Prikaz rada jednostavnog jezičnog procesora:

Prevođenje izvornog programa u ciljni program ostvaruje se u dvije osnovne faze rada jezičnog procesora: analiza izvornog programa i sinteza ciljnog programa. Faza analize prethodi fazi sinteze kako bi se ustanovilo ima li u izvornom programu pogrešaka. Nema li pogrešaka u izvornom programu izvodi se sinteza ciljnog programa. Faza analize rastavlja izvorni program u sastavne dijelove, provjerava pravila jezika, prijavljuje pogreške i zapisuje izvorni program primjenom različitih struktura podataka u memoriju računala. Analiza izvornog programa izvodi se u više zasebnih koraka: leksička analiza, sintaksna analiza i semantička analiza. U fazi sinteze zapis izvornog programa, koji je generiran u fazi analize, postupno se prevodi u ciljni program- Sinteza ciljnog programa izvodi se u sljedećim koracima: generiranje međukoda, strojno nezavisno optimiranje, generiranje strojnog programa, strojno zavisno optimiranje i priprema strojnog programa za izvođenje.

Leksička analiza:

Pravila programskih jezika određuju dozvoljene oblike osnovnih elemenata jezika. Osnovni elementi programskih jezika su varijable, ključne riječi, konstante, operatori i specijalni znakovi. Pravila koja određuju dozvoljene oblike osnovnih elemenata jezika nazivaju se leksička pravila. Elementi jezika kao što su varijable, ključne riječi i konstante nazivaju se leksičke jedinke. Leksičke jedinke grupiraju se u više različitih klasa. Leksička analiza je linearna analiza znakova izvornog programa koja provjerava jesu li leksičke jedinke u izvornom programu pravilno napisane. Znakovi izvornog programa slijedno se čitaju jedan po jedan s ciljem grupiranja ulaznih znakova u leksičke jedinke, prepoznavanja i određivanja klasa leksičkih jedinki, ispitivanja jesu li zadovoljena pravila koja određuju dozvoljene oblike leksičkih jedinki, izbacivanja bjelina i komentara, te ispisa pograšaka. Leksički analizator prevodi izvorni program u niz leksičkih jedinki. Dobiveni niz leksičkih jedinki je ulazni niz sintaksnog analizatora. Tijekom leksičke analize gradi se tablica znakova u koju se zapisuju dodatni parametri leksičkih jedinki.

Sintaksna analiza:

Programskim jezikom određena su pravila na koji način se koriste jedinke za gradnju ispravne naredbe. Pravila koja određuju dozvoljene strukture naredbi, dozvoljene strukture dijelova programa i dozvoljene strukture čitavog programa nazivaju se sintaksna pravila, dok se dio jezičnog procesora koji provjerava ta pravila naziva sintaksni analizator. Sintaksna analiz aje hijerarhijska analiza. Leksičke jedinke kojie generira leksički analizator grupiraju se u hijerarhijske skupine sa zajedničkim značenjem. Sintaksni analizator provjerava da li niz leksičkih jedinki zadovoljava sintaksnim pravilima zadanu hijerarhijsku strukturu izvornog programa. Najčešće se hijerarhijska struktura programa zadaje primjenom formalne gramatike i opisuje stablom. Sintksni analizator gradi sintaksno stablo ili ispisuje poruku o pogrešci. Sintaksno stablo sprema se na više različitih načina u memoriju računala: izravno spremanje čitavog stabla primjenom za to pogodnih podatkovnih objekata kao što su mreže, izravno dinamičko spremanje dijelova stabla na stog i posredno dinamičko spremanje dijelova stabla na stog putem poziva rekurzivnih potprograma. Za potrebe sintaksne analize uzima se LIFO stog. Sintaksni analizator stalno uspoređuje sadržaj vrha stoga i pročitanu leksičku jedinku izvornog programa, te na temelju tih podataka primjenjuje određene akcije. Akcije sintaksnog analizatora su: 1. Pomak kazaljke za čitanje leksičih jedinki u izvornom programu na sljedeću leksičku jedinku 2. Uzmi (NizZnakova) s vrha stoga 3. Stavi (NizZnakova) na vrh stoga 4. Ispiši (“Sintaksna pogreška!“) ispisuje tekst sintaksne pogreške.

Semantička analiza:

Semantička pravila su interpretacijska pravila koja povezuju izvođenje izvornog programa s ponašanjem računala. Semantika jezika određuje skup dopuštenih značenja. Semantički analizator sačinjava niz potprograma koji provjeravaju deklaraciju varijabli, tipove podataka u izrazima, indeksiranje polja, parametre potprograma, te tijek izvođenja programa. Potprogrami semantičkog analizatora izvode se istodobno s potprogramima sintaksnog analizatora, odnosno potprogrami sintaksnog analizatora pozivaju potprograme semantičkog analizatora. Semantička analiza je most između faze analize i faze sinteze. Potprogrami semantičkog analizatora ugniježđeni su u hijerarhijsku strukturu i vežu se uz sintaksna pravila. Prenošenje parametara je osnovni proces semantičkog analizatora i moguće ga je promatrati na sintaksnom stablu. Promatrajući sintaksno stablo, parametri se prenose u dva osnovna smijera: od vrha stabla prema dnu stabla i od dna stabla prema vrhu stabla.

Generiranje međukoda:

Nakon što uspješno završe leksička, sintaksna i semantička analiza, većina jezičnih procesora generira međukod. Generiranje međukoda je početni korak sinteze ciljnog programa. Međukod je zapis izvornog programa koji se izravno sprema u memoriju računala i koji je jedan od međukoraka u cjelokupnom postupku sinteze ciljnog programa. Slično potprogramima semantičkog analizatora, potprogrami generatora međukoda pozivaju se iz potprograma sintaksnog analizatora. Izborom razine i oblika međukoda nastoji se u što većoj mogućoj mjeri pojednostaviti dva osnovna procesa prevođenja tijekom sinteze ciljnog programa: proces prevođenja izvornog programa u međukod i proces prevođenja međukoda u ciljni program. Osnovne razine međukoda su međukod više razine, srednje razine i niže razine. Naredbe međukoda više razine sliče naredbama viših programskih jezika, dok su naredbe međukoda niže razine slične naredbama strojnog jezika. Međukod srednje razine zadržava neka obilježja kao što su simbolička imena varijabli, dok se složene programske petlje razlažu u više jednostavnih naredbi grananja koje su slične strojnim naredbama grananja. Po svojem obliku međukod se dijeli na linearni, postfiksni i grafički. Naredbe linearnog oblika moguće je slijedno izvoditi na procesoru računala. Naredbe međukoda u postfiksnom obliku nije moguće izravno slijedno izvoditi na procesoru računala. Za određivanje redosljeda njihovog izvođenja potrebno je koristiti potisni stog. Grafički oblik međukoda sličan je sintaksnom stablu.

Strojno nezavisno optimiranje:

Optimiranje je postupak poboljšanja kvalitete generiranog međukoda ili ciljnog programa. Kvaliteta programa mjeri se brzinom njegovog izvođenja i veličinom potrebnog memorijskog prostora. Poboljšana kvaliteta ciljnog programa postiže se nizom pretvorbi. Zahtijeva se da sve primijenjene pretvorbe sačuvaju značenje programa. Pretvorbe se dijele u dvije osnovne grupe: pripremne pretvorbe i postupci optimiranja. Postupci optimiranja obično se grupiraju u dvije klase ovisno o razini međukoda: strojno nezavisno optimiranje i strojno zavisno optimiranje. Strojno nezavisno optimiranje ne uzima u obzir svojstva arhitekture računala na kojemu se izvodi ciljni program. S druge strane strojno zavisno optimiranje uzima u obzir svojstva arhitekture računala. Različite analize moraju osigurati da učinjene promjene u međukodu ili ciljnom programu sačuvaju značenje programa.

Generiranje strojnog programa (GSP):

Izbor strukture međukoda ima utjecaja na osnovni algoritam generatora SP-a. Ako je međukod u postfiksnom obliku redosljed generiranja strojnih naredbi određuje se primjenom potisnog stoga. Ako je međukod u grafičkom obliku redosljed generiranja strojnih naredbi određuje se obilaskom sintaksnog stabla. Budući da su naredbe linearnog međukoda već poredane onim redosljedom kojim se izvode na procesoru računala, algoritam generatora strojnog programa je znatno jednostavniji jer nije potrebno odrediti redosljed generiranja strojnih naredbi. Izbor razine međukoda ima utjecaja na složenost postupka GSP-a. GSP na temelju međukoda niže razine je znatno jednostavnije od GSP-a na temelju međukoda više razine. Izbor oblika međukoda ima utjecaja na učinkovitost generiranog strojnog programa. Generator SP-a priprema strojne naredbe i simulira njihovo izvođenje. Priprema strojne naredbe uključuje određivanje operacije naredbe i njezinih operanada. Strojne naredbe uzima simulator računala. Za potrebe simulacije GSP simulira izvođenje naredbe primjenom modela računala.

Upravljanje tablicom znakova i nadziranje pogrešaka:

Ako se rezultat pojedinog koraka rada jezičnog procesora, kao što su to niz leksičkih jedinki, sintaksno stablo ili međukod spremi u memoriju računala, onda se rad jezičnog procesora raspodijeli i više cjelina, odnosno kaže se da je jezični procesor višeprolazan. Tablica znakova je podatkovna struktura sa zapisima parametara svih leksičkih jedinki koje se nalaze u izvornom programu. Tablice znakova su velike podatkovne strukture koje se dinamički mijenjaju tijekom analize izvornog programa i sinteze ciljnog programa. Otkrivanje pogrešaka u izvornom programu, određivanje mjesta pogreške i ispis odgovarajućih poruka jedan je od osnovnih zadataka jezičnog procesora. Nadzor nad pogreškama prisutan je u svim fazama rada jezičnog procesora. Dio nadzora nad pogreškama je i postupak oporavka od pogreške. Analiza se ne zaustavlja na prvoj pronađenoj pogrešci u izvornom programu, već se ona nastavlja s ciljem pronalaženja ostalih pogrešaka.

Dinamika jezičnih procesora:

JP dijele se s obzirom na dinamiku izvođenja procesa prevođenja izvornog programa u ciljni program i procesa izvođenja ciljnog programa u tri skupine: kompilatori, interpretatori i dinamički interpretatori. Za opis dinamike koristi se Hoareov sustav oznaka CSP. U CSP sustavu oznaka procesi se opisuju nizom događaja. Kompilator prevodi naredbe onim redosljedom kojim su napisane u izvornom programu. Izvođenje ciljnog programa započinje nakon što završi prevođenje. Za razliku od kompilatora interpretator ne prevodi cijeli izvorni program u ciljni program. Interpretator prevede jednu naredbu i odmah je izvede. Dinamički interpretator izvodi paralelno ili konkurentno procese prevođenja izvornog programa i izvođenja ciljnog programa. Procesi su međusobno povezani FIFO spremnikom. Za razliku od interpretatora dinamički interpretator ne prevodi i ne izvodi jednu po jednu naredbu već se prevedene naredbe stavljaju u spremnik. Za razliku od kompilatora dinamički interpretator ne prevodi cijeli izvorni program već se prevodi onoliko naredbi koliko to dozvoljava veličina FIFO spremnika.

Funkcija preslikavanja:

Funkcija preslikavanja određuje koliko se naredbi izvornog programa prevodi u koliko naredbi ciljnog programa. Ako je preslikavanje jedan u jedan onda se jedna naredba izvornog programa prevodi u jednu naredbu ciljnog programa. Prevodi li se jedna naredba izvornog programa u jednu ili više naredi ciljnog programa, preslikavanje se naziova jedan u jedan ili više. Uobičajeno je da JP viših programskih jezika (C, Java) prevodi jednu naredbu izvornog jezika u više strojnih naredbi, te se ta vrsta preslikavanja naziva jedan u više. Tijekom sinteze viših programskih jezika JP prevode naredbe strojnog jezika u naredbe nekog višeg programskog jezika. Budući da se više strojnih naredbi prevodi u samo jednu naredbu višeg progranmskog jezika, preslikavanje se naziva više u jedan. Složenu funkciju preslikavanja imaju prevoditelji viših programskih jezika. Prevoditelji viših programskih jezika su JP koji prevode naredbe jednog višeg programskog jezika u naredbe drugog višeg programskog jezika.

Mjesto izvođenja ciljnog programa JP-a:

JP dijele se u dvije skupine ovisno o tome da li generiraju ciljni program za računalo koje prevodi izvorni program ili za neko drugo računalo. Uobičajeno je da JP generira ciljni program koji se izvodi na istom računalu. Premosni prevoditelji su JP koji prevode izvorni u ciljni program na jednom računalu, a generirani ciljni program se izvodi na drugom računalu.

Jezik izgradnje jezičnog procesora:

JP je program koji se gradi primjenom jednog od programskih jezika. Samoprevoditelj je JP koji je izgrađen primjenom izvornog jezika: JPAA->B odnosno izvorni jezik A i jezik izgradnje A su identični jezici. JP moguće je izgraditi primjenom postupka samopodizanja. Samopodizanje je postupak prenošenja JP s računala A na računalo B koje je različite arhitekture. Posebni programski alati automatiziraju izgradnju JP-a. Najpoznatiji alati su programi LEX i YACC. Njihov zadatak je generiranje programa koji su JP-i. Oba programa grade JP-e primjenom višeg programskog jezika C, odnosno jezik C je ciljni jezik. U JP-u LEX izvorni jezik su regularne definicije, a u JP-u YACC izvorni jezik je BNF sustav oznaka. LEXCRegularne definicije->C YACCCBNF->C

Format zapisa:

Slobodan format zapisa ima programski jezik C, kao i većina današnjih viših programskih jezika. Slobodni format omogućuje korisniku da proizvoljno odredi strukturu i oblik zapisa izvornog programa. Čvrst format zapisa ima programski jezik FORTRAN. Jezici s čcrstim formatom zapisa imaju unaprijed određenu strukturu i oblik svakog reda izvornog programa. Čvrst format bio je posljedica zapisivanja naredbi izvornog programa na bušene kartice.

Broj prolaza JP-a:

JP su jednoprolazni ili višeprolazni. Nijedan korak rada jednoprolaznog JP-a ne sprema rezultat svog izvođenja izravno u memoriju računala. Pojedini koraci rada višeprolaznog JP-a spremaju rezultat svog izvođenja izravno u memoriju računala. Dvoprolaznost označava dva prolaza kroz čitavu podatkovnu strukturu pomoću koje je spremljen izvorni program. Prvi prolazak događa se tijekom leksičke analize, a drugi tijekom sintaksne analize. Spreme li se rezultati i ostalih koraka rada JP-a izravno u memoriju računala jezični procesor je višeprolazan.

Osnovne klase leksičkih jedinki:

Leksičke jedinke su nizovi znakova izvornog programa. Leksički analizator razvrstava leksičke jedinke u više klasa: ključne riječi, operatori, specijalni znakovi, identifikatori i konstante. Leksički analizator generira niz leksičkih jedinki koje čita sintaksni analizator kao niz kodnih znakova koji se nazivaju uniformni znakovi. Za potrebe sintaksne analize nije potrebno razlikovati pojedine identifikatore i konstante. Leksički analizator odredi za sve identifikatore jedinstveni kod uniformnog znaka. Za sve konstante također se odredi jedinstveni kod uniformnog znaka. Budući da je za potrebe sintaksne analize potrebno razlikovati leksičke jedinke koje su u klasi ključnih riječi, operatora i specijalnih znakova, leksički analizator odredi različite uniformne znakove za različite leksičke jedinke u tim klasama. Leksičke jedinke koje su u klasi klj. rij, oper i spec zn dio su sintaksnih pravila, a njihovi uniformni znakovi su završni znakovi gramatike. Leksički analizator čita znakove izvornog programa, grupira ih u leksičke jedinke i generira niz uniformnih znakova.

Dinamika izvođenja leksičke analize:

Suradnja leksičkog i sintaksnog analizatora ostavruje se putem poziva potprograma ili razmejnom 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 između sintaksnog i leksičkog analizatora izvodi se u tri koraka: 1. Sin.an. poziva lek.an. sa zahtijevom za sljedećim uniformnim znakom. 2. Lek.an. čita znakove izvornog programa sve dok ne odredi klasu sljedeće leksičke jedinke. 3. Lek.an. vraća sin. analizatoru odgovarajući uniformni znak. Ako se suradnja ostvari razmjenom čitave čitave tablice uniformnih znakova, onda je lek.an. zasebni prolaz jezičnog procesora. Nakon što spremi tablicu uniformnih znakova u zasebnu datoteku prestaje rad lek.an. Rad sin.an. pokreće se kasnije čitanjem uniformnih znakova iz datoteke koju je prethodno pripremio lek.an.

Nejednoznačnost u leksičkoj analizi:

Ako je moguće niz znakova grupirati na više različitih načina onda je leksička analiza nejednoznačna. Nejednoznačnost se razrješava različitim postupcima: uporabom različitih znakova za višeznačne operatore, prenošenjem sintaksnom analizatoru svih mogućih načina grupiranja znakova u leksičke jedinke, ispitivanjem šire okoline niza znakova izvornog programa i dinamičkom razmjenom podataka između leksičkog i sintaksnog analizatora. Ako se sin.an.predaju svi mogući načini grupiranja znakova, onda se uzima onaj način grupiranja koji zadovoljava sintaksna pravila i omogućuje daljnji rad jezičnog procesora. Ispitivanje šireg konteksta niza znakova uzima u obzir znakove koji su neposredno ispred ili iza leksičke jedinke. Dinamička razmjena podataka između lek. I sin.an. moguće je ostvariti na dva načina. Prvi način razmjene podataka zasniva se na prekidanju rada lek.an. u slučaju kada dobiveni niz leksičkih jedinki ne zadovoljava sintaksna pravila. Nakon što prekine rad lek.an., sin.an. zahtjeva od lek.an. da pokuša grupirati isti niz znakova na drugačiji način. Drugi način razmjene podataka zasniva se na pozivanju sin.an. svaki puta kad se dogodi nejednoznačnost u leksičkoj analizi. Lek.analizator zahtijeva od sin.analizatora dodatne podatke o sintaksnoj strukturi koji omogućuju razrješavanje nejednoznačnosti u postupku grupiranja znakova.

Leksičke pogreške i postupci oporavka od pogreške:

Ako nijedan prefiks niza znakova izvornog programa nije moguće grupirati ni u jednu od leksičkih jedinki onda leksički analizator prijavi pogrešku. Rad JP-a ne zaustavlja se na prvoj pronađenoj pogrešci u izvornom programu, već se pokreće postupak oporavka od pogreške. Postupak oporavka od pogreške preuredi niz znakova izvornog programa tako da se omogući nastavak rada JP-a. Postupci oporavka od pogreške zasnivaju se na sljedećim postupcima promjene niza znakova izvornog programa: odbacivanje uzastopnih znakova, odbacivanje ne nužno uzastopnih znakova, dodavanje znakova u niz, zamjena znakova u nizu novim znakovima i zamjena mjesta znakova u nizu. Navedenim postupcima promjene želi se pronaći prefiks niza znakova izvornog programa koji zadovoljava leksička pravila.

Izgradnja leksičkog analizatora:

Za izgradnju lek.an. karakteristična su dva pristupa koja ovise o složenosti postupaka grupiranja i određivanja klase leksičkih jedinki. Prvi pristup se koristi postupkom grupiranja znakova u leksičke jedinke na temelju prekidnih znakova i postupkom određivanja klase leksičke jedinke na temelju tablice leksičkih jedinki. Drugi pristup koristi konačni automat. Grupiranje znakova izvornog programa u leksičke jedinke, određivanje klase leksičkih jedinki i provjera da li su zadovoljena leksička pravila izvode se kao jedinstven proces. Nakon posljednjeg pročitanog znaka leksičke jedinke, lek.an. ostvaren konačnim automatom donosi odluku da li leksička jedinka zadovoljava leksička pravila i kojoj klasi pripada.

Zadaci sintaksnog analizatora:

Sintaksna analiza je središnji korak rada JP-a. Sintaksni analizator slijedno čita uniformne znakove leksičkih jedinki, grupira ih u sintaksne cjeline, provjerava sintaksna pravila, stvara hijerarhiju sintaksnih cjelina, određuje mjesto sintaksnih pogrešaka, opisuje sintaksne pogreške, izvodi postupak oporavka od pogreške i gradi sintaksno stablo.

Grupiranje uniformnih znakova u sintaksne cjeline:

Sin.analizator grupira uniformne znakove leksičkih jedinki u sintaksne cjeline. Osnovne sintaksne cjeline su izrazi, naredbe, blokovi naredbi i programl. Izrazi su dijelovi naredbi programskog jezika. Izrazi se zadaju primjenom infiksnog, prefiksnog ili postfiksnog sustava oznaka. U većini programskih jezika naredbe su najznačajnije sintaksne cjeline. Struktura naredbe zadana sintaksnim pravilima je jednoobrazna za sve tipove naredbi ili višeobrazna. Većina programskih jezika koristi višeobraznu strukturu. Naredbe imaju jednostavnu ili složenu strukturu. Naredbe programskog jezika FORTRAN, APL i SNOBOL imaju jednostavnu strukturu. Naredbe složene strukture ugnježđuju druge naredbe. Blok naredbi jest slijed naredbi izvornog programa omođen graničnicima. Graničnici su ključne riječi begin i end, zagrade {}... Cjelokupni program čini sintaksnu cjelinu koja točno određuje strukturu glavnog programa i potprograma, te način deklaracije varijabli i potprograma. Četiri osnovne strukture programa su: zasebna deklaracija potprograma, ugnježđeni potprogrami, zasebna deklaracija podataka i nestrukturirane naredbe.

Ako je izvorni program zasnovan na zasebno deklariranim potprogramima onda je moguće glavni program i potprograme spremiti u različite datoteke. Ako je izvorni program zasnovan na ugnježđenim potprogramima onda se svi potprogrami deklariraju unutar glavnog programa. Ako je izvorni program zasnovan na zasebno deklariranim podacima onda se u zasebnom dijelu deklariraju svi podaci, a u zasebnom dijelu zapisuju sve naredbe. Ako je zasnovan na nestrukturiranim naredbama onda je izvorni program niz naredbi bez oznaka koje odvajaju naredbe glavnog programa od naredbi potprograma.

Provjera sintaksnih pravila:

Sin.analizator provjerava da li struktura izraza, naredbi i cjelokupnog izvornog programa zadovoljava zadana sintaksna pravila. Za opis sintaksnih pravila koristi se više različitih sustava oznaka, od regularnih izraza, formalnih gramatika, pa sve do posebno uvedenih sustava oznaka kao što su BNF i COBOL. Sin.analizator moguće je programski ostvariti na mnogo različitih načina.

Stvaranje hijerarhijske strukture (Sintaksni analizator):

Sin.analizator gradi hijerarhijsku strukturu na temelju značenja pojedinih sintaksnih cjelina. Hijerarhijska struktura jednoznačno se određuje za sve sintaksne cjeline jer je ona od velikog značaja za sem.analizator i generator ciljnog programa. Hijerarhijska struktura izraza određuje se na temelju prednosti operatora, zagrada, pravila asocijativnosti operatora itd.

Određivanje mjesta i opis pogreške (Sintaksni analizator):

Budući da u izvornom programu prevladavaju sintaksne pogreške postupak određivanja mjesta i opis sin.pogrešaka je značajniji od postupaka određivanja mjesta i opisa leksičkih semantičkih pogrešaka. Većina postupaka parsiranja precizno određuje mjesto pogreške. Zahtijeva se da postupak određivanja mjesta i opisa pogrešaka precizno odredi mjesto pogreške, kratko i jasno opiše pogrešku, te da značajno ne uspori rad sin.analizatora. Većina LL(k) i LR(k) parsera omogućuje precizno određivanje mjesta pogreške u izvornom programu.

Postupak oporavka od pogreške (Sintaksni analizator):

Nakon što pronađe prvu pogrešku sin.analizator nastavlja daljnju analizu u cilju pronalaženja ostalih pogrešaka. Tijekom postupka oporavka od pogreške sin.analizator promijeni stanje tako da se omogući daljnji rad sin.analizatora. Ne obavi li sin.an. na odgovarajući način postupak oporavka od pogreške u nastavku analize nastaju lažne pogreške. Lažne pogreške nije učinio sam korisnik već one nastaju zbog neodgovarajuće promjene stanja sin.analizatora. Postupci oporavka od pogreške zasnivaju se na različitim algoritmima: traženje sinkronizacijskog znaka, algoritmi lokalnih promjena, dodatne produkcije koje uključuju pogreške i algoritmi globalnih promjena. Ako koristi algoritam traženja sinkronizacijskog znaka onda sin.an. izbacuje redom sve uniformne znakove do prvog sinkronizacijskog znaka. Ako koristi algoritam lokalnih promjena onda sin.an. zamjenjuje, dodaje ili izbacuje uniformne znakove u cilju postizanja prefiksa neanaliziranog dijela izvornog programa koji zadovoljava sintaksna pravila. Ako se koriste dodatne produkcije koje uključuju pogreške onda se skup produkcija gramatike proširuje dodatnim produkcijama. Algoritmi globalnih promjena minimiziraju broj promjenau cjelokupnom izvornom programu potrebnih da se postigne sintaksno ispravan izvorni program.

Podatkovna struktura sintaksnog analizatora:

Podatkovnu strukturu sin.an. čine globalni i lokalni podaci. Osim sin.an. globalne podatke koriste i ostali koraci rada JP-a. Lokalne podatke koristi isključivo sin.an. za potrebe izvođenja izabranog procesa parsiranja. Globalnu strukturu podataka sačinjavaju tablica znakova i stog. Tablica uniformnih znakova jest osnovna i najznačajnija podatkovna struktura sin.an. Tablica sadrži niz uniformnih znakova zapisanih onim redosljedom kojim su leksičke jedinke zadane u izvornom programu. Tablica ključnih riječi, operatora i specijalnih znakova je druga po važnosti tablica koju koristi sin.an. Ako se u tablici uniformnih znakova koristi jedinstveni kodni znak KROS, sin.an. primjenom tablice ključnih riječi, operatora i spec.zn određuje koja leksička jedinka je zadana u izvornom programu. Sin.an. rijetko koristi tablicu identifikatora. Bez obzira na primijenjenu tehniku programskog ostvarenja procesa parsiranja većina sintaksnih i semantičkih analizatora razmjenjuje podatke primjenom stoga. Stog se gradi izravno za potrebe procesa parsiranja, ili se koristi stog jezika izgradnje JP-a u slučaju primjene tehnike rekurzivnog spusta. Lokalna struktura podataka sin.an. gradi se za potrebe procesa parsiranja. Lokalna struktura ovisi o primijenjenoj tehnici parsiranja.

Opis sintaksnih pravila:

Sintaksna pravila najčešće se definiraju primjenom kontekstno neovisne gramatike, regularnih izraza, BNF sustava oznaka i sustava COBOL.

BNF sustav oznaka:

BNF sustav oznaka sličan je sustavu oznaka kontekstno neovisne gramatike. Osnovni elementi jezika definiraju se primjenom pravila, izraza, konstanti, varijabli, znaka jednakosti, operatora nadovezivanja i operatora izbora. BNF sustav oznaka dozvoljava rekurzivne definicije. Pravila imaju lijevu i desnu stranu odvojenu znakom jednakosti. Lijevu stranu čini točno jedna varijabla. Desnu stranu čini više izraza odvojenih operatorom izbora. Dozvoljava se da desna strana nema nijedan izraz ili znak. ::= ili -> znak jednakosti, <v> varijabla v, *znak do znaka* nadovezivanje, I operator izbora.

Sustav oznaka COBOL:

1965. godine za opis jezika COBOL koristi se sljedeći sustav oznaka: *Mala slova* varijabla, *Velika slova* konstanta koju je moguće izostaviti, *Podcrtana velika slova* konstanta koju nije moguće izostaviti, *Znak do znaka* nadovezivanje, [ ] neobavezni izbor jedne od mogućnosti zadane u zagradama [], { } obavezni izbor jedne od mogućnosti zadane u zagradama {}, ... ponavljanje prethodne sintaksne cjeline.

Sintaksna analiza zasnovana na Co-No tablici:

Sintaksna analiza primjenom Co-No tablica (Current operator – Next operator) jedna je od najstarijih načina analize i prevođenja izvornog programa u ciljni program. Osnovna prednost postupka zasnovanog na Co-No tablici jest brzina, dok je nedostatak neučinkovito korištenje memorijskog prostora. U naredbama je moguće zadati aritmetičke izraze koji imaju najviše dva operanda. Koriste se operacije zbrajanja, oduzimanja, množenja i dijeljenja. Znak pridruživanja jest ->. Ispravnost izvornog programa i prevođenje u strojni program zasniva se na poznavanju dva podatka: lijevog operatora i desnog operatora.

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. Gradnja stabla nastavlja se primjenom produkcija gramatike na čvorove generativnog stabla koji su označeni nezavršnim znakovima gramatike. Ako je oznaka čvora jednaka lijevoj strani produkcije, onda se ta produkcija primijeni na promatrani čvor. Gradnja generativnog stabla nastavlja se sve dok ima čvorova označenih nezavršnim znakovima gramatike. Izgradi li se generativno stablo koje ima listove označene završnim znakovima niza w, niz w jest u zadanom jeziku. Tijekom parsiranja znakovi se čitaju slijeva nadesno, te se na temelju pročitanog znaka i znakova lijeve i desne strane produkcije primijeni odgovarajuća produkcija gramatike.

S-gramatika:

Kontekstno neovisna gramatika je S-gramatika ako i samo ako su zadovoljena sljedeća svojstva: 1. Desna strana bilo koje produkcije započinje završnim znakom gramatike. 2. Desna strana nijedne produkcije nije prazni niz epsilon. 3. Ako više produkcija ima isti nezavršni znak na lijevoj strani, onda desne strane tih produkcija započinju različitim završnim znakovima. Zadovoljava li gramatika G=(V,T,P,S) pravila S-gramatike za zadanu gramatiku moguće je izgraditi učinkovit deterministički potisni automat M=({q0}, ∑, Γ, δ, q0 Z0, O) koji prihvaća praznim stogom.

Potisni automat za S-gramatiku:

Za zadanu S-gramatiku G=(V,T,P,S) moguće je izgraditi deterministički potisni automat M=({q0}, ∑, Γ, δ, q0 Z0, O) koji prihvaća 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 gramtaike uvećan oznakom kraja niza I , ∑= TU{ I }.

3)Skup znakova stoga potisnog automata čine oznaka dna stoga v, skup nezavršnih znakova gramatike V i skup svih onih završnih znakova gramatike T' oji su na desnim stranama produkcija, ali ne isključivo na krajnje lijevim mjestima, Γ={v}UVUT'.

4) Na početku rada na stogu je oznaka dna stoga v i početni nezavršni znak S.

5)Budući da potisni automat ima samo jedno stanje, funkcija prijelaza definira se 2D tablicom. Retci tablice označavaju znakove stoga, a stupci ulazne znakove. Elementi tablice označaqvaju akciju potisnog automatana ulaznom nizu i akciju na stogu. Tijekom rada potisni automat čita vrh stoga i znak na ulazu, te se na temelju tih podataka i izgrađene tablice odlučuje za akciju na stogu i ulaznom nizu. Tablica se popunjava na temelju produkcija gramatike na sljedeći način:

5-A) Neka je produkcija oblika: *A->bα* gdje je A nezavršni znak gramatike, b je završni znak gramatike i α je niz završnih i nezavršnih znakova gramatike. U redak koji je označen nezavršnim znakom A i stupac koji je označen završnim znakom b zapisuju se sljedeće akcije potisnog automata: *Zamijeni(αr); Pomakni;* gdje akcija *Zamijeni(αr)* nezavršni znak A koji je u tom trenutku na vrhu stoga zamijeni nizom α. Znakovi niza α stavljaju se na stog obrnutim redosljedom (αr) od redosljeda kojim su navedeni u produkciji, odnosno krajnje desni znak stavlja se prvi na stog. Akcija *Pomakni* miče glavu za čitanje na sljedeći znak u ulaznom nizu. Ako je produkcija oblika: *A->b* onda akcija *Zamijeni* postaje akcija *Izvuci*. Akcija *Izvuci* uzima sa stoga nezavršni znak A koji je u tom trenutku na vrhu stoga.

5-B) Ako je završni znak b znak stoga, element tablice koji je u stupcu b i retku b određuje sljedeće akcije potisnog automata: *Izvuci; Pomakni;*

5-C) Element tablice koji je označen oznakom dna stoga v i oznakom kraja ulaznog niza I određuje akciju prihvaćanja niza: *Prihvati;*

5-D) Svi ostali elementi tablice koji nisu popunjeni u prethodnim koracima označavaju da se niz ne prihvaća: *Odbaci;*

Q-gramatika:

Ako se produkcijama S-gramatike dodaju epsilon-produkcije onda je moguće uz ispunjenje određenih uvjeta ostvariti učinkovit postupak parsiranja. Proširena gramatika koja omogućava gradnju učinkovitog postupka parsiranja naziva se Q-gramatika. Kontekstno neovisna gramatika je Q-gramatika ako i samo ako vrijedi: 1. Desna strana bilo koje produkcije započinje završnim znakom gramatike ili je jednaka praznom nizu epsilon. 2. Ako više produkcija ima ista nezavršni znak na lijevoj strani onda njihovi skupovi PRIMIJENI nemaju zajedničkih elemenata. Za zadanu Q-gramatiku G=(V,T,P,S) moguće je izgraditi deterministički potisni automat M=({q0}, ∑, Γ, δ, q0 Z0, O).

Potisni automat za Q-gramatiku:

Za zadanu Q-gramatiku G=(V,T,P,S) moguće je izgraditi deterministički potisni automat M=({q0}, ∑, Γ, δ, q0 Z0, O).

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 gramtaike uvećan oznakom kraja niza I , ∑= TU{ I }.

3)Skup znakova stoga potisnog automata čine oznaka dna stoga v, skup nezavršnih znakova gramatike V i skup svih onih završnih znakova gramatike T' oji su na desnim stranama produkcija, ali ne isključivo na krajnje lijevim mjestima, Γ={v}UVUT'.

4) Na početku rada na stogu je oznaka dna stoga v i početni nezavršni znak S.

5)A Pravilima za produkcije oblika A -> bα, A->b i A->ε dodaju se sljedeća dva nova pravila: Ako je produkcija oblika: *A->ε* onda se u redak tablice A i sve stupce određene znakovima skupa *PRIMIJENI(A->ε)* zapisuju akcije: *Izvuci; Zadrži;* gdje akcija *Zadrži* ostavlja glavu za čitanje na istom ulaznom znaku. Ako je za nezavršni znak A zadana ε-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;*

5)B Ako je završni znak b znak stoga, element tablice koji je u stupcu b i retku b određuje sljedeće akcije potisnog automata: *Izvuci; Pomakni;*

5)C Element tablice koji je označen oznakom dna stoga v i oznakom kraja ulaznog niza I određuje akciju prihvaćanja niza: *Prihvati;*

5)D Svi ostali elementi tablice koji nisu popunjeni u prethodnim koracima označavaju da se niz ne prihvaća: *Odbaci;*