**PPJ 2. MI TEORIJA**

1. **Objasniti parsiranje od dna prema vrhu metodom prednosti operatora, relaciju prednosti, akcije parsera i određivanje uzorka za zamjenu.**

Ova metoda parsiranja vrijedi za konteksno neovisnu *operatorsku* gramatiku. Konteksno neovisna gramatika je *operatorska*  akko produkcije nisu epsilon produkcije i na jednom mjestu desne strane produkcije dva susjedna znaka nisu nezavršna.

Za završne znakove gramatike *a* i *b*  definira se *relacija prednosti*  koja se koristi u postpuku parsiranja. Relacije prednosti su:

|  |  |
| --- | --- |
| **RELACIJA** | **ZNAČENJE** |
| *a ← b* | Prednost *a* je manja od prednosti *b* |
| *a↔b* | *a* i *b* su jednake prednosti |
| *a→b* | Prednost *a* je veća od prednosti *b* |

Relacija prednosti se zadaje u tablici gdje su reci i stupci tablice označeni završnim znakovima gramatike. Relacija prednosti definira se za sve završne znakove gramatike, uključujući oznaku dna stoga i oznaku kraja niza . 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 akcije *Pomakni* ili *Reduciraj:*

Akcija ***Pomakni***

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

Akcija ***Reduciraj***

Ako je prednost završnog znaka na vrhu stoga veća od prednosti znaka u ulaznom nizu, onda parser primjenjuje akciju *Reduciraj.*

**Određivanje uzorka za zamjenu**

Parser uzima jedan po jedan znak sa stoga sve do završnog znaka koji ima manju prednost od prednosti prethodno uzetog znaka.

1. **Objasniti postupke nadziranja i oporavka od pogrešaka tijekom LR parsiranja.**

Dva su osnovna postupka nadziranja pogrešaka: *postupak traženja sinkronizacijskog znaka* i *postupak lokalnih promjena*.

***Postupak traženja sinkronizacijskog znaka*** prekida sintaksnu analizu programske cjeline u kojoj je pronađena pogreška. Postupak oporavka od pogreške skida sa stoga stanja koja pripadaju nezavršnom znaku (npr. A) s pogreškom (sve do stanja **s**  za koji je definirano NOVOSTANJE(s, A)). Pomakne se kazaljka s ulaznog niza do sinkronizacijskog znaka.

***U postupku lokalnih promjena*** svi elementi tablice *LR* parsera koji označavaju akciju *Odbaci ()* se zasebno analiziraju. Na temelju pravila izvornog jezika, pretpostavi se najvjerojatnija pogreška korisnika.

1. **Opisati postupak izgradnje potisnog automata za prijevodnu gramatiku.**

Prijevodna gramatika dijeli skup završnih znakova formalne gramatike na ulazne i izlazne završne znakove.

Potisni automat *M=(Q, Σ, Γ, δ, q0 , Z0 , ᴓ )* prihvaća nizove praznim stogom.

1. Potisni automat ima samo jedno stanje koje je ujedno i početno stanje, Q={q0}
2. Skup ulaznih znakova potisnog automata jednak je skupu završnih znakova gramatike uvećan oznakom kraja niza  **,** Σ=T U {}.
3. Prethodno definiranom skupu znakova stoga dodaju se *izlazni završni znakovi*. Izlazni završni znak je znak stoga akko je na desnoj strani produkcije od krajnje lijevog nezavršnog ili ulaznog završnog znaka koji se stavlja na stog.
4. Skup znakova stoga potisnog automata čine oznaka dna stoga , skup nezavršnih znakova gramatike *V* i skup onih završnih znakova gramatike *T'* koji su na desnim stranama produkcija, ali ne isključivo na krajnje lijevim mjestima.
5. Tablica potisnog automata se popunjava na sljedeći način:

----------------------------------------------------------------------

1. **Objasniti razlike u ostvarenju LR(0), SLR(1), LALR() i LR(1) parsera te navesti njihove prednosti i nedostatke.**
2. **Nabrojati i objasniti formalne metode semantičkog analizatora.**
3. **Definirati L-atributnu prijevodnu gramatiku.**

Atributna prijevodna gramatika je L – atributna akko vrijedi:

1. Vrijednost *vrijednost nasljednog svojstva* znaka desne strane produkcije računa se na temelju *vrijednosti nasljednih svojstava* nezavršnog znaka lijeve strane produkcije i na temelju vrijednosti svojstava dense 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 nezavršnog znaka* računa se na temelju nasljednih svojstava tog istog završnog znaka.
4. **Objasniti konstrukciju epsilon-NKA u postupku izgradnje SLR parsera.**

Neka je *G=(V, T, P, S)* konteksno neovisna gramatika za koju vrijedi da početni nezavršni znak *S* nije na desnoj strani nijedne produkcije (definira se novi poč. znak gramatike *S'* i produkcija *S' -> S,* koja omogućuje prestanak rada parsera).

Automat ε-NKA *M=(Q, Σ, δ, q0, F)* gradi se na sljedeći način:

1. Skup stanja je skup svoj *LR*  stavki gramatike *G* uvećan početnim stanjem *q0*. Osim početnog stanja *q0,* sva ostala stanja su *LR* stavke.
2. Budući da su sva stanja *LR* stavke ulaznog niza γ, onda su sva stanja u skupu dozvoljenih stanja *F=Q.*
3. Funkcija prijelaza δ se definira na sljedeći način:
4. Početni prijelaz iz S omogućuje početak rada eps-NKA
5. Ako automat pročita znak X, onda se oznaka točke pomakne iza tog znaka. Znak X je završni ili nezavršni znak gramatike.
6. Ako je nezavršni znak *B*  desno od točke, onda ε prijelaz kreće u analizu tog nezavršnog znaka.
7. **Definirati LL(1) gramatiku i kratko opisati konstrukciju potisnog automata za LL(1) gramatiku.**

Kontekstno neovisna gramatika je LL(1) gramatika ako i samo ako vrijedi :

Ako više produkcija ima isti nezavršni znak na lijevoj strani, onda njihovi skupovi PRIMIJENI nemaju zajedničkih elemenata.

Za zadanu *LL(1)* gramatiku *G=(V, T, P, S)* moguće je izgraditi deterministički potisni automat *M=({q0}, Σ, Γ, δ, q0 , Z0 , ᴓ )*. Osim nadopune koraka 5a, svi ostali koraci izgradnje su jednaki koracima gradnje potisnog automata za *Q* gramaiku:

1. Pravila za produkcije A--> b α, A--> b i A-->ε dodaju se dva nova pravila:
2. Ako je produkcija oblika:

A--> α i ako niz započinje nezavršnim znakom gramatike, onda se u redak tablice *A* i sve stupce određene znakovima skupa *PRIMJENI(A🡪* α*)* zapisuju akcije:

*Zamijeni (*αr*);*

*Zadrži;*

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- produkcije:

*Izvuci;*

*Zadrži;*

1. **Objasniti sintaksom vođenu semantičku analizu.**

Budući da su *sintaksom upravljani jezični procesori* zasnovani na primjeni *atributne prijevodne gramatike*, sintaksni analizator je središnji proces analize izvornog i sinteze ciljnog programa. Sintaksni analizator upravlja radom ostalih koraka rada jezičnog procesora. Ako zatreba uniformni znak, onda sintaksni analizator pokrene izvođenje leksičkog analizatora. Nadalje, sintaksni analizator pokreće izvođenje semantičkih akcija u cilju provjere semantičkih pravila, generiranja međukoda, optimiranja međukoda i generiranja ciljnog programa. Semantičke akcije su potprogrami namijenjeni analizi pojedinih sintaksnih cjelina. Tijekom gradnje sintaksnog stabla, sintaksni analizator grupira znakove u hijerarhijske cjeline i pokreće izvođenje semantičkih akcija pridruženih tim sintaksnim cjelinama.

1. **Navesti i objasniti tri najčešće primjenjivana formalna modela semantičkog analizatora.**

Tri najčešći primjenjivana formalna modela semantičkog analizatora zasnivaju se na *prevođenju jezika*, *simulaciji na apstraktnom stroju* i *skupu aksioma*.

Najjednostavniji formalni model semantičkog analizatora zasniva se na *prevođenju izvornog jezika* u jezik koji ima definiranu semantiku.

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

Treći formalni model koristi *skup aksioma.* Aksiomi su logičke tvrdnje koje korisnik zadaje u dijelovima izvornog programa, a njima izriče očekivani rezultat izvođenja tog dijela izvornog programa. Nedostatak je dvorstuko opterećenje korisnika: korisnik zadaje naredbe izvornog programa kojima definira način rješavanja problema, a primjenom aksioma definira koje logičke tvrdnje je potrebno zadovoljiti.

1. **Opisati algoritam za izračunavanje IS\_FOLLOWED\_BY( *Ispred* ) relacije.**

Neka je zadana gramatika koja ima početni nezavršni znak *<S>*. Relacija:

*Ispred (A, B)*

Vrijedi akko je znak A neposredno ispred znaka *B*u barem jednom međunizu generiranom iz početnog nezavršnog znaka *<S>*.

1. **Objasniti postupak određivanja relacija prednosti na temelju zadane gramatike.**
2. **Opisati primjenu lijevog faktoriranja u pretvorbi pravila produkcija tako da zadovoljavaju uvjete LL(1) gramatike.**
3. Neka gramatika ima *n* produkcija oblika:

*<A>--> αβ1*

*<A>--> α β2*

*-------*

*<A>--> α βn*

gdje je *<A>* nezavršni znak gramatike, *α* i *βi* su nizovi nezavršnih znakova i završnih znakova gramatike. Skupovi *PRIMIJENI* različitih nizova *βi*  nemaju zajedničkih završnih znakova. Budući da sve produkcije započinju istim znakom *α* , svih *n* produkcija ima iste završne znakove *ZAPOČINJE(<α>)* u skupovima *PRIMIJENI* , odnosno gramatika nije *LL(1)* gramatika.

1. Prethodno zadanih *n* produkcija zamijeni se sljedećim *n+1* produkcijama koje zadovoljavaju uvjete *LL(1)* gramatike:

*<A>--> α <Nastavak>*

*<Nastavak>--> β1*

*<Nastavak>--> β2*

*---------------*

*<Nastavak>--> βn*

1. **Navesti algoritam za izračunavanje FIRST skupova za produkcije.**

Za znakove gramatike *A* i *B* vrijedi relacija:

*ZapočinjeIzravnoZnakom (A, B)*

Akko je barem jedna od zadanih produkcija gramatike oblika:

*A--> αB β,*

gdje je *α* niz praznih znakova *αε*, a *β* je proizvoljni niz znakova gramatike.

1. **Objasniti LALR metodu za gradnju tablice LR parsera.**

Program YACC koristi *LALR (Look Ahead-LR)* postupak gradnje tablice *LR* parsera. LALR postupak gradi znatno manju tablicu *LR* parsera od kanonskog *LR* postupka. Osnovna ideja *LALR* postupka je grupiranje stanja označenih istim *LR(0)* stavkama u jedinstveno stanje. Prednost *LALR* parsera je manji broj stanja, a nedostatak mogućnost nastanka proturječja.

1. **Definirajte tipove nejednoznačnosti kod LR parsiranja i opišite razrješavanje nejednoznačnosti kod YACC-a.**

Ako se zada nejednoznačna gramatika, onda program YACC razrješava nejednoznačnost na sljedeći način:

1. *Reduciraj/Reduciraj* proturječje razrješava se primjenom one redukcije koja je prije zadana u pravilima prevođenja
2. *Pomakni/Reduciraj* proturječje razrješava se primjenom akcija *Pomakni.*

YACC razrješava *Pomakni/ReducirajI* proturječje na temelju prednosti produkcija i završnih znakova.

1. **Opišite postupak konstrukcije epsilon-NKA koji prihvaća skup svih mogućih živih prefiksa za zadanu gramatiku.**