Prevođenje programskih jezika

[Rješenja zbirke zadataka]

2012.

1. UVOD

1. Navedite prednosti i nedostatke uporabe jezičnih procesora.

PREDNOSTI

- jednostavnost korištenja programa za korisnika
- bez potrebe za poznavanjem arhitekture računala
- približavanje jezika programiranja području primjene
- lakši razvoj i razumjevanje programa
- lakše otklanjanje grešaka i dokumentiranje
- prenosivost i skraćeno vrijeme rješavanja problema

NEDOSTACI

- produljenje vremena potrebnog za prevođenje korisničkog jezika u izvodivi strojni program
- strojni program je neučinkovit jer je generiran iz korisničkog programa koji je oslobođen svih detalja arhitekture računala
- nemogućnost korištenja svih posebnosti građe računala na kojoj se izvodi program
- potrebno izgraditi razvojnu okolinu koja omogućuje ispitivanje i nadzor izvođenja korisničkih programa

2. Navedite tri jezika koji su vezani za definiciju jezičnog procesora.

L_i izvorni jezik

L_c ciljni jezik

L_g jezik izgradnje

$$\text{JP}\ \begin{matrix} L_i \rightarrow L_c \\ L_g \end{matrix}$$

3. Navedite koja pravila se susreću kod programskih jezika te ih objasnite.

Pravila određuju dozvoljene oblike osnovnih elemenata jezika (*varijable, ključne riječi, konstante, operatori...*) (LEKSIČKA PRAVILA)

- ***varijabla*** niz slova i brojaka koji započinje slovom
- ***ključne riječi*** rezervirane riječi
- ***cjelobrojna konstanta*** niz dekadskih znamenki
- ***decimalna konstanta*** ima barem jedan broj ispred točke

- ***način na koji se koriste leksičke jedinke za gradnju ispravne naredbe*** ***(*** **ako(**<izraz>**)** ***)***
- ***struktura čitavog programa***
- ***deklaracija varijabli se nalazi na početku programa***

4. Objasnite samoprevodioca.

Samoprevoditelj je jezični procesor koji je izgrađen primjenom izvornog jezika:

$$JP\ {}_{A}^{A\to B}$$

Izvorni jezik *A* i jezik izgradnje *A* su identični jezici.

5. Objasnite cilj i osnovne korake postupka samopodizanja.

Samopodizanje je postupak prenošenja jezičnog procesora s računala ***a*** na računalo ***b*** koje je različite arhitekture.

Za računalo A postoji jezični procesor $JP_A^{L\to A}$. Strojni jezik A moguće je izravno izvesti na računalu A.
Želimo izgraditi jezični procesor za računalo B $JP_B^{L\to B}$.

Prvo gradimo ***samoprevoditelja*** - $JP_L^{L\to B}$. Samoprevoditelj se prevede primjenom jezičnog procesora $JP_A^{L\to A}$ na računalu i dobije se premosni prevoditelj $JP_A^{L\to B}$. Zatim na računalu A pokrenemo premosni prevoditelj $JP_A^{L\to B}$ te se njegovom primjenom prevodi samoprevoditelj $JP_L^{L\to B}$ u jezični procesor $JP_B^{L\to B}$ te se taj jezični procesor može izvoditi na računalu B.

6. Objasnite što je funkcija preslikavanja jezičnog procesora i navedite njezine vrste.

Funkcija preslikavanja određuje koliko se naredbi izvornog programa prevodi u koliko naredbi ciljnog programa.
1 – 1; 1 – M; M – 1; *složena funkcija preslikavanja;*

7. Objasnite podjelu jezičnih procesora s obzirom na broj prolazaka kroz izvorni program.

Jednoprolazni procesori ne spremaju rezultat svog izvođenja direktno u memoriju, već se pojedini elementi (niz leksičkih jedinki, sintaksno stablo, međukod...) dinamički i pojedinačno razmjenjuju između pojedinih koraka rada jezičnog procesora.
Višeprolazni procesori spremaju rezultat svog izvođenja izravno u memoriju. Npr. prvi prolazak je tijekom leksičke analize, a drugi tijekom sintaksne analize.

8. Navedite i objasnite korake analize izvornog programa.

Analiza rastavlja izvorni program u sastavne dijelove, provjerava pravila jezika, prijavljuje greške i zapisuje izvorni program primjenom različitih struktura podataka u memoriju računala.

LEKSIČKA ANALIZA

Linearna analiza znakova izvornog programa koja provjerava jesu li leksičke jedinke u izvornom programu pravilno napisane.
Leksički analizator prevodi izvorni program u niz leksičkih jedinki.

SINTAKSNA ANALIZA

Provjerava dali su pravilno napisane strukture naredbi, strukture djelova programa i struktura cijelog programa (*SINTAKSNA PRAVILA*).
Provjerava da li niz leksičkih jedinki zadovoljava sintaksnim pravilima zadanu hijerarhijsku strukturu izvornog programa.
Sintaksni analizator gradi sintaksno stablo ili ispisuje poruku o pogrešci.

SEMANTIČKA ANALIZA

Semantička pravila su interpretacijska pravila koja povezuju izvođenje izvornog programa s ponašanjem računala. Semantika jezika određuje skup dopuštenih značenja. (*npr. FLOAT + INT = FLOAT*).
Semantički analizator sačinjava niz potprograma koji provjeravaju deklaraciju varijabli, tipove podataka, indeksiranje, parametre potprograma i tijek izvođenja programa. Ti potprogrami se izvode istovremeno s potprogramima sintaksnog analizatora.
Semantička analiza je most između faze analize i faze sinteze.

9. Objasnite razredbu jezičnih procesora s obzirom na dinamiku izvođenja procesa prevođenja.

Kompilator prevodi naredbe onim redoslijedom kojim su napisane u izvornom programu. Izvođenje ciljnog programa započinje nakon što završi prevođenje. Dinamika kompilatora opisuje se događajima procesa prevođenja izvornog programa i događajima izvođenja ciljnog programa.

Interpretator ne prevodi cijeli izvorni program u ciljni program, već prevede jednu naredbu i odmah je izvede. Naredbe se ne prevode onim redoslijedom kojim su navedene u izvornom programu, već je redoslijed prevođenja određen redoslijedom njihova izvođenja.

Dinamički interpretator izvodi paralelno ili konkurentno procese prevođenja izvornog programa i izvođenje. Procesi su međusobno povezani FIFO spremnikom. Proces prevođenja izvornog programa stavlja prevedene naredbe u spremnik, a preoces izvođenja ciljnog programa uzima prevedene naredbe iz spremnika i izvodi ih. Dinamički interpretator čine tri procesa.

10. Nabrojite i ukratko objasnite svaki od osnovnih koraka u izgradnji jezičnih procesora.

Definicija jezičnog procesora – precizno određuje pravila izvornog jezika, svojstva ciljnog jezika, svojstva arhitekture računala, brzine rada, jezik izgradnje...
Strukturiranje jezičnog procesora – organizacija jezičnog procesora i razrada svih koraka rada.
Programsko ostvarenje jezičnog procesora – izbor programskog jezika, gradnja programa jezičnog procesora i ispitivanje programa jezičnog procesora.
Ocjena svojstava jezičnog procesora – ispitivanje dosljednosti te zadovoljavanje definiranih svojstava.
Održavanje jezičnog procesora – poboljšavanje i prilagođavanje jezičnog procesora, te ispravljanje grešaka.

11. Koristeći Hoareov sustav oznaka CSP, opišite vrste jezičnih procesora s obzirom na dinamiku izvođenja.

KOMPILATOR

$$e_1^p \to e_2^p \to e_3^p \to \cdots \to e_N^p \to e_{x_1}^i \to e_{x_2}^i \to e_{x_3}^i \to \cdots \to e_{x_M}^i \to STOP$$

INTERPRETATOR

$$e_{x_1}^p \to e_{x_1}^i \to e_{x_2}^p \to e_{x_2}^i \to \cdots \to e_{x_M}^p \to e_{x_M}^i \to STOP$$

DINAMIČKI INTERPRETATOR

$$PREVOĐENJE \gg SPREMNIK^B \gg IZVOĐENJE$$

12. Objasnite kompilatore i interpretatore, odnosno njihovu razliku, ne koristeći Hoareov sustav oznaka CSP.

Kompilator prevodi naredbe onim redoslijedom kojim su napisane u izvornom programu. Izvođenje ciljnog programa započinje nakon što završi prevođenje. Dinamika kompilatora opisuje se događajima procesa prevođenja izvornog programa i događajima izvođenja ciljnog programa.

Interpretator ne prevodi cijeli izvorni program u ciljni program, već prevede jednu naredbu i odmah je izvede. Naredbe se ne prevode onim redoslijedom kojim su navedene u izvornom programu, već je redoslijed prevođenja određen redoslijedom njihova izvođenja.

Dinamički interpretator izvodi paralelno ili konkurentno procese prevođenja izvornog programa i izvođenje. Procesi su međusobno povezani FIFO spremnikom. Proces prevođenja izvornog programa stavlja prevedene naredbe u spremnik, a preoces izvođenja ciljnog programa uzima prevedene naredbe iz spremnika i izvodi ih. Dinamički interpretator čine tri procesa.

13. Navedite faze rada jezičnog procesora od kojih se sastoje faze analize izvornog programa i faze sinteze ciljnog programa.

FAZA ANALIZE:

- Leksička analiza
- Sintaksna analiza
- Semantička analiza

FAZA SINTEZE:

- Generiranje međukoda
- Strojno nezavisno optimiranje
- Generiranje strojnog programa
- Strojno zavisno optimiranje
- Priprema strojnog programa za izvođenje

2. LEKSIČKA ANALIZA

23. Opišite dinamiku izvođenja leksičke analize.

Suradnja leksičkog analizatora i sintaksnog analizatora ostvaruje se:

- Putem poziva potprograma
- Razmjenom tablice uniformnih znakova

Ako se suradnja ostvari ***putem poziva potprograma,*** onda je leksički analizator potprogram sintaksnog analizatora. Sintaksni analizator poziva leksički analizator u trenutku kada se zahtijeva dohvat novog uniformnog znaka. Razmjena podataka izvodi se u tri koraka:

1. Sintaksni analizator poziva leksički analizator sa zahtjevom za sljedećim uniformnim znakom
2. Leksički analizator čita znakove izvornog programa sve dok ne odredi klasu sljedeće leksičke jedinke
3. Leksički analizator vraća sintaksnom analizatoru uniformni znak

Ako se suradnja ostvari ***razmjenom čitave tablice uniformnih znakova***, onda je leksički analizator zasebni prolaz jezičnog procesora. Nakon što spremi tablicu uniformnih znakova u zasebnu datoteku, prestaje rad leksičkog analizatora. Rad sintaksnog analizatora pokreće se čitanjem uniformnih znakova iz datoteke koji je prethodno pripremio leksički analizator.

24. Opišite program simulator leksičkog analizatora zasnovan na tablici prijelaza ε-NKA.

1) u program simulator ugradi se tablica prijelaza ε-NKA $M=(Q, \sum, \delta, p_o, F)$

2) u ulazni spremnik spremi se cijeli izvorni program

3) neka je R skup stanja ε-NKA – na početku rada skup stanja $R = \varepsilon\text{-OKRUŽENJE}(p_o)$

4) čitamo znak (a) po znak i računamo $R = \varepsilon\text{-OKRUŽENJE}(\delta(Q, a))$ te izabiremo onaj q_i iz skupa P ($P = R \cap F \neq \{\}$) kojemu je pridružen regularni izraz r_i koji je naveden prije ostalih regularnih izraza koji su navedeni u skupu P – korak 4 ponavljamo sve do prijelaza skupa R u prazni skup ($R \neq \{\}$)

5) nakon što skup R postane prazan na temelju vrijednosti varijable *Izraz* određuje se klasa jedinke

25. Opišite način izrade globalne tablice znakova.

Tablica se može izgraditi primjenom ***linearne liste*** (koja ima dvije kazaljke na početak i kraj radi dodavanja i pretraživanja), ***uređene liste*** (koja je poredana abecednim redoslijedom), ***binarnog stabla*** i ***tehnikom raspršenog adresiranja*** (računanje vrijednosti ključa).

26. Nabrojite i objasnite osnovne klase leksičkih jedinki.

- ***Ključne riječi*** (*npr. ako, onda, inače...*)
- ***Operatori*** (*npr. zbrajanje, oduzimanje, množenje...*)
- ***Specijalni znakovi*** (*npr. zagrade, zarez, točka...*)
- ***Identifikatori*** (*npr. imena varijabli, polja, programa, potprograma...*)
- ***Konstante*** (*npr. cjelobrojne, znakovne, s posmačnim zarezom...*)

27. Opišite program simulator leksičkog analizatora zasnovan na tablici prijelaza DKA.

1) u program simulator ugradi se tablica prijelaza DKA $M' = (Q', \sum', \delta', p_o', F')$ koji se izgradi na temelju ε-NKA $M=(Q, \sum, \delta, p_o, F)$. Doda se i neprihvatljivo stanje Ø. Nema li ε-NKA definiranih prijelaza, DKA prijelazi u stanje Ø bez obzira na znak

2) u ulazni spremnih spremi se cijeli izvorni program

3) program simulator započinje rad čitanjem krajnje lijevog znaka ulaznog spremnika i koristi četiri varijable (kazaljke) tijekom analize niza:

- ***Početak*** početak neanaliziranog dijela niza
- ***Završetak*** posljednji pročitani znak
- ***Posljednji*** posljednji znak najduljeg prepoznatog niza
- ***Izraz*** vrijednost oznake regularnog izraza najduljeg niza

4) čita znak (a) po znak te računa prijelaze $q' = \delta'(q', a)$ i izabire onaj q_i iz skupa $P=\{q_o,q_1,q_2...q_n\} \cap F$ kojemu je pridružen regularni izraz r_i. Čitanje se prekida kada se ispuni uvjet $q' = Ø$.

5) nakon što DKA prijeđe u stanje Ø na temelju vaerijable Izraz određuje se klasa leksičke jedinke

6) analizom svih znakova ulaznog spremnika završava rad simulatora – ima li još znakova koji nisu analizirani, simulator nastavlja korakom 4

28. Objasnite kako leksički analizator utvrđuje leksičku pogrešku i navedite dva postupka oporavka od pogreške.

Postupak odbacivanja krajnje ljevog znaka – ako niz znakova *w* nema nijedan prefiks koji je deifiniran barem jednim od zadanih regularnih izraza, odbaci se krajnje ljevi znak a_1 i ispiše taj znak i nastavlja s analizom ostatka niza.

29. Navedite namjenu programa Lex i opišite strukturu ulazne datoteke Lexa.

Omogućuje automatiziranje izgradnje leksičkog analizatora i koristi se za rješavanje problema koje je moguće svesti na problem prihvaćanja regularnih jezika. Ulazna datoteka se zadaje na sljedeći način:

```
%{
        DEKLARACIJE
%}

        REGULARNE DEFINICIJE

%%
        PRAVILA PREVOĐENJA
%%

        POMOĆNE PROCEDURE
```

30. Opišite podatkovne strukture leksičkog analizatora.

Podatkovnu strukturu leksičkog analizatora čine *izvorni program*, *tablica uniformnih znakova* i *tablica znakova* (tablica identifikatora, tablica konstanti i tablica ključnih riječi).

TABLICA UNIFORMNIH ZNAKOVA

U tu tablici su zapisani uniformni znakovi onim redoslijedom kojim su leksičke jedinke zadane u izvornom programu. Nakon što leksički analizator odredi klasu leksičke jedinke, njezin uniformni znak zapisuje se u tablicu uniformnih znakova. Zapis u tablici može imati dvije kazaljke. ***Prva kazaljka*** je sam uniformni znak leksičke jedinke i pokazuje na jednu od tablica (*tablicu uniformnih znakova [IDN], tablicu konstanti [KON], ili tablicu ključnih riječi, operatora i specijalnih znakova [KROS]*). ***Druga kazaljka*** pokazuje na mjesto zapisa leksičke jedinke u tablici odabranoj primjenom prve kazaljke.
Tablica znakova funkcijski se razlaže na tri tablice:

- Tablica identifikatora
- Tablica konstanti
- Tablica ključnih riječi, operatora i specijalnih znakova

31. Objasnite metode opisa leksičkih jedinki, pravila određivanja klasa leksičkih jedinki i postupak grupiranja leksičkih jedinki koje se koriste za izgradnju generatora leksičkog analizatora.

Leksička pravila zadaju se ***primjenom regularnih izraza i regularnih definicija***. ***Određivanje klasa laksičkih jedinki*** zasniva se na sljedeća dva pravila:

1) Za sve leksičke jedinke definira se zasebni regularni izraz
2) Ako je niz definiran primjenom dvaju regularnih izraza, uzima se onaj koji je prije zapisan u listi regularnih izraza

Tijekom ***grupiranja leksičkih jedinki***, traži se najdulji prefiks niza w koji je definiran barem jednim regularnim izrazom.

32. Navedite i ukratko objasnite pet zadataka leksičkog analizatora.

Leksički analizator slijedno čita tekst izvornog programa, znak po znak, stvara učinkovit zapis znakova izvornog programa, odbacuje znakove koji se ne koriste u daljnjim koracima rada jezičnog procesora, grupira znakove u leksičke jedinke, određuje klase leksičkim jedinkama, provjerava leksička pravila, pronalazi greške, određuje mjesto pogreške, zapisuje parametre leksičkih jedinki u tablicu znakova i čuva tekstualnu strukturu izvornog programa.

Leksički analizator:

- ***određuje klasu leksičkih jedinki***
- ***kodira znakove izvornog programa*** i za to koristi jedan od standardnih kodova (*npr. ASCII kod*)
- ***odbacuje sve znakove*** koji se ne koriste u daljnjim fazama rada jezičnog procesora
- ***provjerava dali leksička jedinka zadovoljava pravila klase u koju je svrstana***; za opis pravila klase najčešće se koriste regularni izrazi
- ***određuje mjesto pogreške*** u izvornom programu i opiše pogrešku
- ***stvara zapise u tablici znakova*** za sve leksičke jedinke izvornog programa; ***tablica znakova*** čini osnovnu podatkovnu strukturu leksičkog analizatora. U tablicu znakova se spremaju parametri leksičkih jedinki (*npr. tip konstante*)

33. Navedite i na primjeru objasnite dva osnovna načina razrješavanja nejednoznačnosti u leksičkoj analizi.

Nejednoznačnost se razrješava na dva načina – ***pretraživanjem desnog konteksta*** i ***pretraživanjem lijevog konteksta***.

Pretraživanje desnog konteksta zadaje se : **r / r'** gdje su r i r' regularni izrazi. Niz u ulaznom spremniku grupira se u laksičku jedinku definiranu izrazom r ako i samo ako iza niza definiranog regularnim izrazom r slijedi niz koji je definiran izrazom r'.

DO / (slovo+brojka)* = (slovo+brojka)*,

Za ***pretraživanje lijevog konteksta*** definiraju se ***dodatna stanja***. Ulazak i izlazak iz dodatnih stanja zadaje se u akcijama pridruženim pojedinim regularnim izrazima, a zadaje se: **<ImeStanja> r** što označava da je niz definiran regularnim izrazom r ako i samo ako je simulator u dodatnom stanju *ImeStanja*.

A = - 5 + C

34. Navedite i objasnite varijable koje koristi simulator zasnovan na tablici prijelaza DKA. U ovisnosti o navedenim varijablama, objasnite postupak simulatora za grupiranje i određivanje klase leksičke jedinke.

- ***Početak*** početak neanaliziranog dijela niza
- ***Završetak*** posljednji pročitani znak
- ***Posljednji*** posljednji znak najduljeg prepoznatog prefiksa niza
- ***Izraz*** vrijednost oznake regularnog izraza najduljeg niza

Ukoliko je ***Izraz == 0*** simulator nije pronašao nijedan prefiks niza znakova izvornog programa koji je definiran barem jednim regularnim izrazom. Ako je ***Izraz != 0***, klasa leksičke jedinke određuje se na temelju vrijednosti varijable ***Izraz***.

35. Objasnite postupak razrješenja nejednoznačnosti u leksičkoj analizi pretraživanjem lijevog konteksta.

Za ***pretraživanje lijevog konteksta*** definiraju se ***dodatna stanja***. Ulazak i izlazak iz dodatnih stanja zadaje se u akcijama pridruženim pojedinim regularnim izrazima, a zadaje se: **<ImeStanja> r** što označava da je niz definiran regularnim izrazom r ako i samo ako je simulator u dodatnom stanju *ImeStanja*.

A = - 5 + C

36. Općenito objasnite (ne na primjeru) postupak primjene pretraživanja desnog konteksta za razrješavanje nejednoznačnosti u leksičkoj analizi. Potrebno je definirati kako se u regularnim izrazima zadaje desni kontekst, kako se na osnovi regularnih izraza sa zadanim desnim kontekstom stvara konačni automat te kako se dobiveni konačni automat koristi za leksičku analizu.

Pretraživanje desnog konteksta zadaje se : **r / r'** gdje su r i r' regularni izrazi. Niz u ulaznom spremniku grupira se u laksičku jedinku definiranu izrazom r ako i samo ako iza niza definiranog regularnim izrazom r slijedi niz koji je definiran izrazom r'. U postupku izgradnje ε-NKA, na temelju regularnih izraza ***r/r'*** definira se regularni izraz ***rr'***. Tijekom rada program simulator koristi regularne izraze r i ***rr'***. Pronađe li simulator prefiks niza znakova koji je definiran regularnim izrazom ***rr'***, grupiraju se samo znakovi koji su definirani regularnim izrazom r.

37. Navedite pravila za određivanje klase leksičke jedinke i grupiranje znakova u leksičke jedinke.

1) Za sve leksičke jedinke definira se zasebni regularni izraz

2) Ako je niz definiran primjenom dvaju regularnih izraza, uzima se onaj koji je prije zapisan u listi regularnih izraza

3) Kod grupiranja leksičkih jedinki traži se najdulji prefiks niza w koji je definiran barem jednim regularnim izrazom

38. Općenito definirajte (ne na primjeru) ulaze i izlaze iz programa generatora leksičkog analizatora i programa leksičkog analizatora ako je leksički analizator ostvaren kao zasebni prolaz jezičnog procesora.

GENERATOR LEKSIČKOG ANALIZATORA

ULAZ: opis procesa leksičkog analizatora (regularne definicije, stanja leksičkog analizatora, imena leksičkih jedinki, pravila leksičkog analizatora)
IZLAZ: izvorni kod leksičkog analizatora napisan u jeziku izgradnje

LEKSIČKI ANALIZATOR

ULAZ: izvorni program
IZLAZ: tablica uniformnih znakova

39. Navedite strukture podataka pogodne za ostvarenje tablice znakova i asimptotsku složenost osnovnih operacija nad tablicom znakova za svaku predloženu strukturu podataka.

LINEARNA LISTA

TRAŽENJE $O(n)$
DODAVANJE $O(1)$

UREĐENA LISTA

TRAŽENJE $O(log_2 n)$
DODAVANJE $O(n)$

BINARNO STABLO

TRAŽENJE $O(log_2 n)$
DODAVANJE $O(1)$

RASPRŠENO ADRESIRANJE

TRAŽENJE $O(1)$
DODAVANJE $O(1)$

40. Opišite algoritam leksičkog analizatora zasnovan na tablici prijelaza DKA.

1) u program simulator ugradi se tablica prijelaza DKA $M' = (Q', \sum', \delta', p_0', F')$ koji se izgradi na temelju ε-NKA $M=(Q, \sum, \delta, p_0, F)$. Doda se i neprihvatljivo stanje Ø. Nema li ε-NKA definiranih prijelaza, DKA prijelazi u stanje Ø bez obzira na znak
2) u ulazni spremnih spremi se cijeli izvorni program
3) program simulator započinje rad čitanjem krajnje lijevog znaka ulaznog spremnika i koristi četiri varijable (kazaljke) tijekom analize niza:

- ***Početak*** početak neanaliziranog dijela niza
- ***Završetak*** posljednji pročitani znak
- ***Posljednji*** posljednji znak najduljeg prepoznatog niza
- ***Izraz*** vrijednost oznake regularnog izraza najduljeg niza

4) čita znak (a) po znak te računa prijelaze $q' = \delta'(q', a)$ i izabire onaj q_i iz skupa $P=\{q_0,q_1,q_2...q_n\} \cap F$ kojemu je pridružen regularni izraz r_i. Čitanje se prekida kada se ispuni uvjet $q' = Ø$.
5) nakon što DKA prijeđe u stanje Ø na temelju vaerijable Izraz određuje se klasa leksičke jedinke
6) analizom svih znakova ulaznog spremnika završava rad simulatora – ima li još znakova koji nisu analizirani, simulator nastavlja korakom 4

3. SINTAKSNA ANALIZA

65. Definirajte relacije *IspodZnaka* i *ReduciranZnakom* za parsiranje tehnikom *Pomakni-Pronađi*.

Relacija ***IspodZnaka(A, x)*** vrijedi ako je ispunjen jedan od sljedećih dva uvjeta:

1) znak ***A*** je izravno ispred znaka ***B*** na desnoj strani barem jedne produkcije zadane gramatike – vrijedi relacija ***IzravnoIspredZnaka(A, B)*** a znak ***x*** započinje barem jedan niz generiran iz znaka ***B*** – vrijedi ***x*** ∈ ***ZAPOČINJE(B)***

2) ***A*** je oznaka dna stoga ***∇*** i ***x*** ∈ ***ZAPOČINJE(<S>)*** , gdje je ***<S>*** početni nezavršni znak gramatike

Relacija ***ReduciranZnakom(A, x)*** vrijedi ako je ispunjen jedan od sljedećih dva uvjeta:

1) znak ***A*** je izravno ispred znaka ***B*** na desnoj strani barem jedne produkcije zadane gramatike – vrijedi relacija ***IzravnoIspredZnaka(A, B)*** a znak ***x*** započinje barem jedan niz generiran iz znaka ***B*** – vrijedi ***x*** ∈ ***ZAPOČINJE(B)***

2) ***A*** je oznaka dna stoga ***∇*** i ***x*** ∈ ***ZAPOČINJE(<S>)*** , gdje je ***<S>*** početni nezavršni znak gramatike

66. Navedite uvjete pod kojima je kontekstno neovisna gramatika ujedno i *S-gramatika*.

Kontekstno neovisna gramatika je ***S-gramatika*** ako:

1) desna strana bilo koje produkcije započinje **završnim** znakom gramatike
2) desna strana nijedne produkcije **nije prazni niz ε**
3) ako više produkcija ima isti nezavršni znak na lijevoj strani, onda s desne strane tih produkcija započinju različitim završnim znakovima

67. Navedite uvjete pod kojima je kontekstno neovisna gramatika ujedno i *LL(1)-gramatika*.

Kontekstno neovisna gramatika jest ***LL(1)-gramatika*** ako vrijedi:

ako više produkcija ima isti nezavršni znak na lijevoj strani, onda njihovi skupovi ***PRIMJENI*** nemaju zajedničkih elemenata

68. Navedite postupak izgradnje kanonskog *LR- parsera* na temelju izgrađenog DKA.

1) gramatiku **G** proširimo tako da dodamo novo početno stanje ***S'*** i novu produkciju $S' \rightarrow S$

2) dvije tablice – ***Akcija*** (stupci su završni znakovi gramatike i završni znak ⊥) i ***NovoStanje*** (stupci su nezavršni znakovi)

3) tablica ***Akcija*** se popunjava:

   a. ako je *LR(1)* stavka $A \rightarrow \alpha \bullet a\beta, \{a_1, a_2, a_3 \ldots a_n\}$ u stanju ***s*** i ako je $\delta(s, a) = t$ prijelaz DKA, onda se definira akcija:
   $$Akcija[s, a] = Pomakni(t)$$
   Akcija ***Pomakni(t)*** stavlja na vrh stoga stanje ***t***.

   b. ako je *LR(1)* stavka $A \rightarrow \alpha \bullet, \{a_1, a_2, a_3 \ldots a_n\}$ u stanju ***s,*** onda za sve završne znakove iz skupa $\{a_1, a_2, a_3 \ldots a_n\}$ definira akcija:
   $$Akcija[s, a] = Reduciraj(A \rightarrow \alpha)$$

   c. ako je *LR(1)* stavka $S' \rightarrow S \bullet, \{\perp\}$ u stanju ***s,*** onda se definira akcija:
   $$Akcija[s, \perp] = Prihvati();$$

4) tablica ***NovoStanje*** se popunjava:

   a. ako je $\delta(s, a) = t$ prijelaz DKA, onda se za nezavršni znak ***A*** i stanje ***s*** definira akcija:
   $$Akcija[s, A] = Stavi(t);$$

5) svi ostali elementi tablice označavaju akciju ***Odbaci()***

6) početno stanje označeno je LR(1) stavkom $S' \rightarrow \bullet S, \{\perp\}$

69. Objasnite postupak određivanja relacija prednosti na temelju zadane gramatike.

1) ako je na desnoj strani produkcije završni znak l neposredno ispred nezavršnog znaka **<C>:**

<A> → --- l **<C>** ---

i ako nezavršni znak **<C>** generira međuniz u kojem je krajnje lijevi završni znak d:

<C> → d---

onda definiramo za znakove l i d relaciju prednosti:

$$l \Leftarrow d$$

2) ako je na desnoj strani produkcije završni znak l neposredno ispred završnog znaka d **:**

<A> → --- l d ---

ili ako je između njih samo jedan nezavršni znak:

<A> → --- l **<B>** d ---

onda definiramo za znakove l i d relaciju prednosti:

$$l \Leftrightarrow d$$

3) ako je na desnoj strani produkcije nezavršni znak **<B>** neposredno ispred završnog znaka d **:**

<A> → --- **<B>** d ---

i ako nezavršni znak **<B>** generira međuniz u kojem je krajnje desni završni znak l:

<B> → --- l

onda definiramo za znakove l i d relaciju prednosti:

$$l \Rightarrow d$$

70. U pseudokodu sličnom jeziku C napišite algoritam parsiranja tehnikom prednosti operatora.

```
while (true) {
        if ( Prednost (Vrh_Stoga) <= Prednost (Ulazni_Znak) )
                Pomakni();
        if ( Prednost (Vrh_Stoga) > Prednost (Ulazni_Znak) )
                Reduciraj();
}
```

71. Navedite i kratko opišite postupke pretvorbe produkcija u produkcije LL(1)-gramatike.

1) **lijevo izlučivanje**

neka gramatika ima n produkcija oblika:

<A> → $\alpha\beta_1$

<A> → $\alpha\beta_n$

Skupovi ***PRIMJENI*** različitih nizova $\boldsymbol{\beta_i}$ nemaju zajedničkih završnih znakova.
Prethodno zadanih ***n*** produkcija se zamjeni sa ***n+1*** produkcija:

<A> → α**<*Nastavak*>**

<*Nastavak*> → β_1

<*Nastavak*> → β_n

2) **zamjena nezvaršnih znakova**

neka gramatika ima n produkcija oblika:

<A> → α_1

<A> → α_n

gdje bilo koja produkcija ima nezavršni znak **<A>** na lijevoj strani.
Neka je zadana produkcija:

<B> → β**<A>**γ

Prethodno zadanih ***n+1*** produkcija se zamjeni sa ***n*** produkcija:

<B> → $\beta\alpha_1\gamma$

<B> → $\beta\alpha_n\gamma$

3) **razrješavanje lijeve rekurzije**

neka je za nezavršni znak **<A>** zadano ***m*** izravno lijevih rekurzivnih produkcija:

<A> → **<A>**α_i ***1 ≤ i ≤ m***

i ***n*** produkcija koje nisu izravno lijevo rekurzivne:

<A> → β_j ***1 ≤ j ≤ n***

Lijevo rekurzivne produkcije se zamjene sljedećim produkcijama:

<A> → β_j**<*Ponovi*>** ***1 ≤ j ≤ n***

<*Ponovi*> → α_i**<*Ponovi*>** ***1 ≤ j ≤ m***

<*Ponovi*> → ε

72. Općenito definirati (ne na primjeru) postupak lijevog izlučivanja koji se koristi pri pretvorbi produkcija u produkcije LL(1)-gramatike.

Neka gramatika ima n produkcija oblika:

<A> → $\alpha\beta_1$

<A> → $\alpha\beta_n$

<A> je završni znak gramatike, a α i β su nizovi nezavršnih i završnih znakova.
Skupovi ***PRIMJENI*** različitih nizova $\boldsymbol{\beta_i}$ nemaju zajedničkih završnih znakova.
Prethodno zadanih ***n*** produkcija se zamjeni sa ***n+1*** produkcija:

<A> → α**<Nastavak>**

<Nastavak> → β_1

<Nastavak> → β_n

koje zadovoljavaju uvjete ***LL(1)-gramatike***.

73. Objasnite postupak uklanjanja lijeve rekurzije tijekom pretvorbe produkcija u LL(1) oblik.

Neka je za nezavršni znak **<A>** zadano ***m*** izravno lijevih rekurzivnih produkcija:

<A> → **<A>**α_i ***1 ≤ i ≤ m***

i ***n*** produkcija koje nisu izravno lijevo rekurzivne:

<A> → β_j ***1 ≤ j ≤ n***

gdje su α_i i β_j nizovi završnih i nezavršnih znakova.
Lijevo rekurzivne produkcije se zamjene sljedećim produkcijama:

<A> → β_j**<Ponovi>** ***1 ≤ j ≤ n***

<Ponovi> → α_i**<Ponovi>** ***1 ≤ j ≤ m***

<Ponovi> → ε

gdje je **<Ponovi>** novi nezavršni znak.

74. Navedite algoritam za izračunavanje ZAPOČINJE skupova za produkcije.

Neka je produkcija oblika:

<X> → **<Z_1><Z_2>---<Z_n><Y>**α

gdje su **<Z_i>** prazni nezavršni znakovi a **<Y>** nije prazni znak.
Skup ***ZAPOČINJE*** za zadanu desnu stranu produkcije računa se na sljedeći način:

ZAPOČINJE(**<X>** → **<Z_1><Z_2>---<Z_n><Y>**α) =
ZAPOČINJE(Z_1) ∪ ***ZAPOČINJE***(Z_2) ∪ --- ∪ ***ZAPOČINJE***(Z_n) ∪ ***ZAPOČINJE***(***Y***)

75. Opišite algoritam za izračunavanje relacije *Ispred*.

Ispred(A, B) vrijedi samo ako je znak ***A*** neposredno ispred znaka ***B*** u barem jednom međunizu generiranom iz početnog nezavršnog znaka ***<S>.***
Relacija ***Ispred*** je produkt triju relacija.
Ako za znakove ***A*** i ***B*** vrijedi relacija ***Ispred(A, B)*** onda za neki par znakova ***X*** i ***Y*** vrijedi:

$$Kraj(A, X) \wedge IzravnoIspredZnaka(X, Y) \wedge ZapočinjeZnakom(Y, B)$$

76. Navedite korake u računanju PRIMJENI skupova za produkcije.

Skupovi ***PRIMJENI*** računaju se primjenom izračunatih skupova ***ZAPOČINJE*** i ***SLIJEDI***. Ako produkcija ne generira prazan znak (ε) onda se skup ***PRIMJENI*** izračuna pomoću skupa ***ZAPOČINJE***. Inače (ako produkcija generira prazan znak) skup ***PRIMJENI*** se računa kao:

$$PRIMJENI(1) = ZAPOČINJE(\varepsilon) \cup SLIJEDI(lijeva\_strana\_produkcije)$$

77. Opišite postupak računanja skupova SLIJEDI za prazne nezavršne znakove.

Izravno na temelju tablice ***Ispred*** moguće je odrediti skupove ***SLIJEDI*** za sve prazne nezavršne znakove.

78. Navedite i ukratko opišite podatkovnu strukturu sintaksnog analizatora.

Podatkovnu strukturu čine ***globalne*** i ***lokalne*** varijable. ***Globalnu strukturu*** podataka čine ***tablica znakova i stog***. ***Tablica unfiormnih znakova*** je osnovna i najznačajnija struktura podatak sintaksnog analizatora, potom slijedi ***tablica ključnih riječi, operatora i specijalnih znakova (KROS)*** i ***zatim tablica identifikatora*** koju sintaksni analizator rijetko koristi.
Lokalna struktura gradi se za potrebe procesa parsiranja i ona ovisi o primjenjenoj tehnici parsiranja.

79. Opišite kako se izvodi nadziranje i oporavak od pogreške kod LR-parsiranja.

Postupak traženja sinkronizacijskog znaka prekida sintaksnu analizu programske cijeline u kojoj je pronađena pogreška. ***Postupak oporavka od pogreške*** uzima sa stoga dio stanja koja pripadaju analizi nezavršnog znaka (npr. ***<Naredba>***). Kazaljka ulaznog niza pomiče se na znak koji slijedi iza znakova generiranih iz nezavršnog znaka (npr. ***<Naredba>***), odnosno u ulaznom spremniku se traži znak ulančavanja naredbi (***;***) ili oznaka kraja bloka naredbi (***{ }***).

U ***postupku lokalnih promjena*** svi elementi tablice ***LR*** parsera koji označavaju akciju ***Odbaci()*** zasebno se analiziraju. Na temelju pravila izvornog jezika pretpostavi se najvjerojatnija pogreška korisnika. Na temelju pretpostavljene pogreške odrede se ***akcije promjene sadržaja stoga i ulaznog niza***.

80. Objasnite parsiranje od dna prema vrhu metodom *Pomakni-Reduciraj*.

Na temelju koda samo jednog znaka na vrhu stoga moguće je odrediti koji je niz znakova na vrhu stoga, te je zbog toga moguće primjeniti akciju ***Reduciraj()*** bez potrebe čitanja znakova na vrhu stoga i njihove usporedbe sa znakovima na desnim stranama produkcija. ***Kodirani znakovi gramatike čine skup stanja znakova stoga***. Parserom se upravlja ***primjenom tablica Pomakni/Reduciraj i tablicom Stavi***.

Tablica Pomakni/Reduciraj na temelju kodiranog znaka na vrhu stoga i pročitanog znaka ulaznog niza odlučuje se o primjeni akcije ***Pomakni*** ili ***Reduciraj***.

Tablica Stavi određuje kod znaka koji se stavlja na vrh stoga. Njeni reci su označeni znakovima stoga a stupci znakovima gramatike.

81. Navedite i definirajte korake algoritma izgradnje kanonskog LR(1)-parsera.

1) skup stanja je skup svih LR(1) stavki uvećan početnim stanjem q_o

2) skup ulaznih znakova je unija skupova završnih i nezavršnih znakova gramatike ($\Sigma = T \cup V$)

3) budući da su sva stanja valjane LR(1) stavke onda su i sva stanja u skupu dozvoljenih stanja (F = Q)

4) funkcija prijelaza δ definira se na sljedeći način:

a. $\delta(q_o,\varepsilon) = \{ S \rightarrow \bullet\ \alpha, \{\perp\} \mid S \rightarrow$ produkcija $\}$
(ovaj prijelaz omogućuje početak rada ***ε-NKA***)

b. $\delta((A \rightarrow \alpha \bullet X\beta, \{a_1, a_2, ..., a_n\}), X) = \{ A \rightarrow \alpha X \bullet \beta, \{ a_1, a_2, ..., a_n\}\}$
(ako automat pročita znak ***X***, onda se oznaka točke pomakne iza toga znaka)

c. $\delta((A \rightarrow \alpha \bullet B\beta, \{a_1, a_2, ..., a_n\}), \varepsilon) = \{ B \rightarrow \bullet\ \gamma, T \mid B \rightarrow \gamma$ produkcija $\}$
(ako je nezavršni znak ***B*** desno od točke, onda ***ε-prijelaz*** pokreće analizu tog znaka)
T je skup svih završnih znakova ***b*** i oznake kraja niza $\perp$ za koje vrijedi:

i. Znak ***b*** započinje niz β ($b \in$ ZAPOČINJE (β))

ii. Ako je moguće iz niza znakova niza β generirati prazni niz ε, odnosno $\beta \Longrightarrow \varepsilon$

82. Opišite postupak izgradnje potisnog automata za S-gramatiku.

1) PA ima samo jedno stanje (koje je i početno) ***Q*** = { ***qo*** }

2) skup ulaznih znakova PA je skup završnih znakova i oznaka kraja niza ($\Sigma = T \cup \perp$)

3) skup znakova stoga PA su oznaka dna stoga ∇, skup nezavršnih znakova ***V***, skup završnih znakova koji su na desnim stranama produkcija ali ne isključivo na krajnje lijevim mjestima

4) na početku rada na stogu je ∇ i početni znak ***S***

5) funkcija prijelaza definira se tablicom (reci su znakovi stoga, a stupci ulazni znakovi). Elementi tablice su akcije PA, a tablica se popunjava:

   a. $A \rightarrow b\alpha$
      Zamjeni (α^r ***);***
      Pomakni;

   b. $A \rightarrow b$
      Izvuci;
      Pomakni;

   c. ako je završni znak b znak stoga, element tablice [b, b] :
      Izvuci;
      Pomakni;

   d. element tablice [∇, ⊥] određuje akciju prihvaćanja niza:
      Prihvati;

   e. svi ostali elementi označavaju da se niz ne prihvaća:
      Odbaci;

83. Definirajte LL(1)-gramatiku i kratko opišite konstrukciju potisnog automata za LL(1)-gramatiku.

Kontekstno neovisna gramatika jest ***LL(1)-gramatika*** ako vrijedi:

ako više produkcija ima isti nezavršni znak na lijevoj strani, onda njihovi skupovi ***PRIMJENI*** nemaju zajedničkih elemenata

1) PA ima samo jedno stanje (koje je i početno) **Q** = { ***q0*** }

2) skup ulaznih znakova PA je skup završnih znakova i oznaka kraja niza ($\Sigma = T \cup \perp$)

3) skup znakova stoga PA su oznaka dna stoga ∇, skup nezavršnih znakova ***V***, skup završnih znakova koji su na desnim stranama produkcija ali ne isključivo na krajnje lijevim mjestima

4) na početku rada na stogu je ∇ i početni znak ***S***

5) tablica PA se popunjava:

a. $A \rightarrow b\alpha$

Zamjeni (α^r ***);***

Pomakni;

b. $A \rightarrow b$

Izvuci;

Pomakni;

c. $A \rightarrow \varepsilon$

Izvuci;

Zadrži;

d. $A \rightarrow \alpha$

Zamjeni (α^r ***);***

Zadrži;

e. ako je za nezavršni znak A zadana prazna produkcija i ako u prethodnim koracima nije popunjen neki od elemenata u retku A, onda se za taj element tablice definira akcija ***Odbaci;*** ili akcije ε-preodukcije:

Izvuci;

Zadrži;

f. ako je završni znak b znak stoga, element tablice [b, b] :

Izvuci;

Pomakni;

g. element tablice [∇, $\perp$] određuje akciju prihvaćanja niza:

Prihvati;

h. svi ostali elementi označavaju da se niz ne prihvaća:

Odbaci;

84. Navedite i općenito objasnite (ne na primjeru) sve (6) akcije potisnog automata konstruiranog na osnovi LL(1)-gramatike.

1) ***Zamjeni*** (α^r)
   nezavršni znak ***A*** koji je na vrhu stoga zamjeni nizom znakova α
2) ***Izvuci***
   uzima sa vrha stoga nezavršni znak ***A***
3) ***Pomakni***
   miče glavu za čitanje na sljedeći znak ulaznog niza
4) ***Zadrži***
   ostavlja glavu za čitanje na istom ulaznom znaku
5) ***Prihvati***
   akcija prihvaćanja ulaznog niza
6) ***Odbaci***
   akcija odbacivanja (neprihvaćanja) ulaznog niza

85. Navedite sve zadatke sintaksnog analizatora.

Sintaksni analizator

a) slijedno čita uniformne znakove leksičkih jedinki
b) grupira uniformne znakove u sintaksne cjeline
c) provjerava sintaksna pravila
d) stvara hijerarhiju sintaksnih cjelina
e) određuje mjesto sintaksnih pogrešaka i opisuje sintaksne pogreške
f) izvodi postupak oporavka od pogrešaka
g) gradi sintaksno stablo

86. Općenito definirajte (ne na primjeru) ulaze i izlaze iz programa generatora sintaksnog analizatora i programa sintaksnog analizatora ako je sintaksni analizator ostvaren kao zasebni prolaz jezičnog procesora.

Generator sintaksnog analizatora na ulaz prima opis procesa sintaksnog analizatora (nezavršne i završne znakove gramatike, sinkronizacijski završni znakovi i produkcije gramatike). Izlaz generatora bit će izvorni kod (u jeziku izgradnje) sintaksnog analizatora.

Sintaksni analizator na ulazu dobiva niz leksičkih jedinki koje čita i stvara sintaksno stablo (koje je izlaz sintaksnog analizatora).

87. Objasnite pojmove parsiranje, parsiranje od dna prema vrhu i parsiranje od vrha prema dnu.

Parsiranje je postupak prepoznavanja niza i gradnja generativnog stabla na temelju zadanih produkcija kontekstno neovisne gramatike.

Parsiranje od vrha prema dnu započinje gradnju generativnog stabla vrhom generativnog stabla koji je označen početnim nezavršnim znakom gramatike.

Parsiranje od dna prema vrhu započinje gradnju generativnog stabla listovima. Listovi stabla označeni su završnim znakovima gramatike. Gradnja stabla nastavlja se primjenom desnih strana produkcija gramatike na prethodno izgrađene čvorove.

88. Neovisno poredajte gramatike LL(1), S i Q te gramatike LALR(1), SLR(1), LR(0) i LR(1) uzlazno po općenitosti.

1) S gramatika
2) Q gramatika
3) LL(1) gramatika

---

1) LR(0) gramatika
2) SLR(1) gramatika
3) LR(1) gramatika
4) LALR(1) gramatika

89. Objasnite namjenu programa YACC te dijelove ulazne datoteke za program YACC.

Yacc program je generator parsera i koristi se za rješavanje problema koje je moguće svesti na problem prihvaćanja kontekstno neovisnih jezika.

U ulaznoj datoteci se definira sintaksni analizator čiju definiciju čine:

deklaracije

%%
pravila prevođenja (središnji dio)
%%
---
pomoćne C procedure

90. Ukratko objasnite postupak oporavka od pogreške kod LR-parsiranja koji se zasniva na traženju sinkronizacijskih znakova.

Postupak traženja sinkronizacijskog znaka prekida sintaksnu analizu programske cijeline u kojoj je pronađena pogreška. ***Postupak oporavka od pogreške*** uzima sa stoga dio stanja koja pripadaju analizi nezavršnog znaka (npr. ***<Naredba>***). Kazaljka ulaznog niza pomiče se na znak koji slijedi iza znakova generiranih iz nezavršnog znaka (npr. ***<Naredba>***), odnosno u ulaznom spremniku se traži znak ulančavanja naredbi (***;***) ili oznaka kraja bloka naredbi (***{ }***). Nakon što se sa stoga uzmu stanja koja pripadaju analizi znaka ***<Naredba>*** na vrh stoga se stavlja stanje ***S*** za koje je definirana akcija prijelaza u neko drugo stanje primjenom nezavršnog znaka ***<Naredba>***. Na stog se stavlja stanje:

NovoStanje [s, <Naredba>]

91. Navedite pet različitih vrsta sustava oznaka za opis sintaksnih pravila.

1) kontekstno neovisna gramatika
2) regularni izrazi
3) BNF sustav oznaka
4) COBOL sustav oznaka
5) Co-No tablice

92. Opišite postupak sintaksne analize zasnovane na tablici Co-No.

Ispravnost izvornog programa i prevođenje u strojni program zasniva se na poznavanju dva podatka – ***lijevog*** i ***desnog operatora***. Postupak analize izvornog programa i generiranja ciljnog programa zadaje se ***dvodimenzionalnom tablicom*** (reci su lijevi operator a stupci desni). Ako je par operatora dozvoljen sintaksnim pravilima onda element tablice (koji je određen tim parom) označava jednu od akcija generatora ciljnog programa (u protivnom kao element tablice zapisuje se oznaka pogreške).

93. Objasnite parsiranje od dna prema vrhu tehnikom *Pomakni-Reduciraj*. Opišite tablice koje se koriste u parsiranju.

Na temelju koda samo jednog znaka na vrhu stoga moguće je odrediti koji je niz znakova na vrhu stoga, te je zbog toga moguće primjeniti akciju ***Reduciraj()*** bez potrebe čitanja znakova na vrhu stoga i njihove usporedbe sa znakovima na desnim stranama produkcija. ***Kodirani znakovi gramatike čine skup stanja znakova stoga***. Parserom se upravlja ***primjenom tablica Pomakni/Reduciraj i tablicom Stavi***.

Tablica Pomakni/Reduciraj na temelju kodiranog znaka na vrhu stoga i pročitanog znaka ulaznog niza odlučuje se o primjeni akcije ***Pomakni*** ili ***Reduciraj***.

Tablica Stavi određuje kod znaka koji se stavlja na vrh stoga. Njeni reci su označeni znakovima stoga a stupci znakovima gramatike.

94. Opišite algoritme na kojima se zasnivaju postupci oporavka od pogreške kod sintaksne analize.

Postupak traženja sinkronizacijskog znaka prekida sintaksnu analizu programske cijeline u kojoj je pronađena pogreška. ***Postupak oporavka od pogreške*** uzima sa stoga dio stanja koja pripadaju analizi nezavršnog znaka (npr. ***<Naredba>***). Kazaljka ulaznog niza pomiče se na znak koji slijedi iza znakova generiranih iz nezavršnog znaka (npr. ***<Naredba>***), odnosno u ulaznom spremniku se traži znak ulančavanja naredbi (***;***) ili oznaka kraja bloka naredbi (***{ }***).

U ***postupku lokalnih promjena*** svi elementi tablice ***LR*** parsera koji označavaju akciju ***Odbaci()*** zasebno se analiziraju. Na temelju pravila izvornog jezika pretpostavi se najvjerojatnija pogreška korisnika. Na temelju pretpostavljene pogreške odrede se ***akcije promjene sadržaja stoga i ulaznog niza***.

95. Objasnite parsiranje od dna prema vrhu metodom prednosti operatora, relaciju prednosti, akcije parsera i određivanje uzorka za zamjenu.

Za završne znakove gramatike ***a*** i ***b*** definira se ***relacija prednosti*** koja se koristi u postupku parsiranja. Tablica u kojoj su zadane relacije prednosti koristi se tijekom parsiranja niza. Parser uspoređuje završni znak na vrhu stoga i znak u ulaznom nizu te na temelju relacije prednosti donosi odluku o primjeni akcija ***Pomakni*** ili ***Reduciraj***.

Ako je prednost završnog znaka na vrhu stoga jednaka ili manja od prednosti znaka u ulaznom nizu, onda parser primjenjuje akciju ***Pomakni***.

Ako je prednost završnog znaka na vrhu stoga veća od prednosti znaka ulaznog niza, onda se primjenjuje akcija ***Reduciraj***.

Određivanje uzorka za zamjenu obavlja se tako da parser uzima jedan po jedan znak sa stoga sve do završnog znaka koji ima manju prednost od prednosti prethodno uzetog znaka.

96. Objasnite razlike u ostvarenju parsera LR(0), SLR(1), LALR i LR(1) te navedite njihove prednosti i nedostatke.

LR(0)	parsiranje od dna prema vrhu generiranje niza zamjenom krajnje desnog nezavršnog znaka čita još 0 znakova ulaznog niza kod redukcije
SLR(1)	najjednostavniji postupak nemogućnost primjene na velik skup jezika obuhvaća nešto širi skup jezika od LR(0) klase
LALR	primjena na veći skup jezika od SLR(1) obuhvaća nešto manji skup jezika od LR(1)
LR(1)	parsiranje od dna prema vrhu čita još 1 znak ulaznog niza kod redukcije

97. Objasnite konstrukciju ε-NKA u postupku izgradnje SLR(1)-parsera.

1) skup stanja je skup svih LR(1) stavki uvećan početnim stanjem q_o

2) skup ulaznih znakova je unija skupova završnih i nezavršnih znakova gramatike ($\Sigma = T \cup V$)

3) budući da su sva stanja valjane LR(1) stavke onda su i sva stanja u skupu dozvoljenih stanja (F = Q)

4) funkcija prijelaza δ definira se na sljedeći način:

a. $\delta(q_o,\varepsilon) = \{ S \rightarrow \bullet\ \alpha, \{\perp\} \mid S \rightarrow$ produkcija $\}$
(ovaj prijelaz omogućuje početak rada ***ε-NKA***)

b. $\delta((A \rightarrow \alpha \bullet X\beta, \{a_1, a_2, ..., a_n\}), X) = \{ A \rightarrow \alpha X \bullet \beta, \{ a_1, a_2, ..., a_n\}\}$
(ako automat pročita znak ***X***, onda se oznaka točke pomakne iza toga znaka)

c. $\delta((A \rightarrow \alpha \bullet B\beta, \{a_1, a_2, ..., a_n\}), \varepsilon) = \{ B \rightarrow \bullet\ \gamma, T \mid B \rightarrow \gamma$ produkcija $\}$
(ako je nezavršni znak ***B*** desno od točke, onda ***ε-prijelaz*** pokreće analizu tog znaka)
T je skup svih završnih znakova ***b*** i oznake kraja niza $\perp$ za koje vrijedi:

i. Znak ***b*** započinje niz β ($b \in$ ZAPOČINJE (β))

ii. Ako je moguće iz niza znakova niza β generirati prazni niz ε, odnosno $\beta \Longrightarrow \varepsilon$

98. Navedite i općenito objasnite (ne na primjeru) sve četiri akcije potisnog automata konstruiranog na osnovi LR(1) gramatike.

Pomakni (t)	stavlja stanje **t** na vrh stoga i pomiče glavu na sljedeći znak u ulaznom nizu
Reduciraj (A→ α)	izvodi zamjenu na vrhu stoga (s određenim nezavršnim znakom)
Stavi (t)	na stog stavlja stanje **t** na temelju tablice ***NovoStanje***
Prihvati ()	akcija prihvaćanja ulaznog niza

99. Navedite zahtjeve koje mora ispuniti detekcija pogrešaka u sintaksnom analizatoru.

1) precizno određivanje mjesta pogreške

2) kratak i jasan opis greške

   a. oznaka mjesta pogreške u izvornom programu
   (ispisati liniju programa)

   b. odrediti vrstu pogreške
   (kratki tekst objašnjenja)

3) postupci detekcije pogreški ne smiju značajnije usporiti rad sintaksnog analizatora

100. Objasnite sustav oznaka COBOL.

mala slova	varijabla
velika slova	konstanta koju je moguće izostaviti
podcrtana velika slova	konstanta koju nije moguće izostaviti
znak do znaka	nadovezivanje
[ ]	neobavezan izbor jedne od mogućnosti
{ }	obavezan izbor jedne od mogućnosti
•••	ponavljanje prethodne sintaksne cjeline

101. Opišite postupak oporavka od pogrešaka u sintaksnom analizatoru.

Algoritam traženja sinkronizacijskih znakova
Sintaksni analizator izbacuje redom sve uniformne znakove do prvog sinkronizacijskog znaka.

Algoritam lokalnih promjena
Sintaksni analizator zamjenjuje, dodaje ili izbacuje uniformne znakove u cilju postizanja prefiksa neanaliziranog dijela izvornog programa koji zadovoljava sintaksna pravila

Dodatne produkcije koje uključuju pogreške
Skup gramatike proširuje se dodatnim produkcijama. Dodatne produkcije definiraju pogreške koje često nastaju u izvornom programu. Ako sintaksni analizator primjeni jednu od dodatnih produkcija, onda se opiše pogreška definirana tom dodatnom produkcijom.

102. Objasnite akcije parsera od dna prema vrhu koji koristi tehniku *Pomakni-Pronađi*. Opišite proturječja koja se pojavljuju.

Akcije parsera:

Pronađi	u ovisnosti o vrhu stoga određuje koja će se redukcija primjeniti (ili će se odbaciti niz)
Pomakni	pomiče glavu za čitanje ulaznog niza
Odbaci	ulazni niz se ne prihvaća

4. SEMANTIČKA ANALIZA

205. Opišite postupak izgradnje potisnog automata za prijevodnu gramatiku.

1) PA ima samo jedno stanje (koje je i početno) **Q** = { ***q0*** }

2) skup ulaznih znakova PA je skup završnih znakova i oznaka kraja niza ($\Sigma = T \cup \perp$)

3) skup znakova stoga PA su ***oznaka dna stoga*** ∇, skup ***nezavršnih znakova V***, skup ***završnih znakova*** koji su na desnim stranama produkcija ali ne isključivo na krajnje lijevim mjestima, ***izlazni završni znakovi*** (samo ako je na desnoj strani produkcije desno od krajnje ljevog nezavršnog ili ulaznog završnog znaka)

4) na početku rada na stogu je ∇ i početni znak ***S***

5) Elementi tablice su akcije PA, a tablica se popunjava:
   a. ***<A>*** $\rightarrow \xi b \phi \alpha$
      Izlaz($\xi\phi$);
      Zamjeni(α^r);
      Pomakni;
   b. za produkciju **<A>** $\rightarrow \xi$, u redak tablice ***A*** i sve stupce određene znakovima skupa ***PRIMJENI(A*** $\rightarrow \varepsilon$***)***
      Izlaz(ξ);
      Izvuci;
      Zadrži;
   c. za produkciju **<A>** $\rightarrow \xi\alpha$, u redak tablice ***A*** i sve stupce određene znakovima skupa ***PRIMJENI(A*** $\rightarrow \alpha$***)***
      Izlaz(ξ);
      Zamjeni(α^r);
      Zadrži;
   d. ako je izlazni završni znak {ξ} znak stoga
      Izlaz(ξ);
      Izvuci;
      Zadrži;
   e. element tablice [$\nabla, \perp$] određuje akciju prihvaćanja niza:
      Prihvati;
   f. svi ostali elementi označavaju da se niz ne prihvaća:
      Odbaci;

206. Objasnite što je provjera vrijednosti obilježja i opišite pojedine postupke za provjeru vrijednosti obilježja.

Ako se tijekom ***provjere vrijednosti obilježja*** ustanovi pogreška, onda semantički analizator ispisuje poruku o pogrešci, pridruži obilježju vrijednost *Pogreška*, pokrene postupak oporavka od pogreške i nastavi analizu izvornog programa.

Provjera vrijednosti obilježja naredbi deklaracija
U tablicu identifikatora zapisuju se vrijednosti obilježja. Produkcije gramatike proširuju se izlaznim završnim znakovima semantičkih akcija i svojstvima. Semantičkim akcijama definira se provjera vrijednosti obilježja naredbi deklaracija.

Provjera vrijednosti obilježja izraza
Ako je produkcijom definirana varijabla onda se vrijednost svojstva V određuje na temelju vrijednosti obilježja koje je zapisano u tablici indetifikatora tijekom deklaracije varijabli.

Provjera vrijednosti obilježja ulančanih naredbi
Naredbe se ulančavaju primjenom operatora **;** . Obilježjima naredbi pridružuju se dvije vrijednosti – ***BezPogreške*** i ***Pogreška***. Logičkom izrazu naredbe uvjetnog grananja pridružuje se vrijednost ***LogičkaVrijednost***.

207. Opišite L-atributne prijevodne gramatike.

1) Vrijednost nasljednog svojstva znaka desne strane produkcije računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije koji su lijevo od zadanog znaka.

2) Vrijednost izvedenog svojstva nezavršnog znaka lijeve strane produkcije računa se na temelju vrijednosti nasljednih svosjtava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije.

3) Vrijednost izvedenog svojstva izlaznog završnog znaka računa se na temelju nasljednih svojstava tog istog izlaznog završnog znaka.

208. Nabrojite i objasnite formalne modele semantičkog analizatora.

Prevođenje izvornog jezika u jezik koji ima definiranu semantiku je najjednostavniji formalni model.

Izvođenje na apstraktnom stroju zasniva se na skupu pravila primjenom kojih se simulira izvođenje izvornog programa. Apstraktno računalo definira se stanjima. Izvođenje programa simulira se skupom funkcija koje mijenjaju stanje apstraktnog računala.

Skup aksioma je treći model. Aksiomi su logičke tvrdnje koje korisnik zadaje u različitim dijelovima izvornog programa, a njima izriče očekivani rezultat izvođenja tog dijela programa. Semantička ispravnost programa zasniva se na provjeri istovjetnosti statičkih logičkih tvrdnji i rezultata dinamičkog izvođenja programa.

209. Definirajte L-atributnu prijevodnu gramatiku, odnosno navedite sva tri pravila računanja vrijednosti svojstava kod L-atributne prijevodne gramatike.

1) Vrijednost ***nasljednog svojstva znaka desne strane produkcije*** računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije koji su lijevo od zadanog znaka.

2) Vrijednost izvedenog svojstva ***nezavršnog znaka lijeve strane produkcije*** računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije.

3) Vrijednost ***izvedenog svojstva izlaznog završnog znaka*** računa se na temelju nasljednih svojstava tog istog izlaznog završnog znaka.

210. Navedite i objasnite tri najčešće primjenjivana formalna modela semantičkog analizatora.

Prevođenje izvornog jezika u jezik koji ima definiranu semantiku je najjednostavniji formalni model.

Izvođenje na apstraktnom stroju zasniva se na skupu pravila primjenom kojih se simulira izvođenje izvornog programa. Apstraktno računalo definira se stanjima.

Izvođenje programa simulira se skupom funkcija koje mijenjaju stanje apstraktnog računala.

Skup aksioma je treći model. Aksiomi su logičke tvrdnje koje korisnik zadaje u različitim dijelovima izvornog programa, a njima izriče očekivani rezultat izvođenja tog dijela programa. Semantička ispravnost programa zasniva se na provjeri istovjetnosti statičkih logičkih tvrdnji i rezultata dinamičkog izvođenja programa.

211. Objasnite sintaksom vođenu semantičku analizu.

Sintaksom upravljani jezični procesori zasnovani su na primjeni atributne prijevodne gramtike. Semantičke akcije su potprogrami namjenjeni analizi pojedinih sintaksnih cjelina. Tijekom gradnje sintaksnog stabla, sintaksni analizator grupira znakove u hijerarhijske cjeline i pokreće izvoeđenje semantičkih akcija pridruženih tim sintaksnim cjelinama.
Semantičke akcije ugrađuju se u produkcije formalne gramatike kao izlazni završni znakovi.
Produkcijama gramatike pridružuju se pravila računanja vrijednosti svojstava. Prijevodna gramatika proširena svojstvima i pravilima računanja vrijednosti tih svojstava naziva se ***atributna prijevodna gramatika***.

212. Definirajte atributnu prijevodnu gramatiku.

U atributnoj prijevodnoj gramatici završnim i nezavršnim znakovima dodjeljuju se svojstva i pravila računanja tih svojstava. Pravilima računanja svojstava određen je i smjer njihovog prijenosa po sintaksnom stablu.
Ako se njihove vrijednosti prenose od dna prema vrhu onda se svojstva nazivaju ***izvedena***, u suprotnom se nazivaju ***nasljedna svojstva***.

213. Opišite algoritam provjere jednakosti tipova obilježja temeljen na provjeri jednakosti strukture obilježja.

Za potrebe ispitivanja jednakosti dva obiljažja koristi se postupak ***ujednačavanja***.
Ujednačavanje obiljažja ***s*** i ***t*** je postupak utvrđivanja jednakosti dva obilježja na način da se varijable u oba obilježja zamjene odgovarajućim zajedničkim vrijednostima. Tijekom postupka ujednačavanja uspoređuju se čvorovi stabala strukture. Ako su čvorovi stabala označeni istim konsturktorom ili ako je jedan od čvorova označen varijablom onda se oni združe u jednu grupu.

214. Objasnite kako se obrađuju izvedena svojstva izlaznih završnih znakova koji se ne stavljaju na stog.

Izlazni završni znak ***{Rezultat}*** pokreće izvođenje semantičke akcije koja ispisuje vrijednost aritmetičkog izraza. Izlaznom završnom znaku ***{Rezultat}*** dodjeljuje se svojstvo sa značenjem rezultata aritmetičkog izraza generiranog iz nezavršnog znaka ***<S>***. ***{Rezultat$_r$}*** je oznaka izlaznog završnog znaka ***{Rezultat}*** kojemu je dodjeljeno svojstvo ***r***. Vrijednost aritmetičkog izraza računa se korak po korak primjenom produkcija gramatike. Međurezultati računanja prenose se sintaksnim stablom od ulaznih završnih znakova do izlaznog završnog znaka ***{Rezultat}***. Izračunata vrijednost aritmetičkog izraza pridruži se svojstvu ***r*** izlaznog završnog znaka ***{Rezultat$_r$}***.

215. Navedite i objasnite algoritam ispitivanja jednakosti obilježja konstantnih vrijednosti.

```
Jednakost ( s, t )
{
     ako ( (s i t su jednake jednostavne vrijednosti) ||
           (s i t su ista korisnička imena) )
           vrati ISTINITO;
     inače ako (s==Polje(s1, s2) && t==Polje(t1, t2) )
          vrati (Jednakost(s1, t1) & Jednakost (s2, t2) );
     inače ako (s==Kazaljka(s1)  && t==Kazaljka(t1) )
          vrati (Jednakost(s1, t1));
     inače ako (s==s1→s2  && t== t1→t2 )
          vrati (Jednakost(s1, t1) & Jednakost (s2, t2) );
     inače ako (s==s1×s2 && t==t1×t2)
          vrati (Jednakost(s1, t1) & Jednakost (s2, t2) );
     inače
          vrati NEISTINITO;
}
```

216. Navedite po dva primjera leksičkih, sintaksnih i semantičkih pravila programskog jezika C ili sličnog jezika.

LEKSIČKA

imena varijabli počinju slovom
komentari se označavaju // ili /*
ključne riječi (if, break, else....) pišu se malim slovima

SINTAKSNA

ulančavanje naredbi pomoću ;
jednak broj otvorenih i zatvorenih zagrada ({}, ())

SEMANTIČKA

indeks polja mora biti cjelobrojan
ako je funkcija tipa ***int*** onda i povratna vrijednost mora biti istog tipa
---

217. Opišite korake gradnje atributnog generativnog stabla. Definirajte potpuno atributno generativno stablo.

1) primjenom produkcija prijevodne gramatike izgradi se generativno stablo za zadani niz ulaznih završnih znakova
2) svojstvima ulaznih završnih znakova pridruže se pročitane vrijednosti
3) nasljednim svojstvima početnog nezavršnog znaka pridruže se početne vrijednosti koje su definirane zajedno s produkcijama gramatike
4) pretražuje se generativno stablo
    a. traži se svojstvo koje nema izračunatu vrijednost ali su zato izračunate vrijednosti svih svojstava na temelju kojih se računa vrijednost izabranog svojstva
    b. izračuna se vrijednost izabranog svojstva
    c. postupak se nastavlja traženjem sljedećeg svojstva

Ako je moguće izračunati vrijednost svih svojstava koja su dodjeljena svim znakovima gramatike onda je ***atributno generativno stablo potpuno***.

218. Objasnite kako se obrađuju svojstva izvorišta koja nemaju dostupne vrijednosti.

Budući da vrijednosti svojstava izvorišta nisu dostupne, potisni automat stavlja na stog kazaljku koja pokazuje na listu polja dodjeljenih svojstvima odredišta. Vrijednost kazaljki prenose se od vrha generativnog stabla prema dnu sve dok vrijednost svojstava izvorišta ne postanu dostupne.

219. Navedite uvjete pod kojima je atributna prijevodna gramatika ujedno i L-atributna prijevodna gramatika.

1) Vrijednost ***nasljednog svojstva znaka desne strane produkcije*** računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije koji su lijevo od zadanog znaka.

2) Vrijednost izvedenog svojstva ***nezavršnog znaka lijeve strane produkcije*** računa se na temelju vrijednosti nasljednih svojstava nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava znakova desne strane produkcije.

3) Vrijednost ***izvedenog svojstva izlaznog završnog znaka*** računa se na temelju nasljednih svojstava tog istog izlaznog završnog znaka.

220. Objasnite razliku između izvedenih i nasljednih svojstava. Kako se izvedena i nasljedna svojstva spremaju na stog tijekom parsiranja od vrh prema dnu?

Ako se vrijednosti svojstava prenose od dna prema vrhu sintaksnog stabla, onda se svojstva nazivaju ***izvedena***.
Ako se vrijednosti svojstava prenose od vrha prema dnu sintaksnog stabla, onda se svojstva nazivaju ***nasljedna***.
Parser na stog stavlja ***oznake semantičkih akcija***. Zajedno s oznakom semantičke akcije na stog se stavljaju ***vrijednosti svojstava*** te oznake.

221. Navedite zadatke semantičkog analizatora.

1) popunjava tablicu znakova vrijednostima obilježja sintaksnih cjelina
2) osigurava prijenos vrijednosti obilježja po sintaksnom stablu
3) određuje početne uvjete
4) određuje mjesto semantičke pogreške
5) opisuje semantičke pogreške
6) obrađuje makro naredbe
7) obrađuje naredbe koje se izvode tijekom prevođenja