ldots B_k x_k$, gde je $x_j \in \Sigma^*$, $B_j \in N$ i $j=\emptyset,1,\ldots,k$. Dalje neka je $A_i \to y_0 C_1 y_1 C_2 y_2 \ldots C_e y_e$ gde je $y_j \in \Delta^*$, $C_j \in \Gamma$ i $j=\emptyset,1,\ldots,e$.

$$v(A_i) = y_0 v(C_1) y_1 v(C_2) y_2 \ldots v(C_e) y_e \in \Delta^*$$

gde je $v(C_j)$ - vrednost od C_j.

(c) Vrednost završnog simbola je taj isti simbol.

<u>Definicija 25</u> $\tau(U)$ je prevodjenje definisano uopštenom šemom SO-prevodjenja U.

$$\tau(U) = \{(x,y) \mid y \in U(x)\}.$$

Nastavak prethodnog primera

$$L(G_u) = \{a,b\}^* = \{e,a,b,aa,ab,ba,bb,\ldots\}$$

$$L(G_i) = \{x,y\}^* \cup p \{a,b\}^* k \cup \{z\}^*$$

Neka je u prevodjenje definisanom datom uopštenom šemom U SO-prevodjenja.

$$u(w) = \{w', pwk, w''\}\ ;\ w \in L(G_u) = \{a,b\}^*$$

reč w'dobija se iz reči w tako što se redom:

- slova a zamene slovom x odnosno
- slova b zamene slovom y

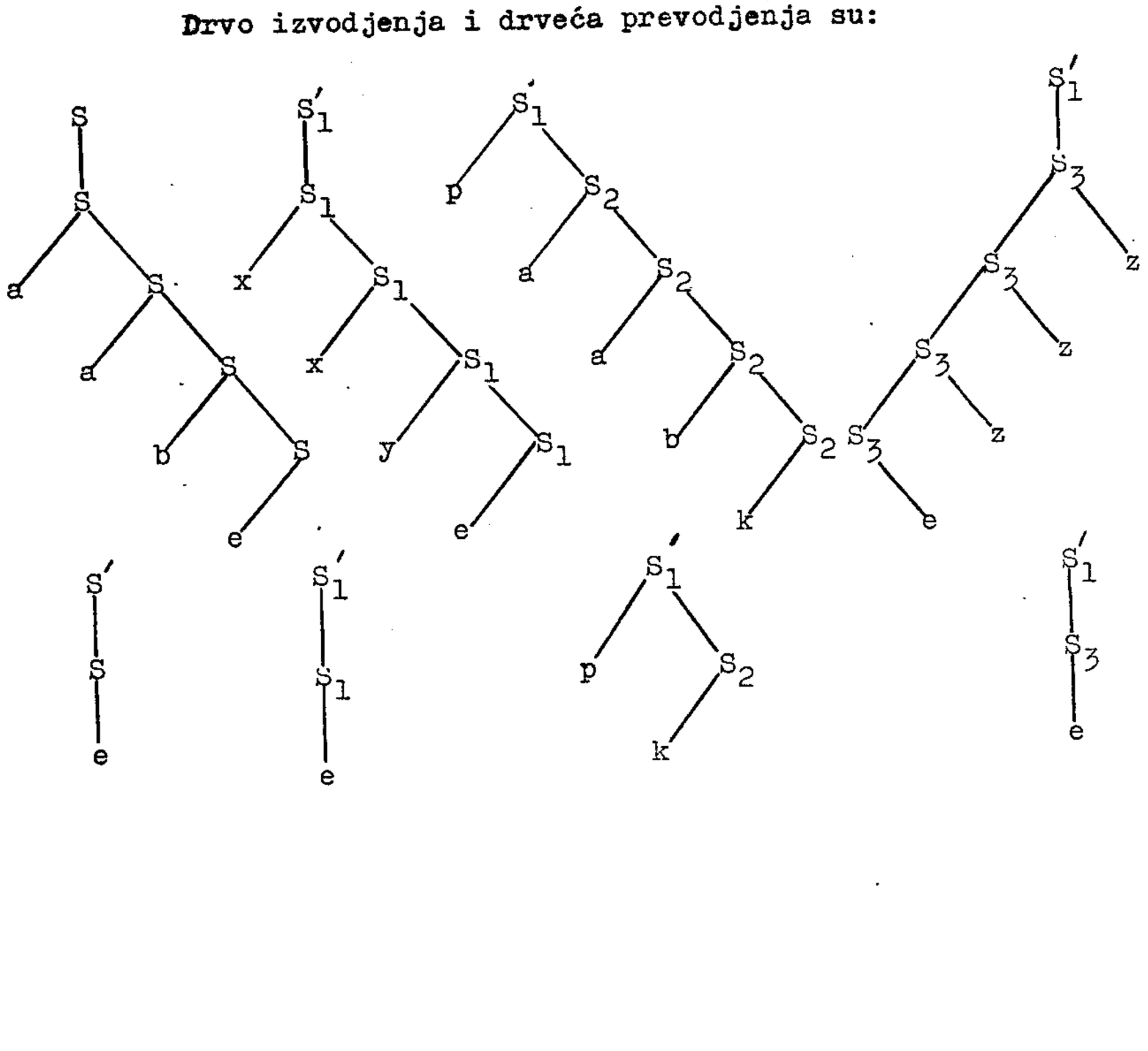
reč w''dobija se iz reči w tako što se redom slova a i slova b zamene slovom z.

Na primer

$$u(aab) = \{xxy, paabk, zzz\}$$

$$u(e) = \{e, pk\} = \{e, pk, e\}$$

Drvo izvodjenja i drveća prevodjenja su:



4.8. Primena sintaksno-orijentisanog prevodjenja u teoriji formalnih gramatika i jezika

Teorema 26: Ako je $\tau(T)$ - sintaksno-orijentisano prevodjenje definisano šemom sintaksno-orijentisanog prevodjenja T onda postoji konstanta C takva da za svaku nisku x iz oblasti definisati prevodjenje $\tau(T)$ postoji niska y$\neq$e iz skupa vrednosti prevodjenja $\tau(T)$ za koju važi: $|y| \leqslant c|x|$

Dokaz: Ovu teoremu dokazali su 1969. godine Aho i Ullman [3].

Primer

Pretpostavimo da je:

$$\tau(T) = \{(\text{celobrojni zapis broja } x, \text{binarni zapis broja } x) \mid x \in N\}$$

sintaksno-orijentisano prevodjenje.

$\tau(T)$ je skup parova:

$$\tau(T) = \{(\emptyset,\emptyset),(1,1),(2,1\emptyset),(3,11),(4,100),\ldots,(x,y),\ldots\}$$

Obeležimo sa x i y redom duzine niski x i y.

Pomoću n-cifara dekadnog odnosno binarnog brojnog sistema registruje se $1\emptyset^n$ odnosno 2^n različitih brojeva. Na osnovu jednakosti:

$$10^{n_1} = 2^{n_2}$$

može se izračunati odnos izmedju dužina zapisa celih i binarnih brojeva. Zaista:

$$\log 1\emptyset^{n_1} = \log 2^{n_2}$$

$$n_1 \log 1\emptyset = n_2 \log 2$$

$$n_2 = n_1 \frac{\log 1\emptyset}{\log 2} < 4n_1$$

Znači, binarni zapis broja x je najviše 4 puta duži od celobrojnog zapisa istog broja x.

Odavde sledi da je u navedenom primeru konstanta $C \leqslant 4$ i da važi:

$$|y| \leqslant 4|x|$$

A.B.Gladkij je dokazao [101] da kontekstno-zavisna gramatika

$$G = (\{I,A,B,C,D,E,F\}, \{*\}, P, I)$$

gde je: $P = \{$

(1) $I \rightarrow ABBDF$

(2) $BD \rightarrow DCB$

(3) $BC \rightarrow CB$

(4) $AD \rightarrow AAE$

(5) $EC \rightarrow AE$

(6) $EB \rightarrow BE$

(7) $EF \rightarrow BBDF$

(8) $DF \rightarrow *$

(9) $B \rightarrow *$

(10) $A \rightarrow *$

(11) $I \rightarrow * \}$

generiše jezik $N_2 = \{n^2 \mid n=1,2,\ldots\}$. Reči jezika N_2 su prirodni brojevi - kvadrati kodirani primitivnim kodom, tj.

$$N_2 = \{*, \underbrace{****}_{9}, \ldots \underbrace{**\ldots*}_{n^2}, \ldots\}$$

Primer

a) $I \overset{11}{\Rightarrow} *$

b) $I \overset{1}{\Rightarrow} ABBDF$

$\overset{10}{\Rightarrow} *BBDF$

$\overset{9}{\Rightarrow} **BDF$

$\overset{9}{\Rightarrow} ***DF$

$\overset{8}{\Rightarrow} ****$

c) $I \overset{1}{\Rightarrow} ABBDF$

$\overset{2}{\Rightarrow} ABDCBF$

$\overset{2}{\Rightarrow} ADCBCBF$

$\overset{3}{\Rightarrow} ADCCBBF$

$\overset{4}{\Rightarrow} AAECCBBF$

$\overset{5}{\Rightarrow} AAAECBBF$

$\overset{5}{\Rightarrow} AAAAEBBF$

$$\overset{6}{\Rightarrow} AAAABEBF$$

$$\overset{6}{\Rightarrow} AAAABBEF$$

$$\overset{7}{\Rightarrow} AAAABBBBDF$$

$$\vdots$$

$$\Rightarrow \text{⁂⁂⁂⁂⁂⁂⁂⁂⁂} \quad .$$

Postavlja se pitanje da li se jezik N_2 može generisati kontekstno-slobodnom gramatikom. Odgovor je negativan.

Teorema 27: Jezik $N_2 = \{\text{⁂}, \text{⁂⁂⁂⁂}, \underbrace{\text{⁂...⁂}}_{9}, \ldots, \underbrace{\text{⁂...⁂}}_{n^2}, \ldots\}$

nije kontekstno-slobodan.

Dokaz: Jezik $N_1 = \{\text{⁂}, \text{⁂⁂}, \text{⁂⁂⁂}, \ldots, \underbrace{\text{⁂...⁂}}_{n}, \ldots\}$

je kontekstno-slobodan.

Zaista, gramatika $G_1 = (\{S\}, \{\text{⁂}\}, P, S)$

gde je $P = \{ S \rightarrow \text{⁂},\ S \rightarrow S\text{⁂} \}$

je kontekstno-slobodna i generiše jezik N_1.

Pretpostavimo suprotno, tj. da je jezik:

$$N_2 = \{\text{⁂}, \text{⁂⁂⁂⁂}, \underbrace{\text{⁂...⁂}}_{9}, \ldots, \underbrace{\text{⁂...⁂}}_{n^2}, \ldots\}$$

kontekstno-slobodan.

Onda je

$$\Psi(T) = \{(\underbrace{\text{⁂...⁂}}_{n}, \underbrace{\text{⁂...⁂}}_{n^2}) \mid n=1,2\ldots\}$$

sintaksno-orijentisano prevodjenje pa mora da postoji konstanta c takva da za svaku reč x jezika N_1 postoji reč y jezika N_2 za koju važi da je:

$$|y| \leqslant c|x| .$$

Medjutim, kako je:

$$|y| = n^2, \text{ a}$$
$$|x| = n$$

važi:

$$n^2 \leqslant cn$$

tj.

$$n \leqslant c$$

što znači da je konstanta c neograničena odnosno nepostoji - što je suprotno teoremi Aho - Ullmana. Odavde sledi da je polazna pretpostavka pogrešna tj. jezik N_2 nije kontekstno-slobodan.

Teorema 28: Jezici $N_m = \{\underbrace{* \ldots *}_{n^m} \mid n=1,2,\ldots\}$, $m = 2,3,\ldots$ nisu kontekstno-slobodni

Dokaz: Sličan je dokazu prethodne teoreme •

5. PROGRAMSKA REALIZACIJA ŠEME SINTAKSNO-ORIJENTISANOG PREVODJENJA (UNIVERZALNOG KOMPILATORA)

5.1. Metoda medjujezika

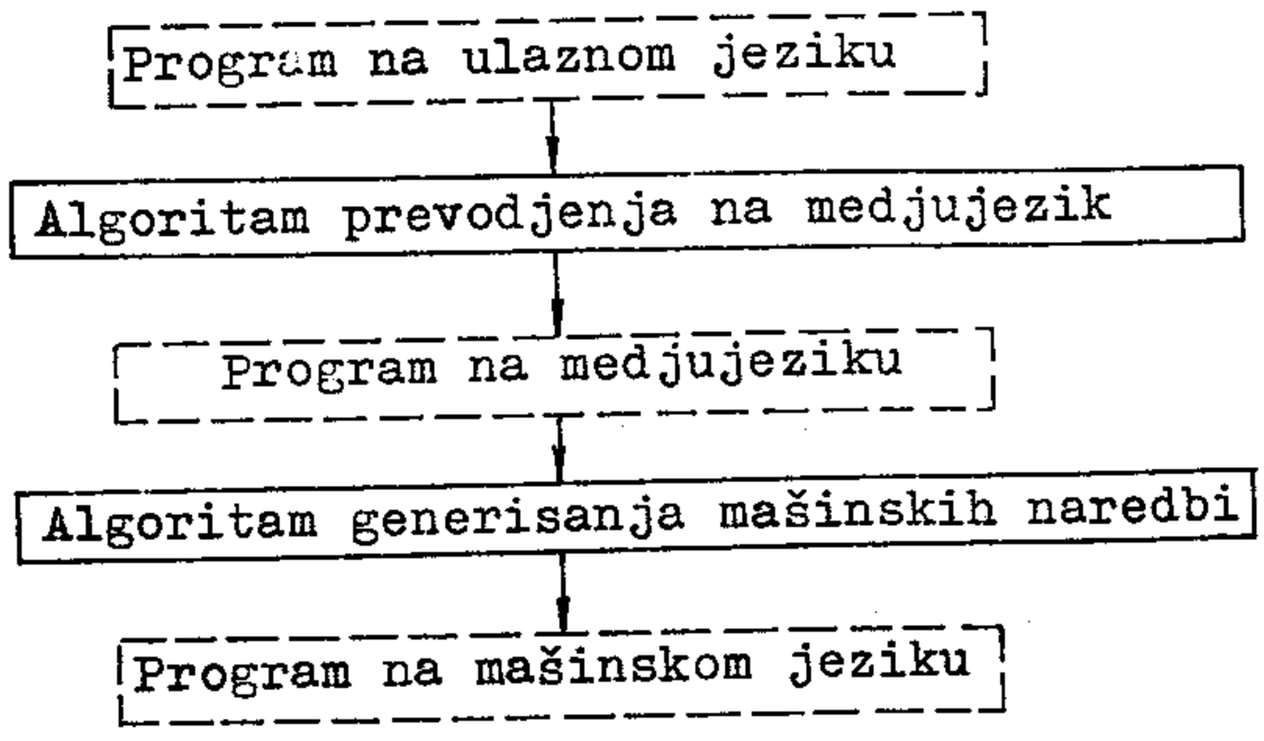
Prve metode prevodjenja zasnivale su se na specifičnostima konkretnog ulaznog jezika i specifičnostima konkretnog računara za koji je pravljen prevodilac. Pri koncipiranju i realizaciji prevodioca za drugi programski jezik i/ili računar gotovo sav posao se mora iz početka (iznova) raditi. Kako je sav taj posao veoma obiman i težak (na primer da bi se napravio jedan FORTRAN-prevodilac potrebno je desetak čovek-godina) postalo je neophodno tražiti nove puteve - metode prevodjenja koji neće zavisiti kako od specifičnosti programskog jezika tako i od specifičnosti računara. Medjutim, do današnjeg dana nisu pronadjene metode prevodjenja potpuno nezavisne od specifičnosti programskog jezika računara. Uvodjenjem u proces prevodjenja mašinski-nezavisnog medjujezika, veliki deo algoritama prevodjenja učinjen je nezavisnim od specifičnosti računara. Manji deo algoritma prevodjenja - generisanje kôda (mašinskih naredbi) ostao je i dalje zavisan od specifičnosti računara.

Opšta šema prevodjenja metodom medjujezika prikazana je na slici 5.1.

Algoritam prevodjenja ulaznog jezika na medjujezik izvršava sintaksnu analizu i deo semantičke analize koja je u vezi sa prevodjenjem na medjujezik.

Algoritam generisanja mašinskih naredbi izvršava deo semantičke analize vezane za raspoznavanje operacija medjujezika i izvršava za svaku operaciju odgovarajući potprogram koji obrazuje niz mašinskih naredbi odgovarajućih datoj operaciji.

Potprogrami o kojima je reč nazivaju se semantički potprogrami jer izražavaju semantiku operacija medjujezika.



Slika 5.1.

5.2. Metoda semantičkih potprograma

Direktne metode prevodjenja na mašinski jezik ne zavise od specifičnosti konkretnog računara ali bitno koriste specifičnosti konkretnog ulaznog jezika. Navedeni nedostatak direktnih metoda prevodjenja je prepreka stvaranju, na njihovoj osnovi, opštih algoritama prevodjenja primenljivih na najrazličitije programske jezike. Opšti algoritmi prevodjenja omogućavaju jednostavniju realizaciju prevodilaca kao i potpunu (ili bar delimičnu) automatizaciju procesa pisanja prevodilaca.

U cilju stvaranja opštih algoritama prevodjenja neophodno je razdvojiti etape sintaksne i semantičke analize programa. Razlog za to je nezavisnost sintakse ulaznog jezika od specifičnosti računara. Prirodno je da algoritam sintaksne analize bude nezavisan od računara. Ako se još iskoristi jedna od standardnih formi opisa sintakse ulaznog jezika može se sastaviti algoritam sintaksne analize nezavisan od specifičnosti ulaznog jezika.

Još od prvih FORTRAN-kompilatora koristi se metoda prevodjenja kod koje je sintaksna analiza u odredjenom stepenu

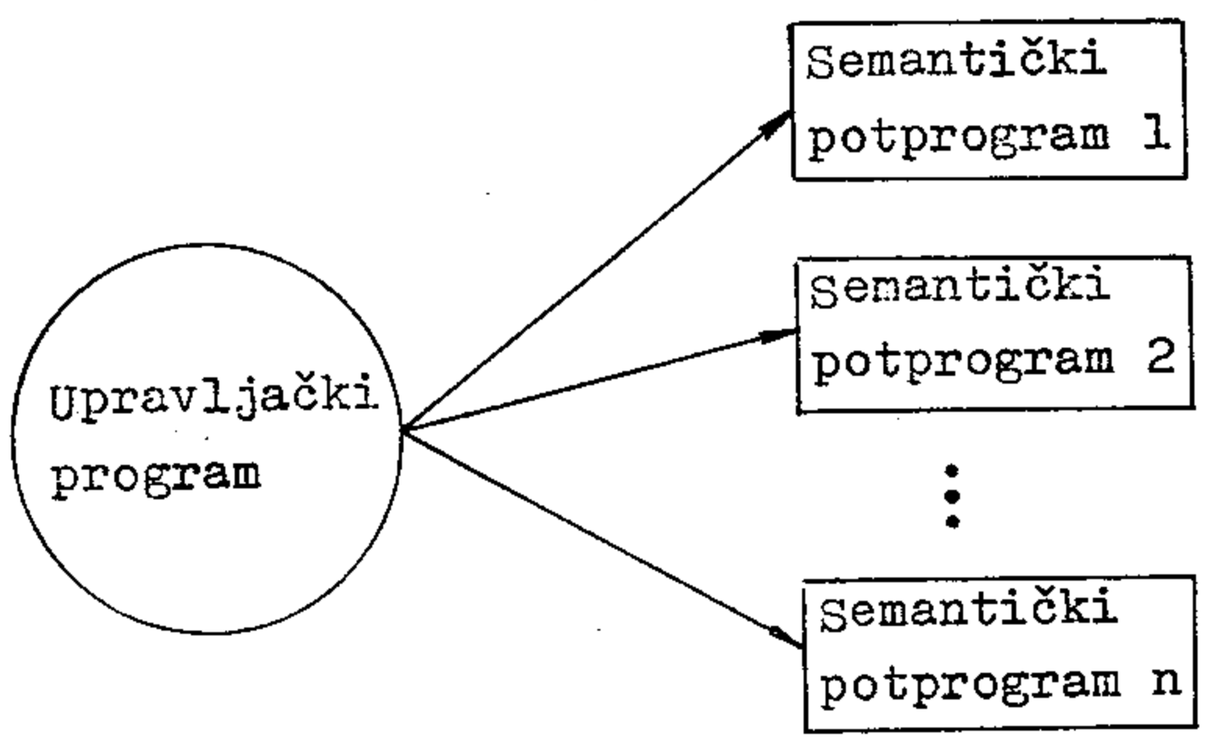
odvojena od semantičke analize. Ovakva metoda prevodjenja naziva se metoda semantičkih potprograma. Ona se često koristi kod najsavremenijih prevodilaca - kompilatora. Na primer, na metodi semantičkih potprograma zasnovan je Kompleks ALGOL-kompilator [115].

Opšta ideja metode semantičkih potprograma sastoji se u podeli programa prevodioca u dve grupe. Prvu grupu obrazuju semantički potprogrami. Svaki semantički potprogram odgovara odredjenoj sintaksnoj konstrukciji ulaznog jezika i služi za obradu takve konstrukcije. Na primer, semantički potprogram naredbe (operatora) dodeljivanja obradjuje naredbe (operatore) dodeljivanja. Slično semantički potprogram naredbe ciklusa obradjuje naredbe ciklusa itd.

Drugi deo prevodioca čini upravljački program koji služi za raspoznavanje i izdvajanje konstrukcija programa na ulaznom jeziku i pozivanje odgovarajućeg semantičkog potprograma. Znači, upravljački program izvršava sintaksnu analizu.

Semantički potprogrami neposredno generišu mašinske naredbe ili izdaju što jednostavnije konstrukcije i pozivaju za njihovu dalju obradu druge potprograme.

Na slici 5.2. prikazana je struktura prevodioca zasnovanog na metodi semantičkih potprograma.



Slika 5.2.

Postoje prevodioci kod kojih je izvršena modifikacija gore izložene ideje. Kod njih ne postoji upravljački program, a

potprogrami po potrebi pozivaju jedan drugog, pri čemu se uvek prvo izvršava potprogram koji odgovara najopštijoj konstrukciji (na primer, bloku u ALGOL-u) ulaznog jezika.

Metoda semantičkih potprograma orijentisana je na specifikaciji sintakse ulaznog jezika. Osnovni nedostatak metode semantičkih potprograma je isti kao direktne metode - velika zavisnost prevodioca od specifičnosti ulaznog jezika. Skup sintaksnih konstrukcija i semantika ulaznog jezika definišu skup semantičkih potprograma i algoritam prevodjenja.

Medjutim, privrženost prevodioca jednom odredjenom programskom jeziku ima i jednu prednost: Prevodilac je obično veoma efikasan što je za odredjene primene od posebnog značaja.

5.3. Univerzalni kompilator

Stremljenja stvaranju metoda prevodjenja, nezavisnih od konkretnog ulaznog jezika inicirala su razradu metoda prevodjenja, orijentisanih na odredjene formalne načine opisa sintakse ulaznog jezika.

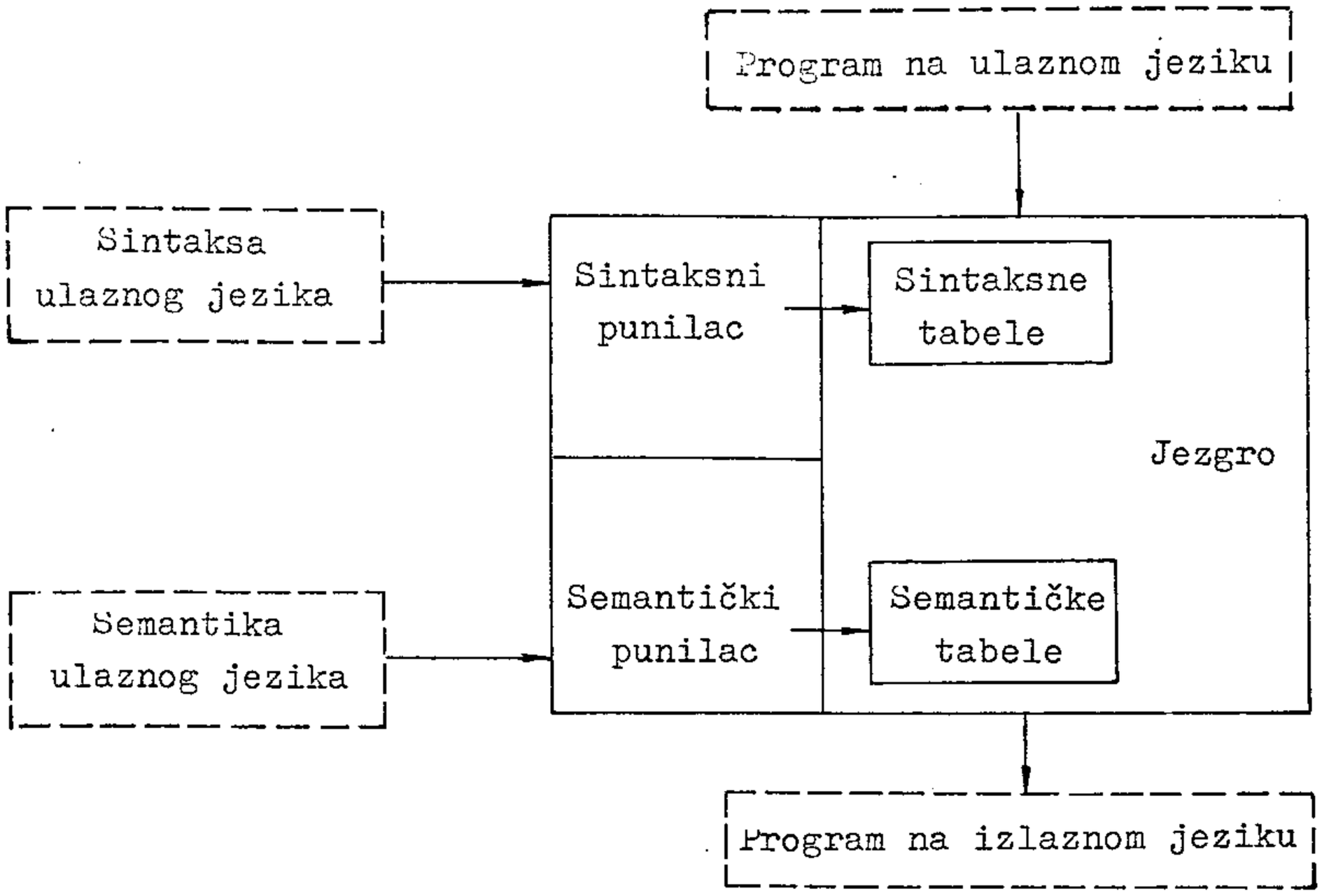
Takve metode prevodjenja nazivaju se sintaksno-orijentisane (vodjene), a prevodioci zasnovani na njima sintaksno-orijentisani (vodjeni).

Svaka metoda sintaksno-orijentisanog prevodjenja primenjiva je za odredjenu klasu ulaznih jezika. Faze sintaksne i semantičke analize mogu se jasno uočiti i razdvojiti. U algoritmima sintaksne i semantičke analize mogu se takodje izdvojiti delovi koji ne zavise od ulaznog jezika. Navedene činjenice u principu dozvoljavaju stvaranje kompilatora, univerzalnog za odredjenu klasu ulaznih jezika. Sintaksa i semantika ulaznog jezika su ulazne veličine (podaci, parametri) univerzalnog kompilatora.

Na slici 5.3. prikazana je struktura jednog hipotetičnog univerzalnog kompilatora.

Hipotetički univerzalni kompilator sastoji se iz:

- sintaksnog punioca,
- semantičkog punioca i
- jezgra.



Slika 5.3.

Sintaksni punilac učitava sintaksu ulaznog jezika i transformiše je u više sintaksnih tabela kao pogodan način unutrašnje reprezentacije sintakse ulaznog jezika.

Semantički punilac na sličan način učitava semantiku ulaznog jezika i transformiše je u više semantičkih tabela.

Jezgro se sastoji iz programa sintaksne i semantičke analize nezavisnih od ulaznog jezika. Programi sintaksne i semantičke analize, koristeći tabličnu reprezentaciju sintakse i semantike datog ulaznog jezika, prevode program na ulaznom jeziku u program na izlaznom jeziku.

Do sad je u svetu napravljeno više sintaksnih punilaca koji za datu sintaksu ulaznog jezika automatski generišu sintaksni analizator, sposoban da izvrši sintaksnu analizu programa na ulaznom jeziku.

Rad na semantičkom puniocu još uvek se nalazi u fazi eksperimentisanja. Osnovni razlog za to je složenost formalnog

opisa semantike jezika. Na rešavanju ovih problema se danas u svetu izuzetno puno radi [59] .

5.4. Programska realizacija univerzalnog kompilatora

Univerzalni kompilator o kojem će dalje biti reči, napisan je na FORTRAN-jeziku, a realizovan je na računaru PDP-11/70. Univerzalni kompilator bilo bi jednostavnije napisati na programskom jeziku PASCAL ili C ali ih u vreme njegove realizacije nije bilo u Beogradu.

Principi konstruisanja i rada univerzalnog kompilatora, sa ulaženjem u detalje samo tamo gde je to najpotrebnije ili najinteresantnije, biće izloženi na primeru jezika uprošćene aritmetičke naredbe. Ništa se suštinski nebi promenilo da je uzet u razmatranje neki daleko složeniji programski jezik.

5.4.1. Sintaksni punilac

Sintaksa ulaznog jezika zadata je modifikovanom Backus--ovom notacijom. Prirodno je da metalingvističke formule Backus-a čine posebnu datoteku. U našem primeru to je datoteka [200,100] PRIM.SIN. Datoteka koja sadrži sintaksu ulaznog jezika obrazuje se pomoću EDIT-ora sa priznakom sadržaja SIN je ulazni podatak sintaksnog punioca.

```
<NAREDBA>::=<PROMENLJIVA>=<IZRAZ>
<IZRAZ>::=<SABIRAK>!+<SABIRAK>!<IZRAZ>+<SABIRAK>
<SABIRAK>::=<CINILAC>!<SABIRAK>*<CINILAC>
<CINILAC>::=<OSNOVA>!<CINILAC>^<OSNOVA>
<OSNOVA>::=<IDENTIFIKATOR>!(<IZRAZ>)
<IDENTIFIKATOR>::=<PROMENLJIVA>!<KONSTANTA>
<PROMENLJIVA>::=<SLOVO>!<PROMENLJIVA><SLOVO>!<PROMENLJIVA><CIFRA>
<KONSTANTA>::=<CEO-BROJ-BEZ-ZNAKA>
<CEO-BROJ-BEZ-ZNAKA>::=<CIFRA>!<CEO-BROJ-BEZ-ZNAKA><CIFRA>
<SLOVO>::=A!B!C!D!E!F!G!H!I!J!K!L!M!N!O!P!Q!R!S!T!U!V!W!X!Y!Z
<CIFRA>::=0!1!2!3!4!5!6!7!8!9
<RED>::=<NAREDBA>;
<RED>
```

Sve rečenice jezika uprošćene aritmetičke naredbe završavaju se specijalnim znakom tačka-zarez (;). To je zato da bi se sasvim pojednostavio postupak otkrivanja kraja rečenice. Tačka-zarez (;) je samo prividno metalingvistička konstanta. Nju ne sme da koristi programer već jedino univerzalni kompilator koji je dopisuje na kraj svake naredbe ulaznog jezika.

Metalingvistička promenljiva <RED> je početni simbol.

5.4.2. Sintaksne tabele

Sintaksni punilac obrazuje tzv. sintaksne tabele u kojima će se čuvati informacije o sintaksi ulaznog jezika. Jedne tabele obrazuju se neposredno, a druge iz više koraka. U cilju uštede memorijskog prostora i smanjenja dimenzija tabela, informacije se kôdiraju prirodnim brojevima.

Osnovne dve tabele su:

- tabela metalingvističkih konstanti i
- tabela metalingvističkih promenljivih.

Konstante i promenljive kodiraju se redosledom pojavljivanja u sintaksi ulaznog jezika. Da bi se u kasnijim obradama promenljive razlikovale od konstanti kôd promenljivih uvećan je za 1000.

Konstante su dužine 1, a promenljive su različite dužine veće ili jednake 3. U cilju ubrzanog poredjenja promenljivih korisno je imati informaciju o njihovoj dužini.

Zbog moguće različite dužine zapisa promenljivih podesnije je tabelu promenljivih realizovati u obliku jednodimenzionalnog, a ne dvodimenzionalnog niza. Na taj način se dobija izvesna ušteda memorijskog prostora.

Iz razloga uniformosti tabela konstanti se takodje realizuje u obliku jednodimenzionalnog niza.

Neophodne informacije za buduća kôdiranja i dekôdiranja čine indeksi članova nizova od kojih počinje nova konstanta, odnosno promenljiva. U navedenim tabelama dotični indeksi označeni su strelicama, a u konkretnoj realizaciji čuvaju se u listi LISTA.

Tabela konstanti

→	1	=
	2	3
→	3	+
	4	6
→	5	*
	6	8
→	7	↑
	8	10
→	9	(
	10	12
→	11	)
	12	13
→	13	A
	14	18
→	15	B
	16	19
→	17	C
	18	20
→	19	D
	20	21
→	21	E
	22	22
→	23	F
	24	23
→	25	G
	26	24
→	27	H
	28	25
→	29	I
	30	26

⋮

Tabela promenljivih

→	1	9
	2	<
	3	N
	4	A
	5	R
	6	E
	7	D
	8	B
	9	A
	10	>
	11	1001
→	12	13
	13	<
	14	P
	15	R
	16	O
	17	M
	18	E
	19	N
	20	L
	21	J
	22	I
	23	V
	24	A
	25	>
	26	1002
→	27	7
	28	<
	29	/
	30	Z

⋮

realizovane u obliku jednodimenzionalnog niza.

```
TABELA KONSTANTI
KONSTANTA-KOD

=- 3
+- 6
*- 8
^-10
(-12
)-13
A-18
B-19
C-20
D-21
E-22
F-23
G-24
H-25
I-26
J-27
K-28
L-29
M-30
N-31
O-32
P-33
Q-34
R-35
S-36
T-37
U-38
V-39
W-40
X-41
Y-42
Z-43
0-44
1-45
2-46
3-47
4-48
5-49
6-50
7-51
8-52
9-53
;-55
```

```
TABELA PROMENLJIVIH
DUZINA-PROMENLJIVA-KOD

 9-<NAREDBA>-1001
13-<PROMENLJIVA>-1002
 7-<IZRAZ>-1004
 9-<SABIRAK>-1005
 9-<CINILAC>-1007
 9-<OSNOVNI>-1009
15-<IDENTIFIKATOR>-1011
11-<KONSTANTA>-1014
 7-<SLOVO>-1015
 7-<CIFRA>-1016
20-<CEO-BROJ-BEZ-ZNAKA>-1017
 5-<RED>-1054
```

izdate u obliku dvodimenzionalnog niza.

Sledeća tabela je tabela kôdiranih formula. Da bi se lakše čitala tabela kodiranih formula uporedo je navedena tabela formula. Tabela kodiranih formula dobija se iz datoteke koja sadrži sintaksu ulaznog jezika. Složene formule rastavljene su na niz odgovarajućih prostih formula. Proste formule kodirane su u skladu sa sadržajem tabela konstanti i promenljivih.

TABELA KODIRANIH FORMULA

1	1001	1002	3	1004	
2	1004	1005			
3	1004	6	1005		
4	1004	1004	6	1005	
5	1005	1007			
6	1005	1005	8	1007	
7	1007	1009			
8	1007	1007	10	1009	
9	1009	1011			
10	1009	12	1004		13
11	1011	1002			
12	1011	1014			
13	1002	1015			
14	1002	1002	1015		
15	1002	1002	1016		
16	1014	1017			
17	1017	1016			
18	1017	1017	1016		
19	1015	18			
20	1015	19			
21	1015	20			
22	1015	21			
23	1015	22			
24	1015	23			
25	1015	24			
26	1015	25			
27	1015	26			
28	1015	27			
29	1015	28			
30	1015	29			
31	1015	30			
32	1015	31			
33	1015	32			
34	1015	33			
35	1015	34			
36	1015	35			
37	1015	36			
38	1015	37			
39	1015	38			
40	1015	39			
41	1015	40			
42	1015	41			
43	1015	42			
44	1015	43			
45	1016	44			
46	1016	45			
47	1016	46			
48	1016	47			
49	1016	48			
50	1016	49			
51	1016	50			
52	1016	51			
53	1016	52			
54	1016	53			
55	1054	1001		55	

TABELA FORMULA

1	<NAREDBA><PROMENLJIVA>=<IZRAZ>
2	<IZRAZ><SABIRAK>
3	<IZRAZ>+<SABIRAK>
4	<IZRAZ><IZRAZ>+<SABIRAK>
5	<SABIRAK><CINILAC>
6	<SABIRAK><SABIRAK>*<CINILAC>
7	<CINILAC><OSNOVA>
8	<CINILAC><CINILAC>^<OSNOVA>
9	<OSNOVA><IDENTIFIKATOR>
10	<OSNOVA>(<IZRAZ>)
11	<IDENTIFIKATOR><PROMENLJIVA>
12	<IDENTIFIKATOR><KONSTANTA>
13	<PROMENLJIVA><SLOVO>
14	<PROMENLJIVA><PROMENLJIVA><SLOVO>
15	<PROMENLJIVA><PROMENLJIVA><CIFRA>
16	<KONSTANTA><CEO-BROJ-BEZ-ZNAKA>
17	<CEO-BROJ-BEZ-ZNAKA><CIFRA>
18	<CEO-BROJ-BEZ-ZNAKA><CEO-BROJ-BEZ-ZNAKA><CIFRA>
19	<SLOVO>A
20	<SLOVO>B
21	<SLOVO>C
22	<SLOVO>D
23	<SLOVO>E
24	<SLOVO>F
25	<SLOVO>G
26	<SLOVO>H
27	<SLOVO>I
28	<SLOVO>J
29	<SLOVO>K
30	<SLOVO>L
31	<SLOVO>M
32	<SLOVO>N
33	<SLOVO>O
34	<SLOVO>P
35	<SLOVO>Q
36	<SLOVO>R
37	<SLOVO>S
38	<SLOVO>T
39	<SLOVO>U
40	<SLOVO>V
41	<SLOVO>W
42	<SLOVO>X
43	<SLOVO>Y
44	<SLOVO>Z
45	<CIFRA>0
46	<CIFRA>1
47	<CIFRA>2
48	<CIFRA>3
49	<CIFRA>4
50	<CIFRA>5
51	<CIFRA>6
52	<CIFRA>7
53	<CIFRA>8
54	<CIFRA>9
55	<RED><NAREDBA>;

Redovi tabele formula su proste formule. Formule su navedene onim redom kojim se javljaju u sintaksi ulaznog jezika. Algoritam sintaksne analize je jednostavniji i brže se izvršava ako se skup formula tabela rastavi na niz tzv. karakterističnih podskupova (podtabela) formula. Karakteristično podskup formula sastoji se iz formula koje imaju jednak prvi simbol desne strane formule tzv. vodeći simbol. Vodeći simbol definiše karakteristično podskup formula. Slično, karakteristična podtabela formula sastoji se iz redova koji imaju jednak treći elemenat.

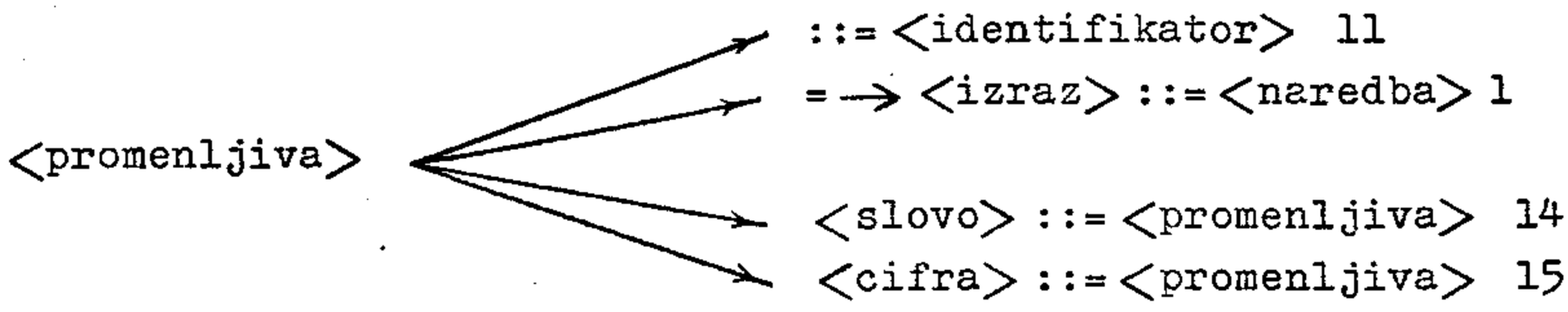
Vodeći simbol <promenljiva> definiše sledeći skup formula:

11 <identifikator> ::= <promenljiva>
1 <naredba> ::= <promenljiva> = <izraz>
14 <promenljiva ::= <promenljiva> <slovo>
15 <promenljiva> ::= <promenljiva> <cifra>

Navedene formule mogu se dalje transformisati na tzv. inverzan oblik. Inverzan oblik se dobija tako što se redom univerzalni simbol ::=, leva strana formule i redni broj formule dopišu na kraj desne strane formule:

<promenljiva> ::= <identifikator> 11
<promenljiva> = <izraz> ::= <naredba> 1
<promenljiva> <slovo> ::= <promenljiva> 14
<promenljiva> <cifra> ::= <promenljiva> 15

odnosno dalje prikazati u obliku drveta čiji je koren vodeći simbol <promenljiva>.



Sortiranjem tabele formula tako da elementi treće kolone (vodeći simboli) obrazuju neopadajući niz tabela formula rastavlja se na niz karakterističnih podtabela formula.

Izvršeno kôdiranje elemenata tabele formula prirodnim brojevima već pri sortoranju tabele formula pokazuje prednost. (Daleko je lakše porediti celobrojne od znakovnih podataka).

TABELA KODIRANIH FORMULA TABELA FORMULA

3	1004	6	1005		3	<IZRAZ>+<SABIRAK>
10	1009	12	1004	13	10	<OSNOVA>(<IZRAZ>)
19	1015	18			19	<SLOVO>A
20	1015	19			20	<SLOVO>B
21	1015	20			21	<SLOVO>C
22	1015	21			22	<SLOVO>D
23	1015	22			23	<SLOVO>E
24	1015	23			24	<SLOVO>F
25	1015	24			25	<SLOVO>G
26	1015	25			26	<SLOVO>H
27	1015	26			27	<SLOVO>I
28	1015	27			28	<SLOVO>J
29	1015	28			29	<SLOVO>K
30	1015	29			30	<SLOVO>L
31	1015	30			31	<SLOVO>M
32	1015	31			32	<SLOVO>N
33	1015	32			33	<SLOVO>O
34	1015	33			34	<SLOVO>P
35	1015	34			35	<SLOVO>Q
36	1015	35			36	<SLOVO>R
37	1015	36			37	<SLOVO>S
38	1015	37			38	<SLOVO>T
39	1015	38			39	<SLOVO>U
40	1015	39			40	<SLOVO>V
41	1015	40			41	<SLOVO>W
42	1015	41			42	<SLOVO>X
43	1015	42			43	<SLOVO>Y
44	1015	43			44	<SLOVO>Z
45	1016	44			45	<CIFRA>0
46	1016	45			46	<CIFRA>1
47	1016	46			47	<CIFRA>2
48	1016	47			48	<CIFRA>3
49	1016	48			49	<CIFRA>4
50	1016	49			50	<CIFRA>5
51	1016	50			51	<CIFRA>6
52	1016	51			52	<CIFRA>7
53	1016	52			53	<CIFRA>8
54	1016	53			54	<CIFRA>9
55	1054	1001	55		55	<RED><NAREDBA>;
11	1011	1002			11	<IDENTIFIKATOR><PROMENLJIVA>
1	1001	1002	3	1004	1	<NAREDBA><PROMENLJIVA>=<IZRAZ>
14	1002	1002	1015		14	<PROMENLJIVA><PROMENLJIVA><SLOVO>
15	1002	1002	1016		15	<PROMENLJIVA><PROMENLJIVA><CIFRA>
4	1004	1004	6	1005	4	<IZRAZ><IZRAZ>+<SABIRAK>
2	1004	1005			2	<IZRAZ><SABIRAK>
6	1005	1005	8	1007	6	<SABIRAK><SABIRAK>*<CINILAC>
5	1005	1007			5	<SABIRAK><CINILAC>
8	1007	1007	10	1009	8	<CINILAC><CINILAC>^<OSNOVA>
7	1007	1009			7	<CINILAC><OSNOVA>
9	1009	1011			9	<OSNOVA><IDENTIFIKATOR>
12	1011	1014			12	<IDENTIFIKATOR><KONSTANTA>
13	1002	1015			13	<PROMENLJIVA><SLOVO>
17	1017	1016			17	<CEO-BROJ-BEZ-ZNAKA><CIFRA>
16	1014	1017			16	<KONSTANTA><CEO-BROJ-BEZ-ZNAKA>
18	1017	1017	1016		18	<CEO-BROJ-BEZ-ZNAKA><CEO-BROJ-BEZ-ZNAKA><CIFRA>

Sortirana tabela formula na neposredan način daje informaciju o karakterističnim podtabelama formula. Medjutim, tabela formula odnosno podtabele formula na posredan način daju informacije o strukturi drveta čiji su koreni vodeći simboli. Drugačije rečeno, medjusobne veze formula unutar tabele formula, odnosno podtabele formula su implicitno odredjene. U cilju što eksplicitnijeg izražavanja medjusobnih veza formula podesno je tabelu formula, odnosno podtabele formula prikazati u obliku četiri liste medjusobno povezane na tri načina (slika 5.4.)

Nazovimo redom navedene liste:

- GLAVA,
- VRAT,
- TELO i
- RUKA.

Lista GLAVA sadrži elemente jezika, tj. metalingvističke konstante i promenljive. Članovi liste navedeni su u rastućem redosledu njihovih kôdova tj. prvo konstante pa onda promenljive.

VRAT sadrži redom broj reda koji sadrži sledeći elemenat formule date u inverznom obliku. Ako je VRAT(i) = Ø onda GLAVA(i) nije vodeći simbol.

TELO sadrži elemente formule date u inverznom obliku koji slede iza vodećeg simbola. Univerzalni simbol ::= je nepotreban pa je izostavljen. Da bi se redni broj proste formule mogao razlikovati od kôda elementa jezika uveden je simbol kraj formule. Simbol kraj formule razdvaja vodeći simbol od rednog broja formule.

Kôd simbola kraj formule je 9999 i različit je od kôda bilo kog elementa jezika.

RUKA sadrži redom broj reda od kojeg počinje alternativni deo formule.

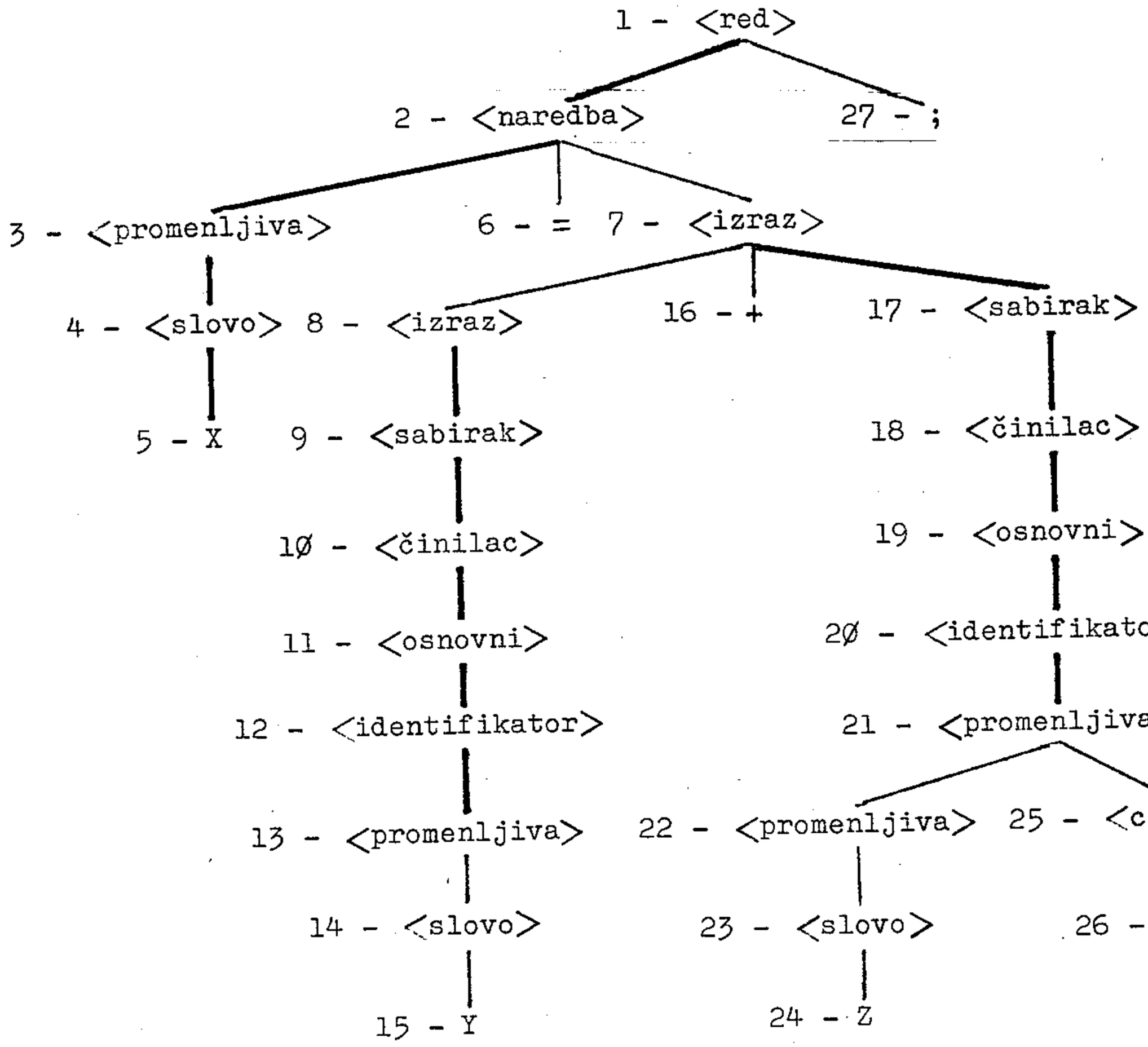
R.broj	GLAVA	VRAT	TELO	RUKA
1	=	ø	<sabirak>	ø
2	+	1	<izraz>	ø
3	*	ø	kraj	ø
4	↑	ø	3	ø
5	(	5	<izraz>	ø
⋮				
43	;	ø	<slovo>	ø
44	<naredba>	118	kraj	ø
45	<promenljiva>	122	3ø	ø
46	<izraz>	138	<slovo>	ø
•				
55	<red>	ø	<slovo>	ø
56			kraj	ø
•				
122			<identifikator>	125
123			kraj	ø
124			11	ø
125			=	13ø
126			<izraz>	ø
127			<naredba>	ø
128			kraj	ø
129			1	ø
130			<slovo>	134
131			<promenljiva>	ø
132			kraj	ø
133			14	ø
134			<cifra>	ø
135			<promenljiva>	ø
136			kraj	ø
137			15	ø
•				
176			16	ø
177			<cifra>	ø
178			<ceo-broj-bez-znaka>	ø
179			kraj	ø
18ø			18	ø

Slika 5.4.

Algoritam sintaksne analize ima zadatak da konstruiše drvo izvodjenja čiji je koren početni simbol, a krošnja ulazna niska.

Niz temena (a_o, a_1,..., a_n), $n \geqslant 1$, drveta izvodjenja naziva se lanac od temena a_o do temena a_n, ako za svako i, $1 \leqslant i \leqslant n$ postoji poteg koji izlazi iz temena a_{i-1} i ulazi u teme a_i i teme a_i je krajnje levi brat.

Na primer:



je drvo izvodjenja niske X=Y+Z9;

Niz temena (1,2,3,4,5) je lanac od temena 1 obeleženog sa <red> do temena 5 obeleženog sa X.

Niz temena (7,17,18,19,20,21) nije lanac. Niz temena (8,9,10,11,12,13) je lanac itd. Obeležja temena prvog i drugog niza temena su ista!

Pri konstruisanju drveta izvodjenja neophodno je znati da li postoji lanac (proizvoljne dužine) izmedju:

a) početnog simbola* i prvog simbola ulazne niske,

b) promenljive i elementa jezika.

Izmedju korena i vodećeg simbola proste formule postoji lanac dužine 1:

(koren, vodeći simbol).

Sintaksa ulaznog jezika, tj. skup metalingvističkih formula definiše relaciju LANAC nad skupom elemenata jezika. Relaciju LANAC podesno je prikazati u obliku kvadratne tabele LANAC čiji su članovi logičke konstante i imaju vrednost T (istina) odnosno F (laž) u zavisnosti od toga da li postoji,odnosno ne postoji lanac izmedju odgovarajućih elemenata jezika.

Kako je koren metalingvističke formule uvek metalingvistička promenljiva to je:

LANAC (metalingvistička konstanta, element jezika)=F

Tabela LANAC je oblika :

	KON.	PRO.
KON.	F	F/T
PRO.	F	F/T

Leva polovina tabele LANAC konstantno sadrži F (laž) - nije od značaja i može se izostaviti.

Kvadratna tabela LANAC redukuje se na pravougaonu tabelu LANAC oblika:

---

* Nema potrebe da se izražavamo sasvim strogo, recimo:"korena drveta obeleženog početnim simbolom ...".

KON.	F/T
PRO.	F/T

Znači, redovi tabele LANAC odgovaraju elementima jezika, tj. konstantama i promenljivima, a kolone tabele LANAC odgovaraju isključivo promenljivima.

Tabela LANAC obrazuje se u dva koraka. Prvi korak: odabrati redom sve proste formule na sledeći način:

- koren odredjuje kolonu tabele,
- vodeći simbol odredjuje red tabele.

U preseku odgovarajuće kolone i reda upisati T. U nepopunjene rubrike tabele upisati F. Drugi korak: Analizirati redom sadržaj svih kolona. Kad god se naidje na vrednost T prepisati sadržaj reda sa oznakom kolone u red u kome je nadjena vrednost T.

Tabela LANAC obrazovana u prvom koraku definiše relaciju LANAC dužine 1 izmedju korena i vodećeg simbola metalingvističkih formula. Tabela LANAC obrazovana u drugom koraku definiše relaciju LANAC proizvoljne dužine izmedju bilo koje metalingvističke promenljive i elementa jezika, tj. tranzitivno zatvaranje prvobitne relacije. Tabela LANAC ništa ne govori o broju lanaca ili koja temena drveta izvodjenja čine lanac.

Lista GLAVA sadrži oznake redova tabela LANAC.

Lista ZAGLAVLJE sadrži oznake kolona tabele LANAC. Na mestu metalingvističkih konstanti postavljene su nule.

Liste GLAVA, ZAGLAVLJE, VRAT i LISTA su iste dužine pa je podesno da se izdaju zajedno. Slično važi za liste TELO i RUKA.

Po pravilu, broj članova tabele LANAC čija je vrednost F, veći je od onih čija je vrednost T. U cilju uštede memorijskog prostora podesno je registrovati pozicije, isključivo članova tabele LANAC čija je vrednost T. Vrednosti članova niza VEKTOR definišu pozicije članova tabele LANAC čija je vrednost T.

Sintaksne tabele obično se popunjavaju postupno. Navodimo najznačajnije korake.

1

GLAVA	ZAGLAVLJE	VRAT	LISTA
1001	0	0	1
1002	0	0	12
3	0	0	1
1004	0	0	27
1005	0	0	36
6	0	0	3
1007	0	0	47
8	0	0	5
1009	0	0	58
10	0	0	7
1011	0	0	69
12	0	0	9
13	0	0	11
1014	0	0	86
1015	0	0	99
1016	0	0	108
1017	0	0	117
18	0	0	13
19	0	0	15
20	0	0	17
21	0	0	19
22	0	0	21
23	0	0	23
24	0	0	25
25	0	0	27
26	0	0	29
27	0	0	31
28	0	0	33
29	0	0	35
30	0	0	37
31	0	0	39
32	0	0	41
33	0	0	43
34	0	0	45
35	0	0	47
36	0	0	49
37	0	0	51
38	0	0	53
39	0	0	55
40	0	0	57
41	0	0	59
42	0	0	61
43	0	0	63
44	0	0	65
45	0	0	67
46	0	0	69
47	0	0	71
48	0	0	73
49	0	0	75
50	0	0	77
51	0	0	79
52	0	0	81
53	0	0	83
1054	0	0	139
55	0	0	85

2

GLAVA	ZAGLAVLJE	VRAT	LISTA
3	0	0	1
6	0	0	3
8	0	0	5
10	0	0	7
12	0	0	9
13	0	0	11
18	0	0	13
19	0	0	15
20	0	0	17
21	0	0	19
22	0	0	21
23	0	0	23
24	0	0	25
25	0	0	27
26	0	0	29
27	0	0	31
28	0	0	33
29	0	0	35
30	0	0	37
31	0	0	39
32	0	0	41
33	0	0	43
34	0	0	45
35	0	0	47
36	0	0	49
37	0	0	51
38	0	0	53
39	0	0	55
40	0	0	57
41	0	0	59
42	0	0	61
43	0	0	63
44	0	0	65
45	0	0	67
46	0	0	69
47	0	0	71
48	0	0	73
49	0	0	75
50	0	0	77
51	0	0	79
52	0	0	81
53	0	0	83
55	0	0	85
1001	0	0	1
1002	0	0	12
1004	0	0	27
1005	0	0	36
1007	0	0	47
1009	0	0	58
1011	0	0	69
1014	0	0	86
1015	0	0	99
1016	0	0	108
1017	0	0	117
1054	0	0	139

3

GLAVA	ZAGLAVLJE	VRAT	LISTA
3	0	0	1
6	0	0	3
8	0	0	5
10	0	0	7
12	0	0	9
13	0	0	11
18	0	0	13
19	0	0	15
20	0	0	17
21	0	0	19
22	0	0	21
23	0	0	23
24	0	0	25
25	0	0	27
26	0	0	29
27	0	0	31
28	0	0	33
29	0	0	35
30	0	0	37
31	0	0	39
32	0	0	41
33	0	0	43
34	0	0	45
35	0	0	47
36	0	0	49
37	0	0	51
38	0	0	53
39	0	0	55
40	0	0	57
41	0	0	59
42	0	0	61
43	0	0	63
44	0	0	65
45	0	0	67
46	0	0	69
47	0	0	71
48	0	0	73
49	0	0	75
50	0	0	77
51	0	0	79
52	0	0	81
53	0	0	83
55	0	0	85
1001	1	0	1
1002	2	0	12
1004	3	0	27
1005	4	0	36
1007	5	0	47
1009	6	0	58
1011	7	0	69
1014	8	0	86
1015	9	0	99
1016	10	0	108
1017	11	0	117
1054	12	0	139

4

GLAVA	ZAGLAVLJE	VRAT	LISTA
3	0	0	1
6	0	1	3
8	0	0	5
10	0	0	7
12	0	5	9
13	0	0	11
18	0	10	13
19	0	13	15
20	0	16	17
21	0	19	19
22	0	22	21
23	0	25	23
24	0	28	25
25	0	31	27
26	0	34	29
27	0	37	31
28	0	40	33
29	0	43	35
30	0	46	37
31	0	49	39
32	0	52	41
33	0	55	43
34	0	58	45
35	0	61	47
36	0	64	49
37	0	67	51
38	0	70	53
39	0	73	55
40	0	76	57
41	0	79	59
42	0	82	61
43	0	85	63
44	0	88	65
45	0	91	67
46	0	94	69
47	0	97	71
48	0	100	73
49	0	103	75
50	0	106	77
51	0	109	79
52	0	112	81
53	0	115	83
55	0	0	85
1001	1	118	1
1002	2	122	12
1004	3	138	27
1005	4	143	36
1007	5	151	47
1009	6	159	58
1011	7	162	69
1014	8	165	86
1015	9	168	99
1016	10	171	108
1017	11	174	117
1054	12	0	139

RB	TELO	RUKA
1	1005	0
2	1004	0
3	9999	0
4	3	0
5	1004	0
6	13	0
7	1009	0
8	9999	0
9	10	0
10	1015	0
11	9999	0
12	19	0
13	1015	0
14	9999	0
15	20	0
16	1015	0
17	9999	0
18	21	0
19	1015	0
20	9999	0
21	22	0
22	1015	0
23	9999	0
24	23	0
25	1015	0
26	9999	0
27	24	0
28	1015	0
29	9999	0
30	25	0
31	1015	0
32	9999	0
33	26	0
34	1015	0
35	9999	0
36	27	0
37	1015	0
38	9999	0
39	28	0
40	1015	0
41	9999	0
42	29	0
43	1015	0
44	9999	0
45	30	0
46	1015	0
47	9999	0
48	31	0
49	1015	0
50	9999	0
51	32	0
52	1015	0
53	9999	0
54	33	0
55	1015	0
56	9999	0
57	34	0
58	1015	0
59	9999	0
60	35	0
61	1015	0
62	9999	0
63	36	0
64	1015	0
65	9999	0
66	37	0
67	1015	0
68	9999	0
69	38	0
70	1015	0
71	9999	0
72	39	0
73	1015	0
74	9999	0
75	40	0
76	1015	0
77	9999	0
78	41	0
79	1015	0
80	9999	0
81	42	0
82	1015	0
83	9999	0
84	43	0
85	1015	0
86	9999	0
87	44	0
88	1016	0
89	9999	0
90	45	0

91	1016	0	136	9999	0
92	9999	0	137	15	0
93	46	0	138	6	0
94	1016	0	139	1005	0
95	9999	0	140	1004	0
96	47	0	141	9999	0
97	1016	0	142	4	0
98	9999	0	143	1004	146
99	48	0	144	9999	0
100	1016	0	145	2	0
101	9999	0	146	8	0
102	49	0	147	1007	0
103	1016	0	148	1005	0
104	9999	0	149	9999	0
105	50	0	150	6	0
106	1016	0	151	1005	154
107	9999	0	152	9999	0
108	51	0	153	5	0
109	1016	0	154	10	0
110	9999	0	155	1009	0
111	52	0	156	1007	0
112	1016	0	157	9999	0
113	9999	0	158	8	0
114	53	0	159	1007	0
115	1016	0	160	9999	0
116	9999	0	161	7	0
117	54	0	162	1009	0
118	55	0	163	9999	0
119	1054	0	164	9	0
120	9999	0	165	1011	0
121	55	0	166	9999	0
122	1011	125	167	12	0
123	9999	0	168	1002	0
124	11	0	169	9999	0
125	3	130	170	13	0
126	1004	0	171	1017	0
127	1001	0	172	9999	0
128	9999	0	173	17	0
129	1	0	174	1014	177
130	1015	134	175	9999	0
131	1002	0	176	16	0
132	9999	0	177	1016	0
133	14	0	178	1017	0
134	1016	0	179	9999	0
135	1002	0	180	18	0

POCETNI SIMBOL
1054

	LANAC	LANAC
=	FFFFFFFFFFFF	FFFFFFFFFFFF
+	FFTFFFFFFFFF	FFTFFFFFFFFF
*	FFFFFFFFFFFF	FFFFFFFFFFFF
^	FFFFFFFFFFFF	FFFFFFFFFFFF
(	FFFFFTFFFFFF	FFTTTTFFFFFF
)	FFFFFFFFFFFF	FFFFFFFFFFFF
A	FFFFFFFFTFFF	TTTTTTTFTFFT
B	FFFFFFFFTFFF	TTTTTTTFTFFT
C	FFFFFFFFTFFF	TTTTTTTFTFFT
D	FFFFFFFFTFFF	TTTTTTTFTFFT
E	FFFFFFFFTFFF	TTTTTTTFTFFT
F	FFFFFFFFTFFF	TTTTTTTFTFFT
G	FFFFFFFFTFFF	TTTTTTTFTFFT
H	FFFFFFFFTFFF	TTTTTTTFTFFT
I	FFFFFFFFTFFF	TTTTTTTFTFFT
J	FFFFFFFFTFFF	TTTTTTTFTFFT
K	FFFFFFFFTFFF	TTTTTTTFTFFT
L	FFFFFFFFTFFF	TTTTTTTFTFFT
M	FFFFFFFFTFFF	TTTTTTTFTFFT
N	FFFFFFFFTFFF	TTTTTTTFTFFT
O	FFFFFFFFTFFF	TTTTTTTFTFFT
P	FFFFFFFFTFFF	TTTTTTTFTFFT
Q	FFFFFFFFTFFF	TTTTTTTFTFFT
R	FFFFFFFFTFFF	TTTTTTTFTFFT
S	FFFFFFFFTFFF	TTTTTTTFTFFT
T	FFFFFFFFTFFF	TTTTTTTFTFFT
U	FFFFFFFFTFFF	TTTTTTTFTFFT
V	FFFFFFFFTFFF	TTTTTTTFTFFT
W	FFFFFFFFTFFF	TTTTTTTFTFFT
X	FFFFFFFFTFFF	TTTTTTTFTFFT
Y	FFFFFFFFTFFF	TTTTTTTFTFFT
Z	FFFFFFFFTFFF	TTTTTTTFTFFT
0	FFFFFFFFFTFF	FFTTTTTTFTTF
1	FFFFFFFFFTFF	FFTTTTTTFTTF
2	FFFFFFFFFTFF	FFTTTTTTFTTF
3	FFFFFFFFFTFF	FFTTTTTTFTTF
4	FFFFFFFFFTFF	FFTTTTTTFTTF
5	FFFFFFFFFTFF	FFTTTTTTFTTF
6	FFFFFFFFFTFF	FFTTTTTTFTTF
7	FFFFFFFFFTFF	FFTTTTTTFTTF
8	FFFFFFFFFTFF	FFTTTTTTFTTF
9	FFFFFFFFFTFF	FFTTTTTTFTTF
;	FFFFFFFFFFFF	FFFFFFFFFFFF
<NAREDBA>	FFFFFFFFFFFT	FFFFFFFFFFFT
<PROMENLJIVA>	TTFFFFTFFFFF	TTTTTTTFFFFT
<IZRAZ>	FFTFFFFFFFFF	FFTFFFFFFFFF
<SABIRAK>	FFTTFFFFFFFF	FFTTFFFFFFFF
<CINILAC>	FFFTTFFFFFFF	FFTTTFFFFFFF
<OSNOVA>	FFFFTFFFFFFF	FFTTTFFFFFFF
<IDENTIFIKATOR>	FFFFFTFFFFFF	FFTTTTFFFFFF
<KONSTANTA>	FFFFFFTFFFFF	FFTTTTTFFFFF
<SLOVO>	FTFFFFFFFFFF	TTTTTTTFFFFT
<CIFRA>	FFFFFFFFFFTF	FFTTTTTTFFTF
<CEO-BROJ-BEZ-ZNAKA>	FFFFFFFTFFTF	FFTTTTTTFFTF
<RED>	FFFFFFFFFFFF	FFFFFFFFFFFF

VEKTOR

```
 203  503  504  505  506  701  702  703  704  705  706  707
 709  712  801  802  803  804  805  806  807  809  812  901
 902  903  904  905  906  907  909  912 1001 1002 1003 1004
1005 1006 1007 1009 1012 1101 1102 1103 1104 1105 1106 1107
1109 1112 1201 1202 1203 1204 1205 1206 1207 1209 1212 1301
1302 1303 1304 1305 1306 1307 1309 1312 1401 1402 1403 1404
1405 1406 1407 1409 1412 1501 1502 1503 1504 1505 1506 1507
1509 1512 1601 1602 1603 1604 1605 1606 1607 1609 1612 1701
1702 1703 1704 1705 1706 1707 1709 1712 1801 1802 1803 1804
1805 1806 1807 1809 1812 1901 1902 1903 1904 1905 1906 1907
1909 1912 2001 2002 2003 2004 2005 2006 2007 2009 2012 2101
2102 2103 2104 2105 2106 2107 2109 2112 2201 2202 2203 2204
2205 2206 2207 2209 2212 2301 2302 2303 2304 2305 2306 2307
2309 2312 2401 2402 2403 2404 2405 2406 2407 2409 2412 2501
2502 2503 2504 2505 2506 2507 2509 2512 2601 2602 2603 2604
2605 2606 2607 2609 2612 2701 2702 2703 2704 2705 2706 2707
2709 2712 2801 2802 2803 2804 2805 2806 2807 2809 2812 2901
2902 2903 2904 2905 2906 2907 2909 2912 3001 3002 3003 3004
3005 3006 3007 3009 3012 3101 3102 3103 3104 3105 3106 3107
3109 3112 3201 3202 3203 3204 3205 3206 3207 3209 3212 3303
3304 3305 3306 3307 3308 3310 3311 3403 3404 3405 3406 3407
3408 3410 3411 3503 3504 3505 3506 3507 3508 3510 3511 3603
3604 3605 3606 3607 3608 3610 3611 3703 3704 3705 3706 3707
3708 3710 3711 3803 3804 3805 3806 3807 3808 3810 3811 3903
3904 3905 3906 3907 3908 3910 3911 4003 4004 4005 4006 4007
4008 4010 4011 4103 4104 4105 4106 4107 4108 4110 4111 4203
4204 4205 4206 4207 4208 4210 4211 4412 4501 4502 4503 4504
4505 4506 4507 4512 4603 4703 4704 4803 4804 4805 4903 4904
4905 5003 5004 5005 5006 5103 5104 5105 5106 5107 5201 5202
5203 5204 5205 5206 5207 5212 5303 5304 5305 5306 5307 5308
5311 5403 5404 5405 5406 5407 5408 5411
```

5.4.3. Semantički punilac

Semantika ulaznog jezika zadaje se tzv. tekstualnom* (deskriptivnom) notacijom. Ovu notaciju izmislio je Irons [51]. U ovom radu koristi se notacija koja predstavlja poboljšanje izvorne Irons-ove notacije.

Neka kontekstno-slobodna gramatika $G=(N,\Sigma,P,S)$ definiše ulazni jezik.

Svakom pravilu izvodjenja

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

pridružuje se tekst β koji predstavlja semantiku (značenje) pravila izvodjenja $A \rightarrow \alpha$.

Ne postoji pravilo definisanja teksta β. Drugačije rečeno, šta će biti tekst β zavisi od:

---

* U literaturi se još nije ustalio neki standardni naziv.

- pravila izvodjenja $A \rightarrow \alpha$ ili još tačnije od niske α
- izlaznog jezika.

Postoje pravila pisanja teksta β i ona zavise od realizacije semantičkog punioca. Za ovu realizaciju važe sledeća pravila pisanja:

- tekst β završava se priznakom kraj teksta (!) koji ulazi u sastav teksta,
- složenom pravilu izloženja $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ odgovara niz tekstova: $\beta_1, \beta_2, \dots, \beta_n$.
- jedan tekst može se pisati u više redova,
- više tekstova može se pisati u jednom redu,
- reči (niske simbola) izlaznog jezika pišu se pod znacima navoda,
- nezavršnim simbolima A_1, A_2, A_3, ... niske α redom odgovaraju priznaci G1, G2, G3, ... Završni simboli niske α indirektno se uzimaju u obzir,
- priznaci T1, T2 i T3 redom služe za prelazak na novi i poziciranje u okviru tekućeg reda,
- priznak S_n služi za dodeljivanje lokalnog simbola n koji može da bude radni (privremeni) memorijski registar ili obeležje (simbolička adresa),
- priznak P_n služi za upis u stog lokalnog simbola n
- priznak U služi za čitanje lokalnog simbola svrha stoga.

Navedeni priznaci zadovoljavaju potrebe definisanja semantike pravila izvodjenja. Bes teškoće mogu se uvesti novi priznaci s raznim drugim mogućnostima.

Tekstovi, koji predstavljaju semantiku pravila izvodjenja, navedeni istim redosledom kao odgovarajuća pravila izvodjenja predstavljaju semantiku ulaznog jezika. Prirodno je da niz tekstova čini posebnu datoteku. U našem primeru to je datoteka [202,100]PRIM.SEM Datoteke čiji je sadržaj semantika ulaznog jezika obrazuju se pomoću EDIT-ora sa priznakom sadržaja SEM ulazni su podatak semantičkog punioca.

Semantika ulaznog jezika [202,100]PRIM.SEM je:

```
G2T2'AUM'T3G1!
G1!G1!G2T2'AUM'T3S1G1T2'SAB'T3S1!
G1!G2T2'AUM'T3S1G1T2'MNO'T3S1!
G1!G2T2'AUM'T3S1G1T2'STEPEN'T3S1!
T2'MUA'T3G1!G1!
G1!G1!
G1!G1G2!G1G2!
G1!
G1!G1G2!
'A'!'B'!'C'!'D'!'E'!'F'!'G'!'H'!
'I'!'J'!'K'!'L'!'M'!'N'!
'O'!'P'!'Q'!'R'!'S'!'T'!'U'!'V'!'W'!'X'!'Y'!'Z'!
'0'!'1'!'2'!'3'!'4'!'5'!'6'!'7'!'8'!'9'!
G1!
```

5.4.4. Semantičke tabele

Semantički punilac obrazuje tzv. semantičke tabele u kojima će se čuvati informacije o semantici ulaznog jezika.

Tekstovi koji predstavljaju semantiku pravila izvodjenja čuvaju se u tabeli tekstova. U cilju uštede memorijskog prostora tabela tekstova se realizuje u obliku jednodimenzionalnog niza SEMPRA.

Članovi niza SEMPRA su redom slova tekstova.

SEMPRA: 12345678901234567...

G2T2'AUM'T3G1!G1!G1!G2T2'AUM'T3S1G1...

Vreme pristupa odredjenom tekstu može se smanjiti ako se upamte vrednosti inteksa članova niza SEMPRA koji sadrže prvo slovo tekstova. Vrednosti indeksa čuvaju se u nizu SPISAK

SPISAK:	1	2	3	4	5	6
	1	15	18	21	...	...

5.4.5. Jezgro

Jezgro univerzalnog kompilatora sastoji se iz programa:

- sintaksne analize,
- semantičke analize (generisanja kôda).

Navedeni programi ne zavise od ulaznog jezika. Sintaksa i semantika ulaznog jezika su ulazni podaci programa sintaksne, odnosno semantičke analize. Do današnjeg dana izmišljeno je niz algoritama (metoda) sintaksno-orijentisanog (vodjenog) prevodjenja. Svaki od njih pošavši od sintakse ulaznog jezika pokušava za datu nisku ulaznih simbola da konstruiše drvo izvodjenja.

U slučaju uspešnog konstruisana drveta izvodjenja niska ulaznih simbola je sintaksno ispravna, suprotno niska ulaznih simbola je leksički ili sintaksno neispravna. Rezultat sintaksne analize je:

- drvo izvodjenja ili
- izveštaj o leksičkoj, odnosno sintaksnoj grešci.

Postoji niz mogućih reprezentacija drveta izvodjenja što je inače predmet proučavanja teorije strukture podataka i teorije grafova.

Algoritam semantičke analize (generisanja kôda) pošavši od semantike ulaznog jezika za prethodno dobijeno drvo izvodjenja daje prevod ulazne niske. Prevod može da bude niska na izlaznom jeziku ili niska na odredjenom medjujeziku. U slučaju da se želi izvršiti optimizacija prevoda podesnije je da prevod bude na medjujeziku.

Moguće je da sintaksno ispravna ulazna niska sadrži semantičke greške. Rezultat semantičke analize je:

- prevod na izlaznom (medju) jeziku ili
- izveštaj o semantičkoj grešci.

Dobre strane opisane organizacije jezgra kompilatora su:

- medjusobna nezavisnost algoritama sintaksne i semantičke analize;
- nezavisnost algoritma sintaksne i semantičke analize od specifičnosti konkretnog ulaznog jezika;
- opštost, tj. mogućnost korišćenja raznih reprezentacija, algoritama analize bez velikih izmena u programu;

Pri realizaciji jezgra kompilatora korišćeni su:

- za sintaksnu analizu Irons-ov algoritam;
- za semantičku analizu modifikovan Irons-ov algoritam; prilagodjen specifičnostima tekstova za definisanje semantike pravila izvodjenja;
- za reprezentaciju drveta izvodjenja tzv. linearna reprezentacija.

Navedene algoritme i reprezentaciju drveta nećemo razmatrati zbog dužine.

Izlazni jezik je mašinski-orijentisan jezik nastavnog računara.

Linerarna reprezentacija drveta izvodjenja niske:

X=Y+Z9;

je:

```
 1      54
 2      55
 3       7
 4       9
 5      11
 6      15
 7       1
 8      13
 9      44
10       4
11      -2
12       2
13      56
14       7
15       9
16      11
17      13
18      43
19      55
20       1
21     -10
22      12
23      42
24     -19
```

Prevod na mašinski-orijentisan jezik nastavnog računara naredbe:

```
X=Y+Z9
```

proceduralno-orijentisanog jezika je:

```
MUA     Z9
AUM     0031
MUA     Y
SAB     0031
AUM     X
```

Primeri:

```
X=Y-Z9
 - NIJE ZAVRSAN SIMBOL
X=12.
 . NIJE ZAVRSAN SIMBOL
X=+Y+Z9
MUA     Z9
AUM     0031
MUA     Y
SAB     0031
AUM     X
X=++Y
 SINTAKSNA GRESKA
1=A
SINTAKSNA GRESKA
A=-(-A)
- NIJE ZAVRSAN SIMBOL
```

```
ZET=IKS+IPSILON
        1      23
        2      39
        3      29
        4      37
        5      34
        6      37
        7      27
        8      30
        9      33
       10      32
       11       5
       12       7
       13       9
       14      11
       15      14
       16     -10
       17      14
       18      -9
       19      14
       20      -8
       21      14
       22      -7
       23      14
       24      -6
       25      14
       26      -5
       27      13
       28      27
       29       4
       30     -11
       31       2
       32       5
       33       7
       34       9
       35      11
       36      14
       37      -4
       38      14
       39      -3
       40      13
       41      27
       42      55
       43       1
       44     -29
       45      14
       46      -2
       47      14
       48      -1
       49      13
       50      44
       51     -42
              MUA         IPSILON
              AUM         0031
              MUA         IKS
              SAB         0031
              AUM         ZET
```

```
A+A
        1       5
        2       7
        3       9
        4      11
        5      13
        6      19
        7       3
        8      -1
        9      55
       10       1
       11      -7
       12      13
       13      19
       14      -9
              MUA         A
              AUM         A
```

```
A=(((A)))
        1       2
        2       5
        3       7
        4       9
        5      11
        6      13
        7      19
        8       2
        9       5
       10       7
       11      10
       12      -1
       13       2
       14       5
       15       7
       16      10
       17      -8
       18       2
       19       5
       20       7
       21      10
       22     -13
       23      55
       24       1
       25     -19
       26      13
       27      19
       28     -23
              MUA         A
              AUM         A
```

6. ISTORIJSKE NAPOMENE O PROGRAMSKIM JEZICIMA I PREVODIOCIMA

Reč algoritam vodi poreklo od imena uzbeskog matematičara alj-Horezmija koji je živeo oko 820. godine. Alj-Horezmi je doprineo širenju, u zapadnoj Evropi, indijskog načina zapisa brojeva pomoću arapskih cifara i računanju sa njima. Leibnitz je koristio reč algoritam u smislu "postupak izračunavanja zbira dva broja". Vremenom je pojam algoritam primao sve šira i opštija značenja do današnjeg "uredjen skup pravila formulisanih u cilju rešavanja odredjenog zadatka".

Prekretnica u teoriji algoritama nastaje uvodjenjem formalizacije, tj. napuštanjem prirodnih jezika i prelaskom na formalne jezike za opisivanje algoritama. Programski jezici pripadaju klasi formalnih jezika koji služe za opisivanje podataka i obrade podataka na računaru. Teorijsku osnovu programskih jezika čine algoritamski sistemi /rekurzivne funkcije, Turing-ova i Post-ova mašina, normalni algoritmi Markov-a itd./ namenjeni za opisivanje algoritama definisanih kao konstruktivno zadata preslikavanja reči apstraktnih azbuka. Razvoju algoritamskih sistema svojim rezultatima su puno doprineli: Frege, Russell, Gödel, Turing, Post, Kleene, Church i Markov. Za razliku od algoritamskih sistema, programski jezici su orijentisani na opisivanje procesa koji se sastoje iz konkretnih operacija računara i zavise od režima rada računara /naprimer: ulaz i izlaz podataka, rad u razdeljenom vremenu itd./.

Na inicijativu Rutishauser-a 1951. godine počinje se sa koncipiranjem i razradom algoritamskih jezika. Naziv algoritamski jezici je tek 1958. godine uveo Botenbruch u cilju njihove primene kao programskih jezika. U početku, glavna pažnja je posvećena notaciji programskih jezika. Jedan od prvih načina pisanja programskih jezika dao je Ljapunov 1956. godine. Kasnije su u prvi plan izbili problemi vezani za semantiku programskih jezika. Opšta pitanja semantike programskih jezika razmatrali su: Mac'Carti, Lendin i Floyd.

Wijngaarden-ovi radovi iz operativne semantike inicirali su nastanak programskog jezika ALGOL-68. Dijkstra je radovima iz teorije transfomacija predikata postavio osnove matematičkog opisa semantike. Strecly i Scott uveli su i razmatrali su denotacionu semantiku koja se recimo može koristiti za definisanje semantike programskog jezika LISP.

6.1. FORTRAN

Programski jezik FORTRAN (FORmula TRANslation) je najstariji proceduralno-orijentisan programski jezik. Prvo zvanično saopštenje o programskom jeziku FORTRAN datira iz 1957. godine [7]. Medjutim, koncipiranje i realizacija FORTRAN-jezika izvršena je znatno ranije. Prvi FORTRAN-kompilator konstruisan je 1954. godine za računar IBM-704. Sa razvojem i primenom računarske tehnike uporedo se razvijao i FORTRAN-jezik. Tako su redom nastajale verzije:

- 1958. FORTRAN-II,
- 1961. FORTRAN-III,
- 1962. FORTRAN-IV,
- 1969. FORTRAN-basic,
- 1966. FORTRAN-standard [4].

FORTRAN-jezik se koristi za rešavanje najraznovrsnijih zadataka ali prvenstveno naučno-tehničke prirode. Osnovni cilj konstruktora FORTRAN-jezika je bio što efikasnije izvršavanje programa što je uglavnom i postignuto. Razlog za ovo je sledeći: u to vreme isključivo se programiralo na simboličkim jezicima, viši programski jezici nisu postojali i skoro ništa se nije znalo o metodama kompilacije. Zato su, prvi pokušaji prevodjenja matematičkih formula davali rogobatne prevode koji su bili višestruko gori - neefikasniji od onih koji bi se dobili neposrednim programiranjem matematičkih formula na simboličkom jeziku. Mnogi kratkovidi stručnjaci kritikovali su i sprečavali pokušaj razvoja viših programskih jezika smatrajući da je to nemoguće, nema smisla itd. Tek kad je J. Backus-u pošlo za rukom da napravi efikasan kompilator - počela je era viših programskih jezika.

Struktura FORTRAN-jezika je jednostavna. Danas, gotovo da nema računara, na kojem nije realizovan FORTRAN-kompilator. Efikasnost izvršavanja FORTRAN-programa je izuzetno velika, zah-

valjujući maksimalnom korišćenju svih tehničkih mogućnosti računara. Medjutim, to se negativno odražava na definicije pojedinih konstrukcija FORTRAN-jezika.

6.2. ALGOL

Definicija programskog jezika ALGOL-60 [76] je izuzetno važan dogadjaj u istoriji programskih jezika. Do sada, ni jedan drugi programski jezik nije imao većeg uticaja na konstrukciju,razvoj i realizaciju budućih programskih jezika. Većina istraživanja u oblasti programskih jezika narednih 10 godina polazila su od ALGOL-60-jezika. Interesantno je primetiti da se programski jezik ALGOL i pored velikog teorijskog značaja relativno retko koristi u svetu (izuzev u SSSR i zemljama socijalističke zajednice) za programiranje.

ALGOL-jezik je koncipirao i realizovao jedan medjunarodni komitet u periodu kraj 50-tih - početak 60-tih godina ovog veka. 1963. godine P. Naur (kao izdavač) objavio je rad "Revised Report on the Algorithmic Language ALGOL-60" u časopisu Comm. ACM,6,1,1-17 koji danas predstavlja klasičan dokument literature iz programskih jezika.

Najvažnije osobine ALGOL-jezika su:

- jednostavnost i elegantnost konstrukcija i načina njihovog definisanja,
- upravljačke naredbe.

ALGOL-jezik je uticao na razvoj programskih jezika:

- PL/I,
- FORTRAN-a (kasnije verzije),
- PASCAL-a,
- BLISS-a itd.

Krajem 60-tih godina drugi komitet pod pokroviteljstvom medjunarodne organizacije IFIP predložio je naslednika programskog jezika ALGOL-60, zapravo programski jezik ALGOL-68 [114].

U slučaju ALGOL-jezika prvi put je upotrebljena jedna formalna gramatika za definisanje sintakse jednog programskog jezika. NFB-gramatika, izložena u radu: "The Syntax and Semantics

of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference" Backus-a objavljenom u zborniku Information Processing, UNESCO, Paris, 125-132 je prvi pokušaj formalnog definisanja sintakse jezika. U ovom radu, takodje se pokušava dati opis semantike jezika ali nažalost bez uspeha.

Važnost ALGOL-jezika ogleda se u sledećem: zadnjih dvadeset godina ALGOL-jezik je osnovan jezik na kojem se objavljuju algoritmi u najpoznatijem svetskom časopisu iz oblasti informatike: Communications of the ACM. U istom časopisu je tek 1968. godine objavljen prvi zapis jednog algoritma na FORTRAN-jeziku.

6.3. BNF - Backus-ova normalna forma ili Backus-Naur-ova forma

Najpoznatija i najčešće korišćena notacija za zapis pravila izvodjenja kontekstno-slobodnih gramatika je Backus-ova notacija odnosno kako se još naziva Backus-ova normalna forma. Backus je ovu "tehniku" prvi put izložio juna 1959. godine u Parizu na jednoj konferenciji posvećenoj stvaranju programskog jezika ALGOL-60.

Backus je bio jedan od tvoraca programskog jezika ALGOL 60 koji se u vreme održavanja konferencije nalazio u fazi "doterivanja". Peter Naur preradio je uvodni izveštaj Backus-a uključujući u njega primedbe i sugestije izrečene na konferenciji. Tako dobijeni izveštaj štampan je pod naslovom ALGOL-60.

1962. godine održana je u Rimu konferencija na kojoj su ispravljene uočene greške i nejednoznačnosti u izveštaju ALGOL-60. Dalja briga za održavanje i razvoj programskih jezika poverena je radnoj grupi WG2.1 koja deluje u okviru medjunarodne federacije za obradu informacija (IFIP). Ova grupa je obrazovana od najpoznatijih svetskih stručnjaka iz računarstva sa zadatkom da prate razvoj programskih jezika.

Programski jezik ALGOL-60 inicirao je uvodjenje jednostavnih, a tako moćnih gramatičkih formalizama u teoriju i praksu programskih jezika. Johan Backus i Peter Naur su najzaslužniji za realizovanje ove ideje koja je krajem 50-tih godina bila tako strana i nejasna, a danas tako prirodna i očigledna.

Donald Knuth je predložio da se naziv Backus-ova normalna forma promeni u Backus-Naur-ova forma zbog nesumnjivih Naur-ovih doprinosa. Mnogi ljudi su ovaj predlog prihvatili tako da su danas u upotrebi oba ova naziva za jedan isti gramatički formalizam. Pojedini autori za ovu notaciju koriste skraćenicu BNF. Primetimo na kraju da je ironijom sudbine ispalo da sama skraćenica gramatičkog formalizma (BNF) izmišljenog u cilju povećanja preciznosti i strogosti definisanja programskih jezika, sadrži izvesnu dozu nejednoznačnosti!

6.4. PASCAL

Preliminarna verzija programskog jezika PASCAL data je 1968. godine. Ona je sastavljena po uzoru na programske jezike ALGOL-60 i ALGOL-W. Nakon toga se pristupilo intenzivnoj izradi PASCAL-kompilatora koji je bio gotov dve godine kasnije. Prve radove o konstrukciji i realizaciji PASCAL-jezika objavio je N.Wirth [94] 1971. godine. Izuzetno veliko interesovanje za konstruisanje PASCAL-kompilatora za razne tipove računara zahtevalo je izvesnu reviziju PASCAL-jezika koja je i izvršena 1973. godine.

6.5. BASIC

Programski jezik BASIC (Beginner's All-purpose Symbolic Instruction Code) koristi se prvenstveno za interaktivni rad sa računarom u sistemu sa razdeljenim vremenom.

Grupa saradnika i stručnjaka Dortmouth koledža pod rukovodstvom profesora J.G. Kemeny-a [55] i T.E. Kurtz-a, 1965. godine konstruisala je i razvila BASIC-jezik za potrebe firme General Elictric.

Prvi BASIC-jezik realizovan je za računare DATANET-30 i GE-235.

1967. godine proširena verzija BASIC-jezika realizovana je za računare GE-400 i GE-635.

Nakon firme General Electric, BASIC-jezik prihvatile

su i druge firme za proizvodnju računara.

Korporacija DEC proizvela je razne verzije BASIC-jezika za računare serije PDP-8, PDP-10, PDP-11, PDP-20 itd.

Slično, firma Hewlett-Packard koja uglavnom proizvodi mini-računare koristi BASIC-jezik na računarima HP-2114, HP-2115, HP-2116B itd.

6.6. COBOL

COBOL (Common Business Oriented Language) je jezik koji se od početka 60-tih godina široko koristi u poslovnoj obradi podataka. Kao i većina drugih programskih jezika COBOL-jezik je prošao niz etapa u svom razvoju. Prva verzija jezika objavljena je 1960. godine, a poslednja verzija 1972. godine.

Po broju realizacija COBOL-jezik zauzima prvo mesto. Medjutim, koncepcija COBOL-jezika, gotovo da nije uticala na dalji razvoj programskih jezika (sa izuzetkom PL/I-jezika). Ova činjenica može se objasniti usmerenošću COBOL-jezika na poslovnu obradu podataka. Poslovna obrada podataka je izuzetno značajna oblast primene računara, koja se odlikuje relativno jednostavnim algoritmima i velikim brojem ulaznih i izlaznih podataka. Medjutim, većina drugih oblasti primene računara upravo imaju suprotne osobine: relativno složeni algoritmi, a mali broj ulaznih i izlaznih podataka. Sada su jasni razlozi malog uticaja COBOL-jezika na razvoj drugih programskih jezika i uopšte teorije programskih jezika i programiranje.

Najvažnija stvar u poslovnoj obradi podataka je organizacija ulaza podataka i izdavanje rezultata. Zato je upravo najveća pažnja u COBOL-jeziku posvećena sredstvima opisa svojstava i struktura teka ulaza-izlaza. Dalje, kako je bar za sad, a verovatno i ubuduće, većina ljudi koji se bave poslovnom obradom podataka, relativno slabo matematički potkovana, težnja je da se sintaksa i semantika COBOL-jezika što više približi odgovarajućem prirodnom (engleskom) jeziku. Ovim je povećana dokumentovanost COBOL-programa.

Veliki broj realizacija COBOL-jezika uslovio je uvodjenje

standardizacije jezika. Prvi standard COBOL-jezika objavljen je 1968. godine [127]. Pri standardizaciji se koristi modularni princip koji omogućava realizaciju COBOL-jezika na najrazličitijim računarima.

6.7. PL/I

PL/I je moćan, višeciljni jezik, koncipiran sredinom 60-tih godina od strane komiteta [5] koji je organizovala korporacija IBM. Prvobitan zadatak komiteta je bio stvaranje naslednika FORTRAN-jezika. Novi jezik morao bi da ima više mogućnosti opisa struktura podataka i što fleksibilniju operativnu sredinu i na taj način bio primenljiv za rešavanje širokog kruga zadataka. Medjutim, komitet je uskoro uvideo da je nemoguće definisati jezik koji bi zadovoljio postavljene zahteve, a zadržao sve osobine FORTRAN-a, pa je odlučeno da se pristupi koncipiranju i realizaciji novog programskog jezika, tj. PL/I-jezik.

Oblast primene PL/I-jezika obuhvata:

- naučno-tehničke kalkulacije,
- poslovnu obradu podataka i
- sistemsko programiranje.

Konstrukcije PL/I-jezika često su napravljene po uzoru na odgovarajuće konstrukcije FORTRAN, ALGOL i COBOL-jezika.

Tako na primer iz FORTRAN-a je uzet:

- mehanizam prenosa parametara,
- mogućnost posebne kompilacije potprograma,
- ulaz-izlaz,
- COMMON-zone itd.

Iz ALGOL-a su preneti:

- blokovska struktura programa i
- mogućnost komponovanja naredbi.

Iz COBOL-a:

- ulaz-izlaz orijentisan na zapise,
- nehomogeni nizovi,
- opisi tipa PICTURE.

Naravno, sve prenete mogućnosti su povećane, a realizovano je i niz novih ideja.

6.8. LISP

Programski jezik LISP (Lisp Processing) je koncipirala i realizovala grupa autora pod rukovodstvom J. Mc'Carthy-ja [71] sa Masačusetskog tehnološkog instituta 1960. godine.

LISP-jezik se koristi za rešavanje najraznovrstnijih problema veštačke inteligencije (teorija igara, dokazi teorema, robotika, obrada prirodnih jezika).

6.9. SNOBOL

Programski jezik SNOBOL je razradjen krajem 60-tih godinama u laboratorijama BELL-TELEPHONE pod rukovodstvom R.Griswold-a [42].

SNOBOL-jezik se koristi za obradu velikog broja podataka oblika niski znakova.

6.10. APL

Programski jezik APL prvi je izložio K. Iverson u knjizi A Programming Language izdatoj 1962. godine. Ime APL je skraćenica dobijena od početnih slova reči naslova navedene knjige. Glavni cilj Iverson-a nije bio konstrukcija "običnog" programskog jezika, koji se može realizovati na računaru, već razrada sistema zapisa - notacije podesne za sažeto izražavanje algoritama. Postavljeni cilj uzrokovao je korišćenje velikog skupa najraznovrsnijih znakova i simbola uključujući gornje i donje indekse itd. Ovakvu notaciju bilo bi teško, ako ne i nemoguće, realizovati na računaru. Medjutim, takav APL-jezik se pokazao kao koristan instrument za precizno formulisanje algoritama, koji se dalje recimo ručno, mogu prevesti na neki od uobičajenih programskih jezika. Kao lep primer primene APL-jezika često se navodi jezgrovit i potpuno formalan opis tehničkog sistema računara IBM-360 koji je izvršio A. Falkoff sa saradnicima 1964. godine [31].

Upotrebljivu verziju programskog jezika APL poznatu kao APL-360 realizovala je 1968. godine [30] grupa specijalista firme IBM. APL-jezik je realizovan kao interaktivan jezik.

6.11 "Praistorijski" prevodioci

Problem prevodjenja aritmetičkih i drugih algebarskih formula analiziran je i rešavan u matematičkoj logici mnogo pre nastanka elektronskih računara i potreba za prevodiocima.

Bezzagradni, tj. poljski zapis uveo je J. Lukasiewicz još 1929. godine.

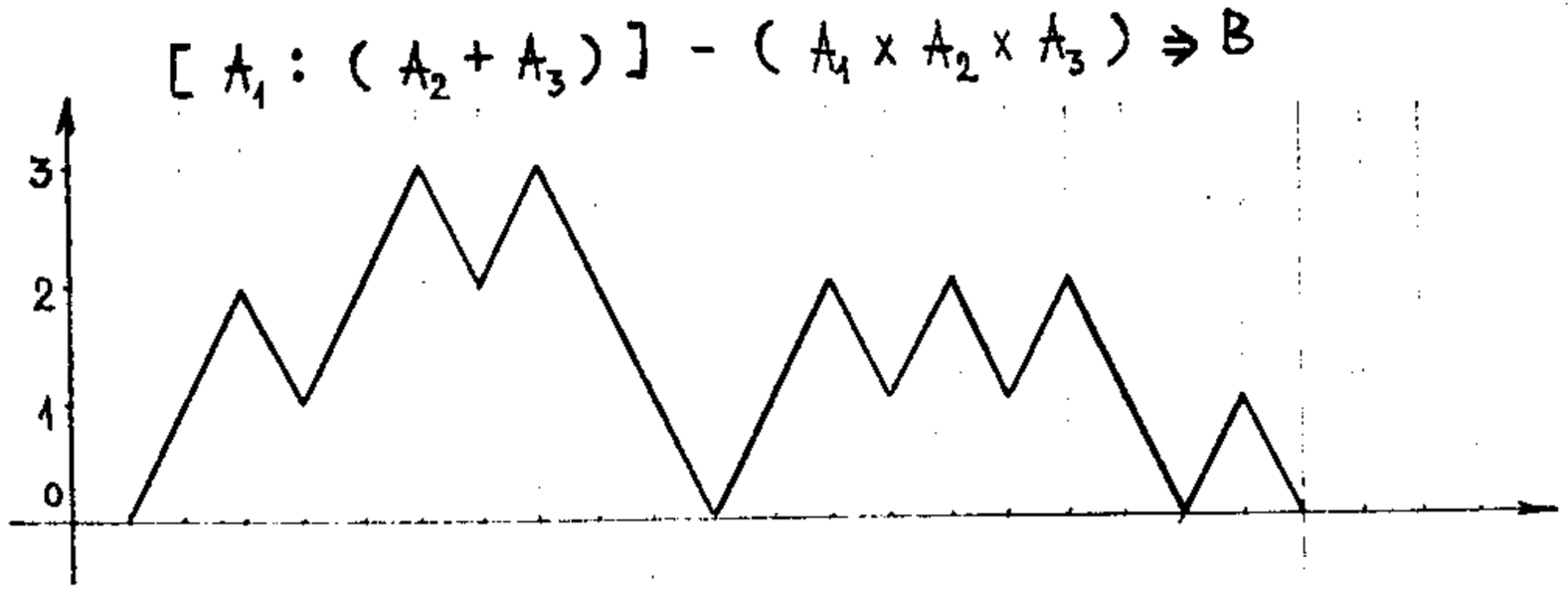
Pravila pisanja i čitanja bezzagradne notacije formalizovali su i dokazali Menger 1932. godine i Schröter 1943. godine.

U pojedinim radovima poznatih matematičara-logičara: Gödel-a, Turing-a, Post-a, Kleene-a i Church-a obradjuje se slična problematika.

1945. godine K. Zuse je pronašao algoritam za utvrdjivanje valjanosti - sintaksne ispravnosti izraza sa zagradama. Zuse je takodje izmislio načine uprošćavanja zapisa izbacivanjem suvišnih minusa ili zagrada.

P. Rosenbloom je 1950. godine izložio jedan formalan jezik kao sredstvo za opis sintakse algebarskih formula.

H. Rutishauser je 1951. godine prvi opisao proces prevodjenja algebarskih formula sa iznošenjem niza operativnih detalja i posebnim naglaskom na opšti značaj i perspektivu procesa prevodjenja. Računar Z4 koji je Rutishauser imao na raspolaganju suviše je malih mogućnosti da bi se na njemu realizovao neki prevodilac pa je Rutishauser izveo kasnije čuveni Gedanken-eksperiment, tj. kontrolu sintaksne ispravnosti zagrada. Na slici 6.1. prikazana je konturna mapa aritmetičkog izraza - faksimil iz istog Rutishauser-ovog rada.



Slika 6.1.

Pravila konstruisanja konture su trivijalna:
- leve zagrade i argumenti dižu konturu,
- desne zagrade i operatori spuštaju konturu.

Izraz je sintaksno ispravan ako je:
- kontura uvek iznad x-ose,
- početak i kraj konture na x-osi.

6.12. Prvi kompilatori

Izvestan broj prvih kompilatora izložen je maja meseca 1954. godine na Simpozijumu* iz automatskog programiranja za cifarske računare koji je organizovao Sekretarijat za istraživanje u pomorstvu Sjedinjenih američkih država.

Navedimo imena autora i nazive kompilatora:

- Adams, Ch. W., Laning, J.H. jr.:
  The MIT systems of automatic coding: Comprehensive, Summer Session, and Algebraic.
- Backus, J.W.,, Herrick, H.:
  IBM-701 speedcoding and other automatic programming systems.
- Brown, J.H. Carr III, J.W.:
  Automatic programming and its development on the MIDAC.
- Goldfinger, R.:
  New York University Compiler System.
- Gorn, S.:
  Planning universal semi-automatic coding.
- Rice, H.G.:
  The APS III compiler for the Datatron-204. i
- nepoznat autor
  The A-2 Compiler System.

Prvi kompilatori su ustvari bili prevodioci aritmetičkih formula. Da bi se olakšao posao kompilatoru formule su bile ili potpuno zagradjene ili nije postojao prioritet izmedju operatora, tj. izvršavale su se redom sleva udesno.

---

* Symposium Automatic Programming Digital Computers, Office of Naval Research.

Sledeći korak u razvoju programskih jezika je bio uvodjenje većih sloboda pisanja aritmetičkih formula, tj. uvodjenje:

- prirodnog prioriteta operatora i
- neobaveznosti zagrada.

Ovo je prvi put ostvareno u slučaju FORTRAN-jezika [7] realizovanog za računare firme IBM. Najzasluženiji za to bili su: Böhm, Backus i Sheridan.

Primer

Redosled izvršavanja operacija aritmetičkog izraza je implicitno odredjen. Medjutim, pre generisanja kôda, redosled izvršavanja operacija mora da bude sasvim eksplicitno odredjen.

Prvi FORTRAN-kompilator koristio je jedan veoma neprirodan ali efikasan algoritam za odredjivanje redosleda izvršavanja operacija. Istorijski značaj ovog algoritma je veliki i iz tog razloga biće na primeru objašnjen.

Zamislimo da aritmetički operator višeg prioriteta "slabo" razdvaja argumente sa obe strane, dok aritmetički operator nižeg prioriteta "jako" razdvaja argumente sa obe strane. Kako obično postoje tri nivoa prioriteta izmedju aritmetičkih operacija, mogu se postići tri nivoa razdvajanja dopisivanjem nijedne, jedne ili dve zagrade s obe strane operatora zavisno od prioriteta operatora, tj.:

A**↑**	A**↑↑**B
A**↑**B	A)**↑**(B
A+B	A))+((B

Dalje, na početku i kraju aritmetičkog izraza dopisuju se po dve leve odnosno desne zagrade. Ispred svake leve zagrade i argumenta dopisuje se aritmetički operator koji označava jačinu zagrade, odnosno argumenta. Uočimo aritmetički izraz:

A+B**↑**C

umetanjem zagrada dobija se:

((A))+((B)**↑**(C))

umetanjem aritmetičkih operatora dobija se tzv. normalni izraz:

+(**↑**(**↑↑**A))+(**↑**(**↑↑**B)**↑**(**↑↑**C))

Normalni izraz se analizira sleva udesno u cilju obrazovanja tzv. niza nivoa.

Članovi niza nivoa su oblika:

⊙	m
l	

l je tekući nivo,

⊙ je operator,

m je redni broj leve zagrade ili operand.

Prvi član niza nivoa je:

⊙	l
ø	

Nailaskom na levu zagradu, tekući nivo se povećava za jedan. Slično, nailaskom na desnu zagradu, tekući nivo se smanjuje za jedan.

Za slučaj ranije dobijenog normalnog izraza dobija se sledeći niz nivoa:

$$\frac{+\,|\,1}{ø\,|} \left(\frac{*\,|\,2}{1\,|} \left(\frac{**\,|\,A}{2\,|}\right)\right) \frac{+\,|\,3}{ø\,|} \left(\frac{*\,|\,4}{3\,|} \left(\frac{**\,|\,B}{4\,|}\right) \frac{*\,|\,5}{3\,|} \left(\frac{**\,|\,C}{5\,|}\right)\right)$$

Sledeći korak je generisanje nesortiranog niza trojki:

$$C_i O_i N_i$$

C_i je vreme izvršavanja operacije,

O_i je operacija i

N_i je operand.

Svaka trojka odgovara jednom nivou.

Za razmatran primer biće:

ø+1, 1*2, 2**A, ø+3, 3*4, 4**B, 3*5, 5**C .

Dalje se niz trojki sortira, spaja i pojednostavljuje:

ø+1	$U_o=+U_1+U_3$	$U_o=U_1+U_3$
ø+3		
1*2	$U_1=*U_2$	$U_1=U_2$

2**A	U_2=**A	U_2=A
3*4	U_3=*U_4*U_5	U_3=U_4*U_5
3*5		
4**B	U_4=**B	U_4=B
5**C	U_5=**C	U_5=C

Proces prevodjenja se završava procesom zamenjivanja odozdo na gore:

$$U_3=B*C$$

$$U_0=A+U_3$$

6.13. Sekvencijalni kompilatori

Sredinom 50-tih godina postalo je sasvim jesno da je prevodjenje aritmetičkih formula prvi korak rešavanja složenog problema kompilacije univerzalnih programskih jezika.

U to vreme tehničke mogućnosti računara - brzina rada i veličina memorije bile su male. Zato je glavni cilj konstruktora prevodioca bio pronaći što efikasnije metode kompilacije.

Prve metode prevodjenja glavnu pažnju su posvećivale sekvencijalnim formulama. Ideja prevodjenja aritmetičkih izraza sastoji se u sledećem: traži se desna zagrada, a zatim izvršava odgovarajuća obrada sdesna ulevo do leve zagrade. Postupak se iz početka ponavlja sa preskakanjem već obradjenih delova. Na ovu ideju prvi je došao Bottenbruch 1957. godine. Nju su realizovali Adams [1], Schlesinger i Ershov [28] pri konstrukciji kompilatora 1958. godine.

Većina sekvencijalnih metoda kompilacije koristi medjumemoriju koju razni autori nazivaju:

- poslednji u - prvi iz memorija,
- stog (stack),
- ostava (cellar)
- potiskujuća ili potisna memorija (pushdown store) itd.

Bauer i Samelson su 1960. godine uveli tablicu prelaza (transition table). Na Kolokvijumu iz logike 1960. godine održanom u Berlinu uvedeni su potisni automati (pushdown automata).

Ovo su godine nastanka ALGOL-58 i 60 jezika. Prvi ALGOL-kompilator konstruisao je Paul za računar ZUSE-Z22, čija se memorija sastojala iz jednog doboša kapaciteta 2000 registara.

6.14. Sintaksno-kontrolisani kompilatori

Očigledno je da tablice prelaza i potisni automati održavaju strukturu jezika. Problem njihovog mehaničkog konstruisanja iz formalnog opisa strukture jezika prvi je uočio Paul [78], a rešio Mainz u svojoj doktorskoj disertaciji odbranjenoj 1962. godine. Isti problem samo za ograničenu klasu jezika rešio je Floyd [34] 1963. godine.

Eickel, Floyd, Graham i Irons su 1964. godine definisali kontekstno-ograničene gramatike.

Prvi rad u kome je izloženo konstruisanje tabela za (m,n) sleva udesno kontekstno-ograničeni prepoznavač objavio je Eickel [26].

Najopštiju klasu LR(k) gramatika koje se mogu obradjivati determinističkim potisnim automatima, koji omogućavaju pristup svim simbolima u stogu i k-simbolima udesno prvi je definisao Knuth [59].

6.15. Kompilatori zasnovani na prioritetu

Na ideju upotrebe prioriteta dva susedna simbola za upravljanje prepoznavačima prvi je došao Perlis još 1956. godine.

1963. godine Floyd je definisao prioritetne gramatike za koje se prepoznavači relativno lako konstruišu.

Floyd-ove gramatike prioriteta operatora su ekvivalentne prostim prioritetnim gramatikama Wirth-a i Weber-a, odnosno (1,1) slučaj su opštih (m,n)-gramatika koje su takodje definisali Wirth i Weber. 1966. godine Mc'Keeman je uopštio prost prioritet uvodjenjem (1,2)(2,1) prioriteta.

Neliac i Alcor Illinois [41] kompilator napravljeni su na osnovu prioriteta izmedju operatora.

6.16. Sintaksno-orijentisani kompilatori

Osnovne ideje o sintaksno-orijentisanoj kompilaciji prvi je dao Glennie [36] 1960. godine.

Gramatičku analizu sa eksplicitnim rekursivnim silaženjem prvi je izložio Lucas [68] **1961. godine. On je opisao pojednostavljen ALGOL-60 kompilator skupom rekursivnih potprograma. Svaki potprogram odgovara jednoj metalingvističkoj formuli** Backus-a, **kojima je opisana sintaksa programskog jezika ALGOL-60.**

1961. godine Irons [51] **je opisao algoritam gramatičke analize vodjen tablicama koje** definišu sintaksu jezika. Specifičnost Irons-ovog algoritma ogleda se u kombinovanju metoda gore - dole i dole - gore.

Irons-ova ideja bila je izvanredna. Za nju su se zainteresovali mnogi matematičari, inženjeri i lingvisti i počeli su da je proučavaju i produbljuju. Navedimo imena nekolicine najpoznatijih naučnika: Brooker, Morris, Ledley, Wilson, Barnett i Futrelle koji su na tome radili.

Do sada je konstruisano niz sintaksno-orijentisanih kompilatora. Najpoznatiji su:

META - autor Schorre

ALGOL - za firmu Burroughs

SHARE-7090 **ALGOL itd.**

Godine 1965 **Foster je našao opšte pravilo transformisanja gramatika na oblik LL(1). Zbog značaja ovaj rad je objavljen tek 1968. godine.**

Knuth **je 1967. godine definisao LL(1) gramatike. Njegovi rezultati su iz istih razloga objavljeni tek** 1971. godine.

LITERATURA

Literatura na engleskom jeziku:

[1] Adams, E. S. i Schlesinger, S. I. /1958/. Simple automatic coding systems, Comm. ACM 1:7, 5-9.

[2] Aho, A. V. /1973/. Currents in the theory of computing, Prentice-Hall, Enlewood Cliffs.

[3] Aho, A. V. i Ullman, J. D. /1972/. The theory of parsing, translation and compiling, Prentice-Hall, Englewood Cliffs.

[4] ANSI /1966/. American national standard FORTRAN (ANS X3.9-1966), American National Standards Institute, New York.

[5] ANSI /1976/. American national standard programming language PL/I (ANS X3.53 - 1976), American National Standards Institute, New York.

[6] Backhouse, R. C. /1979/. Syntax of programming languages, Prentice-Hall, London.

[7] Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Haibt, L. M., Herrick, H. L., Nelson, R. A., Sayre, D., Sheridan, P. B., Hughes, R. A. i Nutt, R. /1957/. The FORTRAN automatic coding system, Proc. AFIPS 1957 Western Joint Computer Conf., Spartan Books, Baltimore, 188-198.

[8] Bandat, K. /1968/. On the formal definition of PL/I, Proc. Spring Joint Computer Conf., Thompson, Wash. D. C., 363-373.

[9] Barnett, M. P. i Futrelle, R. P. /1962/. Syntactic analysis by digital computer, Comm. ACM 5, 515-526.

[10] Bauer, F. L. i Eickel, J. /1974/. Compiler Construction: An Advanced Course, Springer-Verlag, New York.

[11] Berztiss, A. T. /1971/. Data structures, Academic Press, New York.

[12] Brillinger, P. C. i Cohen, D. J. /1972/. Introduction to data structures and non-numeric computation, Prentice-Hall, Englewood Cliffs.

[13] Bron, C. i de Vries, W. /1976/. A PASCAL compiler for PDP11 minicomputers, Software-Practice and Experience 6:1, 109-116.

[14] Brooker, R. A., MacCallum, I. R., Morris, D. i Rohl, J. S. /1963/. The compiler-compiler, Annual Review of Automatic Programming 3, 229-275.

[15] Brooker, R. A. i Morris, D. /1962/. A general translation program for phrase-structure languages, J. ACM 9:1, 1-10.

[16] Cheatham, T. E. Jr. /1965/. The TGS-II translator generator system, Proc. IFIP Congress 65, North Holland, Amsterdam, 592-593.

[17] Cheatham, T. E. Jr. i Sattley, K. /1964/. Syntax directed compiling, Proc. AFIPS 1964 Spring Joint Computer Conf., Spartan Books, Baltimore, 31-57.

[18] Cleaveland, J. C. i Uzgalis, R. C. /1977/. Grammars for programming languages, American Elsevier, New York.

[19] Cocke, J. i Schwartz, J. T. /1970/. Programming languages and their compilers, Courant Institute of Mathematical Sciences, New York.

[20] Conway, M. E. /1963/. Design of a separable transition diagram compiler, Comm. ACM 6:7, 396-408.

[21] Conway, R. W. i Wilcox, T. R. /1973/. Design and implementation of a diagnostic compiler for PL/I, Comm. ACM 16:3, 169-179.

[22] Day, A. C. /1978/. Compatible FORTRAN, Cambridge University Press, Cambridge.

[23] Day, A. C. /1972/. FORTRAN techniques, Cambridge University Press, Cambridge.

[24] Dijkstra, E. W. /1960/. ALGOL 60 translation, Supplement ALGOL Bulletin 10.

[25] Earley, J. /1970/. An efficient context-free parsing algorithm, Comm. ACM 13:2, 94-102.

[26] Eickel, J. /1962/. Generation of parsing algorithms for Chomsky 2-type languages, Mathematisches Institut der Technischen Universitat München, Bericht Nr. 6401.

[27] Eickel, J., Paul, M., Bauer, F. L. i Samelson, K. /1963/. A syntax controlled generator of formal language processors, Comm. ACM 6:8, 451-455.

[28] Ershov, A. P. /1958/. On programming of arithmetic operations, Comm. ACM 1:8, 3-6.

[29] Ershov, A. P. /1971/. The ALPHA automatic programming system, Academic Press, New York.

[30] Falkoff, A. i Iverson, K. /1968/. The APL-360 terminal system, Academic Press, New York.

[31] Falkoff, A., Iverson, K. i Sussenguth, E. /1964/. A formal description of System/360, IBM Syst. J., 3,3,198-263.

[32] Feldman, J. A. /1966/. A formal semantics for computer languages and its application in a compiler-compiler, Comm. ACM 9:1, 3-9.

[33] Feldman, J. A. i Gries, D. /1968/. Translator writing systems, Comm. ACM 11:2, 77-113.

[34] Floyd, R. W. /1963/. Syntactic analysis and operator precedence, J. ACM 10:3, 316-333.

[35] Freiburghouse, R. A. /1969/. The multics PL/I compiler, AFIPS Conf. Proc. Fall Joint Computer Conference 35, 187-208.

[36] Glennie, A. /1960/. On the syntax machine and the construction of a universal compiler, Carnegie-Mellon University, Techn. Report No. 2.

[37] Graham, R. M. /1964/. Bounded context translation, Proc. AFIPS Spring JCC 40, 205-217.

[38] Graham, R. M. /1975/. Principles of system programming, John Wiley & Sons, New York.

[39] Grau, A. A., Hill, U. i Langmaack, H. /1967/. Translation of ALGOL 60, Springer-Verlag, New York.

[40] Gries, D. /1971/. Compiler construction for digital computers, John Wiley & Sons, New York.

[41] Gries, D., Paul, M. i Wiehle, H. R. /1965/. Some techniques used in the ALCOR ILLINOIS 7090, Comm. ACM 8:8, 496-500.

[42] Griswold, R. E., Poage, J. i Polonsky, I. /1971/. The SNOBOL4 programming language, Prentice-Hall, Englewood Cliffs.

[43] Gross, M. i Lentin, A. /1970/. Introduction to formal grammars, Springer-Verlag, Berlin.

[44] Hartmann, A. C. /1977/. A concurrent PASCAL compiler for minicomputers, Springer-Verlag, Berlin.

[45] Hoare, C. A. R. i Wirth, N. /1973/. An axiomatic definition of the programming language PASCAL, Acta Informatica 2:4, 335-356.

[46] Hopgood, F. R. A. /1969/. Compiling Techniques, American Elsevier, New York.

[47] Horowitz, L. P., Karp, R. M., Miller, R. M. i Winograd, S. /1966/. Index register allocation, J. ACM 13:1, 43-61.

[48] Hugnes, C. E. /1978/. Advanced programming techniques, John Wiley & Sons, New York.

[49] IBM /1968/. FORTRAN IV (H) compiler program logic manual, Form Y28-6642-3, IBM, New York.

[50] Ingerman, P. Z. /1966/. A syntax oriented translator, Academic Press, New York.

[51] Irons, E. T. /1961/. A syntax directed compiler for ALGOL 60, Comm. ACM 4:1, 51-55.

[52] Jensen, K. i Wirth, N. /1975/. PASCAL user manual and report, Springer-Verlag, New York.

[53] Kain, R. Y. /1972/. Automata theory:machines and languages, McGraw-Hill, New York.

[54] Katzan, H. J. /1970/. Advanced programming, Van Nostrand, New York.

[55] Kemeny, J. G. i Kurtz, T. E. /1967/. BASIC programming, John Wiley & Sons, New York.

[56] Kernighan, B. W. /1975/. RATFOR - a preprocessor for a rational FORTRAN, Software - Practice and Experience 5:4, 395-406.

[57] Kernighan, B. W. i Plauger, P. J. /1976/. Software tools, Addison-Wesley, Reading.

[58] Kernighan, B. W. i Plauger, P. J. /1978/. The elements of programming style, McGraw-Hill, New York.

[59] Knuth, D. E. /1968/. Semantics of context-free languages, Math. Systems Theory 2:2, 127-145.

[60] Knuth, D. E. /1973/. The art of computer programming, Addison-Wesley, Reading.

[61] Kreitzberg, C. B. /1972/. The elements of FORTRAN style, Harcourt Brace Jovanovich, New York.

[62] Ledley, R. S. i Wilson, J. B. /1962/. Automatic-programming language translation through syntactical analysis. Comm. ACM 5, 145-155.

[63] Lee, J. A. N. /1974/. Anatomy of a compiler, Van Nostrand, New York.

[64] Lewis, P. M. II i Rosenkrantz, D. J. /1971/. An ALGOL compiler designed using automata theory, Proc. Symp. on Computers and Automata, Microwave research Inst., Polytechnic Inst. of New York, 75-88.

[65] Lewis, P. M. II, Rosenkrantz, D. J. i Stearns, R. E. /1976/. Compiler design theory, Addison-Wesley, Reading.

[66] Lewis, P. M. II i Stearns, R. E. /1968/. Syntax-directed transduction, J. ACM 15:3, 465-488.

[67] Lowry, E. i Medlock, C. W. /1969/. Object code optimization, Comm. ACM 12:1, 13-22.

[68] Lucas, P. /1961/. The structure of formula-translators, Mailüfterl Vienna, Austria, ALGOL Bulletin Suppl. No. 16.

[69] Lucas, P. i Walk, P. /1969/. On the formal description of PL/I, Ann. Rev. Aut. Prog., 6, 3, 105-182.

[70] Maginnis, J. B. /1972/. Elements of compiler construction, ACC, New York.

[71] McCarthy, J. /1965/. LISP 1.5 Programmer's manual, MIT Press, Cambridge.

[72] McClure, R. M. /1965/. TMG-a syntax directed compiler, Proc. 20th ACM National Conf., 262-274.

[73] McKeeman, W. M., Horning, J. J. i Wortman, D. B. /1970/. A compiler generator, Prentice-Hall, Englewood Cliffs.

[74] Mickunas, M. D. i Schneider, V. B. /1973/. A parser-generating system for constructing compressed compilers, Comm. ACM 16:11, 669-676.

[75] Nakata, I. /1967/. On compiling algorithms for arithmetic expressions, Comm. ACM 10:8, 492-494.

[76] Naur, P. /1963/. Revised report on the algorithmic language ALGOL 60, Comm. ACM 6:1, 1-17.

[77] Parezanović, N. i Janković, B. /1977/. An approach to designing problem-oriented languages, Anglo-Yugoslav symposium, Herceg Novi.

[78] Pollack, B. W. /1972/. Compiler techniques, Auerbach Press, Philadelphia.

[79] Pratt, T. W. /1975/. Programming languages: design and implementation, Prentice-Hall, Englewood Cliffs.

[80] Randell, B. i Russell, L. J. /1964/. ALGOL 60 implementation, Academic Press, New York.

[81] Ritchie, D. M., Kernigham, B. W. i Lesk, M. E. /1975/. The C programming language, Prentice-Hall, Englewood Cliffs.

[82] Rohl, J. S. /1975/. An introduction to compiler writing, Macdonald and Jane's, London.

[83] Rosen, S. /1967/. Programming systems and languages, McGraw-Hill, New York.

[84] Russel, D. L. i Sue, J. Y. /1976/. Implementation of a PASCAL compiler for the IBM 360, Software-Practice and Experience 6:3, 371-376.

[85] Rustin, R. /1972/. Design and optimization of compilers, Prentice-Hall, Englewood Cliffs.

[86] Salomaa, A. /1973/. Formal languages, Academic Press, New York.

[87] Sammet, J. E. /1968/. Programming languages: history and fundamentals, Prentice-Hall, Englewood Cliffs.

[88] Stojković, V. /1977/. The program simulator of the machine oriented language of one hypotetical machine, VI Balkan mathematical congrese, Varna.

[89] Stojković, V. /1977/. The pumping lema and its application to regular sets. XII Yugoslav international symposium on information processing, Bled.

[90] Velašević, D. /1980/. Right-to-left code generation for arithmetic expressions, Informatica 1, 22-27.

[91] Warshall, S. i Shapiro, R. M. /1964/. A general purpose table driven compiler, Proc. AFIPS 1964 Spring Joint Computer Conf., Spartan Books, Baltimore, 59-65.

[92] Wegner, P. /1972/. The Vienna definition language, Computing Surveys 4:1, 5-63.

[93] Weingarten, F. W. /1973/. Translation of computer languages, Holden-Day, San Fransisco.

[94] Wirth, N. /1971/. The programming language PASCAL, Acta Informatica 1:1,35-63.

[95] Wirth, N. /1971/. The design of a PASCAL compiler, Software-Practice and Experience 1:4, 309-333.

[96] Wortman, D. B., Khaiat, P. J. i Laskar, D. M. /1976/. Six PL/I compilers, Software-Practice and Experience 6:3, 411-422.

[97] Wulf, W., Johnsson, R. K., Weinstock, C. B., Hobbs, S. O. i Geschke, C. M. /1975/. The design of an optimizing compiler, American Elsevier, New York.

Literatura na ruskom jeziku:

[98] **Алферова З. В., Лихачева Г. Н. и Шураков В. В. Математическое обеспечение эвм. Статистика, Москва, 1974.**

[99] Болъе Л. Методы построения компиляторов. Сб. Языки программирования. Мир, Москва, 1972.

[100] Братчиков И. Л. Синтаксис языков программирования. Наука, Москва, 1975.

[101] Гладкий А. В. Формалъные грамматики и языки. Наука, Москва, 1973.

[102] Глушков В. М., Цейтлин Г. Е. и Ющенко Е. Л. Алгебра языки программирование. Наукова думка, Киев, 1974.

[103] Демин В. Ф., Добролюбов Л. В. и Степанов В. А. Системы программирования на АЛГОЛе. Наука, Москва, 1977.

[104] Криницкий Н. А., Миронов Г. А. и Фролов Г. Д. Программирование и алгоритмические языки. Наука, Москва, 1979.

[105] Лебедев В. Н. Введение в системы программирования. Статистика, Москва, 1975.

[106] Летичевский А. А. Синтаксис и семантика формальных языков. Кибернетика, 1968, 4.

[107] Оллонгрен А. Определение языков программирования интерпретирующими автоматами. Мир, Москва, 1977.

[108] Редъко В. Н. Интерпретированные языки и интерпретаторы. Кибернетика, 1969, 5.

[109] Редъко В. Н. и Ющенко Е. Л. Алгоритмические языки и транслирующие системы. Кибернетика, 1967, 5.

[110] Савинков В. М. и Сидоренко Л. А. Синтаксический транслятор анализирующего типа. Статистика, Москва, 1972.

[111] Фостер Дж. Автоматический синтаксический анализ. Мир, Москва, 1975.

[112] Фуксман А. Л. Технологические аспекты создания программных систем. Статистика, Москва, 1979.

[113] Хигман Б. Сравнителъное изучение языков программирования. Мир, Москва, 1974.

[114] Цейтина Г. С. АЛГОЛ 68 методы реализации. Изд. Ленинградского университета, Ленинград, 1976.

[115] Шура-Бура М. Р. и Любимский Э. З. Транслятор АЛГОЛ-60, Журн. вычисл. матем. и матем. физ. 4, 1, 1964.

[116] Шураков В. В. Математическое обеспечение ЭВМ. Изд. Киевский институт народного хозяйства им. Д. С. Коротченко, Киев, 1971.

Literatura na srpskohrvatskom jeziku:

[117] Alagić S. Principi programiranja. Svjetlost, Sarajevo, 1976.
[118] Alagić S. Matematički aspekti programskih jezika. VI Kongres MFAJ, Novi Sad, 1975.
[119] Bitrakov D. FORTRAN IV programski jezik. Književne novine, Beograd, 1976.
[120] Vuletić V. i Ljubović Ć. Programiranje /FORTRAN/. Svjetlost, Sarajevo, 1975.
[121] Žokalj M. FORTRAN IV. Statistički zavod, Beograd, 1969.
[122] Janković B. i Parezanović N. Generativne gramatike metaprogramskih jezika. VI Kongres MFAJ, Novi Sad, 1975.
[123] Mijajlović Ž. i Jovanović A. Jezik za algoritamski sistem Turingovih mašina. VII Kongres MFAJ, Bečići, 1980.
[124] Parezanović N., Timotić M. i Trajković D. Simbolički jezik SJ-1 za CER-11. XII Jugoslovenska konferencija ETAN-a, 1968.
[125] Parezanović N. Algoritmi i programski jezik FORTRAN IV, Matematički institut, Beograd, 1970.
[126] Parezanović N. Programski jezik FORTRAN IV. Privredno-finansijski vodič, Beograd, 1973.
[127] Parezanović N. i Petrić Z. Programski jezik COBOL. Privredno-finansijski vodič, Beograd, 1973.
[128] Parezanović N. Osnovi računske tehnike. Privredno-finansijski vodič, Beograd, 1974.
[129] Parezanović N. i Janković B. Programski jezik BASIC. Privredno-finansijski vodič, Beograd, 1977.
[130] Parezanović N. Uvod u programiranje. Privredno-finansijski vodič, Beograd, 1980.
[131] Prešić S. Elementi matematičke logike. Zavod za izdavanje udžbenika SR Srbije, Beograd, 1972.
[132] Stefanini B. FORTRAN V. Školska knjiga, Zagreb, 1978.
[133] Stojaković M. Algoritmi i automati. Radnički univerzitet Radivoj Cirpanov, Novi Sad, 1972.
[134] Stojković V. Svodjenje kontekstno-slobodnih gramatika. XI Jugoslovenski medjunarodni simpozijum iz obrade podataka, Bled, 1976.
[135] Stojković V. Bekusova notacija i primena. Praksa 7/8, 1977.
[136] Stojković V. Sintaksno-vodjena analiza. II bosansko-hercegovački simpozijum iz informatike, Jahorina, 1978.

[137] Stojković V. Prevodjenje aritmetičkog izraza sa uobičajenog na savršen zapis. III bosansko-hercegovački simpozijum iz informatike, Jahorina, 1979.

[138] Stojković V. Izračunavanje vrednosti iskaznih formula iskazne algebre /0,1/ pomoću računara. XXIII Jugoslovenska konferencija iz ETAN-e, Maribor, 1979.

[139] Stojković V. Realizacija kompilatora jednog nastavnog proceduralno-orijentisanog jezika. IV bosansko-hercegovački simpozijum iz informatike, Jahorina, 1980.

[140] Stojković V. Neke osobine sintaksno-orijentisanih prevodjenja. VII Kongres MFAJ, Bečići, 1980.

[141] Trahtenbrot B. Što su algoritmi. Školska knjiga, Zagreb, 1978.

[142] Hrnjaković Lj. COBOL-viši jezik za programiranje. Beograd, 1974.

[143] Čomski N. Gramatika i um. Nolit, Beograd, 1972.

[144] Čuljat K. Organizacija i funkcioniranje digitalnih kompjutera. Stvarnost, Zagreb, 1971.

PROGRAM UNIVERZALNOG KOMPILARORA

```
      IMPLICIT INTEGER(A-Z)
C
      LOGICAL*1
     1    IMEDAT(20),
     2    TACKA,SLOVGS,SLOVOI,SLOVON,SLOVOE,SLOVOM,
     3    LANAC(110,40)
C
      INTEGER*2
     1    ULAZ(80),
     2    RBP(200),
     3    SINFRA(110,40),
     4    MLF(20),
     5    PROM(1500),
     6    KONST(200),
     7    LISTA(100),
     8    ZAGLAVLJE(100),
     9    GLAVA(100),
     1    JRAT(100),
     2    TELO(500),
     3    RUKA(500),
     4    VEKTOR(700),
     5    SEMPRA(1500),
     6    SPISAK(200)
C
      EQUIVALENCE
     1    (SEMPRA(1),PROM(1)),
     2    (DUZLAN,DUZLIS,DUZGLA,DUZVRA),
     3    (SIRPRA,REALIP),
     4    (DUZTEL,DUZRUK)
C
      DATA
     1    LUZ/':'/,
     2    DUZ/'>'/,
     3    ILI/'!'/,
     4    DVETACKE/':'/,
     5    JEDNAKOST/'='/,
     6    NAVOD/1H'/,
     7    RAZMAK/' '/,
     8    DOLAR/'$'/,
     9    TACKA/'.'/,
     1    SLOVGS/'S'/,
     2    SLOVOI/'I'/,
     3    SLOVON/'N'/,
     4    SLOVOE/'E'/,
     5    SLOVOM/'M'/
```

```
      MAXDP=40
      REALDP=0
      RED=0
      PLUZ=0
      DUZKON=0
      DUZPRO=0
      L=0
      WRITE(5,4001)
4001  FORMAT(' NAVESTI IME DATOTEKE CIJI JE SADRZAJ'
     1       ' OPIS SINTAKSE JEZIKA ? ',$)
      READ(5,4003)DUZID,IMEDAT
4003  FORMAT(Q,20A1)
      IMEDAT(DUZID+1)=TACKA
      IMEDAT(DUZID+2)=SLOVOS
      IMEDAT(DUZID+3)=SLOVOI
      IMEDAT(DUZID+4)=SLOVON
      DUZID=DUZID+4
      CALL ASSIGN(1,IMEDAT,DUZID)
      WRITE(5,400)(IMEDAT(N),N=1,DUZID-4)
400   FORMAT(' OPIS SINTAKSE JEZIKA ',20A1/)
1     READ(1,100,END=20)DUZINA,ULAZ
100   FORMAT(Q,80A1)
      WRITE(5,333)(ULAZ(N),N=1,DUZINA)
333   FORMAT(' ',80A1)
      IF(ULAZ(1).NE.LUZ)GOTO 2
      RED=RED+1
      RBP(RED)=RED
      KOLONA=0
      DO 10 N=1,MAXDP
10    SINPRA(RED,N)=0
2     I=0
3     I=I+1
      IF(I.GT.DUZINA)GOTO 1
      ZNAK=ULAZ(I)
      IF(ZNAK.EQ.RAZMAK)GOTO 3
      IF(ZNAK.EQ.NAVOD)GOTO 1101
      IF(ZNAK.EQ.LUZ)GOTO 9
      IF(ZNAK.EQ.DUZ)GOTO 8
      IF(ZNAK.EQ.ILI)GOTO 7
      IF(ZNAK.EQ.DVETACKE)GOTO 6
4     IF(PLUZ.EQ.1)GOTO 5
      N=-1
11    N=N+2
      IF(N.GT.DUZKON)GOTO 12
      IF(KONST(N).NE.ZNAK)GOTO 11
      KOD=KONST(N+1)
      GOTO 13
1101  I=I+1
      ZNAK=ULAZ(I)
      GOTO 4
12    DUZKON=DUZKON+2
      KONST(DUZKON-1)=ZNAK
      L=L+1
      KOD=L
      KONST(DUZKON)=KOD
      GLAVA(L)=KOD
      ZAGLAVLJE(L)=0
      VRAT(L)=0
      LISTA(L)=DUZKON-1
13    KOLONA=KOLONA+1
      IF(KOLONA.GT.REALDP)REALDP=KOLONA
      SINPRA(RED,KOLONA)=KOD
```

```
      GOTO 3
5     J=J+1
      MLP(J)=ZNAK
      GOTO 3
      IF(ULAZ(I+1).NE.DVETACKE)GOTO 4
      IF(ULAZ(I+2).NE.JEDNAKOST)GOTO 4
      I=I+2
      GOTO 3
7     RED=RED+1
      RBP(RED)=RED
      KOLONA=1
      DO 39 N=1,MAXDP
39    SINPRA(RED,N)=0
      SINPRA(RED,1)=SINPRA(RED-1,1)
      GOTO 3
8     J=J+1
      MLP(J)=ZNAK
      DUZMLP=J
      J=1
14    IF(J.GT.DUZPRO)GOTO 17
      IF(PROM(J).NE.DUZMLP)GOTO 16
      K=J
      DO 15 N=1,DUZMLP
      K=K+1
15    IF(PROM(K).NE.MLP(N))GOTO 16
      KOD=PROM(K+1)
      GOTO 19
16    J=J+PROM(J)+2
      GOTO 14
17    PROM(J)=DUZMLP
      DO 18 N=1,DUZMLP
      J=J+1
      DUZPRO=DUZPRO+1
18    PROM(J)=MLP(N)
      L=L+1
      KOD=L+1000
      PROM(J+1)=KOD
      DUZPRO=DUZPRO+2
      GLAVA(L)=KOD
      ZAGLAVLJE(L)=0
      VRAT(L)=0
      LISTA(L)=J-DUZMLP
19    KOLONA=KOLONA+1
      IF(KOLONA.GT.REALDP)REALDP=KOLONA
      SINPRA(RED,KOLONA)=KOD
      PLUZ=0
      GOTO 3
9     PLUZ=1
      J=1
      MLP(J)=ZNAK
      GOTO 3
20    DUZPRA=RED-1
      DUZLIS=L
      WRITE(5,1231)
1231  FORMAT(/' TABLICA KONSTANTI'/
     1     ' KONSTA-KOD'/)
      WRITE(5,1232)(KONST(N),N=1,DUZKON)
1232  FORMAT(' ',A1,'-',I2)
      WRITE(5,1233)
1233  FORMAT(/' TABLICA PROMENLJIVIH'/
     1     ' DUZIN.-PROMENLJIVA-KOD'/)
```

```
      N=1
1234  WRITE(5,1235)PROM(N)
1235  FORMAT(' ',I2,'-',$)
      WRITE(5,12351)(PROM(J),J=N+1,N+PROM(N))
12351 FORMAT('+',A1,$)
      J=N+PROM(N)
      WRITE(5,12353)PROM(J+1)
12353 FORMAT('+','-',I4)
      IF(PROM(J+2).EQ.0)GOTO 1236
      N=J+2
      GOTO 1234
1236  WRITE(5,12361)
12361 FORMAT(/' KODIRANA PRAVILA'/)
      DO 1240 I=1,DUZPRA
      DO 1238 K=1,REALDP
1238  IF(SINPRA(I,K).EQ.0) GOTO 1240
1240  WRITE(5,1239)RBP(I),(SINPRA(I,J),J=1,K-1)
1239  FORMAT(' ',I3,15I5)
      WRITE(5,1242)
1242  FORMAT(/'  GLAVA ZAGLAVLJE VRAT LISTA'/)
      WRITE(5,1243)(GLAVA(I),ZAGLAVLJE(I),VRAT(I),LISTA(I),I=1,DUZLIS)
1243  FORMAT(' ',4I6)
      DO 23 I=1,DUZPRA-1
      DO 23 J=I+1,DUZPRA
      DO 24 K=2,SIRPRA
      IF(SINPRA(I,K).EQ.SINPRA(J,K))GOTO 24
      IF(SINPRA(I,K).LT.SINPRA(J,K))GOTO 23
      L=RBP(I)
      RBP(I)=RBP(J)
      RBP(J)=L
      DO 22 K=1,SIRPRA
      L=SINPRA(I,K)
      SINPRA(I,K)=SINPRA(J,K)
22    SINPRA(J,K)=L
      GOTO 23
24    CONTINUE
23    CONTINUE
      WRITE(5,12361)
      DO 2340 I=1,DUZPRA
      DO 2338 K=1,SIRPRA
2338  IF(SINPRA(I,K).EQ.0)GOTO 2340
2340  WRITE(5,1239)RBP(I),(SINPRA(I,J),J=1,K-1)
      DO 81 I=1,DUZLIS-1
      DO 81 J=I+1,DUZLIS
      IF(GLAVA(I).LE.GLAVA(J))GOTO 81
      K=GLAVA(I)
      GLAVA(I)=GLAVA(J)
      GLAVA(J)=K
      K=ZAGLAVLJE(I)
      ZAGLAVLJE(I)=ZAGLAVLJE(J)
      ZAGLAVLJE(J)=K
      K=VRAT(I)
      VRAT(I)=VRAT(J)
      VRAT(J)=K
      K=LISTA(I)
      LISTA(I)=LISTA(J)
      LISTA(J)=K
81    CONTINUE
      WRITE(5,1242)
      WRITE(5,1243)(GLAVA(I),ZAGLAVLJE(I),VRAT(I),LISTA(I),I=1,DUZLIS)
      J=0
      DO 30 I=1,DUZLIS
```

```
      IF(GLAVA(I).LT.1000)GOTO 30
      J=J+1
      ZAGLAVLJE(I)=J
30    CONTINUE
      WRITE(5,1242)
      WRITE(5,1243)(GLAVA(I),ZAGLAVLJE(I),VRAT(I),LISTA(I),I=1,DUZLIS)
      SIRLAN=J
      DO 31 I=1,DUZLAN
      DO 31 J=1,SIRLAN
31    LANAC(I,J)=.FALSE.
      DO 36 I=1,DUZPRA
      DO 32 J=1,DUZLIS
32    IF(SINPRA(I,2).EQ.GLAVA(J))GOTO 33
      PAUSE'X::=YZ... Y NIJE U GLAVI'
33    RED=J
      DO 34 J=1,DUZLIS
34    IF(SINPRA(I,1).EQ.GLAVA(J))GOTO 35
      PAUSE'X::=YZ... X NIJE U GLAVI'
35    KOLONA=ZAGLAVLJE(J)
36    LANAC(RED,KOLONA)=.TRUE.
      WRITE(5,3601)
3601  FORMAT(/' LANAC '/)
      DO 3602 I=1,DUZLAN
3602  WRITE(5,3603)(LANAC(I,J),J=1,SIRLAN)
3603  FORMAT(' ',72L1)
      DO 43 M=1,2
      DO 43 J=1,SIRLAN
      DO 43 I=1,DUZLAN
      IF(.NOT.LANAC(I,J))GOTO 43
      DO 44 K=1,DUZLIS
44    IF(ZAGLAVLJE(K).EQ.J)GOTO 45
      PAUSE'X::=YZ... X NIJE U ZAGLAVLJU'
45    DO 46 L=1,SIRLAN
      IF(LANAC(I,L))GOTO 46
      LANAC(I,L)=LANAC(K,L)
46    CONTINUE
43    CONTINUE
      WRITE(5,3601)
      DO 4301 I=1,DUZLAN
4301  WRITE(5,3603)(LANAC(I,J),J=1,SIRLAN)
      L=1
      I=1
      J=2
50    DO 49 K=1,DUZLIS
      IF(SINPRA(1,2).EQ.GLAVA(K))GOTO 60
49    CONTINUE
      PAUSE'X::=YZ... Y NIJE U GLAVI'
60    VRAT(K)=L
      M=L
      PSDSP=SINPRA(I,2)
53    J=J+1
51    IF(SINPRA(I,J).EQ.0)GOTO 52
      TELO(L)=SINPRA(I,J)
      RUKA(L)=0
      L=L+1
      GOTO 53
52    TELO(L)=SINPRA(I,1)
      RUKA(L)=0
      TELO(L+1)=9999
      RUKA(L+1)=0
      TELO(L+2)=PSDSP
      RUKA(L+2)=0
```

```
      L=L+3
      I=I+1
      N=M
      IF(I.GT.DUZPRA)GOTO 80
      J=2
      IF(SINPRA(I,2).NE.PSIOP)GOTO 50
54    J=J+1
55    IF(TELO(M).NE.SINPRA(I,J))GOTO 63
      M=M+1
      GOTO 54
63    IF(RUKA(M).NE.0)GOTO 64
      RUKA(M)=L
      M=N
      GOTO 61
64    M=RUKA(M)
      GOTO 55
80    DUZTEL=L-1
      WRITE(5,1242)
      WRITE(5,1243)(GLAVA(I),ZAGLAVLJE(I),VRAT(I),LISTA(I),I=1,DUZLIS)
      WRITE(5,8080)
8080  FORMAT(/'      NO       TELO       RUKA'/)
      WRITE(5,8081)(I,TELO(I),RUKA(I),I=1,DUZTEL)
8081  FORMAT(' ',I6,6X,I6,6X,I6)
      WRITE(5,450)SINPRA(DUZPRA+1,1)
450   FORMAT(/' POCETNI SIMBOL  '/' ',I6)
      K=0
      DO 82 I=1,DUZLAN
      DO 82 J=1,SIRLAN
      IF(.NOT.LANAC(I,J))GOTO 82
      K=K+1
      VEKTOR(K)=I*100+J
82    CONTINUE
      DUZVEK=K
      WRITE(5,8200)
8200  FORMAT(/' VEKTOR'/)
      WRITE(5,8201)(VEKTOR(I),I=1,DUZVEK)
8201  FORMAT(' ',16I5)
      J=0
      K=0
      POC=1
      PN=1
      WRITE(5,4002)
4002  FORMAT(' NAVESTI IME DATOTEKE CIJI JE SADRZAJ',
     1       ' OPIS SEMANTIKE JEZIKA ? ',$)
      READ(5,4003)DUZID,IMEDAT
      IMEDAT(DUZID+1)=TACKA
      IMEDAT(DUZID+2)=SLOVOS
      IMEDAT(DUZID+3)=SLOVOE
      IMEDAT(DUZID+4)=SLOVOM
      DUZID=DUZID+4
      CALL ASSIGN(2,IMEDAT,DUZID)
      WRITE(5,401)(IMEDAT(I),I=1,DUZID-4)
401   FORMAT(' OPIS SEMANTIKE JEZIKA ',20A1/)
85    READ(2,100,END=90)DUZINA,ULAZ
      I=0
      WRITE(5,333)(ULAZ(N),N=1,DUZINA)
86    I=I+1
      IF(I.GT.DUZINA)GOTO 85
      IF(PN.EQ.-1)GOTO 87
      IF(ULAZ(I).EQ.RAZMAK)GOTO 86
87    J=J+1
      SEMPRA(J)=ULAZ(I)
```

```
      IF(SEMPRA(J).EQ.NAVOD)PN=-PN
      IF(ULAZ(I).NE.BOLAN UL U 86
      I=I+1
      SPISAK(J)=POC
      POC=J+1
      PN=1
      GO TO 86
      ULAZ(I)=
      DUZSPI=K
      CALL ASUIGM(  ,DUM,DAT )
      WRITE(3,9991)SEMPRA(DUZPRA+1,1)
 9991 FORMAT(1X,I5)
      WRITE(3,9991)DUZLIS
      WRITE(5,55661)DUZLIS
55661 FORMAT(' GLAVA SADRZI=',I5,' ELEMENATA')
      WRITE(3,9992)(GLAVA(I),I=1,DUZLIS)
 9992 FORMAT(1X,20I5)
      WRITE(3,9992)(ZAGLAVLJE(I),I=1,DUZLIS)
      WRITE(3,9992)(VRAT(I),I=1,DUZLIS)
      WRITE(3,9991)DUZKON
      WRITE(5,55662)DUZKON
55662 FORMAT(' KONST SADRZI=',I5,' ELEMENATA')
      WRITE(3,9993)(KONST(I),I=1,DUZKON)
 9993 FORMAT((5X,A1,I5,7(4X,A1,I5)))
      WRITE(3,9991)DUZTEL
      WRITE(5,55663)DUZTEL
55663 FORMAT(' TELO SADRZI=',I5,' ELEMENATA')
      WRITE(3,9992)(TELO(I),I=1,DUZTEL)
      WRITE(3,9992)(RUKA(I),I=1,DUZRUK)
      WRITE(3,9991)DUZVEK
      WRITE(5,55664)DUZVEK
55664 FORMAT(' VEKTOR SADRZI=',I5,' ELEMENATA')
      WRITE(3,9992)(VEKTOR(I),I=1,DUZVEK)
      WRITE(3,9991)DUZSPI
      WRITE(5,55665)DUZSPI
55665 FORMAT(' SPISAK SADRZI=',I5,' ELEMENATA')
      WRITE(3,9992)(SPISAK(I),I=1,DUZSPI)
      WRITE(3,9991)DUZSEM
      WRITE(5,55666)DUZSEM
55666 FORMAT(' SEMPRA SADRZI=',I5,' ELEMENATA')
      WRITE(3,9994)(SEMPRA(I),I=1,DUZSEM)
 9994 FORMAT(1X,100A1)
      WRITE(3,9995)
 9995 FORMAT(' KRAJ')
      STOP
      END
      IMPLICIT INTEGER*2 (A-Z)
C
      LOGICAL*1 PUT
C
      COMMON TELO(500),RUKA(500),ZAVRSNI(200),SEMPRA(1500),SPISAK(200),
     1 NIZ(700),GLAVA(100),ZAGLAVLJE(100),VRAT(100),NISKA(80),ULAZ(80),
     2 DRVO(500),DUZGLA,DUZNIZ
C
      COMMON /C/PAMTII,PAMTIJ,PAMTIK,
     1 REZULTAT,INDEKS,SKRETNICA,CILJ,POVRATAK,OZNAKA
C
      COMMON /D/GRANICA,ZONA(500,9)
C
      INTEGER*2 IRPILE(300),ICPILE(300),ILPILE(300),ITPILE(300),
     1 CIFRA(10),IZLAZ(50),IPPILE(300),ISPILE(300)
```

```
      EQUIVALENCE
     1     (PAMTII,SETII),
     2     (PAMTIJ,SETIJ),
     3     (PAMTIK,SETIK)
C
      DATA
     1     SLOVOG/'G'/,
     2     SLOVOP/'P'/,
     3     SLOVOS/'S'/,
     4     SLOVOT/'T'/,
     5     SLOVOU/'U'/,
     6     DOLAR/'$'/,
     7     NAVOD/1H'/,
     8     RAZMAK/' '/,
     9     TACKAZAREZ/';'/,
     1     KRAJ/9999/,
     2     CIFRA/'0','1','2','3','4','5','6','7','8','9'/
C
      CALL ASSIGN(3,'POM.DAT')
      READ (3,99991)POCETNISIMBOL
      WRITE(6,99991)POCETNISIMBOL
99991 FORMAT(1X,I5)
      READ(3,99991)DUZGLA
      WRITE(6,99991)DUZGLA
      READ(3,99992)(GLAVA(I),I=1,DUZGLA)
      WRITE(6,99992)(GLAVA(I),I=1,DUZGLA)
      READ(3,99992)(ZAGLAVLJE(I),I=1,DUZGLA)
      WRITE(6,99992)(ZAGLAVLJE(I),I=1,DUZGLA)
      READ(3,99992)(VRAT(I),I=1,DUZGLA)
      WRITE(6,99992)(VRAT(I),I=1,DUZGLA)
99992 FORMAT(1X,20I5)
      READ(3,99991)DUZZAV
      WRITE(6,99991)DUZZAV
      READ(3,99993)(ZAVRSNI(I),I=1,DUZZAV)
      WRITE(6,99993)(ZAVRSNI(I),I=1,DUZZAV)
99993 FORMAT((5X,A1,I5,9(4X,A1,I5)))
      READ(3,99991)DUZTEL
      WRITE(6,99991)DUZTEL
      READ(3,99992)(TELO(I),I=1,DUZTEL)
      WRITE(6,99992)(TELO(I),I=1,DUZTEL)
      READ(3,99992)(RUKA(I),I=1,DUZTEL)
      WRITE(6,99992)(RUKA(I),I=1,DUZTEL)
      READ(3,99991)DUZNIZ
      WRITE(6,99991)DUZNIZ
      READ(3,99992)(NIZ(I),I=1,DUZNIZ)
      WRITE(6,99992)(NIZ(I),I=1,DUZNIZ)
      READ(3,99991)DUZSPI
      WRITE(6,99991)DUZSPI
      READ(3,99992)(SPISAK(I),I=1,DUZSPI)
      WRITE(6,99992)(SPISAK(I),I=1,DUZSPI)
      READ(3,99991)DUZSEM
      WRITE(6,99991)DUZSEM
      READ(3,99994)(SEMPRA(I),I=1,DUZSEM)
      WRITE(6,99994)(SEMPRA(I),I=1,DUZSEM)
99994 FORMAT(1X,100A1)
C
      GRANICA=0
888   DO 887 I=1,80
887   NISKA(I)=0
880   I=0
      READ(5,886,END=500)DUZULA,ULAZ
886   FORMAT(Q,80A1)
```

```
      WRITE(6,4567)(ULAZ(J),J=1,DUZULA)
4567  FORMAT(' ',80A1)
      DUZULA=DUZULA+1
      ULAZ(DUZULA)=TACKAZAREZ
      DO 883 J=1,DUZULA
      IF(ULAZ(J).EQ.RAZMAK)GOTO 883
      DO 885 K=1,DUZZAV,2
885   IF(ULAZ(J).EQ.ZAVRSNI(K))GOTO 884
      WRITE(5,882)ULAZ(J)
      WRITE(6,882)ULAZ(J)
882   FORMAT(' ',A1,' NIJE ZAVRSAN SIMBOL.')
      GOTO 888
884   I=I+1
      NISKA(I)=ZAVRSNI(K+1)
883   CONTINUE
      DUZNIS=I
C
      CILJ=POCETNISIMBOL
      J=1
      K=1
      CALL SADRZAJ(NISKA(1),NEKOR2,NEKOR3,INDEKS)
      IF(INDEKS.EQ.0)GOTO 102
      REZULTAT=0
      OZNAKA=0
      SKRETNICA=0
      POVRATAK=1
C     MAIN LOOP
10    PAMTII=INDEKS
      PAMTIJ=J
      PAMTIK=K
220   IF(TELO(INDEKS+1).EQ.KRAJ)GO TO 50
      J=J+1
      IF (NISKA(J).EQ.TELO(INDEKS))GOTO 80
C     META-COMPONENT
      IF(.NOT.PUT(NISKA(J),TELO(INDEKS)))GOTO 40
C     META
30    CALL UPIS
      REZULTAT=0
      SKRETNICA=0
      CILJ=TELO(INDEKS)
      CALL SADRZAJ(NISKA(J),NEKOR2,NEKOR3,INDEKS)
      IF(INDEKS.EQ.0)GOTO 102
      POVRATAK=2
      GOTO 10
C     BACK-UP
40    J=SETIJ
      K=SETIK
C     ALT
50    IF(RUKA(INDEKS).EQ.0)GOTO 59
      INDEKS=RUKA(INDEKS)
      GOTO 220
59    INDEKS=SETII
C     RESULT
60    IF(TELO(INDEKS+1).NE.KRAJ)GOTO 100
      REZULTAT=TELO(INDEKS+2)
      SKRETNICA=0
      IF(TELO(INDEKS).NE.CILJ)GOTO 61
      SKRETNICA=1
      OZNAKA=-K
61    IF(.NOT.PUT(TELO(INDEKS),CILJ))GOTO 90
```

```
70      CALL UPIS
        REZULTAT=0
        SKRETNICA=0
        CALL SADRZAJ(TELO(INDEKS),NEKOR2,NEKOR3,INDEKS)
        IF(INDEKS.EQ.0)GOTO 102
        POVRATAK=3
        GOTO 10
C       OBJEKT
80      CALL UPIS
        REZULTAT=0
        SKRETNICA=0
        INDEKS=INDEKS+1
        POVRATAK=4
        GOTO 10
C       TEST-SWITCH
90      IF(SKRETNICA.EQ.1)GOTO 110
C       RECOVER
100     IF(RUKA(INDEKS).EQ.0)GOTO 101
        INDEKS=RUKA(INDEKS)
        GOTO 60
101     J=SETIJ
        K=SETIK
        SAVE2=4
        GO TO 130
C       NORMAL RETURN
120     SAVE2=0
C       REMAINDER OF RETURN PROCEDURE
130     SLEDECI=POVRATAK
        IF((SLEDECI+SAVE2).EQ.1)GOTO 121
        SAVE1=OZNAKA
        CALL ISPIS
        IF(SLEDECI.EQ.2)GOTO 150
        OZNAKA=SAVE1
        GOTO 151
150     REZULTAT=SAVE1
151     SKOK=SLEDECI+SAVE2
        GOTO(121,80,110,110,102,40,90,40),SKOK
C       OUTPUT SECTION
102     WRITE(5,103)
        WRITE(6,103)
103     FORMAT(' SINTAKSNA GRESKA')
        GOTO 888
C       NORMAL
110     IF(REZULTAT.EQ.0)GOTO 120
        DRVO(K)=REZULTAT
        K=K+1
        GOTO 120
121     DRVO(K)=OZNAKA
C       WRITE(5,122)
C       WRITE(6,122)
C122    FORMAT('USPESNA ANALIZA')
        DO 170 I=1,K
C       WRITE(5,123)I,DRVO(I)
C170    WRITE(6,123)I,DRVO(I)
C123    FORMAT(' ',2I6)
170     CONTINUE
250     ISP=0
6       NIVO=0
        I=0
        ISPEC=0
        IPROD=0
```

```
      RED=0
      KOLONA=0
      KOUNT=0
      IJ=1
310   IF(DRVO(K).LT.0)K=-DRVO(K)
      I=I+1
      IPPILE(I)=IPROD
      IRPILE(I)=RED
      ICPILE(I)=KOLONA
      ILPILE(I)=NIVO
      ITPILE(I)=KOUNT
      NIVO=NIVO+1
      ISPEC=ISPEC+1
      IPROD=ISPEC
      IFLAG=0
      RED=DRVO(K)
      KOLONA=0
      KOUNT=K
20    KOLONA=KOLONA+1
      ZNAK=PRONACI(RED,KOLONA)
      IF(ZNAK.EQ.DOLAR)GOTO 340
      IF(ZNAK.EQ.SLOVOG)GOTO 330
      IF(IFLAG.EQ.0)GOTO 11
      IF(ZNAK.NE.NAVOD)GOTO 12
      IFLAG=0
      GOTO 20
12    IZLAZ(IJ)=ZNAK
      IJ=IJ+1
      GOTO 20
11    IF(ZNAK.NE.NAVOD)GO TO 13
      IFLAG=1
      GOTO 20
13    IF(ZNAK.EQ.SLOVOT)GOTO 14
      IF(ZNAK.EQ.SLOVOU)GOTO 1170
      IF(ZNAK.EQ.SLOVOP)GOTO 1150
      IF(ZNAK.EQ.SLOVOS)GOTO 1150
      GOTO 20
C     TAB OUTPUT
14    KOLONA=KOLONA+1
      IF(PRONACI(RED,KOLONA).EQ.CIFRA(2))GOTO 19
      IF(PRONACI(RED,KOLONA).EQ.CIFRA(3))GOTO 17
      IF(PRONACI(RED,KOLONA).EQ.CIFRA(4))GOTO 15
      GOTO 20
15    IF(IJ.LE.20)GOTO 155
      JIM=3
      GOTO 22
155   DO 16 LAA=IJ,19
16    IZLAZ(LAA)=RAZMAK
      IJ=20
      GOTO 20
17    IF(IJ.LE.10)GOTO 156
      JIM=4
      GOTO 22
156   DO 18 LAA=IJ,9
18    IZLAZ(LAA)=RAZMAK
      IJ=10
      GOTO 20
19    IF(IJ.EQ.1)GOTO 20
      JIM=1
22    DO 21 LAA=IJ,50
21    IZLAZ(LAA)=RAZMAK
      IJ=1
```

```
      WRITE(5,211)IZLAZ
      WRITE(6,211)IZLAZ
211   FORMAT(' ',50A1)
      GOTO(20,290,155,156),JIM
C     LABEL OUTPUT OR STORE OUTPUT
1150  KOLONA=KOLONA+1.
      N1=IPROD
      N1=N1-(N1/10)*10
      N2=(IPROD-N1)/10
      N2=N2-(N2/10)*10
      N3=(IPROD-(N2*10)-N1)/100
      IF(ZNAK.EQ.SLOVOF)GOTO 160
      IZLAZ(IJ)=SLOVOS
      IZLAZ(IJ+1)=CIFRA(N3+1)
      IZLAZ(IJ+2)=CIFRA(N2+1)
      IZLAZ(IJ+3)=CIFRA(N1+1)
      IZLAZ(IJ+4)=PRONACI(RED,KOLONA)
      IJ=IJ+5
      GOTO 20
160   ISPILE(ISP+1)=SLOVOS
      ISPILE(ISP+2)=CIFRA(N3+1)
      ISPILE(ISP+3)=CIFRA(N2+1)
      ISPILE(ISP+4)=CIFRA(N1+1)
      ISPILE(ISP+5)=PRONACI(RED,KOLONA)
      ISP=ISP+5
      GOTO 20
1170  IZLAZ(IJ)=ISPILE(ISP-4)
      IZLAZ(IJ+1)=ISPILE(ISP-3)
      IZLAZ(IJ+2)=ISPILE(ISP-2)
      IZLAZ(IJ+3)=ISPILE(ISP-1)
      IZLAZ(IJ+4)=ISPILE(ISP)
      IJ=IJ+5
      ISP=ISP-5
      GOTO 20
330   KOLONA=KOLONA+1
      ZNAK=PRONACI(RED,KOLONA)
      DO 31 J=1,10
31    IF(ZNAK.EQ.CIFRA(J))GOTO 32
32    K=J+KOUNT-1
      GOTO 310
340   RED=IRPILE(I)
      KOLONA=ICPILE(I)
      IPROD=IPPILE(I)
      NIVO=ILPILE(I)
      KOUNT=ITPILE(I)
      I=I-1
      IF(NIVO.NE.0)GOTO 20
      JIM=2
      GOTO 22
290   GOTO 888
500   STOP
      END
      SUBROUTINE SADRZAJ(SIMBOL,RED,KOLONA,KAZALO)
      IMPLICIT INTEGER(A-Z)
      COMMON TELO(500),RUKA(500),ZAVRSNI(200),SEMPRA(1500),SPISAK(200),
     1 NIZ(700),GLAVA(100),ZAGLAVLJE(100),VRAT(100),NISKA(80),ULAZ(80),
     2 DRVO(500),DUZGLA,DUZNIZ
      POCETAK=1
      KRAJ=DUZGLA
      VECBILA=0
2     SREDINA=(POCETAK+KRAJ)/2
      IF(SREDINA.LT.1.OR.SREDINA.GT.DUZGLA)GOTO 5
```

```
      IF(VECBILA.EQ.SREDINA)GOTO 5
      VECBILA=SREDINA
      IF(SIMBOL-GLAVA(SREDINA))1,4,3
1     KRAJ=SREDINA-1
      GOTO 2
3     POCETAK=SREDINA+1
      GOTO 2
4     RED=SREDINA
      KOLONA=ZAGLAVLJE(RED)
      KAZALO=VRAT(RED)
      RETURN
5     IF(SIMBOL.NE.0)GOTO 15
      RED=0
      KOLONA=0
      KAZALO=0
      RETURN
15    WRITE(5,200)SIMBOL
      WRITE(6,200)SIMBOL
200   FORMAT(' ',A1,' NIJE U GLAVI')
      STOP
      END
```

```
      LOGICAL FUNCTION PUT*1(SIMB1,SIMB2)
      IMPLICIT INTEGER*2 (A-Z)
      COMMON TELO(500),RUKA(500),ZAVRSNI(200),SEMPRA(1500),SPISAK(200),
     1 NIZ(700),GLAVA(100),ZAGLAVLJE(100),VRAT(100),NISKA(80),ULAZ(80),
     2 DRVO(500),DUZGLA,DUZNIZ
      CALL SADRZAJ(SIMB1,RED,NEKOR3,NEKOR4)
      CALL SADRZAJ(SIMB2,NEKOR2,KOLONA,NEKOR4)
      CLAN=(RED*100)+KOLONA
      POCETAK=1
      KRAJ=DUZNIZ
      VECBILA=0
2     SREDINA=(POCETAK+KRAJ)/2
      IF(SREDINA.LT.1.OR.SREDINA.GT.DUZNIZ)GOTO 5
      IF(VECBILA.EQ.SREDINA)GOTO 5
      VECBILA=SREDINA
      IF(CLAN-NIZ(SREDINA))1,4,3
1     KRAJ=SREDINA-1
      GOTO 2
3     POCETAK=SREDINA+1
      GOTO 2
4     PUT=.TRUE.
      RETURN
5     PUT=.FALSE.
      RETURN
      END
```

```
      SUBROUTINE UPIS
      IMPLICIT INTEGER*2 (A-Z)
      INTEGER*2 Z(9)
      COMMON/C/PAMTII,PAMTIJ,PAMTIK,
     1 REZULTAT,INDEKS,SKRETNICA,CILJ,POVRATAK,OZNAKA
      COMMON/D/GRANICA,ZONA(500,9)
      EQUIVALENCE (PAMTII,Z(1))
      GRANICA=GRANICA+1
      DO 1 I=1,9
1     ZONA(GRANICA,I)=Z(I)
      RETURN
      END
```

```
      SUBROUTINE ISPIS
      IMPLICIT INTEGER*2 (A-Z)
      INTEGER*2 Z(9)
      COMMON/C/PAMTII,PAMTIJ,PAMTIK,
     1 REZULTAT,INDEKS,SKRETNICA,CILJ,POVRATAK,OZNAKA
      COMMON/D/GRANICA,ZONA(500,9)
      EQUIVALENCE (PAMTII,Z(1))
      DO 1 I=1,9
    1 Z(I)=ZONA(GRANICA,I)
      GRANICA=GRANICA-1
      RETURN
      END
      INTEGER FUNCTION PRONACI(I,J)
      IMPLICIT INTEGER(A-Z)
      COMMON TELO(500),RUKA(500),ZAVRSNI(200),SEMPRA(1500),SPISAK(200),
     1 NIZ(700),GLAVA(100),ZAGLAVLJE(100),VRAT(100),NISKA(80),ULAZ(80),
     2 DRVO(500),DUZGLA,DUZNIZ
      L=SPISAK(I)
      K=L+J-1
      PRONACI=SEMPRA(K)
      RETURN
      END
```

```
1054
  55
   3     6     8    10    12    13    18    19    20    21    22    23    24    25    26    27    28    29    30    31
  32    33    34    35    36    37    38    39    40    41    42    43    44    45    46    47    48    49    50    51
  52    53    55  1001  1002  1004  1005  1007  1009  1011  1014  1015  1016  1017  1054
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     1     2     3     4     5     6     7     8     9    10    11    12
   0     1     0     0     5     0    10    13    16    19    22    25    28    31    34    37    40    43    46    49
  52    55    58    61    64    67    70    73    76    79    82    85    88    91    94    97   100   103   106   109
 112   115     0   118   122   138   143   151   159   162   165   168   171   174     0
  86
   =     3     +     6     *     8     ž    10     (    12     )    13     ^    18     B    19     C    20     D    21
   E    22     F    23     G    24     H    25     I    26     J    27     K    28     L    29     M    30     N    31
   O    32     P    33     Q    34     R    35     S    36     T    37     U    38     V    39     W    40     X    41
   Y    42     Z    43     0    44     1    45     2    46     3    47     4    48     5    49     6    50     7    51
   8    52     9    53     ;    55
 180
1005  1004  9999     3  1004    13  1009  9999    10  1015  9999    19  1015  9999    20  1015  9999    21  1015  9999
  22  1015  9999    23  1015  9999    24  1015  9999    25  1015  9999    26  1015  9999    27  1015  9999    28  1015
9999    29  1015  9999    30  1015  9999    31  1015  9999    32  1015  9999    33  1015  9999    34  1015  9999    35
1015  9999    36  1015  9999    37  1015  9999    38  1015  9999    39  1015  9999    40  1015  9999    41  1015  9999
  42  1015  9999    43  1015  9999    44  1016  9999    45  1016  9999    46  1016  9999    47  1016  9999    48  1016
9999    49  1016  9999    50  1016  9999    51  1016  9999    52  1016  9999    53  1016  9999    54    55  1054  9999
  55  1011  9999    11     3  1004  1001  9999     1  1015  1002  9999    14  1016  1002  9999    15     6  1005  1004
9999     4  1004  9999     2     8  1007  1005  9999     6  1005  9999     5    10  1009  1007  9999     8  1007  9999
   7  1009  9999     9  1011  9999    12  1002  9999    13  1017  9999    17  1014  9999    16  1016  1017  9999    18
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   0   125     0     0   130     0     0     0     0   134     0     0     0     0     0     0     0     0     0     0
   0     0   146     0     0     0     0     0     0     0   154     0     0     0     0     0     0     0     0     0
   0     0     0     0     0     0     0     0     0     0     0     0     0   177     0     0     0     0     0     0
 368
```

SKUP METALINGVISTIČKIH FORMULA U KODIRANOM OBLIKU

```
 203  503  504  505  506  701  702  703  704  705  706  707  709  712  801  802  803  804  805  806
 807  809  812  901  902  903  904  905  906  907  909  912 1001 1002 1003 1004 1005 1006 1007 1009
1012 1101 1102 1103 1104 1105 1106 1107 1109 1112 1201 1202 1203 1204 1205 1206 1207 1209 1212 1301
1302 1303 1304 1305 1306 1307 1309 1312 1401 1402 1403 1404 1405 1406 1407 1409 1412 1501 1502 1503
1504 1505 1506 1507 1509 1512 1601 1602 1603 1604 1605 1606 1607 1609 1612 1701 1702 1703 1704 1705
1706 1707 1709 1712 1801 1802 1803 1804 1805 1806 1807 1809 1812 1901 1902 1903 1904 1905 1906 1907
1909 1912 2001 2002 2003 2004 2005 2006 2007 2009 2012 2101 2102 2103 2104 2105 2106 2107 2109 2112
2201 2202 2203 2204 2205 2206 2207 2209 2212 2301 2302 2303 2304 2305 2306 2307 2309 2312 2401 2402
2403 2404 2405 2406 2407 2409 2412 2501 2502 2503 2504 2505 2506 2507 2509 2512 2601 2602 2603 2604
2605 2606 2607 2609 2612 2701 2702 2703 2704 2705 2706 2707 2709 2712 2801 2802 2803 2804 2805 2806
2807 2809 2812 2901 2902 2903 2904 2905 2906 2907 2909 2912 3001 3002 3003 3004 3005 3006 3007 3009
3012 3101 3102 3103 3104 3105 3106 3107 3109 3112 3201 3202 3203 3204 3205 3206 3207 3209 3212 3303
3304 3305 3306 3307 3308 3310 3311 3403 3404 3405 3406 3407 3408 3410 3411 3503 3504 3505 3506 3507
3508 3510 3511 3603 3604 3605 3606 3607 3608 3610 3611 3703 3704 3705 3706 3707 3708 3710 3711 3803
3804 3805 3806 3807 3808 3810 3811 3903 3904 3905 3906 3907 3908 3910 3911 4003 4004 4005 4006 4007
4008 4010 4011 4103 4104 4105 4106 4107 4108 4110 4111 4203 4204 4205 4206 4207 4208 4210 4211 4412
4501 4502 4503 4504 4505 4506 4507 4512 4603 4703 4704 4803 4804 4805 4903 4904 4905 5003 5004 5005
5006 5103 5104 5105 5106 5107 5201 5202 5203 5204 5205 5206 5207 5212 5303 5304 5305 5306 5307 5308
5311 5403 5404 5405 5406 5407 5408 5411
```

MATRICA 'LANAC'

```
  55
   1   15   18   21   48   51   78   81  111  123  126  129  132  135  140  145  148  151  156  160
 164  168  172  176  180  184  188  192  196  200  204  208  212  216  220  224  228  232  236  240
 244  248  252  256  260  264  268  272  276  280  284  288  292  296  300
 302
G1T2'AUM'T3G2$G1$G1$G1T2'AUM'T3S1G2T2'SAB'T3S1$G1$G1T2'AUM'T3S1G2T2'MNO'T3S1$G1$G1T2'AUM'T3S1G2T2'ST
EPEN'T3S1$T2'MUA'T3G1$G1$G1$G1$G1$G2G1$G2G1$G1$G1$G2G1$'A'$'B'$'C'$'D'$'E'$'F'$'G'$'H'$'I'$'J'$'K'$'
L'$'M'$'N'$'O'$'P'$'Q'$'R'$'S'$'T'$'U'$'V'$'W'$'X'$'Y'$'Z'$'0'$'1'$'2'$'3'$'4'$'5'$'6'$'7'$'8'$'9'$G
1$
```

SEMANTIČKA PRAVILA