Sveučilište J. J. Strossmayera u Osijeku
Fakultet Primjenjene Matematike i Informatike
Preddiplomski studij Matematika i Računarstvo

Nikola Šarić

Općenito o BNF-u

Završni rad

Mentor: Domagoj Ševerdija

Osijek, 2023.

Sadržaj

1 Uvod

U modernom svijetu koji su u 21. stoljeću preuzele informacijske tehnologije i računarstvo, definiranje, gramatika i razvoj jezika igraju ključnu ulogu u granama industrije kao što su programiranje, analiza podataka i mnogi drugi. Jedan od alata koji se koristi za precizno definiranje gramatike je BNF (Backus-Naur Form). Što je BNF ? BNF je zapravo formalni način zapisivanja gramatike koji omogućuje precizno definiranje strukture jezika. Začetak BNF-a je u 20. stoljeću, točnije 60-ih godina 20. stoljeća kada je John Backus, poznat također po svojoj ulozi u razvoju jezika Fortran, predstavio BNF kao sredstvo za definiranje gramatika budućih programskih jezika. BNF je nakon toga proširen u oblik pod nazivom EBNF (Extended Backus-Naur Form) zbog bolje preciznosti kod definiranja gramatike. Svrha ovog istraživanja jest osigurati sveobuhvatan prikaz BNF (Backus-Naur Form) notacije. Fokus će biti usmjeren prema dubljem razumijevanju osnovne strukture same notacije i njezine svrhe. Nadalje, istraživanje će istaknuti širok spektar primjena BNF notacije u raznovrsnim domenama. Ovaj rad će čitatelju omogućiti da razumije koncept BNF-a i njegov utjecaj na računarstvo i informacijske tehnologije

2 Osnove BNF notacije

BNF (Backus-Naur Form) je notacija koja se razvila u 1950-ima zahvaljujući radu dvojice značajnih istraživača, Johna Backusa i Petera Naura. Ova notacija je prvenstveno stvorena kako bi se precizno definirala sintaksa programskih jezika i omogućila njihovu formalnu analizu. S vremenom se BNF proširila i na druge formalne jezike te komunikacijske protokole. Pritom je postala temeljni alat za razumijevanje, razvoj i analizu struktura unutar tih jezika i sustava.

2.1 Osnovni Elementi BNF-a

BNF notacija koristi niz osnovnih elemenata kako bi precizno opisala sintaksu jezika ili sustava:

2.1.1 Neterminalni simboli

Neterminalni simboli su ključni elementi BNF (Backus-Naur Form) notacije te imaju važnu ulogu u opisivanju strukture jezika ili sustava. Oni predstavljaju apstraktne kategorije ili koncepte unutar jezika i omogućavaju definiranje načina kako se ti koncepti mogu kombinirati ili zamjenjivati u ispravno oblikovane izraze ili nizove. Evo nekoliko ključnih karakteristika neterminalnih simbola:

Apstraktni konepti: Neterminalni simboli predstavljaju apstraktne pojmove unutar jezika. Umjesto da se odnose na konkretne znakove ili riječi, oni označavaju širu kategoriju koja može imati različite konkretne instance. Primjeri u programskim jezicima mogu uključivati <izraz>, <naredba>, <deklaracija> itd.

Strukturni elementi: Neterminalni simboli omogućavaju definiranje temeljnih strukturnih elemenata jezika. Oni predstavljaju način na koji se ti elementi kombiniraju kako bi se stvorili ispravni nizovi. Na primjer, u jeziku koji opisuje aritmetičke izraze, <izraz> može sadržavati kombinacije <broj>, operatora (+, -, *), zagrada i drugih elemenata.

Ponovna uporaba: Jedna od ključnih prednosti neterminalnih simbola je njihova sposobnost za ponovnu upotrebu. Definiranje apstraktnih kategorija omogućava općenitiji pristup jeziku ili sustavu te olakšava razvoj i održavanje. Isti neterminalni simbol može se koristiti u različitim dijelovima gramatike, čime se osigurava konzistentnost i jasnoća.

Hijerarhijska struktura: Neterminalni simboli često tvore hijerarhijsku strukturu. To znači da se složeniji koncepti mogu razlagati na manje dijelove i na taj način olakšavaju definiranje kompleksnih jezičnih konstrukcija. Na primjer, <izraz> se može dalje razložiti na <broj>, <operator> i druge komponente.

Definicija pravila: Neterminalni simboli se koriste u produkcijama (pravilima) koja opisuju kako se ti simboli zamjenjuju drugim simbolima. Ove produkcije čine osnovu za razumijevanje i generiranje ispravnih nizova u jeziku ili sustavu.

U konačnici, neterminalni simboli su gradivni blokovi BNF notacije i omogućavaju precizno opisivanje složenih struktura jezika ili sustava. Njihova apstraktna priroda omogućava fleksibilnost i ponovnu upotrebu te je osnova za definiranje sintakse u mnogim domenama, od programiranja i kompilacija do komunikacijskih protokola i formalnih jezika.

2.2 Terminalni simboli:

Terminalni simboli predstavljaju konkretne znakove ili simbole u jeziku. To mogu biti slova, brojevi, specijalni znakovi ili riječi. Na primjer, "a", "1", "+", itd.

2.3 Produkcije:

Produkcije su osnovna pravila koja opisuju kako se neterminalni simboli mogu razvijati u druge simbole. Svaka produkcija sastoji se od lijeve i desne strane, odvojenih znakom "→". Lijeva strana je neterminalni simbol koji se zamjenjuje, a desna strana je niz terminalnih i/ili neterminalnih simbola koji predstavljaju njegovu zamjenu.

3 Prošireni Backus-Naur oblik (EBNF)

EBNF (Extended Backus-Naur Form) je proširena verzija klasičnog BNF (Backus-Naur Form) formalizma za opisivanje gramatika programskih jezika i drugih formalnih jezika. EBNF dodaje nekoliko dodatnih konstrukcija i notacija kako bi olakšala i poboljšala opisivanje složenijih sintaktičkih struktura, više o tome možete proučiti na [6]

3.1 Razlike između EBNF-a i Osnovnog BNF-a:

3.1.1 Opcionalni Elementi

EBNF omogućava označavanje elemenata kao opcionalnih pomoću uglatih zagrada []. To znači da se taj element može pojaviti nula ili jednom unutar nizova. U osnovnom BNF-u, opcionalni elementi se obično ostvaruju pomoću produkcija.

3.1.2 Ponavljanje

EBNF omogućava označavanje elemenata kao ponavljajućih pomoću zagrada . Ovo omogućava da se element ponavlja proizvoljan broj puta. U osnovnom BNF-u, ponavljanje se obično ostvaruje koristeći produkcije ili nizove simbola.

3.1.3 Skupovi

EBNF omogućava definiranje skupova znakova pomoću uglatih zagrada [ ] i crtice -. Na primjer, [a-z] označava skup malih slova. U osnovnom BNF-u, morate koristiti individualne terminalne simbole za svaki znak.

3.1.4 Grupiranje

EBNF omogućava grupiranje elemenata pomoću običnih zagrada (). Ovo omogućava precizno definiranje prioriteta i redoslijeda izvršavanja. U osnovnom BNF-u, redoslijed se obično implicitno definira pozicijom produkcija.

3.1.5 Posebni Simboli

EBNF dopušta korištenje posebnih simbola kao što su ?, *, + za označavanje opcionalnosti i ponavljanja. Na primjer, a? označava da je simbol a opcionalan. Ovo čini EBNF izraze čitljivijima.

3.1.6 Referenca na simbole

U EBNF-u je često moguće referirati se na neterminalne simbole koristeći njihova imena. Ovo pomaže u održavanju i razumijevanju gramatika.

U konačnici, EBNF je nadogradnja osnovnog BNF-a koja pruža jasnije i čitljivije načine za opisivanje složenih gramatika i sintaktičkih struktura. Ona olakšava rad s gramatikama programskih jezika i drugih formalnih jezika, jer omogućava preciznije definiranje opcionalnosti, ponavljanja i prioritetnih redoslijeda.

4 Primjena BNF-a

BNF (Backus-Naur Form) notacija ima širok spektar primjena u različitim područjima. Evo nekoliko primjera kako se BNF koristi u različitim domenama:

4.1 Programiranje i Kompilatori

BNF je prvi put razvijen za opisivanje sintakse programskih jezika. Koristi se za precizno definiranje gramatike jezika kako bi programeri i kompilatori znali kako interpretirati i analizirati izvorni kod. Programski jezici kao što su C, Java, Python itd., imaju svoje BNF gramatike koje omogućavaju precizno definiranje ispravne sintakse jezika.

4.2 Označni jezik

BNF se također primjenjuje u definiranju jezika za označavanje poput HTML, XML, CSS i drugih. Ove gramatike omogućavaju jasno definiranje struktura dokumenata i načina kako se elementi mogu kombinirati i uređivati. Na primjer, HTML BNF gramatika opisuje kako elementi poput tagova i atributa mogu biti korisnički definirani i organizirani.

4.3 Matematika i formalni jezici

U matematici se BNF koristi za definiranje formalnih jezika kao što su Peano aksiomi ili formalni sustavi kao što je tipičan zapis za definiciju matematičke strukture. BNF se koristi za precizno opisivanje odnosa, operacija i struktura unutar matematičkih sustava.

4.4 Komunikacijski protokoli

BNF se primjenjuje u definiranju komunikacijskih protokola, gdje je bitno precizno definirati kako se podaci šalju i primaju između različitih uređaja ili sustava. To osigurava dosljednost u razmjeni podataka između različitih implementacija.

4.5 Prirodni jezik i gramatika

U lingvistici, BNF se koristi za opisivanje gramatike prirodnih jezika. Ovo omogućava analizu struktura i gramatičkih pravila koji stoje iza rečenica u jeziku.

4.6 Baze podataka

BNF se koristi za definiranje sintakse upita u bazi podataka. SQL jezik za upravljanje bazama podataka koristi BNF gramatiku za precizno opisivanje kako se upiti definiraju i izvršavaju nad bazama podataka.

4.7 Umjetna inteligencija

U području umjetne inteligencije, BNF se može koristiti za definiranje formalnih jezika koji opisuju pravila, zaključke ili operacije u računalnoj logici i programima za donošenje odluka.

BNF notacija pruža snažan okvir za precizno definiranje sintakse u različitim domenama. Njena primjena pomaže osigurati dosljednost, olakšava razumijevanje i analizu te omogućava razvoj alata i programa koji mogu obraditi i interpretirati strukturu i gramatiku jezika ili sustava.

5 Korištenje BNF-a u razvoju programskih jezika

5.1 C Prgramming language

U ovom primjeru koristi se BNF (Backus-Naur Form) notacija za opisivanje gramatike programskog jezika C. Ova gramatika detaljno opisuje strukturu C programa, uključujući deklaracije, tipove podataka, identifikatore i druge elemente jezika.

```
1
2
3   <program> ::= <declaration-list>
4
5   <declaration-list> ::= <declaration>
6                        | <declaration-list> <declaration>
7
8   <declaration> ::= <type-specifier> <identifier> ;
9
10  <type-specifier> ::= int | float | char
11
12  <identifier> ::= <letter>
13                 | <identifier> <letter>
14                 | <identifier> <digit>
15
16  <letter> ::= a | b | ... | z | A | B | ... | Z
17
18  <digit> ::= 0 | 1 | ... | 9
19
```

Slika 1: BNF in C programming language

- `<program>`: Ovaj neterminalni simbol označava čitav C program. Sastoji se od `<declaration-list>`.
- `<declaration-list>`: Ovaj simbol označava listu deklaracija unutar programa, odnosno može biti jedna ili više tipova `<declaration>`.
- `<declaration>`: Ovaj simbol opisuje pojedinačnu deklaraciju. Sastoji se od `<type-specifier>` (koji označava tip podatka), `<identifier>` (koji označava identifikator) i završava se znakom ;.
- `<type-specifier>`: Simbol koji definira tip podataka, kao što su int, float ili char.

- `<identifier>`: Simbol koji označava identifikator, što je ime koje programer dodjeljuje varijablama ili funkcijama. Identifikatori se sastoje od slova ili kombinacije slova i brojeva.
  - `<letter>`: Ovi simboli označavaju abecedu slova (a, b,...,z, A, B, ..., Z)
  - `<digit>`: Ovi simboli označavaju brojeve (0, 1, ..., 9).

Svaka od ovih definicija u BNF notaciji ima svoje pravilo zamjene i moguće kombinacije. Na primjer, <declaration> se može zamijeniti s <type-specifier> koji se slijedi identifikatorom i završava znakom ;. Sve zajedno, ovaj primjer BNF-a za jezik C omogućava precizno opisivanje kako se pravilno definiraju deklaracije u jeziku C, s obzirom na njegove tipove podataka i identifikatore.

5.2 Python

U ovom primjeru koristi se BNF (Backus-Naur Form) notacija za opisivanje gramatike programskog jezika Python. Ova gramatika detaljno opisuje strukturu Python programa, uključujući izjave dodjele, if naredbe, petlje i izraze.

```
1
2
3   <program> ::= <statement-list>
4
5   <statement-list> ::= <statement>
6                      | <statement-list> <statement>
7
8   <statement> ::= <assignment> | <if-statement> | <loop-statement>
9
10  <assignment> ::= <identifier> = <expression>
11
12  <if-statement> ::= if <expression> : <suite>
13                   | if <expression> : <suite> else : <suite>
14
15  <loop-statement> ::= while <expression> : <suite>
16
17  <expression> ::= <term>
18                 | <expression> + <term>
19                 | <expression> - <term>
20
21  <term> ::= <factor>
22           | <term> * <factor>
23           | <term> / <factor>
24
25  <factor> ::= <number> | <identifier>
26
```

Slika 2: BNF in Python

- `<program>`: Ovaj neterminalni simbol označava cijeli Python program. Sastoji se od `<statement-list>`.
- `<statement-list>`: Ovaj simbol predstavlja listu izjava unutar programa, odnosno To može biti jedna ili više `<statement>`.
- `<statement>`: Ovaj simbol označava pojedinačnu izjavu. To može biti izjava dodjele, if naredba ili petlja.
- `<assignment>`: Ovaj simbol opisuje izjavu dodjele vrijednosti varijabli. Sastoji se od `<identifier>` (identifikator varijable) nakon čega slijedi znak "=", a potom i `<expression>` (izraz koji se dodjeljuje varijabli).

- `<if-statement>`: Ovaj simbol označava if naredbu. Može biti samo if ili if-else konstrukcija. Sastoji se od ključne riječi if, zatim uslovnog izraza `<expression>`, dvotočke ":", te bloka koda `<suite>`. Opcionalno, može se dodati blok koda else sa ključnom riječju else i još jednim blokom koda `<suite>`.

- `<loop-statement>`: Ovaj simbol označava petlju while. Sastoji se od ključne riječi while, uslovnog izraza `<expression>`, dvotočke ":", te bloka koda `<suite>`.

- `<expression>, <term>, <factor>`: Ovi simboli označavaju hijerarhijsku strukturu izraza, često korištenu u matematičkim operacijama. Na primjer, `<expression>` može sadržavati `<term>`, kombinacije sa znakovima "+" ili "-", itd.

- `<number>, <identifier>`: Ovi simboli označavaju imena varijabli

Kroz ovu definiciju BNF notacije, precizno se opisuje kako se pravilno konstruiraju Python programi koji koriste osnovne konstrukcije jezika, uključujući dodjelu vrijednosti, grananje (if), petlje (while) i osnovne izraze.

5.3 Java

U ovom primjeru koristi se BNF notacija za opisivanje gramatike programskog jezika Java. Ta gramatika opisuje strukturu Java programa, uključujući deklaracije klasa, metoda, polja i drugih elemenata jezika.

```
<program> ::= <class-declaration>

<class-declaration> ::= class <identifier> { <member-declarations> }

<member-declarations> ::= <member-declaration>
                        | <member-declarations> <member-declaration>

<member-declaration> ::= <field-declaration> | <method-declaration>

<field-declaration> ::= <type> <identifier> ;

<method-declaration> ::= <type> <identifier> ( ) { <statements> }

<type> ::= int | float | boolean | ...

<identifier> ::= <letter>
               | <identifier> <letter>
               | <identifier> <digit>

<letter> ::= a | b | ... | z | A | B | ... | Z

<digit> ::= 0 | 1 | ... | 9
```

Slika 3: BNF in Java

- `<program>`: Ovaj neterminalni simbol označava cijeli Java program. Sastoji se od `<class-declaration>`.
- `<class-declaration>`: Ovaj simbol opisuje deklaraciju klase u Java programu. Definira se ključnom riječju "class", zatim slijedi `<identifier>` (naziv klase), zatim otvorena vitičasta zagrada "", pa <member-declarations> (članovi klase), te na kraju zatvorena vitičasta zagrada "".
- `<member-declarations>`: Ovaj simbol predstavlja listu članova klase. Može sadržavati jedan ili više `<member-declaration>`.

- `<member-declaration>`: Ovaj simbol opisuje pojedinačni član klase. To može biti ili `<field-declaration>` (deklaracija polja) ili `<method-declaration>` (deklaracija metode).
- `<field-declaration>`: Ovaj simbol definira deklaraciju polja. Sastoji se od `<type>` (tip polja), `<identifier>` (naziv polja) i završava se znakom ";".
- `<method-declaration>`: Ovaj simbol opisuje deklaraciju metode. Sastoji se od `<type>` (tip povratne vrijednosti metode), `<identifier>` (naziv metode), zatim otvorene zagrade "()", otvorena vitičasta zagrada "", `<statements>` (naredbe metode) i zatvorena vitičasta zagrada "".
- `<type>`: Ovaj simbol definira tip podataka, poput "int", "float", "boolean" i drugih.
- `<identifier>`: Simbol koji označava identifikator, tj. naziv klase, metode, polja ili varijable. Identifikatori se sastoje od slova i brojeva.

Kroz ovaj primjer, koristeći BNF notaciju, precizno se opisuje kako se ispravno definiraju klase, metode, polja i ostali elementi u Java programima. Ova definicija omogućava precizno tumačenje i generiranje Java koda u skladu s gramatikom jezika.
Više o BNF notaciji i njenim primjerima možete pronaći u [1] i [2]

6 BNF u kompilatorskim alatima

BNF (Backus-Naur Form) i njegove varijacije, kao što je EBNF (Extended Backus-Naur Form), igraju ključnu ulogu u izradi kompilatora i analizatora. Evo kako se BNF koristi u ovim kontekstima:

6.1 Definiranje sintakse programskog jezika

BNF se koristi za precizno definiranje sintakse programskog jezika kroz slijedeći proces

6.1.1 Identifikacija Sintatičkih Konstrukcija

Na početku definiranja sintakse programskih jezika je ključan proces identificiranja sintaktičkih konstrukcija koje specifični jezik podržava, a to su ništa drugo nego operatori, naredbe, deklaracije...

6.1.2 Definiranje neterminalnih simbola

nakon toga slijedi proces stvaranja neterminalnih simbola, a to su, kao što smo prije opisali, apstraktni koncepti koji služe za opisivanje strukture, u ovom slučaju, <operator>, <naredba>, <deklaracija>

6.1.3 Definiranje terminalnih simbola

Nakon toga prirodno slijedi definiranje terminalnih simbola, odnosno konkretni znakovi ili riječi u jeziku, na primjer, "int", "+", "="

6.1.4 Definiranje produkcija

Ključna komponenta su produkcije, odnosno objašnjenje kako se to neterminalni simboli zamjenjuju sa terminalnim simbolima ili nekim drugim neterminalnim simbolima, kao na primjer, <neterminalni simbol> ::= <izraz>, u ovom primjeru se neterminalni simbol <neterminalni simbol> zamjenjuje s <izraz>

6.1.5 Razrada gramatike

Kada se definiranje produkcija proširi na svaki neterminalni simbol dobijemo gramatiku jezika. Ključna stvar kod pravila te gramatike je da moraju biti precizna, jasna i uzajamno iskljuliva kako bi se izbjegle dvosmislice.

6.1.6 Specifičnosti sintakse

BNF omogućuje opisivanje specifičnosti sintakse kao što su ascoijativnost i prioriteti operatora. Na primjer, izraz A*B+C se može opisati koristeći BNF i definiciju o prioritetu operatora

6.1.7 Testiranje sintakse

Nakon ovih 6 koraka smo uspješno definirali gramtiku pomoću BNF-a, Slijdeći korak bi bio testiranje, a to izveli na način da definiramo parser kojem bi bio zadatak da provjerava da je kod u skladu sa definiranom sintaksom, odnosno, parser radi ništa drugo nego čita niz tokena (simvoli, riječi...) iz koda i uspoređuje sa produkcijama da provjeri ispravnost sintakse

6.1.8 Održavanje

Kako se jezik kontantno proširuje i mijenja, tako se i BNF pravila moraju konstantno mijenjati u skladu s promjenom u jeziku.

6.1.9 Otkrivanje grešaka

BNF također pomaže u detekciji grešaka u izvornom kodu. Parseri mogu prepoznati sintaktičke greške koje nisu u skladu s definiranom gramatikom.

U literaturi pod [5] možete pronaći više o formalnom i laboratorijskom pristupu sintaksi BNF-a

6.2 Parseri

Kompilatori razdvajaju kod na manje dijelove koristeći parsere, odnosno vrše leksičku i sintatičku analizu. U neku ruku, Parseri pomoću BNF-a razumiju strukturu izvornog koda. Ukratko, Parseri provjeravaju jesu li nizovi riječi i simbola (tokeni) iz izvornog koda i provjeravaju jesu li u skladu sa gramatikom. U nastavku se detaljnije opisuje kako to parseri koriste gramatiku napisanu pomoću BNF-a

6.2.1 Leksička analiza

Prije nego što parser počne s radom, lexer/skener, koji je također poznat i pod nazivom leksički analizator prolazi kroz izvorni kod i razdvaja ga na komponente, koje nazivamo leksičke jedinice ili tokeni. U to ulati prepoznavanje varijabli, brojeva, operatora, ključnih riječi i drugih elemenata. Lexerov zadatak je da stvori niz tih tokena koji kasnije proslijeđuje parseru.

6.2.2 Sintaktička analiza

Nakon toga slijedi proces sintaktičke analize. U ovom koraku parser prolazi kroz niz tokena proslijeđen nakon leksičke analize i provjerava sintaksu koristeći BNF gramatiku, BNF gramatika, kao što znamo, definira pravila za obrasce u izvornom kodu, U koraku sintatičke analize, upoznajemo se sa dvije ključne komponente parsera, stack i spremnik ulaza:

Stack se koristi za pračenje trenutnog stanja analize.

Spremnik ulaza sadrži preostale tokene koje parser treba obraditi

6.2.3 Analiza odozgo prema dolje

Parseri također, umjesto Sintatičke analize, mogu koristiti odozgo prema gore strategiju analize. U ovoj strategiji parser kreće sa nekim početnim simbolom (Najčešće <program> koji obuhvaća cijeli kod kao što smo vidjeli u primjerima figure 1, figure 2, figure 3) i pokušava pronaći put prema nizovima tokena

6.2.4 Analiza odozdo prema gore

Slično kao u analizi odozgo prema dolje, parseri mogu koristiti analizu odozdo prema gore. U ovoj strategiji parser radi suprotno nego u odozgo prema dolje analizi. Kreće od tokena i pokušava izgraditi AST (Apstraktno stablo sintakse) koristeći produkcije iz BNF-a

6.2.5 Parsiranje po produkcijama

Parser koristi produkcije BNF gramatike kako bi razvojio tokene u sintaktički ispravne konstrukcije. Odnsno, uzmimo za primjer pravilo <izraz> ::= <izraz> * <term>, ovo će parser prepoznati i razdvojiti niz tokena u izraz, operator i term

6.2.6 Generiranje AST-a

slijedeći korak, nakon uspješnog parsiranja je da se generira AST (Apstraktno stablo sintakse) pomoću parsera. Apstraktno stablo sintakse je hijerarhijska struktura koja apstratnko prikazuje strukturu izvornog koda. Sintaktičke konstrukcije su prikazane kroz čvorove stabla, a odnosi između njih su prikazani granama. Najveća korist AST-a je za daljnju analizu i generiranje ciljnog koda

6.3 Semantička analiza

Nakon sintaktičke analize, slijedi semantička analiza, druga ključna faza u procesu kompilacije. Semantička analiza se provodi kako bi se provjerilo ima li izvorni kod smisla s obzirom na programski jezik koji treba opisivati. U nastavku je detaljnije opisan proces semantičke analize:

6.3.1 Priprema podataka

Prije početka, ključan je proces pripreme podataka, odnosno izradnja AST-a (Apstratno stablo sintakse) koje se napravi tijekom sintaktičke analize. Podsjetimo, apstraktno stablo sintakse je struktura koja prikazuje strukturu izvrnog koda na apstraktan način.

6.3.2 Pregledavanje apstraktnog stabla sintakse

U ovom koraku semantički analizator provjerava varijable, izraze i razne druge konstrukcije. Odnosno, pokušava razumijeti značenje svakog dijela izvornog koda

6.3.3 Provjera tipova

Ključna zadaća, ili barem jedna od ključnih zadaća semantičkog analizatora je provjera tipova podataka. Semantička analiza provjerava jesu li operacije i izrazi kompatibilni odgovarajućim tipovima podataka. Na primjer, ne može se zbrajati broj(int) i niz znakova (char)

6.3.4 Praćenje varijabli

Isto tako jedna od bitnijih uloga semantičkog analizatora je praćenje deklaracije i upotreba deklariranih varijabli. Semantička analiza provjerava ispravnost deklariranih varijabli i koriste li se u skladu sa deklaracijom.

6.3.5 Greške

Ako prilikom procesa semantičke analize dođe do neispravnosti u izvornom kodu, analizator generira odgovarajuću grešku, Na primjer, neke od češćih grešaka uključuju nedefiniranje varijabli i pokušaj zbrjanja stringa i broja

6.3.6 Uklanjanje bespotrebnog koda

jedna od zanimljivijih dodatnih mogućnosti semantičkog analizatora je uklanjanje bespotrebnog koda, kao što su, na primjer, nepotrebne deklaracije i nedostupne varijable

6.4 Optimizacija koda

Nakon stvaranja apstratnog stabla sintakse, kompilatori mogu optimizirati kod kako bi ga učinili boljim i efikasnijim. To čine pomoću BNF pravila koja im omogućavaju da prepoznaju obrasce u kodu i optimiziraju ga na temelju tih obrazaca. Bitno je naglasiti da se optimizacija primjenjuje na generiranom međukodu ili na izvornom kodu, a ne na BNF notaciji.

7 Prednosti i izazovi korištenja BNF-a

Korištenje BNF (Backus-Naur Form) za opisivanje gramatike programskih jezika i struktura podataka ima svoje prednosti i ograničenja, a postoje i alternative. Evo razgovora o tome:

7.1 Prednosti Korištenja BNF-a

Notacija BNF-a se koristi u raznim granama informatike i puno različitih koncepta, ovo su samo neki od primjera

7.1.1 Preciznost, jasnost i razumljivost

Jedna od glavnih prednosti korištenja BNF-a je njegova preciznost u definiranju sintakse. BNF notacija omogućava programerima, inženjerima i istraživačima da precizno opišu kako se ispravno napisati izvorni kod ili struktura podataka. To rezultira jasnim i dosljednim specifikacijama jezika ili formata, čime se smanjuje prostor za nesporazume i pogreške. Uz to, BNF pruža jasnost i razumljivost. Njegova jednostavna struktura čini ga prikladnim za programere i istraživače koji žele razumjeti i interpretirati jezične konstrukcije. BNF notacija često olakšava učenje novih jezika ili formata.

7.1.2 Standardizacija

Druga značajna prednost BNF-a je standardizacija. BNF se često koristi za definiranje programskih jezika, jezika za označavanje (kao XML ili JSON), i drugih jezičnih konstrukcija. Kao takav, BNF pomaže uspostaviti formalne standarde za komunikaciju između različitih alata, sustava i programera. Ovo doprinosi interoperabilnosti i razmjeni podataka.

7.1.3 Kompilacija i parsiranje

Korištenje BNF-a također olakšava kompilaciju i parsiranje. Parseri generirani iz BNF gramatike pomažu u razdvajanju i analizi izvornog koda, čime se ubrzava proces razvoja kompilatora i interpretera. Osim toga, BNF omogućava jasno razumijevanje sintakse jezika, što je od suštinskog značaja za implementaciju analizatora i generatora koda. Više o tehnikama parsiranja u kontekstu BNF-a i drugih formalnih jezika možete pronaći u [3], također više o teoriji kompliatora možete pronaći u [4]

7.1.4 Analiza i validacija

Kada su u pitanju analiza i validacija, BNF se može koristiti za provođenje preciznih provjera ispravnosti izvornog koda ili struktura podataka. Na primjer, BNF se često koristi u XML shemama kako bi se definirala dopuštena struktura XML dokumenata.

7.1.5 Primjenjivost

Konačno, BNF je primjenjiv u različitim domenama. Osim u programiranju, BNF se koristi u jezicima za označavanje, matematici, analizi podataka, modeliranju i mnogim drugim područjima informatičkog inženjeringa i znanosti.

7.2 Ograničenja korištenja BNF-a

7.2.1 Ne obrađuje semantiku

BNF se fokusira isključivo na sintaksu jezika i ne bavi se semantikom. To znači da, iako može precizno definirati kako se ispravno napisati izvorni kod, ne može opisati što taj kod radi ili što znače specifične konstrukcije. Semantička analiza je potrebna za razumijevanje značenja koda.

7.2.2 Kompleksnost i održavanje

Za složenije jezike, BNF gramatika može postati vrlo opsežna i teško održiva. Održavanje preciznosti i dosljednosti u velikim gramatikama može biti izazovno.

7.2.3 Ne dopušta minimalne nepreciznosti

BNF zahtijeva potpunu preciznost u definiranju jezičnih pravila. To može biti ograničenje u slučajevima kada je jezik namjerno dizajniran da bude fleksibilan ili tolerantan prema greškama.

7.2.4 Ograničenost

BNF gramatike su ograničene na kontekst-slobodne gramatike (CFG). To znači da ne mogu precizno modelirati neka složenija jezična pravila koja zahtijevaju kontekstno osjetljive gramatike. Ovo je ograničenje u definiranju nekih naprednijih jezika i struktura podataka.

7.3 Alternative BNF-u

BNF je moćna i često korištena notacija za definiranje gramatike i sintakse jezika, ali kao i za sve ostalo u životu, postoje alternative. U nastavku su navedene neke od njih

7.3.1 PEG (Parsing Expression Grammar)

PEG je formalizam za definiranje sintakse koji se temelji na notaciji sličnoj BNF-u, ali ima različita pravila za tumačenje. PEG gramatike su jednostavne i izravne, a PEG parseri su lako razumljivi i efikasni. PEG također može rješavati neka ograničenja BNF-a, poput ambiguiteta.

7.3.2 ANTLR (Another Tool for Language Recognition)

ANTLR je alat za generiranje parsera koji omogućava opisivanje gramatike u obliku pravila i automatski generira parser.

7.3.3 Regex (Regular Expressions)

Regularne izraze često koristimo za definiranje i pretragu sintakse u tekstualnim nizovima. Iako nisu prikladni za definiranje složenih gramatika, regularni izrazi su iznimno korisni za jednostavne obrasce i pretragu teksta.

7.3.4 EBNF (Extended Backus-Naur Form)

EBNF je proširenje BNF-a koje uključuje neke dodatne konstrukcije koje olakšavaju definiranje gramatika. To uključuje opcionalne elemente, ponavljanje i grupiranje konstrukcija. EBNF je često korisniji od klasičnog BNF-a za precizno definiranje složenih gramatika.

7.3.5 YAML i JSON Schema

Ove notacije se često koriste za definiranje struktura podataka u obliku teksta. Iako nisu formalne gramatike, YAML i JSON su jednostavni za čitanje i pisanje te se koriste za razmjenu podataka između različitih aplikacija.

8 Primjeri BNF notacija

8.1 Primjer BNF-a za aritmetički izraz

```
<expression> ::= <term> "+" <expression>
               | <term> "-" <expression>
               | <term>
<term>       ::= <factor> "*" <term>
               | <factor> "/" <term>
               | <factor>
<factor>     ::= "(" <expression> ")"
               | <number>
<number>     ::= <digit>+
```

Slika 4: Aritmetički izraz

Primjer gdje <expression> predstavlja aritmetički izraz koji može zadržavati oduzimanje ili zbrajanje, <term> predstavlja član izraza koji može sadržavati množenje ili dijeljenje. <factor> predstavlja faktor izraza, koji može biti broj ili grupirani izraz unutar zagrada. Dok <number> predstavlja cjelobrojni broj.

8.2 Primjer BNF-a za deklaraciju funkcije u C-u

```
<function-declaration> ::= <type> <identifier> "(" <parameter-list> ")" "{" <statements> "}"
<type>                 ::= "int" | "float" | "char"
<parameter-list>       ::= <parameter>
                         | <parameter> "," <parameter-list>
<parameter>            ::= <type> <identifier>
```

Slika 5: Deklaracija funkcije u C-u

Primjer gdje <function-declaration> predstavlja deklaraciju funkcije u jeziku C, a sastoji se od tipa, identifiera (imena funkcije), otvorene i zatvorene zagrade koje označavaju početak i kraj parametara funkcije. <Type> predstavlja tip podataka koji funkcija vraća, može biti INT, FLOAT ili CHAR. <parameter> predstavlja pojedini parametar funkcije, a sastoji se od tipa parametra i identifikatora koji predstavlja ime parametra, dok <parameter-list> predstavlja listu tih parametara

8.3 Primjer BNF-a za osnovnu XML oznaku

```
1
2
3   <element> ::= "<" <tag-name> ">" <content> "</" <tag-name> ">"
4   <tag-name> ::= <letters>
5   <content> ::= <element> | <text>
6   <text> ::= <characters>
7   <letters> ::= <letter> <letters>
8   <characters> ::= <character> <characters>
9
```

Slika 6: XML oznaka

Primjer gdje <element> predstavlja osnovnu XML oznaku i sastoji se od otvarajuće oznake (<tag-name>) i zatvarajuće oznake (</tag-name>), između kojih se nalazi content(saržaj). <tag-name> predstavlja ime XML elementa i sastoji se od slova, brojeva, donjih crta, i crtica. Ime elementa mora započeti slovom ili donjom crtom. <content> predstavlja sadržaj unutar XML oznake. To može biti ili tekst (<tekst>) ili druga ugniježđena XML oznaka (<element>).

8.4 Primjer BNF-a za JSON objekt

```
<object> ::= "{" <members> "}"
<members> ::= <pair>
            | <pair> "," <members>
<pair> ::= <string> ":" <value>
<string> ::= "\"" <characters> "\""
<value> ::= <string> | <number> | <object> | <array> | "true" | "false" | "null"
```

Slika 7: JSON objekt

Primjer gdje <object> predstavlja JSON objekt i sastoji se od otvarajuće vitičaste zagrade (), <members> i zatvarajuće vitičaste zagrade (). <members> predstavlja parove ključ-vrijednost unutar JSON objekta. Svaki <pair> sastoji se od ključa (stringa), znaka :, i vrijednosti (koja može biti bilo koja JSON vrijednost).

8.5 Primjer BNF-a za SELECT upit u SQL-u

```
<select-query> ::= "SELECT" <columns> "FROM" <table> "WHERE" <condition>
<columns> ::= "*" | <column> "," <columns>
<column> ::= <identifier>
<table> ::= <identifier>
<condition> ::= <expression>
<expression> ::= <column> <operator> <value>
<operator> ::= "=" | "<>" | "<" | ">" | "<=" | ">="
```

Slika 8: SQL SELECT upit

Primjer gdje <select-query> predstavlja osnovnu strukturu SELECT upita. Sastoji se od ključne riječi "SELECT", popisa stupaca za dohvaćanje (<columns>), ključne riječi "FROM" i imena tablice <table> iz koje se dohvaćaju podaci. Također može uključivati uvjete <condidition> nakon ključne riječi WHERE

9 Zaključak

BNF (Backus-Naur Form) notacija je ključna komponenta za definiranje sintakse programskih jezika, formata podataka i drugih struktura u različitim domenama računarstva. Evo sažetka glavnih točaka i važnosti BNF-a.
BNF notacija je formalni jezik koji omogućava precizno definiranje gramatike jezika ili struktura podataka.Sintaksa se opisuje koristeći pravila zamjene koja definiraju kako se jedan izraz može zamijeniti drugim, čime se definira dopuštena struktura jezika.
Neterminalni i terminalni simboli su osnovni elementi BNF-a. Neterminalni simboli predstavljaju apstraktne koncepte, dok terminalni simboli predstavljaju konkretne znakove.
Produkcije su pravila koja definiraju kako se neterminalni simboli zamjenjuju terminalnim ili drugim neterminalnim simbolima.
EBNF (Extended Backus-Naur Form) je proširena verzija BNF-a koja omogućava dodatne značajke poput opcionalnih elemenata, ponavljanja i grupiranja.
BNF igra ključnu ulogu u razvoju kompilatora i analizatora. Precizno definira sintaksu programskih jezika, omogućava generiranje parsera i olakšava analizu izvornog koda.
Standardizacija: BNF se često koristi za standardizaciju sintakse programskih jezika i formata podataka. To olakšava razmjenu podataka i interoperabilnost.
Alternativne notacije: Iako je BNF česta notacija za opisivanje gramatike, postoje i alternative poput PEG, ANTLR, regularnih izraza i drugih, ovisno o specifičnim potrebama i domenama.
Održavanje i promjene: BNF pravila treba pažljivo održavati, posebno ako se gramatika jezika mijenja. Promjene u gramatici zahtijevaju ažuriranje pravila.
U konačnici, BNF notacija je temelj za razumijevanje i opisivanje struktura u računarstvu. Bez precizno definirane sintakse, razvoj programskih jezika, kompilatora, parsera i analizatora bio bi znatno otežan. Stoga, razumijevanje i primjena BNF-a igraju ključnu ulogu u razvoju i analizi različitih računalnih sustava i tehnologija.

Literatura

[1] Compilers: Principles, Techniques, and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman

[2] Formal Languages and Their Machines by Keith D. Jones and Jonathan S. Ostwald

[3] Parsing Techniques: A Practical Guide by Dick Grune and Ceriel J.H. Jacobs

[4] The Art of Compiler Design: Theory and Practice by Thomas Pittman and James Peters

[5] Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach by Kenneth Slonneger and Barry L. Kurtz

[6] (`https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form`)

Popis slika