Programski prevodioci

Pitanja za usmeni deo ispita

---

1. Navesti osnovne tipove gramatika po Čomskom

Gramatika je skup pravila kojima se definišu pravilne konstrukcije jezika. *Noam Čomski* je dao svoju definiciju koja kaže da su **4 elementa** dovoljna za opis gramatike. Dakle, gramatika je definisana sa G = {VN,VT,R,S}. Kao što smo maločas kazali:

V_N - je skup **neterminalnih** simbola,
V_T - je skup **terminalnih** simbola,
R - je rečenični tj. **startni** simbol koji se primenom skupa pravila može transformisati u sve reči jezika opisanog gramatikom. R se sastoji isključivo od neterminalnih simbola, dakle, pripada skupu V_N.
S - je **skup smena**. Tipična smena izgleda ovako: x -> y ili x := y. Skup smena mora poštovati neka **ograničenja:**
- 1.U reči na levoj strani mora da se nalazi bar 1 neterminalni simbol; Dakle, x iz prethodnog primera mora da sadrži bar jedan neterminalni simbol.
- 2. Reč na desnoj strani je bilo koja reč azbuke V; Dakle, y iz prethodnog primera može biti bilo koja reč. Treba napomenuti da su V_N i V_T su **disjunktni**. Drugim rečima, jedan simbol ne može u isto vreme da pripada skupu terminalnih i skupu neterminalnih simbola.

*Najopštija definicija gramatike je ona koju je postavio *Noam Čomski*. Po njemu razlikujemo **4 gramatika**:

Gramatike tipa „nula“

- **Gramatike tipa nula** su one koje odgovaraju definiciji Čomskog.
G = {V_N,V_T,R,S}
S: x -> y

$$x \to y, \quad gde\ je \quad x \in V^* V_n V^* \wedge y \in V^*$$

Gde važi da X pripada skupu V^* s tim što mora da sadrži makar jedan neterminalni simbol **u sredini**. Ovde treba re□i da je V^* ustvari skup, jednak uniji skupova V_N i V_T (unija neterminalnih I terminalnih simbola). Dakle, na po□etku i kraju re□i mogu biti i terminalni i neterminalni simboli, dok u sredini **mora** postojati makar jedan neterminalni simbol. S desne strane tj. (y) može da bude i terminal i neterminal.

Gramatike tipa “jedan”

- **Gramatike tipa jedan** su one gramatike za koje pored pravila za tip “nula” važi i sledeća ograničenje. **Dužina reči** na levoj strani mora biti <= dužini reči na desnoj strani. Dakle matematički (formalno) rečeno |x| <= |y|. Takođe je **zabranjeno preslikavanje** tipa X -> e ili A ->e.

$$Za \quad x \to y \quad vazi \quad |y| \geq |x|$$

Ono što je **zajedničko** za gramatike tipa „nula“ i gramatike tipa „jedan“ je da imaju **isti oblik smena** s leve strane:
X mora biti oblika: V^* V_N V^*. Drugim rečima, na početku i kraju reči mogu biti i terminalni I neterminalni simboli, dok **u sredini** mora biti makar jedan neterminalni simbol. Dakle, gramatika tipa “jedan” sadrži sve karakteristike gramatike tipa “nula” i plus neke svoje osobenosti koje se odnose na dužinu reči.

Gramatike tipa „dva“

- **Gramatike tipa dva** su takozvane **beskonteksne gramatike**. Ograničenje za ove gramatike je da su **na levoj strani samo neterminali** (više se ne dozvoljava da se neterminal nađe u kontekstu.

A -> Y

„A“ mora da pripada skupu V_N $A \to y, \quad A \in V_n, \quad y \in V^*$

Ove gramatike se koriste u **programskim jezicima**.

Gramatike tipa „tri“

- **Gramatike tipa tri** su takođe beskonteksne gramatike tj. **automatne gramatike** ili **regularne gramatike ili *Gramatike sa konačnim brojem stanja***. One sadrže još stroža ograničenja na desnoj strani smena. Dakle, jedino što je **dozvoljeno je**:

A -> a

A –>aB $A \to aB \vee A \to a, \quad A, B \in V_n \wedge a \in V_t \bigcup \{\varepsilon\}$

A,B pripadaju skupu V_N, a „a“ pripada skupu V_T

Na osnovu smena se direkno može konstruisati **graf konačnog automata**. Najstrože definisana gramatika uzrokuje najjednostavnije rešavanje problema prepoznavanja jezika je opisano upravo ovom gramatikom.

Relacije između jezika definisanih ovim gramatikama su:

L(3) <= L(2) <= L(1) <= L(0) $L(3) \subseteq L(2) \subseteq L(1) \subseteq L(0)$

Ovo nam kazuje da je jezik definisan nad gramatikom tipa 3 podskup jezika nad gramatikom 2.

---

2. Magacinski automat kao uređaj za prepoznavanje jezika. Dati šemu i objasniti kako se koristi u prepoznavanju jezika

---

Postoje 4 grupe automata:Objasniti ih sve

1. Tjuringova mašina
2. Linearno ograničeni automati
3. Magacinski automati
4. Konačni automati

Njegova **radna memorija** je organizova na kao **magacin** tj. **stek**.

MA=(Q, V, S, g, q_0, Z_0, F)

Q – Skup stanja operativnog organa

V – Skup simbola na ulaznoj traci(**ulazna zbuka**)

S – Skup magacinskih simbola(**azbuka radnog magacina**)

F – Skup završnih, krajnjih stanja (Podskup skupa Q)

q_0 – Početno stanje operativnog organa

Z_0 – Početni simbol na vrhu magacina

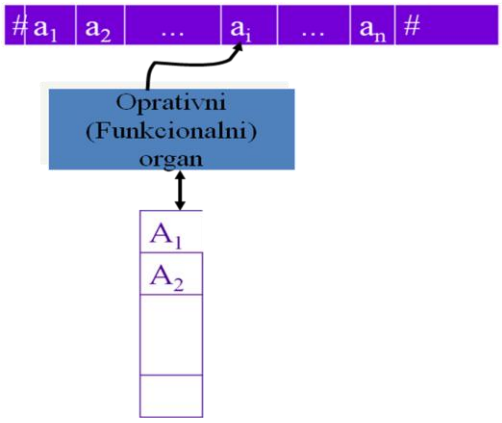
g – Preslikavanje

$$Q \times (V \cup \#) \times S \to \{Q \times S^* \times \{-1,0,+1\}\}$$

$Q\times(V \cup \{\Delta,\Gamma\})\times S \rightarrow Q\times S^* \times\{-1,0,+1\}$

$(q, a_1, \ldots, \overline{a_i}, \ldots, a_n, X_1X_2 \ldots X_m) \rightarrow (p, a_1, \ldots, \overline{a_{i+d}}, \ldots, a_n, X_1X_2 \ldots X_{m-1}\, r)$

$d \in \{-1,01+1\}, r \in S^*$



Konfiguracija magacinskog automata:

$(q, \# a_1 a_2 \cdots \overline{a}_i \cdots a_n \#, X_1 X_2 \cdots X_m)$

$a_i \in V, i = 1,2,\cdots,n$

$X_m - Vrh\ magacina$

Ako je $(p,r,d) \in g(q,a_i,X_m)$

$p \in Q, \quad r \in S^*, \quad d \in \{-1,0,+1\},$

$(q, \# a_1 a_2 \cdots \overline{a}_i \cdots a_n \#, X_1 X_2 \cdots X_m) \mapsto$

$(p, \# a_1 a_2 \cdots \overline{a}_i + d \cdots a_n \#, X_1 X_2 \cdots X_{m-1}, r)$

Proces prepoznavanja se obavlja sve dok se ne naiđe na **završni granični simbol** # i kada se **isprazni stek**, reč je **prepoznata**.
T1: 1N magacinski automati raspoznaju jezik L ako i samo ako je L beskonteksni jezik **tipa 2**.
T2: 2N magacinski automati pored beskonteksnih raspoznaju i neke, **ali ne sve**, konteksne jezike.
T3: Postoje beskonteksni jezici koji ne mogu da se prepoznaju **1D automatima**.
T4: 2D magacinski automati pored beskonteksnih prepoznaju i **neke** konteksne jezike.

Najprimitivniji algoritam za prepoznavanje gramatika tipa „dva“:

Radna memorija je organizovana kao **magacin**, i na početku u njoj je jedan simbol – Z0. Na **ulaznoj traci** je reč koju treba preslikati, ograničena levim i desnim **graničnim simbolom**. Magacinski automat, na osnovu tekućeg stanja operacionog organa, tekućeg simbola i simbola na vrhu steka određuje **novo stanje** **O**peraciojskog **O**rgana (**OO**), pomeranje ulazne glave i reč koja se upisuje na vrh magacina (proizvoljne dužine).

---

3. Navesti osnovne transformacije koda koje se primenjuju u okviru osnovne optimizacije

---

Uvek se vrši prva **osnovna optimizaija**. Cilj je poboljšati kod koji se dobija u prvom prolazu (nakon koga ima puno redundantnosti tj. puno load-store parova natredbi).
Ona je uvek neophodna. Izvršava se tako što se kroz kod prolazi prozorom koji sadrži 2-3 naredbe (koje se optimizuju) tako da se osnovna funkcionalnost polaznog programa ne naruši, a da se poveća efikasnost i ako je moguće smanje memorijski zahtevi.

Dakle, možemo nabrojati osnovne faze optimizacije:
- Eliminisanje redundantnih naredbi
- Optimizacija toka programa
- Mala algebarska uprošćenja
- Korišćenje mašinskih idioma

Eliminisanje redundantnih naredbi

1. Pre svega se izbacuju suvišni parovi naredbi load-store. Ne izbacuju se samo kada se druga nardba u labeli. Na primer u slu□aju da imamo:

```
ST a
Lab LD a
```

2. Zatim, eliminišu se bezefektni parovi

```
Mov R0,A
Mov A,R0
ili
Mov r0,r1
Mov r1,r0
```

3. Eliminišu se strukture u kojima postoji mrtav kod!

```
A := 0;
If a <> 1 goto l1
Goto l2
L1: print
L2: bilo šta
```

U ovom slučaju se „goto l2“ nikada neće izvršiti (0<>1) pa se ta naredba eliminiše

Optimizacija toka programa

U toku generisanja međukoda često se javljaju skokovi na skokove, skokovi na uslovne skokove ili uslovni skokovi na bezuslovne skokove. U mnogim takvim slučajevima moguće su redukcije.

1.

```
Goto l1
L1: goto l2
```

2.

```
If a<b goto l1
L1: goto l2
```

3.

```
Goto l1
L1: if a<b goto l2
L3:
```

Mala algebarska uprošćenja
X := x + 0
X := x*1
Ili
X := Y+0
X := y*1
Direktna redukcija
- Umesto stepenovanja se iskoristi množenje određeni broj puta.
- Šiftovanje u levo/desno se koristi za množenje/deljenje nekim brojem, koji je stepen dvojke.

Korišćenje mašinskih idioma
Koriste se specifičnosti mašine na primer:
- na nivou adresiranja,
- različite grupe registara,
- na nivou implementacije steka
Postoji i **mašinski nezavisna optimizacija**, međutim, ona se uglavnom vrši na nivou međukoda
Kod koji će se ovako generisati treba da je što bliži onom koji bi DOBAR programer ručno napisao. Kompajliranje bi trebalo biti brzo, ali je bitniji kvalitet dobijenog koda.

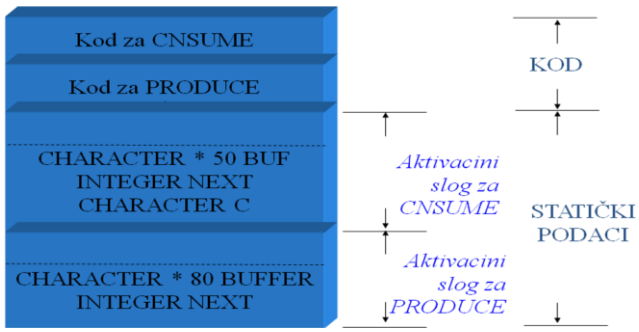
---

4. Objasniti pojam statičke alokacije memorije i navesti sekvencu naredbi koju u tom slučaju generiše kompilator

---

Strategije alokacije memorije
- Statička alokacija svih objekata u vreme kompilacije.
- Alokacija pomoću steka
- Dinamička alokacija pomoću Heap-a.

Statička alokacija memorije
- U toku kompiliranja programa generiše se po jedan aktivacioni slog za svaku od procedura tako da se isti aktivacioni slog koristi kod svakog poziva potprograma.
- Nema mogućnosti za rekurzivne pozive procedura
- Strukture podataka se ne mogu kreirati dinamički.
- Primenjuje se kod implementacije jezika Fortran 77.



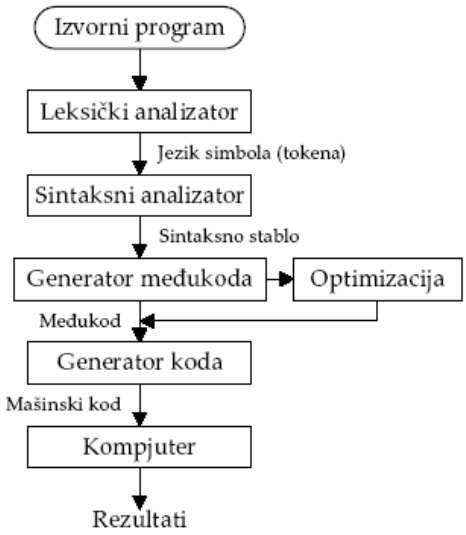
---

5. Dati šemu strukture kompilatora i objasniti sve njegove delove

---

Kompilator je prevodilac koji prevodi kod u celini i generiše mašinski kod. On prolazi kroz faze **sinteze** i **analize**.
Sinteza u sebi sadrži i **optimizaciju koda**. U procesu prevođenja cilj je dobiti kod koji je što bliži asemblerskom kodu koji bi "ručno" pisao DOBAR programer. Dakle, sinteza se sastoji iz 2 faze:
- Generisanje koda
- Optimizacija koda

Analiza je kompleksinija. Ona se sastoji iz 3 faze:
- Leksička analiza
- Sintaksna analiza
- Semantička analiza



Pos := init + r * 60

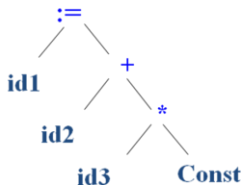
Leksički analizator

id1 := id2 + id3 * const

-Leksički analizator je prva faza kompilatora. Njegova glavna uloga je da čita ulazni kod i gneriše sekvencu simbola (tokena) koju prosleđuje sintaksnom analizatoru. **L**eksički **A**nalizator (**LA**) treba da pri analizi izdvoji leksičke celine (**reči i slova**). On analizira
slovo po slovo dok ne dođe do tzv. **graničnog znaka** (lekseme) koja je tu da bi odvojila jednu celinu od druge (jednu leksemu od druge).
Na osnovu početnog simbola (lekseme) se vrši razvrstavanje:
Ukoliko je prvi znak slovo (leksema) može biti **identifikator** ili **ključna reč**. Ako je ključna reč, onda postoji u **tablici ključnih reči**, u suprotnom je identifikator.

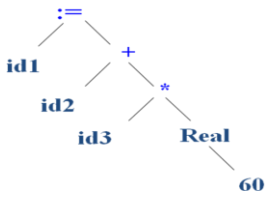
id1 := id2 + id3 * const

Sintaksni analizator



-Postoji 4 tipa gramatika i za svaku svojstven (poseban) **sintaksni analizator**.
Skup pravila se koristi za sintaksnu analizu, i na osnovu njega se generiše **stablo sintaksne analize**.

Semantički analizator



-Za rešavanje nekih semantičkih problema kao što su:

- Svaka oznaka (labela) naredbe na koju se pozivamo mora stvarno i da postoji u programu.
- Identifikatori ne mogu više puta da budu opisani
- Sve promenljive treba da budu odrešene pre korišćenja.
- Stvarni argumenti funkcija treba da budu usklađeni po tipu sa fiktivnim argumentima.

vršimo semantičku nadgradnju sintaksnih analizatora. U algoritam sintaksne analize ubacujemo semantičke rutine, koje se aktiviraju na određenim mestima zavisno od primenjene metode (top-down ili bottom-up).

Generator međukoda

```
temp1 := real(60)
temp2 :=  id3 * temp1
temp3 := id2 + temp2
  id1 := temp3
```

-Generator međukoda generiše međukod, na osnovu koga se može generisati kod za različite ciljne mašine. Na nivou međukoda može da se vrši mašinski nezavisna optimizacija.

Generator koda

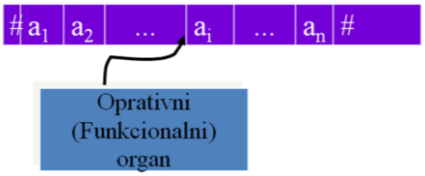
```
MOV     id3, R2
MULF    #60.0, R2
MOV     id2, R1
ADD     R2, R1
MOV     R1, id1
```

-Generator koda generiše odgovarajuće asemblerske naredbe koje se izvršavaju na određenoj mašini.

---

6. Objasniti konačne automate kao uređaje za prepoznavanje jezika

---



To je automat koji nema radnu memoriju. **Konačan automat** možemo opisati sa:

$g: Q \times V \to Q$ $\qquad$ $Q \times (V \cup \#) \to Q$

Na osnovu stanja operastivnog organa i simbola koji čita ulazna glava, menja se stanje automata i ulazna glava pomera za jedno mesto ulevo ili udesno ili ostaje na svom mestu.

MA=(Q, V, g, q_0,F)
Q – Skup stanja operativnog organa
V – Skup simbola na ulaznoj traci
F – Skup završnih, krajnjih stanja (Podskup skupa Q)
q_0 – Početno stanje operativnog organa
g – Preslikavanje

U zavisnosti od: **stanja automata** i **ulaznog simbola** menja se stanje. Moguće je uspostaviti direktnu vezu između **grafa automata** i **jezika** nad određenom gramatikom G koji treba prepoznati.

G ->Graf
Analiza grafa ->$L(G)$ i obrnuto
Graf ->G

Za neki ulazni niz $w \in V^*$ kazemo da je prepoznat automatom ako je $g(q_0, w) \in F$.

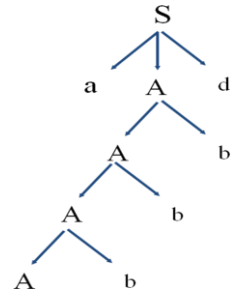
Skup svih reči koje prepoznaje KA nazivamo regularnim i označavaćemo ih sa $T(k)$:

$$T(k) = \{x \mid g(q_0, x) \in F\}$$

7. Objasniti problem leve rekurzije i dati transformacije kojima se on eliminiše

---

Kod osnovnih se smene biraju tako da odgovaraju **levom izvođenju** (preslikavanju). Ako prva smena počinje istim neterminalnim simbolom S->Sa, ulazimo u beskonačnu petlju. Ovaj problem je poznat od imenom **problem leve rekurzije**. I na njega uvek treba obratiti pažnju pri pisanju pravila.



$S \to aAc$
$A \to Aa$
$A \to \varepsilon$

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

$A \Rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$

$A \Rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \mid \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$
$A' \Rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A'$

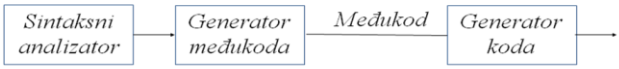
$A \Rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$

$A \Rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$
$A' \Rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

---

8. Objasniti ulogu i mesto međukodova u procesu prevođenja jezika. Dati primere međukodova. Šta je to troadresni međukod? Navesti osnovne naredbe troadresnog međukoda

---



Efekti:

1. Prenosivost koda. Na osnovu međukoda može se generisati kod za različite ciljne mašine
2. Na nivou međukoda može da se vrši mašinski nezavisna optimizacija

Vrste međukodova

- Troadresni međukod (neka vrsta pseudo-asemblerskog jezika)
- Stabla kao statičke ili dinamičke strukture podataka
- Poljska inverzna notacija

Uvođenje **mašinski-nezavisnog** međukoda pruža dve osnovne prednosti, a to su:
1. Prva faza je potpuno nezavisna od mašine za koju se kompilator pravi, pa se pri projektovanju kompilatora za isti jezik a drugi tip računara menja se samo druga faza.
2. Proces optimizacije je moguće izvršiti i na nivou međukoda te i taj proces postaje nezavisan od odredišnog računara.
Međukodna prezentacija ulaznog programa se može smatrati kao program za neku apstraktnu mašinu. Međukod mora da zadovoljava sledeće dve osobine:
- da se lako generiše,
- da se lako transformiše u bilo koji asemblerski jezik.

Naredbe troadresnog međukoda

1. Naredba dodeljivanja x := y *op* z, gde je *op* aritmetički ili logički binarni operator.
2. Naredba dodeljivanja x := *op* y, gde je *op* aritmetički ili logički unarni operator (npr. minus, logička negacija, shift operator, operatori za konverziju tipova i sl.)
3. Nredba kopiranja x := y kojom x dobija vrednost y.
4. Naredba bezuslovnog skoka go to L.
5. Naredba uslovnog skoka if x *relop* y goto L.
6. param x i call p,n za poziv potprograma i return y za povratak iz potprograma. Primer:

```
param x1
param x2
....
param xn
call p,n
```

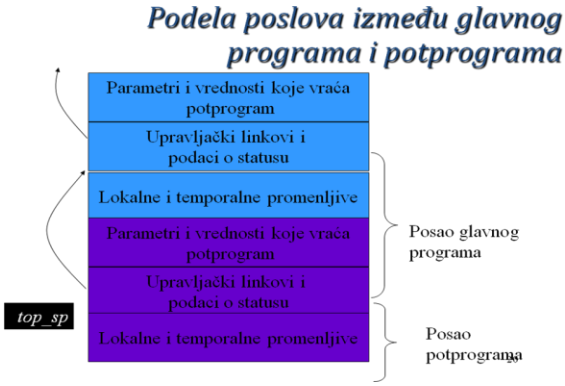
7. Indeksirana dodeljivanja u obliku: x := y [i] i x [i] : = y
8. Dodeljivanje adresa i pointera u obliku: x := &y , x:= *y

---

9. Objasniti pojam alokacije memorije pomoću steka i navesti sekvencu naredbi koju u tom slučaju generiše kompilator

---

- Memorija je organizovana kao stek.
- Prilikom svakog poziva procedure u stek se ubacuje njen aktivacioni slog.
- Slog se izbacuje iz steka kada se procedura završi.
- Ova tehnika omogućava rekurzivne pozive potprograma. U tom slučaju u steku se nalazi više aktivacionih slogova istog potprograma i svaki sadrži odgovarajuće podatke.
- U vreme kompiliranja programa zna se samo veličina aktivacionog sloga ali ne i dubina rekurzije (koliko će aktivacionih slogova jednog potprograma biti generisano).



Sekvenca poziva potprograma

- Glavni program priprema stvarne parametre
- Glavni program pamti adresu povratka i staru vrednost top_sp u aktivacionom slogu pozvanog programa. Posle toga inkrementira se vrednost top_sp na poziciju kao na slici iza polja sa lokalnim podacima glavnog programa i polja sa parametrima potprograma i njegovim podacima o statusu.
- Potprogram pamti podatke o statusu: vrednosto registara i druge informacije o statusu.
- Potprogram inicijalizuje vrednosto lokalnih podataka i započinje izvršavanje

Moguća return sekvenca

- Potprogram smešta rezultate koje vraća u polje aktivacionog sloga neposredno iza sloga glavnog programa.
- Koristeći zapamćene informacije o ststusu potprogram obnavlja stanje procesora i vraća vrednost *top_sp*.
- Kako je vrednost top_sp dekrementirana glavni program može da pristupi rezultatima i preslika ih u svoj aktivacioni slog.

---

10. Dati definiciju formalnog jezika i formalne gramatike. Navesti osnovne tipove gramatika. Dati klasifikaciju formalnih gramatika po Čomskom

---

Formalni jezik

- Formalnim jezikom L nad azbukom V naziva se bilo koji skup reči nad tom azbukom.
- Prema ovoj definiciji formalni jezik je i skup f, prazan skup reči kao i skup { e } koji sadrži samo reč e.

Formalne gramatikeNoam Chomsky

G = (V_n, V_t, S, P)

Važi: $V = V_n \cup V_t \quad i \quad V_n \cap V_t = \phi$

$x \to y, \quad gde\ je \quad x \in V^* V_n V^* \wedge y \in V^*$

P je skup smena oblika:

Jezik: $L(G) = \{w \mid S \xrightarrow{*} w, w \in V_t^*\}$

Postoji 4 tipa gramatika i za svaku svojstven (poseban) **sintaksni analizator**.

- LL(k) gramatikama,
- Operatorskim gramatikama prvenstva,
- Gramatikama prvenstva,
- LR gramatikama.

---

11. Dati najopštiju definiciju LL(1) gramatika

LL(1) gramatike omogucavaju "top-down" postupaksintaksne analize bez vracanja.Osnovna ideja je da se ne pokušava uvek sa primenom prve smene za preslikavanje tekuceg neterminalnog simbola, nego se "pogleda" sledeci simbol iz ulaznog niza i na osnovu toga odluci koja
ce se smena primeniti.
Poreklo imena LL(1)
_ Ulazni niz se analizira s Leva na desno.
_ Nalazi se Levo izvodjenje sintaksnog stabla.
_ Pri odluci koja ce se smena primeniti, "pogleda" se (1) naredni simbol iz ulaznog niza.

Gramatika G jeste LL(1) gramatika ako i samo akoza svaki skup smena koji na levoj strani imaju isti neterminalni simbol važi:
X 4 $\to \alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha n$
1. FIRST(α1) $\cap$FIRST(α2) $\cap \ldots \cap$FIRST(αn) = $\varnothing$
2. Samo jedan od nizova α1, α2 , … , αn se može preslikati u prazan niz ε.
3. Ukoliko se niz αk preslikava u prazan niz, onda
FIRST(α1) $\cap$ FIRST(α2) $\cap \ldots \cap$ FIRST(αk-1) $\cap$
FIRST(αk+1) ... $\cap$ FIRST(αn) $\cap$ FOLLOW(X)= $\varnothing$

---

12. Objasniti osnovne funkcije leksičkog analizatora

Leksički analizator je prva faza kompilatora. Njegova glavna uloga je da čita ulazni kod i gneriše sekvencu simbola (tokena) koju prosleđuje sintaksnom analizatoru.

Dodatni zadaci

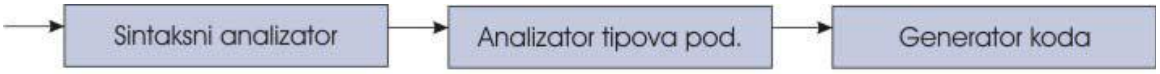
- Izbacuje iz ulaznog koda delove koji nisu značajni za sintaksnu analizu: komentare, praznine (blanko znake), *tab* i *newline* simbole.
- Usklađuje listing gršaka sa ulaznim kodom. Npr. vodi računa o broju newline simbola, što se koristi kod referenciranja na grške.
- Često se realizuje tako da su njegove funkcije razdvojene u dve faze:
    - Ulazna konverzija - Odgovorna za transformaciju ulaznog koda, izbacivanje praznina i sl.
    - Leksička anliza -Odgovorna za generisanje tokena.

Pitanje: Zašto se izdvaja leksički analizator od sintaksnog analizatora

- Jednostavnija realizacija je možda najvažniji razlog.
- Povećanje efikasnosti kompilatora. Posebnim tehnikama baferisanja koje se koriste u realizaciji leksičkog analizatora doprinosi se njegovoj efikasnosti.
- Prenosivost kompilatora – U fazi leksičke analize eliminišu se svi mašinski zavisni elementi i generiše mašinski nezavisna sekvenca tokena koja se prosleđuje sintaksnom analizatoru.

---

13. Objasniti ulogu modula za analizu tipova podataka i kako se realizuje ovaj modul



E →literal {E.type:=char}
E →num {E.type:=integer}
E →id {E.type:=lookup(id.entry)}
E →E1 mod E2 { E.type:=if E1.integer and E2.type=ineger then integer else type_error}
E →E1[E2] {E.type:=if E2.type=integer and E1.type=array(s,t) then t else error}
*t je elementarni tip dobijen iz strukturnog tipa array(s,t)
E →E1^ {E.type:=if E1.type=pointer(t) then t else type_error}
E →E1(E2) {E.type:= if E2.type=s and E1.type=s →t then t else error}

S →id:=E {S.type:=if id.type=E.type then void else type_error}
S →if E then S1 {S.type:=if E.type=boolean then S1.type else error}
S →while E do S1 {S.type:=if E.type=boolean then S1.type else error}
S →S1;S2 {S.type:=if S1.type=void and S2.type=void then void else error}

---

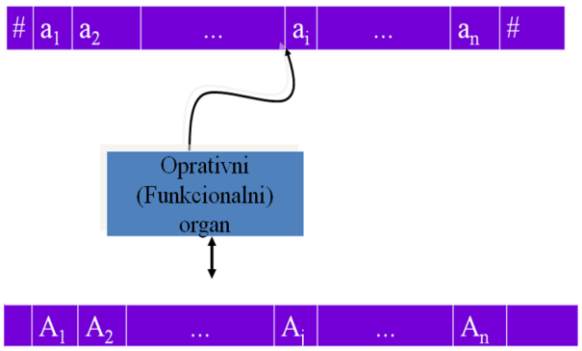
14. Leva faktorizacija

$$A \Rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_{n1} \mid \gamma \qquad A \Rightarrow \alpha A' \mid \gamma \quad A' \Rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

---

15. Tjuringova mašina i Linearno ograničeni automat

Tjuringova mašina



2N Tjuringova mašina

- Operativni organ menja stanje.
- Ulazna glava se pomera za jedno mesto.
- Na radnu traku se upisuje novi simbol.
- Pomera se radna glava.

TM=(Q, V, S, g, q_0, b F)

Q – Skup stanja operativnog organa
V – Skup simbola na ulaznoj traci
S – Skup simbola na radnoj traci
F – Skup završnih, krajnjih stanja (Podskup skupa Q)
q_0 – Početno stanje operativnog organa
b – Oznaka za prazno slovo
g – Preslikavanje

$$Q \times (V \cup \#) \times S \rightarrow \{Q \times (S \setminus \{b\}) \times \{-1,0,+1\} \times \{-1,0,+1\}\}$$

$$Ako\ je\ (P, Y, d_1, d_2) \in g(q, a, X)$$

Ako se mašina nalazi u stanju *q*, pri čemu ulazna glava čita simbol *a*, a radna glava čita simbol *X*, mašina prelazi u novo stanje *P*, upisuje na radnu traku slovo *Y* i pomera ulaznu glavu u smeru d_1, a radnu glavu u smeru.

$$d_1, d_2 \in \{-1,0,+1\}$$

+1 Pomeranje za jedno mesto ulevo
0 Nema pomeranja
-1 Pomeranje za jedno mesto udesno

Mašina je uspešno prepoznala ulazni niz ako se u trenutku kada se ulazna glava pozicionira na granični simbol # TM nađe u nekom od završnih stanja (elemenata skupa X).
TM je deterministička ako se svaki element (q, a, X) preslikava u najviše jedan element (P, Y, d_1, d_2).
Četri klase TM 2N, 1N, 2D, 1D međusobno su ekvivalentni i svaka od njih raspoznaje jezike tipa nula.
TM je pojam ekvivalentan pojmu automata.

Linearno ograničeni automat

Linearno ograničen automat je 2N Tjuringova mašina za koju može da se odredi konstanta k takva da se prilikom prepoznavanja reči:

$$w \in V^*, \quad |w| = n$$

na radnoj traci upisuje niz r pri čemu je:

$$|r| \leq kn$$

Za prepoznavanje svih reči jezka dovoljna je radna traka k puta duža od najduže reči jezika.

Klase 2n i 1N LOA međusobno su ekvivalentne i svaka od njih prepoznaje jezike tipa 1. (Za svaki jezik tipa 1. može se definisati nedeterministički LOA koji ga prepoznaje)
Klase 2D i 1D LOA međusobno su jednake.
Ekvivalentnost 2N i 2D LOA do sada nije dokazana.
Svaka klasa LOA ne prepoznaje sve jezike tipa 1.

---------- *ATRIBUTNE GRAMATIKE*
SINTAKSNO UPRAVLJANO PREVOĐENJE

- Svakom simbolu gramatike se pridružuje skup atributa.
- Atributi mogu da budu generisani ili nasleđeni.
- Ako simbol gramatike zamislimo kao čvor u stablu sintaksne analize, pri čemu se taj čvor implementira kao slog sa više polja, atributi odgovaraju imenima tih polja.
- Atributi mogu da predstavljaju bilo šta string, broj, memorijsku lokaciju ili štagod zamislimo.
- Vrednost atributa određena je semantičkom rutinom koja je pridružena produkcionom pravilu koje se koristi u odgovarajućem čvoru.
- Vrednosti generisanih atributa se izračunavaju kao funkcije atributa potomaka odgovarajućeg čvora.
- Vrednosto nasleđenih atributa se izračunaveju kao funkcije roditeljskih atributa i ostalih atributa na istom nivou.