UDK 004.4'422

SINTAKSIČKA ANALIZA KAO CENTRALNA FAZA U RADU KOMPILATORA

Vesna Simović, dipl. matematičar
Visoka ekonomska škola strukovnih studija Peć u Leposaviću

Rezime: Kompilator se projektuje kao modularan i dobro strukturiran programski sistem. Proces kompilacije, u logičkom pogledu, podeljen je na etape, a svaka etapa na faze. U fizičkom pogledu, proces kompilacije se deli na prolaze. Dve glavne etape procesa kompilacije su analiza i sinteza, one čine samo jezgro procesa kompilacije. U ovom radu obrađena je faza analize – sintaksička analiza.

UVOD

Jedan program može biti zapisan u različitim notacijama, u različitim programskim jezicima. Uloga kompilatora je da transformiše tekst programa iz reprezentacije na visokom programskom jeziku, koja je čitljiva čoveku, u "čitljivu" za računar, što omogućava eventualno izvršavanje programa.

U osnovi, kompilacija je jedna vrsta prevođenja: na osnovu teksta koji je napisan u višem programskom jeziku i koji opisuje neki algoritam, treba sastaviti novi tekst, koji opisuje isti algoritam, ali na mašinskom jeziku mašine za koju se programiralo. U ovom smislu, kompilacija se sastoji u čitanju niske karaktera koja je sastavljena u skladu sa izvesnim sintaksičkim pravilima, sa ciljem da se konstruiše jedna drugačija reprezentacija informacije koju ti karakteri izražavaju.

Vesna Simović, dipl. matematičar

Sintaksička analiza je faza rada kompilatora, koja pripada etapi analize izvornog koda. Tokom analize se izvorni program prevodi u podesnu posrednu reprezentaciju, koja prikazuje strukturu izvodnog programa. U sintaksičkoj analizi se ispituje i rekonstruiše struktura programa kao celine na osnovu niza tokena koje prosleđuje leksički analizator.

1. Sintaksički analizator

Faza sintaksičke analize, koju obavlja sintaksički analizator, je ključni deo u radu prednjeg procesora, a i kompilatora u celini.[1] Sintaksički analizator na ulazu prima tokene koje mu upućuje leksički analizator i proverava njihovu saglasnost sa sintaksičkim pravilima analiziranog jezika. Pravila gramatike su zadata nekom kontekstno slobodnom gramatikom. Na osnovu ovoga, sintaksička analiza je proces utvrđivanja da li se zadata reč jezika može generisati gramatikom ili ne. Tokom provere saglasnosti sa sintaksičkim pravilima, sintaksički analizator gradi *drvo sintaksičke analize* i priprema fazu prevođenja polaznog jezika na međujezik. Pored ovih osnovnih funkcija, sintaksički analizator obrađuje i eventualne sintaksičke greške koje su se pojavile na ulazu.

Gramatike su osnovni alati u specifikaciji jezika, one imaju nekoliko važnih aspekata. Prvo, gramatika služi da postavi strukturu u linearan niz tokena koji su u programu. Drugo, koristeći tehike iz polja formalnih jezika, gramatika može biti zadužena da konstruše sintaksički analizator automatski. Ovo je velika pomoć u konstrukciji kompajlera. Treće, gramatike pomažu programerima da pišu sintaksički ispravne programe i

[1] Duško M. Vitas, Prevodioci i interpretatori (Uvod u teoriju i metode kompilacije programskih jezika), Matematički fakultet, Beograd 2006, str. 173

obezbede odgovore na detaljna pitanja o sintaksi. Tu istu pomoć pružaju i u konstrukciji kompajlera.[2]

Postoje tri opšta tipa sintaksičkog analizatora za gramatike.[3] Univerzalna metoda sintaksičke analize kakva je Cocke-Younger-Kasami algoritam i Earley-ev algoritam koji mogu da analiziraju bilo koju kontekstno slobodnu gramatiku. Ove metode su suviše neefikasne da bi se koristile u produkciji kompilatora. Metode koje se obično koriste u konstrukciji kompilatora se klasifikuju kao metoda naniže (top-down) i metoda naviše (bottom-up). Kao što i sama njihova imena kažu, čvorovi drveta u ovim analizama se generišu naniže (od korena ka listovima) i naviše (od listova ka korenu). U oba slučaja, ulaz u sintaksičkom analizatoru se skanira s leva nadesno sa po jednim simbolom u jedinici vremena.

Obe metode i naniže i naviše su determinističke metode. Intuitivno govoreći, deterministički znači da ne angažujemo pretraživanje, već svaki token dovodi sintaksički analizator jedan korak bliže ciljnoj konstrukciji sintaksičkog drveta. U teoriji formalnih jezika ova definicija je mnogo rigoroznija.

Determinističke metode sintaksičke analize imaju vreme izvršenja linearno zavisno od dužine unosa, tj. reda $O(n)$ gde je n dužina ulaza. Nažalost, one ne rešavaju sve probleme linearnog analizatora jer rade samo za ograničenu klasu gramatike. Gramatike ulaznih jezika imaju veoma male šanse da upravljaju determinističkom metodom, sem naravno ako konstruktor jezika napravio gramatiku koja odgovara ovom metodu.

---

[2] D. Grune, H. Bal, C. Jacobs and K. Langendoen, Modern compiler design, John Wiley & Sons, LTD, str. 110

[3] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Compilers, Principles, Tehniques and Tools, Addison-Wesley publishing company,1986, str.160

Iako je brzina sintaksičkog analizatora od velikog značaja, to nije jedini razlog što se insistira na determinizmu. Gramatika za svaki deterministički sintaksički analizator sigurno je određena kao nedvosmislena, što je veoma važna osobina za gramatiku programskog jezika. Pošto se proizvoljna gramatika često ne slaže sa gramatikom standardnih metoda sintakse, važno je imati tehnike za transformaciju gramatike kako bi bile odgovarajuće.

2. Principi metode naniže

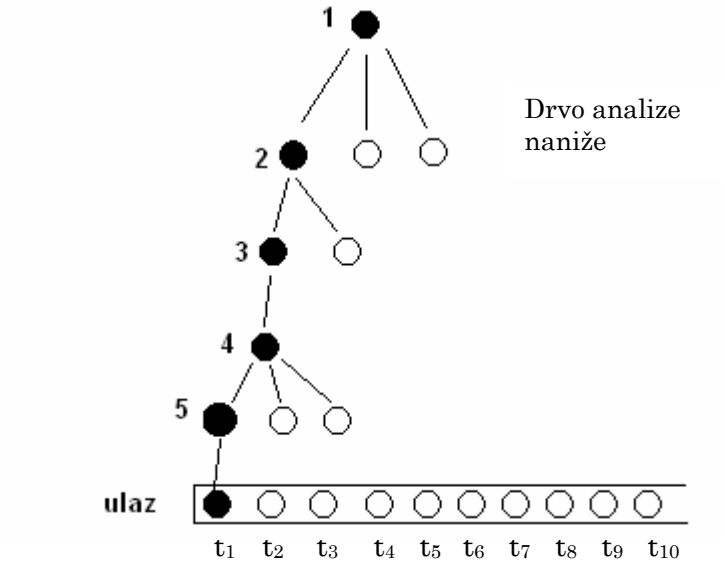
Analiza naniže počinje od početnog simbola gramatike, a tokom analize se postupno generiše izvođenje nalevo niske koja se poredi sleva nadesno sa ulaznim nizom tokena. Poželjno je da ovo izvođenje bude takvo da je u svakom koraku moguće odrediti koje će sledeće pravilo biti primenjeno u izvođenju nalevo na osnovu tekućeg preduvidnog simbola. Poželjno svojstvo gramatike je da se odluka o sledećem pravilu može doneti na osnovu tačno jednog preduvidnog simbola, kako se nebi vraćali unazad.

Gramatike bez vraćanja unazad kod kojih je jedan preduvidni simbol dovoljan za odluku o sledećem pravilu u izvođenju ulevo su $LL(1)$-gramatike. Da li je neka gramatika $LL(1)$ ne određuje se neposredno, već postoji algoritam kojim se to utvrđuje. Ali ne postoji algoritam kojim bi se za proizvoljan jezik utvrdilo da li je $LL(1)$ ili ne, tj. da li za njega postoji $LL(1)$ gramatika. Ovo znači da za proizvoljnu gramatiku nije moguće utvrditi da li je jezik koji se njome generiše $LL(1)$.[4]

[4] Duško M. Vitas, Prevodioci i interpretatori (Uvod u teoriju i metode kompilacije programskih jezika), Matematički fakultet, Beograd 2006, str. 175

Metod naniže počinje konstrukcijom korena drveta koji je ujedno i početni simbol gramatike. Zatim se konstruiše vrh poddrveta pre bilo kog njegovog nižeg čvora.

Kada analizator naniže konstruiše koren, obeležje korena, koje je ujedno i startni simbol gramatike, je već poznato npr. N, ovo je tačno za koren i tokom analize trebamo utvrditi da li je tačno i za ostale čvorove. Koristeći informacije sa ulaza, sintaksički analizator određuje ispravnu alternativu za N, tako što ima uvid u sledeći karakter – preduvidni simbol. Znajući koja se alternativa primenjuje, znamo obeležja svih čvorova dece korena, čije je obeležje N. Analizator zatim konstruiše prvi čvor dete za N, s tim što već zna njegovo obeležje. Proces određivanja ispravne alternative za najlevlje dete se ponavlja na sledećim nivoima, dok najlevlje dete nije konstruisano kao završni simbol. Završni čvor se tada "spaja" sa prvim tokenom t u programu. Ovo se ne dešava slučajno: analiza naniže bira alternative za najlevlji čvor precizno.



Slika 1: Prepoznavanje prvog tokena sa ulaza od strane analizatora naniže

Analizator onda napušta završni čvor i nastavlja sa konstrukcijom sledećeg čvora, ovo na primer može biti drugo dete roditelja prvog tokena. Na slici je sa većim punim krugom prikazan čvor koji je konstruisan kao završni, manji puni krugovi predstavljaju čvorove koji su konstruisani na putu do završnog simbola, a prazni krugovi predstavljaju čvorove čija su obeležja poznata ali koja još nisu konstruisana. O ostatku drveta analize ništa se više nezna, pa taj deo nije ni prikazan. Dakle, glavna uloga analizatora naniže je da izabere ispravne alternative za čvorove koji nisu završni, kako bi se dobio tačan završni simbol.

3. Principi metode naviše

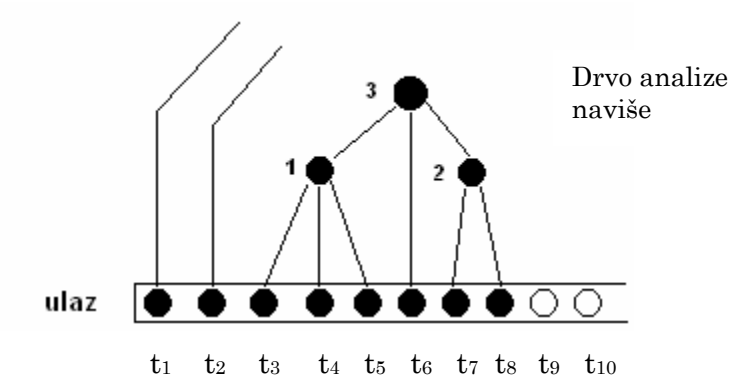
Analiza naviše predstavlja pokušaj da se za ulaznu nisku tokena konstruiše drvo sintaksičke analize polazeći od njegovih listova, postupnim svođenjem ka korenu. Postupak svođenja se sastoji u tome da se u pojedinim koracima analize, desna strana nekog pravila gramatike zameni njegovom levom stranom.

Metoda analize naviše konstruiše čvorove u drvetu sintakse tako što se vrh poddrveta konstruiše posle konstruisanja svih nižih čvorova. Kada se u analizi naviše konstruiše čvor, sva deca čvorovi su već konstruisani, i prisutni su kao poznati, a i obeležje čvora je takođe poznato. Analizator zatim kreira čvor, obeležje za njega i spaja ga sa decom.

Analiza naviše uvek konstruiše čvor koji je na vrhu prvog kompletnog poddrveta na koje se nailazi kada se vrši unos sleva nadesno. Kompletno poddrvo je drvo kod koga su sva deca već konstruisana. Tokeni su razmatrani kao podrveta visine 1 i konsruisani su kako se sa njima sretalo. Novo poddrvo mora takođe biti izabrano od nas kao poddrvo od drveta analize, ali

očigledan problem je što još uvek neznamo celo drvo analize, nego se ono tek konstruiše.

Deca prvog poddrveta se konstruišu samo kao listovi drveta, obeleženi kao završni, a ispravna alternativa je izabrana tako da se poklapa sa njima. Sledećem, drugom poddrvu u ulazu, osnovu predstavljaju sva deca čvorovi koji su već konstruisani. Deca od ovih čvorova mogu da obuhvate i čvorove koji nisu listovi, a koji su kreirani ranijom konstrukcijom čvorova. Takav čvor ima odgovarajuće obelezje koje odgovara ispravnoj alternativi. Ovaj proces se ponavlja dok se sva deca najvišeg čvora ne konstruišu, zatim se konstruiše najviši čvor i analiza je kompletna.



Slika 2: Konstruisanje prvog, drugog i trećeg čvora

Slika 2 prikazuje analizator posle njegovog konstruisanog (prepoznatog) prvog, drugog i trećeg čvora. Veći puni krug prikazuje čvor koji se zadnji konstruisan, a manji puni krugovi čvorove koji su ranije konstruisani. Prvi čvor spaja tokene t_3, t_4 i t_5, drugi spaja t_7 i t_8, a treći spaja prvi čvor, token t_6 i drugi čvor. O postojanju drugih čvorova još se ništa nezna, ali su grane za tokene t_1 i t_2 nacrtane naviše, jer znamo da one ne mogu biti deo manjeg poddtveta od ovog koje spaja tokene od t_3 do t_8. U

suprotnom bi to poddrvo bilo prvo koje je konstruisano. Glavni zadatak analizatora naviše je da ponovo pronađe koren za poddrvo čija su deca već konsruisana, i na kraju nađe koren celog drveta analize.

Gramatike koje imaju svojstvo da se konflikti u analizi naviše mogu razrešiti na osnovu informacija o dotadašnjem toku analize i uvidom u određeni broj preduvidnih simbola, čine klasu $LR(k)$-gramatika, gde k predstavlja broj preduvidnih simbola neophodnih za analizu. Značajne crte $LR(k)$-gramatika na teorijskom planu su:

- postoji algoritam koji odlučuje da li je za dato k data kontekstno slobodna gramatika $LR(k)$;
- bilo koji jezik koji je $LR(k)$, za zadato k, je i $LR(1)$.

Prednosti LR-gramatika se ogledaju i u sledećim crtama:[5]

- za najveći deo konstrukata u programskim jezicima koji se mogu opisati nekom kontekstno slobodnom gramatikom, može se konstruisati i LR- analizator;
- klasa gramatika koje se mogu analizirati metodama LR analize, je pravi nadskup klase gramatika koje se mogu analizirati metodama analize naniže; i
- LR-analizatori otkrivaju sintaksičku grešku najranije moguće tokom skaniranj ulaza sleva nadesno.

Nedostatak metode LR-analize je što je potrebna prekomerna količina manuelnog rada za konstrukciju LR-analizatora realnog programskog jezika. Ovaj nedostatak se prevazilazi tako što postoje programski alati koji omogućavaju generisanje LR-analizatora.

[5] Duško M. Vitas, Prevodioci i interpretatori (Uvod u teoriju i metode kompilacije programskih jezika), Matematički fakultet, Beograd 2006, str. 195

ZAKLJUČAK

Sintaksička analiza je faza u kojoj se ispituje i rekonstruiše struktura programa kao celine na osnovu niza tokena koje prosleđuje lesički analizator. Sintaksička analiza je centralna faza u radu kompilatora: ona upravlja radom leksičke analize i, pri tome, gradi strukturu na osnovu koje će biti moguća semantička analiza programa.

Ulaz za sintaksičku analizu predstavljaju tokeni, koje isporučuje leksički analizator. Ova analiza podrazumeva analizu konteksta u kome se token pojavio. Na izlazu iz ove faze dobija se drvolika struktura kojom je predstavljena struktura programa.

Sintaksički analizator mora biti u stanju da utvrdi na osnovu pročitanih tokena sa ulaza da li oni predstavljaju deo jezika na kome je napisan program koji se kompilira ili ne. Ukoliko token, koji se upravo pojavio na ulazu, nije u skladu sa pravilima jezika, sintaksički analizator će pozvati podsistem za rad sa greškama.

Vesna Simović, dipl. matematičar

REFERENCE:

1. Duško M. Vitas, Prevodioci i interpretatori (Uvod u teoriju i metode kompilacije programskih jezika), Matematički fakultet, Beograd 2006.
2. D. Grune, H. Bal, C. Jacobs and K. Langendoen, Modern compiler design, John Wiley & Sons, LTD.
3. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Compilers, Principles, Tehniques and Tools, Addison-Wesley publishing company,1986.
4. Allen I. Holub,Compiler design in C, Prentice Hall software series, 1990.
5. Andrew W. Appel, Modern compiler implementation in C, Cambridge University press, 1998.
6. Alfred V. Aho, Jeffrey D. Ullman, The theorz of parsing, translation, and compiling, Volume I: Parsing, Prentice-Hall, INC, 1973.
7. Alfred V. Aho, Jeffrey D. Ullman, The theory of parsing, translation, and compiling, Volume II: Compiling, Prentice-Hall, INC, 1973.