TO OROT TEXNOLOGIYALRI UNIVERSITETI

ARIPOVA M.H., QAYUMOVA M.H., BABAMUXAMEDOVA M.Z.

TIZIMLI DASTURIY TA'MINOT

O'QUV QO'LLANMA

TOSHKENT – 2016

TOSHKENT AXBOROT TEXNOLOGIYALRI UNIVERSITETI

ARIPOVA M.H., QAYUMOVA M.H., BABAMUXAMEDOVA M.Z.

TIZIMLI DASTURIY TA'MINOT

O'QUV QO'LLANMA

TOSHKENT – 2016

Aripova M.H., Qayumova M.H., Babamuxamedova M.Z. Tizimli dasturiy ta'minot, O'quv qo'llanma, T.: Toshkent axborot texnologiyalari universiteti, 2016 - 6.

Ushbu kitobda tizimli dasturiy ta`minotning komponentalarini loyixalashtirish va amalga oshirishga masalalari qaralgan.Kitobning asosiy maqsadi «Tizimli dasturiy ta`minot» fani bo'yicha talabalarga o'quv qo'llanma sifatida xizmat qilishdir. Kitobdan fanni mustaqil o'rganish uchun qo'llanma sifatida ham foydalanish mumkin.

Kitobda tizimli dasturiy ta`minot nazariyasining asosiy tushunchalari, tarkibi, struktura va funksiyalari, berilganlar strukturalari, ularni tavsiflash va ulardan foydalanish usullari, formal til va formal grammatika tushunchasi, grammatikaning ta`rifi, Bekus-Naur formasi, anglovchilar, tahlil masalasi, translyatorlar qurishning asosiy tamoyillari, identifikatorlar jadvalini tashkil etish, leksik tahlilchilar, ckanerlarni qurishning tamoyillari, sintaksis tahlilchilarning belgilanishi, sintaksis tahlilchilar, amallar daraxti, sintaksis grafni qurish, berilgan sintaksis uchun sintaksis tahlil dasturini qurish, sintaksis diagrammalardan tahlilchilarni qurish uchun foydalanish, jadvalli-boshqariluvchi va dasturli-boshqariluvchi tahlilchilar, semantik tahlil bosqichlari, dasturlarni ichki tasvirlash usullari, kodni generasiyalashga tayyorgarlik, kodni generasiyalash va optimallash, optimallash usullari kabi asosiy savollar ko'rib chiqiladi.

Taqrizchilar:

TDTU, dotsent t.f.n. Sevinov J.A.

TATU, dotsent t.f.n. Ganiyev A.A.

MUNDARIJA

KIRISH

Ushbu kitob tizimli dasturiy ta`minotning komponentalarini loyixalashtirish va amalga oshirishga mo'ljallangan.

Kitobning asosiy maqsadi «Tizimli dasturiy ta`minot» fani bo'yicha talabalarga o'quv qo'llanma sifatida xizmat qilishdir. Kitobdan fanni mustaqil o'rganish uchun qo'llanma sifatida ham foydalanish mumkin.

Bugungi kunda ishlab chiqarishning zamonaviy vositalari tradision arxitektura kompyuterlari baza tamoyillarida qolgan holda, komanda tizimlaridan integrallashgan muhitlar va dasturlash tizimlarigacha bo'lgan, uzoq rivojlanish va takomillashish yo'lini bosib o'tdilar. Xuddi ana shu xususiyat taklif etilayotgan qo'llanmada o'z aksini topgan.

Hozirgi kunda juda katta sonli turli-tuman dasturlash tillari mavjud. Ularning barchasi o'z tarixiga, o'zining qo'llanilish sohasiga egadirlar. Ammo ushbu tillarning barchasi, asosini formal tillar nazariyasi va grammatikasi aniqlaydigan, bir xil tamoyillar asosida qurilgan. Xuddi ana shu nazariy asoslarga qo'llanmaning bo'limlari bag'ishlangan, chunki translyatorni qurish uchun, barcha translyator ishlashi uchun zarur tamoyillarni bilish kerak bo'ladi.

Va nihoyat barcha zamonaviy kompilyatorlar tizimli dasturiy ta`minotning tarkibiy bo'lagi hisoblanadilar. Ular barcha zamonaviy ishlab chiqarish vositalarining, hoh u tizimli dastur bo'lsin, hoh amaliy dastur bo'lsin, asosini tashkil etadilar.

Kitobdan foydalanish uchun kitobxon zamonaviy dasturlash tillaridan birontasini va kompyuter xotirasida ma`lumotlar va buyruqlarni ifodalashni umumiy usullarini mukammal bilishi talab qilinadi.

Kitobning. markaziy mavzusi – bu kompyuterning arxitekturasi va tizimli dasturlar o'rtasidagi o'zaro aloqalardir.

Kitobda tizimli dasturiy ta`minot nazariyasining asosiy tushunchalari, tarkibi, struktura va funksiyalari, berilganlar strukturalari, ularni tavsiflash va ulardan foydalanish usullari, formal til va formal grammatika tushunchasi, grammatikaning ta`rifi, Bekus-Naur formasi, anglovchilar, tahlil masalasi, translyatorlar qurishning asosiy tamoyillari, identifikatorlar jadvalini tashkil etish, leksik tahlilchilar, ckanerlarni qurishning tamoyillari, sintaksis tahlilchilarning belgilanishi, sintaksis tahlilchilar, amallar daraxti, sintaksis grafni qurish, berilgan sintaksis uchun sintaksis tahlil dasturini qurish, sintaksis diagrammalardan tahlilchilarni qurish uchun foydalanish, jadvalli-boshqariluvchi va dasturli-boshqariluvchi tahlilchilar, semantik tahlil bosqichlari, dasturlarni ichki tasvirlash usullari, kodni generasiyalashga tayyorgarlik, kodni generasiyalash va optimallash, optimallash usullari kabi asosiy savollar ko'rib chiqiladi.

Kitobning 1-bo'limida tizimli dasturiy ta`minot nazariyasining tarkibi, struktura va funksiyalari haqida ma`lumotlarni olish mumkin. SHuningdek ushbu bo'limda boshqaruvchi tizimli dasturlarning asosiy funksiyalari, rivojlanish

yo'llari, operasion tizimlarda jarayonlarni boshqarish, fayllar tizimi va uning asosiy xususiyatlari keltiriladi.

2-bo'lim berilganlar strukturalari, ularni tasvirlash va ulardan foydalanish usullariga bag'ishlanadi.

3-bo'lim formal tillar va formal grammatika haqidagi tushunchalarni batafsil yoritishga bag'ishlanadi. Bo'limda belgilar zanjiri, belgilar zanjiri ustida bajariladigan amallar, til tushunchasi, formal tillar haqida umumiy tushunchalar, tillarni berish usullari, tilning sintaksis va semantikasi, dasturlash tillarining xususiyatlari kabi tushunchalar mukammal yoritiladi.

4-bo'limda grammatikalar ta`rifi, Bekus-Naur formasi, grammatikaning formal ta`rifi, grammatika qoidalarining rekursiv tamoyillari, grammatika berilishining boshqa usullari, grammatikalar va tillarning sinflanishi, Xomskiy grammatikalari, chiqish zanjirlari, sentensial ko'rinish, o'ngtomonli va chaptomonli chiqishlar, chiqish daraxti, chiqish daraxtini qurish usullari, grammatikalarning birqiymatlilik va ekvivalentlik muammolari, ekvivalentlik va birqiymatli bo'lmagan grammatikani bir qiymatli grammatikaga aylantirish, grammatikaning birqiymatli emasligini beruvchi qoidalar haqidagi tushunchalar batafsil yoritiladi.

5-bo'lim anglovchilar, anglovchining umumiy chizmasi, til turlari bo'yicha anglovchilarning sinflanishi, anglovchilarning ko'rinishlari, tahlil masalasi, tahlil muammosi, regulyar tillar va grammatikalar, konteks-ozod tillar va grammatikalar, regulyar tillarning anglovchilari, LL(1) –grammatikalar, faktorlash, kontekst-ozod tillarning anglovchilari, chekli avtomatlar, anglovchilarning ishlash algoritmlariga bag'ishlangan.

6-bo'limda translyatorlarni qurishning asosiy tamoyillari, translyatorlar, kompilyatorlar va interpretatorlarning ta`rifi, belgilanishi, translyasiya bosqichlari, translyator ishining umumiy chizmasi, o'tish tushunchasi, ko'po'tishli va biro'tishli kompilyatorlar, assembler tilining translyatorlari, makroaniqlashlar va makrokomandalar, kompilyator strukturasi nuqtai nazaridan ko'po'tishli va biro'tishli kompilyatorlar haqida ma`lumotlar yoritiladi.

7-bo'limda identifikatorlar jadvali, uni tashkil etish usullari, xesh- funksiya va xesh- manzillash haqidagi ma`lumotlar bilan tanishasiz.

8-bo'lim leksik tahlilchilar, skanerlarni qurish tamoyillari, leksik tahlil masalalari, leksik tahlilning xizmatchi jadvallari, uning echish yo'nalishlari, leksik tahlilchini qurishni avtomatlashtirish masalalarini ifodalashga bag'ishlanadi.

9-bo'lim sintaksis tahlil bosqichini batafsil tushunishga imkon beradi. Ushbu bo'limda sintaksis tahlilchini belgilanishi, tahlil daraxtini amallar daraxtiga aylantirish, amallar daraxti, sintaksis tahlilchini avtomatlashtirish dasturini yaratish, sintaksis grafni qurish, sintaksis grafni qurish qoidalari, berilgan sintaksis uchun grammatik tahlil dasturini qurish, grammatik tahlilning jadvalli-boshqaruv dasturini qurish, yuqorilovchi sintaksis tahlil, asoslar, belgilar jadvali bilan

ishlash, sintaksis xatoliklar vaqtida tiklash haqidagi ma`lumotlarni olishingiz mumkin.

10-bo'limda semantik tahlil bosqichi haqida ma`lumotlarga ega bo'lasiz. Semantik tahlil bosqichi va kodni generasiyalashga tayyorgarlik, dasturlash tillarining leksik birliklarni identifikasiyalash jarayoni, xotirani taqsimlash kabi tushunchalarga to'lik javob olasiz.

11-bo'limda kodni generasiyalash va kodni generasiyalash usullari, sintaksis boshqaruvchi tarjima, dasturlarni ichki tasvirlash usullari, postfiks yozuvlar haqidagi batafsil ma`lumotlarni olishingiz mumkin.

12-bo'limda kodni optimallashning asosiy usullari va umumiy tamoyillari haqidagi ma`lumotlar bilan tanishasiz.

Kitob 12 bo'limdan tashkil topgan. Har bir bo'limdan so'ng o'z bilimingizni sinash maqsadida sinov savollariga javob berishingiz mumkin.

Avtorlar qo`llanmada uchrashi mumkin bo`lgan sintaksis xatoliklar uchun uzr so`raydilar va ushbu xatoliklarni ko`rsatgan o`quvchilarga o`z minnatdorchiliklarini bildiradilar.

1.BO'LIM

Tizimli dasturiy ta`minotning asosiy tushunchalari. Operasion tizimlar va muhitlar

Ingliz tili texnik adabiyotida *System Software* (tizimli dasturiy ta`minot) termini dasturlar va barcha uchun umumiy hisoblangan, kompyuterning texnik qurilmalaridan yangi avtomatik dastur ishlanmalarni yaratish uchun va mavjudlarini bajarilishini tashkil etishda foydalaniladigan, dasturlar majmuasini anglatadi. Tizimli dasturiy ta`minot quyidagi besh guruhga bo'linishi mumkin:

1. Operasion tizimlar (OT).

2. Fayllarni boshqarish tizimlari.

3. Foydalanuvchining OT va dastur muhitlari bilan o'zaroxarakatlarini tashkil etuvchi interfeys qobiqlar.

4. Dasturlash tizimlari.

5. Utilitalar.

Tizimli dasturlarning ushbu guruhlarini alohida qisqacha qarab chiqamiz.

Operasion tizim (OT) deganda, ko'pincha bir tomondan, kompyuter apparaturasi va foydalanuvchi o'rtasidagi interfeysni, boshqa tomondan hisoblash tizimini zahiralarini foydaliroq ishlatilishini ta`minlash va ishonchli hisoblashlarni tashkil etishni boshqaruvchi va qayta ishlovchi dasturlarning majmuasi tushuniladi. Amaliy dasturiy ta`minotning ixtiyoriy komponentasi albatta OT boshqaruvida ishlaydi. Dasturiy ta`minotning OTdan tashqari bironta komponentasi kompyuter apparaturasi bilan bevosita muloqot qila olmaydi.

OT bajaradigan asosiy funksiyalar quyidagilar hisoblanadi:

- foydalanuvchidan (yoki tizim operatoridan) mos tilda ifodalangan topshiriq va buyruqlarni operator direktivalari ko'rinishida yoki mos manipulyator yordamidagi o'ziga xos ko'rsatmalar ko'rinishida (masalan, sichqoncha yordamida) qabul qilish, - va ularni qayta ishlash;

- dastur so'roqlarini bajarilishga qo'yish uchun qabul qilish, to'xtatib turish va boshqa dasturlarni to'xtatish;

- operativ xotiraga bajarilishi kerak bo'lgan dasturlarni yuklash; Dasturlarni inisializasiyalash (unga boshqaruvni uzatish natijasida prosessor dasturni bajaradi); barcha dastur va berilganlarni identifikasiyalash;

- fayllarni boshqarish tizimini ishini ta`minlash, ma`lumotlar omborini boshqarish tizimlarini ishini ta`minlash (MOBT), bu esa o'z navbatida barcha dasturiy ta`minotning foydaliligini oshirish imkonini yaratadi;

- multidasturlash rejimini ta`minlash, ya`ni ikki va undan ortiq dasturlarni bitta prosessorda bajarilishini ta`minlash;

- barcha kiritish/chiqarish amallarini boshqarish va tashkil etishni boshqaruvchi funksiyalarni ta`minlash;

- xotirani taqsimlash, ko'pgina zamonaviy tizimlarda virtual xotirani ham boshqarish;

Masalalarni rejalashtirish va navbatga qo'yishni berilgan mos strategiya va xizmat ko'rsatish dissiplinasi haqida;

- bajarilayotgan dasturlar o'rtasidagi berilganlar va ma`lumotlar almashinuvini tashkil etish;

- bir dasturni boshqa dastur ta`siridan himoya etish; berilganlarni himoyalanishini ta`minlash;

- tizimni vaqtinchalik ishdan chiqish xolatida xizmatlar ko'rsatish;

- Dasturlash tizimlarini, ya`ni foydalanuvchilar ular yordamida o'z dasturlarini yaratishlari uchun, ishlashlarini ta`minlash.

Fayllarni boshqarish tizimining belgilanishi — fayllar ko'rinishida tashkil etilgan berilganlarga qulayroq murojaatni tashkil etish. Xuddi shu fayllarni boshqarish tizimlari tufayli berilganlarga pastdarajali aniq manzillarni ko'rsatish orqali kerakli yozuvga murojaat o'rniga, fayl ismini ko'rsatish va unga yozish orqali mantiqiy murojaatdan foydalaniladi. Barcha zamonaviy OT mos fayllarni boshqarish tizimlariga egadirlar. Lekin tizimli dasturiy ta`minotning ushbu ko'rinishini alohida toifaga ajratish maqsadga muvofiqdir, chunki, qator OT bir nechta fayllar tizimi bilan ishlash imkoniyatiga egadirlar (yoki bir nechtasidan bittasi bilan, yoki bir vaqtning o'zida bir nechtasi bilan). Bu holatda o'rnatilgan fayl tizimlari haqida gap ketmoqda (qo'shimcha fayllarni boshqarish tizimlarini o'rnatish mumkin), va bu ma`noda ular mustaqildirlar. YAna shuni ta`kidlash kerakki, ixtiyoriy fayllarni boshqarish tizimi o'z-o'zidan mavjud emas – u aniq OT ishlashi uchun va aniq fayllar tizimi bilan ishlab chiqarilgan. Barchaga ma`lum bo'lgan fayllar tizimi FAT (file allocation table)[1] fayllarni boshqarish tizimi sifatida ko'pgina variantlarga ega, masalan FAT-16 MS-DOS uchun, super-FAT OS/2 uchun, FAT Windows NT uchun. Boshqacha aytganda, qandaydir fayllar tizimiga mos tashkil etilgan fayllar bilan ishlash uchun, har bir OT uchun mos fayllarni boshqarish tizimlari ishlab chiqarilishi kerak; va bu fayllarni boshqarish tizimi faqat o'zi yaratilgan OT uchun ishlaydi.

OT bilan o'zaroxarakatlarni qulaylashtirish uchun qo'shimcha interfeys qobiqlardan foydalanish mumkin. Ularning asosiy vazifalari — OT ni boshqarish bo'yicha imkoniyatlarni kengaytirish, yoki tizimga qurilgan imkoniyatlarni o'zgartirish, interfeys qobiqlarga misol sifatida grafik interfeysning turli variantlarini aytish mumkin X Window UNIX tizimi oilasiga tegishli, K Desktop Environment Linux ga tegishli), PM Shell yoki Object Desktop OS/2 Presentation Manager grafik interfeysli; va nihoyat, OS Windows oilasi uchun Microsoft interfeyslarning turli variantlarini aytish mumkin, ular Explorerni o'rnini bosishlari va yoki grafik interfeysli NTX ni, yoki OS/2, yoki MAC OS ni eslatishlari mumkin. YAna Microsoft kompaniyasining umumiy interfeysli Explorer nomli dastur moduli bilan amalga oshirilgan OTlari oilasi haqida ta`kidlash kerak (Windows katalogida joylashgan system.init faylida SHELL=EXPLORER.EXE qator mavjud), ya`ni bu tizimlarda faqatgina interfeys qobiqlar o'zgartiriladi, OT esa o'zgarishsiz qoladi; u OTga integrallashtirilgan.

Boshqacha aytganda, operasion muhit dastur interfeyslari bilan aniqlanadi, ya`ni API (application program terface). Amaliy dasturlash interfeysi (API) o'z ichiga jarayonlarni, xotirani va kirish/chiqishni boshqarishni oladi.

Va nihoyat, utilitalar deganda maxsus tizimli dasturlar tushuniladi. Ular yordamida operasion tizimga xizmat ko'rsatish, berilganlarni saqlovchilarni ishlashga tayyorlash, berilganlarni qayta kodlashni bajarish, tashqi saqlovchilarda berilganlarni joylashtirishni optimallash, va boshqa bir qancha hisoblash tizimiga xizmat ko'rsatish bilan bog'liq ishlarni bajarish mumkin. Utilitalarga formatlash dasturlarini, OTning asosiy tizimli fayllarini ko'chirish dasturini, yana Symantec firmasining Pitera Norton ismi bilan ataluvchi utilitalar to'plamini misol sifatida keltirish mumkin. Tabiiyki, utilitalar mos operasion muhitdagina ishlay oladilar.

Servis (service) – bu mos so'roqni bajarish, xizmat ko'rsatishdir.

Kompyuterning tradision arxitekturasi (fon Neyman arxitekturasi) hozirgi zamonaviy hisoblash tizimlarida o'zgarmay va ustunligicha qolmoqda. Xuddi shunday asosda dasturiy ta`minotning translyatorlar, kompilyatorlar va interpretatorlar kabi kompyuterlar uchun dasturiy ta`minot vositalarini ishlab chiqariladigan baza tomoyillari ham o'zgarmay qolmoqda. Bu sohadagi zamonaviy publikasiyalarning yo'qligiga ham shu sabab bo'lsa kerak. SHunga qaramay ishlab chiqarishning zamonaviy vositalari tradision arxitektura kompyuterlari baza tamoyillarida qolgan holda, komanda tizimlaridan integrallashgan muhitlar va dasturlash tizimlarigacha bo'lgan, uzoq rivojlanish va takomillashish yo'lini bosib o'tdilar. Xuddi ana shu xususiyat taklif etilayotgan darslikda o'z aksini topgan.

Hozirgi kunda juda katta sonli turli-tuman dasturlash tillari mavjud. Ularning barchasi o'z tarixiga, o'zining qo'llanilish sohasiga egadirlar. Ammo ushbu tillarning barchasi, asosini formal tillar nazariyasi va grammatikasi aniqlaydigan, bir xil tamoyillar asosida qurilgan. Xuddi ana shu nazariy asoslarga darslikning 3-4-5-6-bo'limlari bag'ishlangan, chunki translyatorni qurish uchun, barcha translyator ishlashi uchun zarur tamoyillarni bilish kerak bo'ladi.

Va nihoyat barcha zamonaviy kompilyatorlar tizimli dasturiy ta`minotning tarkibiy bo'lagi hisoblanadilar. Ular barcha zamonaviy ishlab chiqarish vositalarining, xox u tizimli dastur bo'lsin, xox amaliy dastur bo'lsin, asosini tashkil etadilar.

1.1.Tizimli dasturiy ta`minotining tarkibi, struktura va funksiyalari

Zamonaviy amaliyotda hisoblashlarni bajarish uchun hisoblash mashinalari va tizimlaridan foydalaniladi.

Elektron hisoblash mashinalari (EHM) – bu apparaturalar majmuasi bo'lib, u qandaydir algoritmlar to'plamini xarakatlarini bajarishni ta`minlaydi.

Hisoblash tizimi – bu bir yoki bir necha EHM va dasturlar to'plami bo'lib, o'ziga yuklatilgan funksiyalarni bajarilishini ta`minlaydi.

SHunday qilib, hisoblash tizimini 2 (ikki) bo'lakka bo'linadi:

- dasturiy;
- apparat.

Apparat qismi o'z ichiga hisoblash mashinasining qurilmalarining barcha funksiyalarini kiritadi. Apparat qismining belgilanishi: kiritish va chiqarish amallarini bajarish, saqlash, uzatish va ma`lumot almashuvni amalga oshirishdan iborat.

Dasturiy qismi esa – bu dasturlar to'plami bo'lib, hisoblash tizimiga yuklatilgan funksiyalarni bajarilishini tartibini belgilaydi. Dastur natija olish maqsadida apparatura ishini boshqaradi.

Dasturiy bo'lak, tizimli dasturiy ta`minot (DT) deb ataluvchi biri biriga bog'liq ko'pgina komponentalar to'plamidan tashkil topgan.

«Ta`minot» so'zi hisoblash tizimiga yuklatilgan funksiyalarni amalga oshirilishiga ko'rsatmadir.

Umuman olganda dasturiy ta`minot tizimini ikkita katta bo'lakka bo'linadi:

1. Amaliy DT (foydalanuvchi uchun);
2. Tizimli DT.

Amaliy dasturiy ta`minot – bu foydalanuvchilarning o'zlari uchun o'zlari tomonidan yaratiladigan dasturlardir.

Tizimli dasturiy ta`minot – bu barcha uchun yaratilgan va universal bo'lgan dasturlardir. U xam ikki bo'lakka bo'linadi.:

1. Umumiy tizimli dasturiy ta`minot;
2. Maxsus tizimli dasturiy ta`minot.

Maxsus tizimli dasturiy ta`minot hisoblash tizimining aniq spesifik masalalarini echish uchun umumiy dasturiy ta`minotga qo'shiladi (uchishni boshqarish, xarbiy masalalar va x.k.).

Umumiy tizimli dasturiy ta`minot universal bo'lib keng ommaviy masalalarni echish uchun mo'ljallangan.

Bundan so'ng umumiy tizimli dasturiy ta`minotni ko'rib o'tamiz. U quyidagi tarkibdan iborat:

1. Tizimli qayta ishlovchi dasturlar;
2. Tizimli boshqaruvchi dasturlar;
3. Tizimli qayta ishlovchi dasturlar va tizimli boshqaruvchi dasturlarga qo'shimcha dasturlar;
4. Tekshiruvchi –diagnostik dasturlar;
5. Amaliy dasturlar paketi;
6. Tizimli dasturiy ta`minot xujjatlari majmuasi.

1. Tizimli qayta ishlovchi dasturlar foydalanuvchilarga xizmat ko'rsatish masalalarini ularning talablariga ko'ra echishga mo'ljallangan.

2. Tizimli boshqaruvchi dasturlar hisoblash tizimining barcha funksiyalarini foydaliroq tashkil etish, hisoblash tizimi va foydalanuvchi o'rtasidagi interfeysni tashkil etish uchun mo'ljallangan.

Izoh: Faqatgina tizimli boshqaruvchi dasturlargina apparaturaga bevosita murojat qila oladilar.

Izoh: Tizimli boshqaruvchi dasturlarni operasion tizimlar (OT) deb ataladi.

OT interfeysning quyidagi variantlarini ta`minlashi mumkin:

• komanda interfeysi;

• dastur interfeysi (chaqiriklar tizimi yoki ba`zi bir tizimli funksiyalarni bajarish uchun qism dasturlar ko'rinishida):

• foydalanuvchi interfeysi (darcha, menyu, klavishalar va x.k.)

3. Tizimli qayta ishlovchi dasturlar va tizimli boshqaruvchi dasturlarga qo'shimcha dasturlar tizimli qayta ishlovchi va boshqaruvchi dasturlarni imkoniyatlarini kengaytirish uchun mo'ljallangan.

Ular tarkibiga:

• servis dasturlari;

• instrumental dasturlar kiradi.

Servis dasturlarga:

• dastur qobiqlari (nadstroyki);

• utilitalar kiradi.

Dastur qobig'ining yaxshi tomoni – bu hisoblash tizimining zahiralariga murojatni yaxshilashdan iborat (Windows yo'l ko'rsatuvchisi va x.k.).

Instrumental dasturiy qurilmalarga quyidagilar kiradi:

- SUBD (ma`lumotlar omborlarini boshqarish tizimlari);
- Mashina grafikasi tizimlari va x.k.

4. Tekshiruv-diagnostik dasturlar EHMning ishlatish jarayonidagi nosozliklarni tekshirish, aniqlash va bartaraf etish profilaktikasi uchun mo'ljallangan.

5. Amaliy dasturlar paketi – bu amaliy masalalarni echish uchun mo'ljallangan dasturlar to'plamidir. Ularga– ilmiy hisoblashlar, modellashtirish va x.k. misol bo'ladi.

Har bir amaliy dasturlar paketining o'z tili bo'lib, bu paketga tegishli ishlarning bajarilish tartibi ushbu tilda ifodalanadi.

6. Xujjatlar majmuasi – matnli xujjatlarning GOST (ESKD) ga muvofiq tayyorlangan to'plami bo'lib, ularda tizimli dasturiy ta`minotning mos bo'laklarini ekspluatasiya qilish va o'rnatish haqidagi ma`lumotlar beriladi.

1.1.1.Tizimli qayta ishlovchi dasturlarning asosiy funksiyalari va tarkibi

Tizimli qayta ishlovchi dasturlarning asosiy funksiyalari va tarkibi quyidagilardan iborat:

1.Assembler;
2.Aloqa redaktorlari va yuklovchilar;
3.Makroprosessorlar;
4.Translyatorlar (tarjimonlar);
5.Til konvertorlari;
6.Redaktorlar va matn prosessorlari;
7.Qayta ishlovchilar;
8.Dizassembler;
9.Kross-tizimlar;
10.Kutubxonachilar.

1. Assembler – bu shunday tizimli qayta ishlovchi dastur bo'lib, u biron bir mashinaga mo'ljallangan dasturlash tilida yozilgan dastur matnini ob`ekt kodiga

aylantirish uchun mo'ljallangan. (Assembler tilidagi matn direktivalar va ismlardan tashkil topadi, mashina kodi esa fakat baytlardan tashkil topadi.).

Izoh: Ob`ekt kodi yoki aloqa redaktorining kirishiga, yoki yuklovchining kirishiga kelib tushadi.

2. Aloqa redaktorlari – bu shunday tizimli qayta ishlovchi dastur bo'lib, ular Assembler yordamida alohida olingan ob`ekt modullarini yagona modulga birlashtirishga mo'ljallangan. Aloqa redaktori chegarasida barcha manzil yo'nalishlari yagona adreslar fazosiga o'rnatiladi.

Alohida ob`ekt modullarida har bir ob`ekt moduli aloqa redaktorining chiqishini yuklovchining kirishi deb hisoblaydi.

YUklovchilar dasturni qayta ishlovchi dasturga yuklaydilar va unga boshqaruvni uzatadilar. YAna shu bilan birga ular yuklovchilarni bog'lovchi alohida modullarni birlashtiradilar.

Izoh: YUklovchilar joy o'zgartiruvchi yoki absolyut bo'lishlari mumkin.

- Absolyut yuklovchilar har bir dasturni bittadan fiksirlangan manzil bo'yicha yuklaydilar.
- Joy o'zgartiruvchi yuklovchilar dasturni xotiradagi ixtiyoriy bo'sh joyga joylashtirishlari mumkin.

3. Makroprosessorlar– bu shunday dasturlarki, ular belgili qayta ishlashga mo'ljallangan bo'lib, bu jarayonda qandaydir qisqa frazalarga (makrochaqiriqlarga) uzun frazalar (makrokengaytmalar) mos qo'yiladi. Makroprosessorning kirishida qandaydir makrochaqiriklardan olingan matn bo'lib, chiqishida esa – makrokengaytmalar bo'ladi.

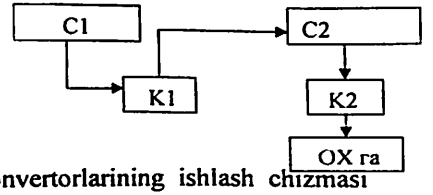
4. Translyatorlar (tarjimonlar) bir tilda yozilgan matnini boshqa tilga o'giradilar. Translyatorlarning quyidagi ko'rinishlarini ajratib ko'rsatish mumkin:

- kompilyatorlar: kirishida yuqori daraja tilida yozilgan dastur matni, chiqishida mashina kodlaridagi aloqa redaktoriga yoki yuklovchiga uzatiladigan dastur.

Xususiyatlari: tarjima funksiyalarini aniq bo'linishi va tarjima qilingan funksiyalarni bajarilishi.

- interpretatorlar – funksiyalar bo'linmaydilar, balki moslashtiriladilar. Interpretator tarjimani va bajarilishni qatorlab va kooperativ bajaradi. Ulardan yozilgan dasturni dialog asosida qayta ishlashda foydalanish qulay.

5.Til konvertorlari bir yuqori daraja dasturlash tilida yozilgan dastur matnini boshqa yuqori daraja dasturlash tiliga aylantirish uchun mo'ljallangan. Ular C1 dasturlash tilida yozilgan dasturni C2 dasturlash tiliga aylantirish uchun kerak (1-rasm).



1-rasm. Til konvertorlarining ishlash chizmasi

Konvertor - bir tildan boshqa o'sha darajadagi tilga o'girish demakdir. Misol sifatida Paskal tilidagi kodni SI tilidagi kodga aylantiruvchi dasturni keltirish mumkin.

6. Matn redaktorlari matnni qayta ishlash uchun keng imkoniyatlarga egaliklari bilan farqlanadilar.

7. Qayta ishlovchilar dasturlarni bajarilish jarayonida uchrashi mumkin bo'lgan xatolarni qidirish va bartaraf etish uchun mo'ljallangan.

8. Dizassembler bu mashina kodlari ketma-ketligini assembler ko'rinishiga o'zgartiradigan dastur.

Izoh: Ular ham ba`zi bir xarakatlar tizimini bajarilishini assembler ko'rinishida ko'rish imkonini yaratadilar.

9. Kross-tizim – bu dastur bir hisoblash mashinasida mashina kodlarida ifodalangan boshqa bir hisoblash mashinasining dasturlarini olish uchun qo'llaniladi. Loyixalashtirilayotgan hisoblash tizimlari arxitekturasini qayta ishlash uchun foydalaniladi.

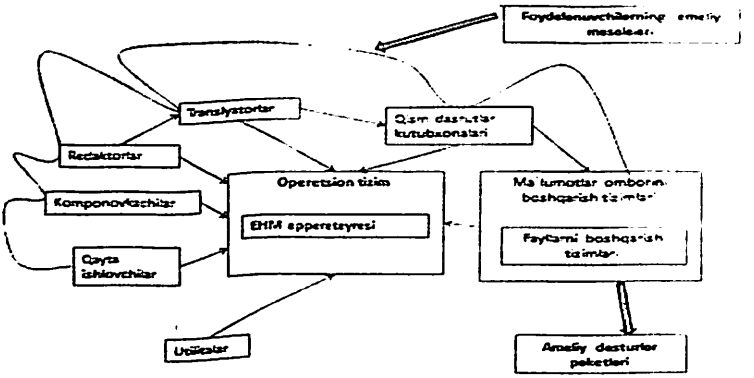
Kompilyator boshlang'ich dasturni bitlar to'plamiga aylantirganligi sababli, ushbu bitlardan boshqa mashinada ham foydalansa bo'ladi. Bir turdagi mashina uchun dasturni tayyorlash boshqa bir kompьyuterda amalga oshirilgan holda, bunday kompilyatorlar kross kompilyatorlar deb ataladi.

10. Kutubxonachilar – kiritilayotgan matn, ob`ekt moduli raqami bo'lishi mumkin bo'lgan kutubxona fayllarini tashkil etish va ularga xizmat ko'rsatish uchun dasturlardir.

1.1.2.Tizimli boshqaruvchi dasturlarning asosiy funksiyalari va rivojlanish imkoniyatlari

YUqorida aytib o'tilganidaek, tizimli boshqaruvchi dasturlarni operasion tizimlar (OT) deb ataladi.

Operasion tizim (OT) deganda, ko'pincha bir tomondan, kompyuter apparaturasi va foydalanuvchi o'rtasidagi interfeysni, boshqa tomondan hisoblash tizimini zahiralarini foydaliroq ishlatilishini ta`minlash va ishonchli hisoblashlarni tashkil etishni boshqaruvchi va qayta ishlovchi dasturlarning majmuasi tushuniladi. Amaliy dasturiy ta`minotning ixtiyoriy komponentasi albatta OT boshqaruvida ishlaydi. 2-rasmda hisoblash tizimining dasturiy ta`minotining umumiy tuzilishi keltirilgan. Dasturiy ta`minotning OTdan tashqari bironta komponentasi kompyuter apparaturasi bilan bevosita muloqot qila olmaydi.



2- rasm. Hisoblash tizimining dasturiy ta`minotining umumiy tuzilishi

OT bajaradigan asosiy funksiyalar quyidagilar hisoblanadi:

- foydalanuvchidan (yoki tizim operatoridan) mos tilda ifodalangan topshiriq va buyruqlarni operator direktivalari ko'rinishida yoki mos manipulyator yordamidagi o'ziga xos ko'rsatmalar ko'rinishida (masalan, sichqoncha yordamida) qabul qilish, - va ularni qayta ishlash;
- dastur so'roqlarini bajarilishga qo'yish uchun qabul qilish, to'xtatib turish va boshqa dasturlarni to'xtatish;
- operativ xotiraga bajarilishi kerak bo'lgan dasturlarni yuklash; dasturlarni inisializasiyalash (unga boshqaruvni uzatish natijasida prosessor dasturni bajaradi); barcha dastur va berilganlarni identifikasiyalash;
- fayllarni boshqarish tizimini ishini ta`minlash, berilganlar omborini boshqarish tizimlarini (SUBD) ishini ta`minlash, bu esa o'z navbatida barcha dasturiy ta`minotning foydaliligini oshirish imkonini yaratadi;
- multidasturlash rejimini ta`minlash, ya`ni ikki va undan ortiq dasturlarni bitta prosessorda bajarilishini ta`minlash;
- barcha kiritish/chiqarish amallarini boshqarish va tashkil etishni boshqaruvchi funksiyalarni ta`minlash;
- xotirani taqsimlash, ko'pgina zamonaviy tizimlarda virtual xotirani ham boshqarish;

Masalalarni rejalashtirish va navbatga qo'yishni berilgan mos strategiya va xizmat ko'rsatish dissiplinasi haqida quyidagilarni ta`kidlash mumkin:

- bajarilayotgan dasturlar o'rtasidagi berilganlar va ma`lumotlar almashinuvini tashkil etish;
- bir dasturni boshqa dastur ta`siridan himoya etish; berilganlarni himoyalanishini ta`minlash;
- tizimni vaqtinchalik ishdan chiqish holatida xizmatlar ko'rsatish;

-dasturlash tizimlarini. ya`ni foydalanuvchilar ular yordamida o'z dasturlarini yaratishlari uchun, ishlashlarini ta`minlash.

Bizga ma`lumki, operasion tizimlar uzluksiz rivojlanib boradilar. Buning imkoniyat va sabablari quyidagilardir:

1. Apparat ta`minotining yangi ko'rinishlarining yuzaga kelishi va yangilanishi.
2. YAngi servislar.
3. O'zgartirishlar kiritish.

Operasion tizimning rivojlanishi zaruriyat, aks holda u yangi dasturlar va yangi qurilmalar bilan ishlay olmaydi.

Misol uchun, Intel prosessorlar Hyper Threadni qo'llanishidan boshlab Windows NT operasion tizimini bozordan siqib chiqara boshladi, ular ikkinchi prosessorni emulyasiya qildilar. Windows NTni yadrosini emulyasiyasi bilan qanday ishlashni «bilmay qolib» tizim yiqildi.

Operasion tizimlarning rivojlanishi ikki yo'nalishda borishi mumkin.

1.sakrab (Windows) va

2.asta sekin (UNIX).

Birinchi holda, avvalgi mavjud dasturlardan to'liq voz kechishga to'g'ri keladi, chunki bu tizimda hech qanday ko'chirib o'tishlar nazarda tutilmagan. Ammo, faqat uning o'zigagina yozilgan yangi dasturlardan butun mahsulot yuzaga keladi. Bu esa dasturiy ta`minotni ishlab chiqaruvchilar (yaratuvchilar) uchun juda foydalidir.

Ikkinchi holda, yangi qurilma imkoniyatlarini eski dasturlar bilan bog'lashga to'g'ri keladi. Operasion tizim va dasturlar imkoniyatlarini kengaytiruvchi dastur zaplatkalarini (patchlar) ishlab chiqarishga to'g'ri keladi.

Operasion tizimlarni ishlab chiqish bo'yicha uch echish yo'li mavjud.

1. YOpiq echim. Barcha ishlab chiqarilayotgan mahsulotlar lisenziyalar bilan himoyalangan va ishlab chiqarilgan firmadan tashqarida o'zgartirila olmaydi. Bunday echim firmadan qurilmalarning o'zgarishiga juda tez e`tibor berishni, marketing va ishlab chiqarishga katta mablag'larni sarflashni talab qiladi.

Bunday yo'nalishning yutug'i bo'lib, o'zi ishlab chiqarayotgan dasturiy mahsuloti uchun firmaning javobgarligi hisoblanadi. Bu yo'nalishga yaxshi xujjatlashtirish, universallik, standartlashtirish xususiyatlari xos. Bunday yo'nalishdan foydalanayotgan taniqli firmalardan Microsoft, SUN, SCO ni ko'rsatish mumkin.

Bunday yo'nalishning kamchiligi firmalarning inertligi hisoblanadi. O'zgarayotgan sharoitlarga tezda ahamiyat bera olmaslik qobiliyati. Operasion tizimning boshqa firma ishlab chiqarayotgan operasion tizim bilan o'zaro aloqa qila olmasligi, ya`ni moslasha olmasligi imkoniyati.

2.Ochik echim. Ishlab chiqarilayotgan echimlar umumiy lisenziyaga ega ochik kodli echimga bo'ysunadilar. Ixtiyoriy xoxlovchi dastur mahsulotining berilish kodini olishi mumkin, agar uning avtor tomonidan ishchi varianti qo'yilgan bo'lsa. Bunday echim dasturiy mahsulot xatosiz ishlashiga kafolat bera olmaydi. Bu echimlar ko'p hollarda yaxshi xujjatlashtirilmagan. Bunday

yo'nalishning yutug'i bo'lib, turli davlat va turli soha mutaxassislarining bir komandada ishlay olishi imkoniyati hisoblanadi. Ishlab chiqaruvchilar komandasining o'zgarish shartlariga tezlikdagi e'tibori. Barcha mumkin bo'lgan platformalarda va ixtiyoriy boshqa tizimlarda ishlay olish imkoniyati. Bunday yo'nalishdan foydalanayotgan tanikli firmalardan RedHat, SuSe, FreeBSD; Novell ni ko'rsatish mumkin.

3.Aralash echim. Ishlab chiqarilayotgan echimlar umumiy koddan tashqari yana shaxsiy ishlab chiqarilayotgan, lisenziya bilan himoyalangan echimdan ham foydalanadilar. Bunday yo'nalish ochiq echimlardan eng yaxshi echimlarni tanlab olish va ular asosida o'zlarining echimlarini tashkil etish imkonini beradi. Bunday yo'nalish har ikkala yo'nalishning eng yaxshi xususiyatlarini hisobga oladi, chunki firma faqatgina o'zining echimlarinigina xujjatlashtirish va unga javobgarlikni bo'yniga olibgina qolmay, balki tanlab olingan ochiq echimlar uchun ham javobgarlikni his etadi. Bunday yo'lni tanlagan firmalar MacOS, BeOs, QNX, Netrino.

1.1.3.Operasion tizimda jarayonlar va ularni boshqarish

«Operasion tizim» atamasi ostida biz funksiyasi hisoblash tizimining zahiralarini taqsimlash va ishlatilishini tekshirishdan iborat bo'lgan dasturlar majmuini tushunamiz. Biz bilamizki, hisoblash tizimida fizik zahiralar, ya`ni aniq qurilma (magnit disklari, operativ xotira, prosessorning ishlash vaqti) bilan bog'liq zahiralar mavjud. YAna shu bilan birga tizimda tizimning yaxshi ishlashini ta`minlovchi mantiqiy zahiralar (gohida ularni virtual deb ataladi), ya`ni aniq qurilma ko'rinishida mavjud bo'lmagan, lekin foydalanuvchiga ba`zi bir manba sifatida taqdim etiladigan zahiralar mavjud. Fizik va mantiqiy zahiralarni biz birgalikda hisoblash tizimining zahiralari deb ataymiz.

Ixtiyoriy operasion tizim (OT) ularni boshqarish hisobiga, ularning xususiyatlarini xarakterlaydigan qandaydir tushunchalarga ega. Bunday tushunchalarga fayl, jarayon, ob`ekt, va x.k. kiradi. Har bir OT o'zining tushunchalari to'plamiga ega. Misol uchun, Windows NT OTda bunday tushunchalarga ob`ekt tushunchasi kiradi va shu tushunchani boshqarish bilan barcha mumkin bo'lgan funksiyalar ko'rsatiladi. Agar biz UNIX OT qarasak, bu erda birinchi navbatda bunday tushuncha bo'lib fayl tushunchasi tushuniladi, ikkinchi navbatda jarayon tushunchasi tushuniladi.

Jarayon - bu shunday tushunchaki, u barcha OTlarda mavjud. Jarayon - bu dastur bo'lib, zahiralarga egalik qilish xuquqiga egadir. Ikki dasturni (foydalanilayotgan kod va berilganlar) va dasturga tegishli barcha zahiralarni (bu operativ xotira fazosi, tashqi xotirada joylashgan berilganlar, boshqa zahiralarga egalik qilish xuquqi, misol uchun, aloqa chiziqlariga) ko'rib chiqamiz. Agar ushbu ikki dasturga tegishli zahiralar to'plami mos tushsa, biz bu ikki dastur haqida ikki jarayon deb ayta olmaymiz, bu – bitta jarayondir. Agar har bir dasturda o'zining zahiralari to'plami mavjud bo'lsa, ular kesishishlari mumkin, lekin mos tushmasalar biz ikki jarayon haqida gapirishimiz mumkin.

Agar, qachonki bir necha jarayonlarning zahiralar to'plami bo'sh bo'lmagan kesishuvga ega bo'lsa, u holda zahiralardan bo'lib foydalanish masalasi yuzaga keladi. Bizda bir necha, har biri o'zining zahirasi sifatida chop etish qurilmasiga ega va ixtiyoriy vaqtda bu zahiraga qandaydir ma`lumotni chop etish buyurtmasi bilan murojaat qilishi mumkin, jarayon bo`lishi mumkin. Jarayonlarni xarakatini boshqarishni ko'rsatgan OTning funksiyalaridan birini chop etish qurilmasi misolida jarayonlarni sinxron ishlashi ko'rsatilgan. Keling, jarayonlarni boshqarish deganda nimalarni tushunamiz, qarab ko'raylik.

Jarayonlarni boshqarish :

1. Markaziy prosessor vaqtidan foydalanishni boshqarish.
2. Kirish bufer va real xotirani boshqa masalalar uchun bo'shatishni boshqarish.
3. Bo'linadigan zahiralarni boshqarish.

Jarayonlarni boshqarishning asosiy muammolari.

Birinchi - markaziy prosessor vaqtidan foydalanishni boshqarish (SP), yoki bu muammoni markaziy prosessorni rejalashtirish deb ataladi, ya`ni qaysi vaqtda qanday masala yoki jarayon markaziy prosessorning aktivligini egallashini boshqarish: qaysi jarayon markaziy prosessorda ishlaydi.

Ikkinchisi - real xotirani boshqa masalalar uchun bo'shatish va kirish buferini boshqarish. Faraz qilaylik, ko'pchilik bir vaqtning o'zida, masalan butun kurs kompyuter atrofida o'tirib, barchasi bir vaqtning o'zida jarayon ko'rinishidagi masalalarni bajarilishga qo'yishdi. Tizimda masalalar to'plami tashkil bo'ldi (kafolat beramiz: yuzdan ortiq). Barcha hisoblash tizimi esa multidasturli rejimda ishlash uchun buncha ko'p masalani qabul qila olmaydi – bu juda ko'p. Bu holda kirish masalalari buferi, yoki kirish jarayonlari buferi, ya`ni o'zining markaziy prosessorda qayta ishlanishini kutayotgan kirish buferi hosil bo'ladi. Endi bu kirish buferidan qayta ishlashni boshlash uchun tanlash navbati muammosi yuzaga keladi. Bu masala buferni rejalashtirish masalasi.

Endi real xotirani boshqa masalalar uchun bo'shatish masalasini rejalashtirishni qarab chiqamiz. Prosessor tomonidan bir necha masala qayta ishlanmokda, va bizning oldimizda real xotirani boshqa masalalar uchun bo'shatish masalasi turibdi. Bu holda qandaydir qayta ishlanayotgan masalalarni tashqi xotira qurilmasiga ko'chirish zaruriyati tug'iladi. Qanday algoritm bo'yicha biz bu masalalarni ko'chiramiz? Ko'chirish strategiyasi qanday bo'ladi? Masalan, har bir juft masalani ko'chirish. Ko'chirish jarayonini ko'proq yoki kamroq foyda bilan tashkil etish – bu muammo.

Uchinchisi - bo'linadigan zahiralarni boshqarish. SHunday zahiralar to'plamlari mavjudki, ularga murojaat aniq bir vaqtlarda turli jarayonlarni nomidan tashkil etiladi. Bu xuddi o'sha chop etish qurilmasi masalasi kabidir. OTning ko'pchilik hollarda xususiyatlarini aniqlaydigan funksiyalaridan biri, bu jarayonlar o'zaroxarakatini tashkil etishni va umumiy zahiralardan foydalanishni ta`minlash funksiyasidir. CHop etish qurilmasi misolidagi muammoni oson echish mumkin,

ammo agar umumiy operativ xotiraga ega ikki dastur bo'lgan holda bo'linadigan zahiralarni boshqarish – bu murakkab masala.

Endi keling, OT konstruksiyasiga qaraymiz. Ixtiyoriy OT yadroga ega. OT yadrosi ko'pincha uning rezident qismi hisoblanadi, ya`ni OTning shunday turg'un qismiki, u jarayonlarni ko'chirishlarda ishtirok etmaydi (u har doim operativ xotirada joylashadi) va OT rejimida ishlaydi, yoki supervizor rejimida ishlaydi (maxsus rejimda). YAdroga ushbu OTga xarakterli bo'lgan asosiy tushunchalarni boshqaruvchi baza qurilmalari kiradi. YAna ba`zi fizik qurilmalarni boshqaruvini ta`minlovchi dasturlar to'plami ham kirishi mumkin. YAdroning funksiyasiga, xususiy holda, uzilishlarni qayta ishlash ham kiradi.

Biz gohida dasturlarni, boshqaruv zahiralari, qurilma drayverlari (fizik yoki mantiqiy) deb ataymiz. Misol uchun OT yadrosiga operativ saqlash qurilmasi drayveri kirishi shart.

So'ngra yadro atrofida hisoblash tizimi zahiralarini boshqarish dasturlari turadi. Birinchi daraja asosan fizik qurilmalar drayverlaridan tashkil topadi. Keyingi daraja - mantiqiy qurilmalarni boshqarishdir va x.k. Bunday darajalar juda ko'p bo'lishi mumkin. YAdrodan qanchalik uzoqlashsak, shunchalik joy bo'lishi ehtimolligi kattaroqdir. Misol tariqasida, qaerdadir bizning chizmada haqiqatda mantiqiy disklarni boshqaruv drayverlari bilan bog'liq, fayllarni boshqarish drayverlari paydo bo'lishi mumkin, ular esa o'z navbatida fizik qurilmalarni boshqarish drayverlari bilan bog'liq va x.k.

Umuman, OT ning barcha komponentalari supervizor rejimida, yoki OT rejimida ishlashi shart emas. YAdrodan kerakli darajada mantiqiy uzoqlashgan komponentlarning ko'pchiligi oddiy foydalanuvchi rejimida ishlashlari mumkin. Xuddi shunday barcha OT komponentalarning rezident rejimida ishlashlari ham shart emas. Ko'pincha ko'pgina funksiyalar uchun bu talab qilinmaydi.

Markaziy prosessor vaqtidan foydalanishni boshqarish

Umuman olganda, OT da masalani aktivligini markaziy prosessorga uzatishning qanday algoritmi tanlovi amalga oshirilganligiga qarab, ushbu OT ning ko'pgina real ekspluatasion xususiyatlari bog'liq. Algoritm tanlovi OT ning ishining foydaliligini baholash uchun foydalaniladigan foydalilik kriteriylari bilan to'liq aniqlanadi. SHuning uchun markaziy prosessor vaqtidan foydalanishni boshqarishni biz operasion tizim turlari misolida ko'rib chiqamiz.

Paketli qayta ishlov tizimlari. Birinchi holat. Mening ixtiyorimda tizimning katta hajmli hisoblash quvvatini talab qiluvchi juda katta masalalar yoki dasturlar mavjud. Bu ko'p hisoblashlarni talab qiladigan va tashqi qurilmalarga kam murojaat etadigan hisob masalalaridir. Bu masalalar bir hisoblash tizimida bajarilishi shart. Bu paket masalalarini bajarilishida foydalilik kriteriysi bo'lib nimalar hisoblanadi? Qanday parametrlar to'plamini olish kerak? Bunday holat uchun hisoblash tizimining ish unumdorligini kriteriysi bo'lib markaziy prosessorning bandligi darajasi hisoblanadi. Agar u kam vaqt bo'sh qolsa (ya`ni kutish rejimida ishlasa, boshqa barcha jarayonlar ma`lumot almashinuvi bilan

band bo'lsalar, yoki OT vaqtni band qilsa), u holda biz bunday tizim foydali ishlamoqda deb ayta olamiz. Bunga biz quyidagi rejalashtirish algoritmidan foydalanib erishishimiz mumkin. Biz qayta ishlash uchun OT ning imkoniyatidan kelib chiqqan (maksimum yoki barcha masalalarni) holda masalalar to'plamini mulьtidasturlash rejimiga qo'yamiz. Markaziy prosessor vaqtini rejalashtirish algoritmi quyidagicha bo'ladi: agar MP jarayonlardan biriga ajratilgan bo'lsa, u holda bu jarayon MPni quyidagi holatlar yuz berguniga qadar band etadi:

1.Tashqi qurilmaga murojaat.
2.Jarayonni tugashi.
3.Jarayonni siklga tushib qolishi.

Ushbu xolatlardan birontasi yuz berganda boshqaruv boshqa jarayonga uzatiladi. Bir jarayondan boshqa jarayonga uzatish soni minimallashtirilgan. Bir jarayondan ikkinchi jarayonga boshqaruvni uzatishda OT bir qancha xarakatlarni to'plamini bajarishi kerak bo'ladi, bu esa vaqtni yo'qotish demakdir, shuning uchun ushbu xarajatlar minimallashtirilgan. OTning bunday ish rejimi **paketli rejim** deb ataladi, bu rejimda ishlayotgan OT esa **paketli operasion tizim** deb ataladi.

Paketli qayta ishlash tizimlari (misol uchun,OC EC) hisoblash xarakteriga ega tezda natija olishni talab qilmaydigan masalalarni echishga mo'ljallangan. Paketli qayta ishlash tizimlarining bosh maqsadi va foydalilik kriteriysi maksimal o'tkazuvchanlik qobiliyati hisoblanadi, ya`ni birlik vaqt ichida maksimal sonli masalalarni echish hisoblanadi. Bu maqsadga erishish uchun paketli qayta ishlash tizimlarida quyidagi harakatlar chizmasidan foydalaniladi: ishning boshida masalalar paketi tashkil etiladi, har bir masala tizimli resurslarga o'zining talablariga ega, bu masalalar paketidagi masalalardan bir vaqtda bajariluvchi masalalar to'plamining mulьtidastur aralashmasi tashkil etiladi. Bir vaqtda bajarilish uchun zahiraga o'z talablariga ega masalalar, ya`ni hisoblash mashinasining qurilmalarini bandligining balansi ta`minlanadigan masalalar tanlanadi. Masalan, multidastur aralashmasida bir vaqtning o'zida hisoblash va interaktiv kiritish-chiqarish masalalarning bor bo'lishi maqsadga muvofiqir. SHunday qilib masalalar paketidan yangi masalani tanlash tizimda yuzaga kelgan ichki xolatlarga bog'liq, ya`ni «foydali» masala tanlanadi. Bunday OT larda u yoki bu masalaning aniq bir vaqt ichida bajarilishiga kafolat bera olmaymiz. Paketli qayta ishlovchi tizimlarda prosessorni bir masalani bajarilishidan boshqa bir masalaning bajarilishiga o'tkazish faqatgina agar aktiv masalaning o'zi prosessordan voz kechsagina amalga oshiriladi. Masalan, kiritish-chiqarish amalini bajarish zaruriyati tufayli. SHuning uchun bitta masala prosessorni uzoq vaqt band etib turishi mumkin. Bu esa interaktiv masalalarning bajarilishini sekinlashtiradi. SHunday qilib, foydalanuvchining paketli qayta ishlash tizimi o'rnatilgan hisoblash mashinasi bilan o'zaro muloqoti shunga kelib taqaladiki, foydalanuvchi masalani olib kelib dispetcher operatorga topshiradi va natijani kun oxirida barcha masalalar paketi bajarilgandan so'nggina oladi. Ko'rinib turibdiki, bunday tartib foydalanuvchining ish unumdorligini pasaytiradi.

Vaqtni taqsimlash tizimlari. Endi shunday holatni tassavur qilaylik, kompyuter sinfida ko'pgina sonli kishilar bor va ularning har biri qandaydir matnni taxrirlamokdalar. Har bir terminal bilan matn redaktorining nusxasi bog'langan. Endi biz qarab ko'ramiz, birinchi holat uchun qo'llangan algoritmni ushbu tizimga qo'llasak nima bo'ladi? Faraz qilaylik, foydalanuvchilardan birontasi terminal oldida uxlab qoldi, va hech qanday aktivlik ko'rsatmayapti. Markaziy prosessor vaqti ushbu jarayon bilan bog'langan, chunki bu jarayon hech qanday almashinuvni bajarmayapti va tugamagan, redaktor ishga tayyor. Bu vaqtda barcha qolgan foydalanuvchilar uxlab yotgan foydalanuvchini uyg'onishini kutib turishga majbur bo'ladilar. Bir biriga bog'liqlik holati vujudga keladi. Bu esa birinchi holat uchun yaxshi bo'lgan algoritm bu tizim uchun xattoki qudratli mashina bo'lgan holda ham yaramasligini anglatadi. SHuning uchun interaktiv masalalar bilan ishlayotgan foydalanuvchilarni ko'psonli ko'rinishi muammosini ta`minlashda boshqa algoritmlardan foydalanish, boshqa unumdorlik kriteriylariga ega algoritmlardan foydalanish maqsadga muvofikdir.

Bunday tizim uchun foydalanuvchining vaqtni kutish kriteriysi mos keladi: qandaydir xarakatni bajarish uchun buyurtma berilgan vaqtdan boshlab, to uning tizimning ushbu buyurtmaga javobini olguniga qadar bo'lgan vaqti; Tizim qanchalik unumli ishlasa, tizimda o'rtacha statistik kutish vaqti shunchalik kam bo'ladi.

Ikkinchi hol uchun masalani qarab chiqamiz. Tizimda bir qancha jarayonlar soni mavjud va rejalashtiruvchini vazifasi markaziy prosessorni vaqtini shunday tashkil etish kerakki, tizimning foydalanuvchining murojatiga reaksiyasi eng kam vaqt olsin, juda bo'lmaganda kafolatlansin. Quyidagi chizma taklif qilinadi. Tizimda kvant vaqti deb ataluvchi Δt parametrdan foydalaniladi, (umumiy holda, kvant vaqti - bu tizimni o'rnatish vaqtida o'zgarishi mumkin bo'lgan qiymatdir). Multidasturli qayta ishlash rejimidagi barcha jarayonlar to'plami ikkita qism to'plamga bo'linadi. Birinchi to'plam xali bajarilishga tayyor bo'lmagan jarayonlardan tashkil topadi; masalan, almashinuvga buyurtma bergan va uning natijasini kutayotgan jarayonlar. YAna bajarilishga tayyor jarayonlar ham bor. Ish quyidagicha amalga oshiriladi: Markaziy prosessorni band etib turgan jarayon quyidagi holatlar yuzaga kelmagunicha aktiv bo'ladi:

1. Almashinuvga buyurtma murojaati.
2. Jarayonni tugashi.
3. Ushbu jarayonga ajratilgan Δt kvanta vaqtining tugashi.

Bu holatlardan birontasi yuz berishi bilanoq, OT rejalashtiruvchisi jarayonlar orasidan bajarilishga tayyor jarayonlardan birini tanlab oladi va va unga markaziy prosessorning zahiralarini uzatadi. Jarayonni esa ushbu berilgan operasion tizimda foydalanilgan rejalashtirish algoritmidan foydalangan holda tanlaydi. Masalan, jarayon tasodifiy ravishda tanlanishi mumkin. Ikkinchi usulning mazmuni shundayki, jarayonlarni ketma-ket ko'rib chiqish amalga oshiriladi, ya`ni biz avval bir jarayonni ishga tushiramiz, keyin u ishini tugatgandan so'ng, markaziy prosessor vaqtini bajarilishga tayyor jarayonlar tartibidagi keyingi jarayonga

beriladi. Navbatdagi masala tanlashdagi uchinchi kriteriy bo'lib, ushbu jarayon markaziy prosessorni band qilmagan vaqt hisoblanadi. Bu holatda tizim shunday vaqti eng katta bo'lgan jarayonni tanlashi mumkin. Bunday algoritmlar OT da amalga oshirilgan bo'lishi kerak, ular sodda bo'lishi kerak, aks xolda tizim unumsiz ishlaydi, o'z-o'ziga (bunday tizimlar mavjud: xususiy xolda Windows oilasi tizimlari ham shu jumladan) ishlaydi.

OT ning bunday turi **vaqtni bo'laklash** OT deb ataladi. Bu OT foydalanuvchining buyurtmasiga tizimning reaksiya vaqti minimallashtirilgan rejimda ishlaydi. Bunday holda, foydalanuvchining buyurtmaga reaksiya vaqti minimal bo'lgani uchun tizimning barcha zahiralari faqat unga ajratilgandek tuyuladi.

Vaqtni bo'laklash OT (masalan, UNIX, VMS) paketli qayta ishlash tizimlarining asosiy kamchiligini, foydalanuvchi –dasturchini o'z masalalarni bajarish jarayonida ishtirok etmasligini, tugatishga mo'ljallangan. Vaqtni bo'laklash OT ning har bir foydalanuvchisiga alohida terminal ajratiladi va u orqali foydalanuvchi o'z dasturi bilan dialogda bo'ladi. Vaqtni bo'laklash tizimlarida har bir masalaga prosessorning kvant vaqti ajratilgani sababli, hech bir masala prosessorni uzoq vaqt band qila olmaydi va javob vaqti kafolatlangan. Agar kvant vaqti juda katta bo'lmagan holda, barcha foydalanuvchilarda mashinani zahiralari uning bir o'ziga ajratilgandek tuyuladi. Vaqtni bo'laklash tizimlari paketli qayta ishlash tizimlariga nisbatan kichik o'tkazuvchanlik qobiliyatiga ega bo'lganliklari uchun bajarilish uchun tizim uchun «foydali» bo'lgan masala emas, balki barcha foydalanuvchi tomonidan yuborilgan masala qabul qilinadi, bundan tashqari hisoblash tizimini prosessorini bir masaladan ikkinchi masalaga tez o'tkazish hisobiga xarajatlar yuzaga keladi. Vaqtni bo'laklash tizimlarini unumdorlik kriteriysi bo'lib, maksimal o'tkazuvchanlik qobiliyati emas, balki foydalanuvchi uchun yaratilgan qulayliklar unumdorligi hisoblanadi.

Real vaqt tizimlari. Endi quyidagi masalani qaraymiz. Faraz qilaylik, avtopilot orqali boshqariluvchan samolet pasayish amalini bajarsin. Har bir samolyotda samolyotdan ergacha bo'lgan balandlikni o'lchab turuvchi asbob bor, Samolyotning ish rejimi shundayki, berilgan dastur asosida uning funksiyalarini kompyuter boshqaradi. Demak, agar bizda avtopilot tizimi mavjud bo'lsa, va samolyot pasaymoqda, u holda tizim uchish balandligini nazorat qilishi kerak. Bu samoletning markaziy kompyuteri bir necha masalalarni echishi mumkin: u uchish balandligini, baklardagi yonilgi miqdorini darajasini, dvigatelni ishini qandaydir ko'rsatkichlarini nazorat qilishi kerak. Bu funksiyalarni har birini boshqaruvi bilan o'z jarayoni shug'ullanadi. Faraz qilaylik, bizda paketli OT, va biz baklardagi yonilg'i darajasini diqqat bilan tekshirayapmiz. Bu holatda avariya holati yuzaga keladi, chunki samolyot pasaya boradi va OT buni sezmaydi ham.

Faraz qilaylik, bizda vaqtni bo'laklash tizimi bo'lsin. Bu tizimning xususiyatlaridan biri jarayondan jarayonga ko'psonli o'tkazishlar tufayli unumsizligidir. YAna o'sha holat: balandlik nolga tenglashayapti, OT esa o'tkazishlar jadvali bilan shug'ullanayapti. Bunday variant ham to'g'ri kelmaydi.

Bunday ko'rinishdagi masalalarni echish uchun ham o'z rejalashtirish qurilmalari kerak. Bu holatda **real vaqt OT** deb ataluvchi, asosiy kriteriysi tizimning u yoki bu holatga reaksiyasini kafolatlash vaqti hisoblangan OTdan foydalaniladi. YA`ni tizimda shunday holatlar to'plami mavjudki, tizim ixtiyoriy holatda ularga ahamiyat beradi va ularni avvaldan berilgan vaqt ichida qayta ishlaydi. Bizning misolimiz uchun bunday holat balandlikni o'lchovchi asbobdan kelib tushayotgan ma`lumotlar bo'lishi mumkin. Bu sinf OT uchun juda oddiy algoritmlardan foydalaniladi. Barcha rejalashtirish ushbu holatni qandaydir qiymatdan oshmagan vaqt ichida qayta ishlash kriteriysiga asoslanadi. Lekin real vaqt OTlari yana o'zining spesifik qurilmasiga ega bo'lib, faqatgina ushbu sodda algoritmdangina emas, balki tizimning ichki qayta qurilishlaridan ham aniqlanadi.

Real vaqt OTlari turli texnik ob`ektlarni boshqarish uchun qo'llaniladi, masalan, stanok, sputnik, ilmiy eksperimental o'rnatishlar yoki texnologik jarayonlar, ya`ni galьvanik liniya, domen jarayoni va x.k. Bu barcha holatlarda ob`ektni boshqaruvchi u yoki bu dastur ma`lum bir chegaraviy vaqt oralig'ida bajarilishi shart, aks holda avariya yuz berishi mumkin: sputnik ko'rinish xonasidan chiqib ketishi, datchiklardan kelayotgan eksperimental berilganlar yo'qolib qolishi, galvanik qoplash qalinligi normaga mos kelmasligi mumkin. SHunday qilib, real vaqt OTlarining unumdorlik kriteriysi dasturni bajarilishga qo'yilish vaqti bilan, natija olish o'rtasidagi berilgan vaqtga chidash qobiliyati hisoblanadi. Bu vaqt tizimning reaksiya vaqti, tizimning mos xususiyati esa - reaktivlik deb ataladi. Bunday tizimlar uchun multidastur aralashmasi avvaldan ishlab chiqilgan dasturlarning fiksirlangan to'plamini tashkil etadi, dasturni bajarilishga tanlovi esa ob`ektning holatidan kelib chiqib yoki rejalashtirilgan ishlar ro'yxatiga mos amalga oshiriladi.

Izoh. Markaziy prosessor vaqtidan foydalanishni boshqarish va markaziy prosessorni rejalashtirish funksiyalariga xulosa sifatida, e`tiboringizni quyidagi ikki faktga qarataman.

Birinchi fakt bu markaziy prosessor vaqtini taqsimlashni rejalashtirish algoritmlari bo'lib, ular hisoblash tizimining ekspluatasiya xususiyatlarini aniqlaydilar. Men turli OT dan turli maqsadlarda foydalanishni taklif qilgan holda maxsus misollarni keltirdim.

Ikkinchi fakt. Biz uch xil OTni ko'rib o'tdik: paketli qayta ishlash tizimlari, vaqtni bo'laklash tizimlari va real vaqt tizimlari. Bugungi kunda real vaqt tizimlari OTlarning alohida sinfi. OS Windows, OS SOLYARIS yoki LINUX kabi tizimlar ham real vaqt tizimlari emas.

Birinchi ikkita paketli qayta ishlash va vaqtni bo'laklash rejimlarini umumiy qabul qilingan OTlarda qo'llash mumkin. Real, katta va OTlar aralash tizimlar hisoblanadi, ya`ni, ularda markaziy prosessorni rejalashtirish elementlarida hisoblash masalalarini boshqaruvchi algoritmlar, va interaktiv masalalarni boshqarish algoritmlari ham qo'llanilgan.

Markaziy prosessor ishini tashkil etishning misoli quyidagi chizmada ko'rinadi. Rejalashtiruvchi ikki darajali chizmada qurilgan. Biz ko'pgina masalalar to'plami hisoblash masalalari va interaktiv masalalardan tashkil topgan deb hisoblaymiz. Birinchi daraja ikki sinf o'rtasidagi ustunlilikni aniqlaydi va Markaziy prosessorni yoki avval hisoblash masalasiga, yoki interaktiv masalaga uzatadi. Ikkinchi daraja esa masalani bir sinf chegarasida qanday tanlash va uni qanday uzishni aniqlaydi. Bunday aralash tizim quyidagicha ishlashi mumkin. Birinchi daraja rejalashtirish quyidagi tamoyil bo'yicha ishlaydi: agar berilgan vaqtda bironta ham bajarilishga tayyor interaktiv masala yo'k bo'lsa, (bu esa real bo'lishi mumkin holat, agar foydalanuvchilar matnni tahrirlash bilan band bo'lsalar), u holda MP hisoblash masalalariga uzatiladi, lekin bitta shart qo'shiladi: interaktiv masala paydo bo'lishi bilanoq, hisoblash masalasi uziladi, va boshqaruv interaktiv masalalar blokiga uzatiladi. Bu esa jarayonlarni boshqarishning birinchi funksiyasiga tegishlidir.

Kiritish buferi va real xotirani boshqa masalalar uchun bo'shatishni boshqarish

Real tizimlarda real xotirani boshqa masalalar uchun bo'shatish buferi moslashadi, ya`ni operativ xotiradan jo'natilayotgan ma`lumotni tashqi qurilmalar fazosi va jarayonlarni kiritish buferi moslashadi. Bu birinchi ko'rsatma.

Ikkinchi ko'rsatma. Zamonaviy OT etarli darajada «yalqov»lar va operativ xotiradan jo'natilayotgan ma`lumotni ko'pincha jarayonlar xotira bloklari birligida emas, balki butun jarayonni ko'chirish orqali amalga oshiradilar. Bu erda ikkita savol tug'iladi: jarayonni ko'chirish kriteriysi qanday va buferdan mulьtiqayta ishlash uchun kerakli jarayonni tanlash kriteriysi qanday? Bu holda eng sodda variant topish vaqtidan foydalanish hisoblanadi. Birinchi holatda, agar biz real xotirani boshqa masalalar uchun bo'shatish sohasiga aktiv holatdagi jarayonni ko'chirish masalasini hal etayotgan bo'lsak, u holda astronomik vaqt bo'yicha qayta ishlash holatida eng ko'p bo'lgan jarayonni olishimiz mumkin. Teskari jarayon esa bunga simmetrik, ya`ni jarayonlarni kiritish buferidan biz u erda eng ko'p bo'lgan jarayonni tanlab olishimiz mumkin. Bular eng sodda va real rejalashtirish algoritmlaridir, lekin kriteriylar o'zgarishi bilan ular ham o'zgarishlari mumkin. Kriteriylardan biri, agar barcha masalalar turli toifalarga bo'lingan bo'lsalar, ya`ni OT masalalari bo'lsalar- bu holda ular birinchi navbatda bajariladilar (va ular o'rtasida topish vaqtini baholash algoritmi mavjud) va barcha qolgan masalalar. Bunday model jamiyatdagi hayotiy nohaqliklarga o'xshab ketadi, ya`ni qudratli odamlarga hamma narsa mumkin, lekin qolgan kuchsizroqlari esa kutib turishlari mumkin bo'lgani kabi.

Bo'laklangan zahirani boshqarish

Biz bu muammoni hal qilishni OS UNIX operasion tizimi misolida qarab chiqamiz.

Faraz qilaylik, ikki jarayon mavjud, va ular umumiy operativ xotira fazosida ishlamoqdalar. Bu holatda bo'laklangan zahiralar turli rejimlarda ishlashlari

mumkin, ya`ni bu ikki jarayon turli mashinalarda joylashishi holati ham yuz berishi mumkin, lekin ular bitta umumiy operativ xotira maydoni bilan bog'langan. Bu holda xotira ishini buferlashtirish muammosi yuzaga keladi, chunki har bir mashina o'zining o'qish-yozish mexanizmlariga ega. Bu erda shunday noxush holat yuzaga keladiki, fizik xotiraning holati uning aniq qiymatiga mos kelmay qoladi. YAna bir qancha ikkita mashinada ishlayotgan OT uchun muammolar yuzaga keladi.

YAna quyidagicha muammo. Faraz qilaylik, bitta mashinada ishlayotgan ikkita jarayon mavjud bo'lsin. SHunday qurilmalar mavjud bo'lishi kerakki, ular bo'laklanayotgan xotiraga murojaatni sinxronlashi, ya`ni shunday sharoit yaratishlari kerakki, har bir operativ xotira bilan ishlayotgan jarayonning ma`lumot almashinuvi aniq bajarilishi kerak. Bu esa bo'laklangan xotiradan har bir ma`lumotni o'qish jaranida bu xotiraga nimadir yozayotgan barcha foydalanuvchilar bu jarayonni tugatgan bo'lishlari kafolatlanishi kerak, ya`ni bo'laklangan xotirada ma`lumot almashinuv sinxronizasiyasi bo'lishi kerak.

Xaqiqatda masalalarni echish jarayonida umumiy xotira kabi bo'laklangan zahiralarga talab qo'yilmaydi, lekin bir vaqtda xarakatlanayotgan jarayonlar bir biriga qandaydir ta`sir ko'rsata olishlari maqsadga muvofiqdir. Bu apparat uzilishlari kabi ta`sirlardir. Ko'pgina OTlarda bularni amalga oshirish uchun jarayonlar orasidagi signallarni uzatish qurilmasi mavjud. Bir jarayon ikkinchisiga aytadi: - signalni boshqa jarayonga uzat. Boshqa jarayonda bu jarayonning bajarilish uzilishi yuzaga keladi, va uzatuv boshqaruvi oldindan aniqlangan qabul qilingan signalni qayta ishlovchi funksiyaga uzatiladi. Bu OTning uchinchi funksiyasidir.

Men sizning e`tiboringizni OTning ekspluatasion xususiyatlariga ta`sir ko'rsatuvchi funksiyalariga qaratdim. Har bir OT shu kabi funksiyalar to'plamiga ega bo'lib, shu OT ning ish faoliyatini ta`minlaydi.

Avval aytganimizdek, har bir OT kiritish/chiqarish buferining ishini ta`minlaydi. Bu esa operasion tizimning asosiy funksiyalaridan biridir. Har bir OT murojatni tashki saqlash qurilmasi (VZU) ga saqlovchi KESH buferlarga ega. Bu esa operasion tizimni optimallashtirish imkonini beradi. Bunday buferlashning mavjudligini belgisi bo'lib, mashinani o'chirishdan avval operasion tizimni tugatish hisoblanadi. Masalan, MS-DOS operasion tizimida ishlab turib, компьyuterni ixtiyoriy vaqtda o'chirish mumkin, chunki unda bunday buferlash yo'q. Windows va UNIX tipidagi operasion tizimlarda mashina ishlab turgan vaqtida uni o'chirish maqsadga mufoviq emas, chunki bu holda ma`lumot yo'qotilishi ehtimolligi mavjud. Bunday buferlash darajasi tizimning real unumdorligini belgilaydi. Bizning fakultetda Pentium-lar paydo bo'lganlarida Windows 95 da ishlashda tizim 486 prosessorda ishlayaptimi yoki Pentiumlardami farqi yo'q. Bu esa tizimning unumdorligi tashqi qurilmalar unumdorligiga bog'liq emasligini anglatadi. Agar UNIX operasion tizimini qarasak, bu farq sezilirli darajada bilinadi, chunki prosessorning tezligi tizim ishining sifatiga Windows 95

ga nisbatan katta ta'sir ko'rsatadi, chunki Windows 95 tizimida tashqi qurilma bilan almashinuvlar ko'proq.

1.1.4.Fayllar tizimi

Biz avval aytib o'tganimizdek, har bir operasion tizim jarayon deb ataluvchi tushunchaga ega. YAna ikkinchi bir tushuncha ham borki, u ham juda muhim bo'lib –bu fayldir. Fayl tizimi - bu operasion tizim komponentasi bo'lib, nomlangan berilganlar to'plamiga murojaatni,,,tashkil etish va saqlashni ta'minlovchidir. Berilganlarning nomlangan to'plami fayllar deb ataladi.

Fayllarni boshqarish tizimining belgilanishi — fayllar ko'rinishida tashkil etilgan berilganlarga qulayroq murojaatni tashkil etish. Xuddi shu fayllarni boshqarish tizimlari tufayli berilganlarga pastdarajali aniq manzillarni ko'rsatish orqali kerakli yozuvga murojaat o'rniga, fayl ismini ko'rsatish va unga yozish orqali mantiqiy murojaatdan foydalaniladi. Barcha zamonaviy OT mos fayllarni boshqarish tizimlariga egadirlar. Lekin tizimli dasturiy ta'minotning ushbu ko'rinishini alohida kategoriyaga ajratish maqsadga muvofiqdir, chunki, qator OT bir nechta fayllar tizimi bilan ishlash imkoniyatiga egadirlar (yoki bir nechtasidan bittasi bilan, yoki bir vaqtning o'zida bir nechtasi bilan).

Fayllarning asosiy xususiyatlari

1.Fayl -bu ismga ega ob`ekt bo'lib, shu ism orqali faylni ichidagi ma`lumotlar bilan ishlovchi ob`ektdir. Ism bu belgilar ketma-ketligi bo'lib, uning uzunligi aniq operasion tizim turiga bog'liqdir.

2.Faylni joylashishiga bog'liq emasligi. Aniq bir fayl bilan ishlash uchun u faylning tashqi qurilmadagi joylashishini bilish talab qilinmaydi.

3.Kirish/chiqish funksiyalari to'plami. Har bir operasion tizim fayllar bilan ma`lumot almashinuvni ta`minlovchi funksiyalar to'plamiga ega. Bu funksiyalar to'plami quyidagilardan tashkil topadi:

1. **Fayl ish uchun ochilgan.** YOki mavjud yoki yangi faylni ochish mumkin. SHunday savol tug'ilishi mumkin. - nima uchun faylni ochish kerak? Nima uchun birdaniga fayldan o'qish va faylga yozish mumkin emas? Xaqiqatda, bu operasion tizimga fayl aniq jarayon bilan ishlashini markaziy ravishda e`lon qilish vositasidir. U esa ushbu ma`lumotlarga asosan qandaydir echim qabul qilishi mumkin. (masalan, boshqa jarayonlar uchun ushbu faylga murojaatni cheklab qo'yishi mumkin.).

2. **O'qish/yozish.** Ko'pincha fayl bilan ma`lumot almashinuv berilganlar bloki ko'rinishida tashkil etilishi mumkin. Ushbu berilganlar bloki ikki xil xususiyatga ega. Bir tomondan ixtiyoriy hisoblash tizimi uchun berilganlar blokining o'lchami aniq berilgan, ya`ni bular apparat -dastur o'lchamlaridir. Ikkinchi tomondan bu berilganlar bloki real almashinuvda dasturchi tomonidan ixtiyoriy ravishda boshqarilishi mumkin. O'qish/yozish funksiyalarida ko'pincha almashinuv uchun berilganlar bloki o'lchami va o'qilishi yoki yozilishi kerak bo'lgan berilganlar bloki soni beriladi. Tanlangan berilganlar blokining o'lchamidan almashinuvlarning unumdorligi bog'liq, faraz qilaylik bir mashina uchun berilganlar blokining unumdorlik o'lchami 256 Kb bo'lsa, siz 128 Kb lik

almashinuvni amalga oshirmoqchi bo'lsangiz, u holda siz mantiqiy bloklarni o'qish uchun ikki marotaba 128 Kb dan murojaat kilasiz. Bu holda siz 256 Kb ni bir martada o'qish o'rniga, bir blokga ikki marotaba murojaat qilasiz va bir safar yarmini, keyingi safar keyingi yarmini o'qiysiz. Bu erda yana ba'zi bir «unimsizlik» elementlari uchrashi ham mumkin, lekin ularni «aqlli» operasion tizim tekislab yuboradi, agar tekislay olmasa, demak bu sizning xatoyingiz bo'ladi.

3. **Fayl ko'rsatkichini boshqarish.** Har bir ochilgan fayl bilan fayl ko'rsatkichi tushunchasi bog'liq. Bu ko'rsatkich komandalarning hisoblagich registri bo'lib, har bir vaqtda keyingi fayl bo'yicha almashinuvni amalga oshirish mumkin bo'lgan nisbiy manzilni ko'rsatadi. Ushbu blok bilan almashinuv tugagandan so'ng ko'rsatkich blokdan tashqariga ko'chiriladi. Fayl bilan ishni tashkil etish uchun ushbu fayl ko'rsatkichini boshqarish talab etiladi. Fayl ko'rsatkichini boshqarish funksiyasi mavjud bo'lib, ko'rsatkichni fayl bo'yicha ixtiyoriy (mumkin bo'lgan chegaralarda) ko'chirish imkonini beradi. Ko'rsatkich bu qandaydir o'zgaruvchi bo'lib, dasturdan murojat qilish mumkin, va u faylni ochish funksiyasi (ushbu o'zgaruvchini tashkil etuvchi) bilan bog'liq.

4. **Faylni yopish.** Bu amal ikkita funksiya orqali amalga oshirilishi mumkin:

1)Faylni yopish va oxirgi qiymatini saqlab qolish.

2)Faylni yo'qotib (o'chirib) tashlash.

Fayl yopilgandan so'ng u bilan barcha aloqalar tugatiladi va u kanonik holatga o'tadi.

Berilganlarni himoyalash

Ko'pgina strategik echimlar apparat darajasidagi kabi, operasion tizim darajasida ham takrorlanadi. Agar biz multidastur rejimini eslasak, uning mavjud bo'lishligining zaruriy shartlaridan biri bu xotira va berilganlarni himoyalashni ta`minlashdan iborat edi. Agar biz fayllar tizimini qarasak, u ham xuddi operasion tizim kabi bir foydalanuvchilik bo'lishi mumkin. Bu holatda berilganlarni himoyalash muammosi bo'lmaydi, chunki operasion tizimda ishlayotgan kishi barcha fayllarni egasi hisoblanadi. MS-DOS yoki Windows 95 bir foydalanuvchilik tizimlarga misol bo'ladi. Mashinani yuklab boshqa foydalanuvchilarning barcha disklardagi fayllarini o'chirib tashlash mumkin, chunki bu tizimlarda hech qanday himoyalash yo'q. Ko'p foydalanuvchilik tizim ko'pgina foydalanuvchilarning tiniq ishlashini ta`minlaydi. MS-DOS operasion tizimi ham multidastur rejimida ishlashi mumkin, lekin juda ham tiniq emas, bir jarayondagi xatolik operasion tizimning va qo'shni jarayonning o'chirilishiga olib kelishi mumkin. Xuddi shunday Windows 95 operasion tizimida ham ko'pfoydalanuvchi ishlashi mumkin, lekin uning ishi ham tiniq emas, chunki operasion tizim ularning barcha xuquqlarini himoyasini ta`minlamaydi. SHunday qilib, ko'pfoydalanuvchilik tizim tashqi ta`sirlardan himoya qilishi kerak. Aslida himoyalash muammosi faqatgina fayl tizimi bilan bog'liq emas. OT barcha sohalarda berilganlarni himoyalashni ta`minlaydi: bu fayllar, jarayonlar, jarayonlarga tegishli bir foydalanuvchi tomonidan qo'yilgan zahiralar. Bu erda men sizning e`tiboringizni shu faktga qarataman, chunki fayllar uchun bu juda muhim nuqta.

Fayllar tizimining asosiy xususiyatlari

Fayllar tizimi fayllar uchun sanab o'tilgan barcha xususiyatlarni o'z ichiga oladi, va yana ba'zi bir qo'shimcha xususiyatlarga ham ega. Bu xususiyatlar fayllar tizimining strukturalik tashkil etilishi bilan bog'liq.

Keling, qandaydir tashqi saqlash qurilmasi (VZU) fazosini qarab chiqaylik va bu fazoda fayllarni joylashtirishni tashkil etishni ko'rib chiqamiz.

Uzluksiz segmentli fayllarni birdarajali tashkil etish

«Birdarajali» termini tizim unikal nomlangan fayllar bilan ishlashni ta'minlashini anglatadi. Tashqi saqlash qurilmasi chegarasida berilganlarni saqlash uchun katalog deb ataluvchi soha ajratiladi. Katalog quyidagi jadvaldagi strukturaga ega:

1-jadval.

Ism	Boshlang'ich blok	Oxirgi blok

«Boshlang'ich blok» berilgan ism bilan boshlanuvchi tashqi saqlash qurilmasidagi nisbiy manzilni ko'rsatadi. «Oxirgi blok» berilgan faylning oxirgi blokini aniqlaydi. Faylni ochish funksiyasi katalogda fayl ismini topish, uning boshlanishi va oxirini aniqlashni amalga oshiradi. (amalda berilganlar ko'rsatilganidan kam joy egallashi mumkin, lekin bu haqida keyinroq to'xtalamiz). Bu xarakat juda oddiy, shu bilan katalogni OT xotirasida saqlash mumkin, bu esa almashinuvlarni kamayishiga olib keladi. Agar yangi fayl tashkil etilayotgan bo'lsa, u bo'sh joyga yoziladi. Ismlar katalogiga o'xshash bo'sh fazolar (fragmentlar) jadvali bo'lishi mumkin.

O'qish/yozish qo'shimcha almashinuvlarsiz amalga oshiriladi, chunki faylni ochishda biz berilganlarni joylashtirish diapazoniga ega bo'lamiz. O'qish ushbu struktura blokiga mos ravishda amalga oshiriladi va hech qanday qo'shimcha ma'lumot talab etilmayli, almashinuv ham mos ravishda tezda amalga oshiriladi.

Endi qarab chiqaylik, bunday faylga qo'shimcha ma'lumot yozmoqchimiz, lekin bo'sh fazo joy yo'q? Bu holda tizim ikki xil yo'l tutishi mumkin. Birinchidan, u sizga joy yo'qligini aytadi va siz o'zingiz nimadir qilishingiz kerak bo'ladi, ya'ni qandaydir Ushbu faylni biror joyga ko'chirib turadigan va kerakli ma'lumotni qo'shadigan jarayonni qo'yasiz. Bunday ko'chirish etarli darajada qimmatga tushadigan funksiya. Ikkinchi imkoniyat - almashinuvni rad etiladi. Bu esa faylni ochish jarayonida avvaldan qo'shimcha joy olib qo'yish kerakligini anglatadi; bu holda fayl tizimi buferning bo'sh o'lchamini tekshiradi va u kam bo'lsa, ushbu faylni joylashtirish uchun bo'sh joy qidiradi.

Bunday ko'ramizki, bunday tashkil etish sodda, almashinuvlarda unumli, lekin fayl uchun joy etishmagan hollarda unimsizlik boshlanadi. Bunday tashqari fayl tizimining uzoq ishlashi davomida diskda xuddi operativ xotiradagi kabi bo'sh joylar fazosi xolati yuz beradi. YA'ni bo'sh joylar mavjud, lekin faylimizni

joylashtirish uchun etarli joy yo'q bo'lgan holat yuzaga keladi. Fayl tizimini bunday tashkil etilishining fragmentasiyasi bilan kurashishda uzoq, og'ir va fayl tizimi uchun xavfli bo'lgan jarayon, ya`ni fayllarni bir-biriga zichlashtirish jarayoni amalga oshiriladi.

Bunday tashkil etish bir foydalanuvchilik fayl tizimi uchun qulay va foydalidir, chunki foydalanuvchilarning ko'pligi holatida bo'sh joylar fazosi yuz beradi. Zichlashtirish jarayonini har doim qo'yish maqsadga muvofiq emas. Boshqa tomondan tizim oddiy va hech qanday qo'shimcha xarajatlar talab qilmaydi.

Fayllar blokli tashkil etilgan fayllar tizimi

Tashqi saqlash qurilmalari fazosi bloklarga bo'lingan. Fayllar tizimida bunday ma`lumotlarni bo'laklashda operativ xotirani varaqli tashkil etishdagi jarayonlar ma`lumotlarini bo'laklash kabi amalga oshiriladi. Umumiy holda, har bir fayl ismi bilan shu fayl berilganlari joylashgan qurilma bloklari raqamlarini to'plami bog'liq. Ushbu bloklarni raqamlari ixtiyoriy tartibga ega, ya`ni bloklar qurilma bo'yicha ixtiyoriy tarqalgan. Bunday tashkil etishda bo'sh joylar fazosi muammosi yo'q, ammo blokni yaxlitlash yo'qotishlari mavjud (agar fayl blokni bitta baytini band qilgan bo'lsa, u holda butun blok band hisoblanadi). SHunday qilib, zichlashtirish muammosi yo'q, va bu tizimdan ko'pfoydalanuvchilik tashkil etishda foydalanish mumkin.

Bu holda har bir fayl bir qancha atributlar to'plami bilan bog'liq: fayl ismi va faylga murojaat etiladigan foydalanuvchi ismi; Bunday tashkil etilishi ismlarni unikalligi muammosidan qutilishga imkon beradi. Bunday tizimda ism unikalligi bir foydalanuvchilik fayllar o'rtasida talab etiladi.

Bunday fayllarni tashkil etish katalog orqali amalga oshiriladi. Katalogni strukturasi quyidagicha bo'ladi. Katalog qatorlardan tashkil topadi; har bir i-chi qator fayl tizimini i-chi blokga mos keladi. Bu qatorda blok band yoki bo'shligi haqidagi ma`lumot saqlanadi. Agar u band bo'lsa, u holda bu qatorda fayl ismi (yoki unga murojaat), foydalanuvchi ismi va boshqa qo'shimcha ma`lumotlar joylashishi mumkin.

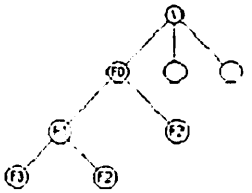
Ma`lumot almashinuv davrida tizim turlicha xarakatlanishi mumkin. YOki faylni ochishda tizim katalog bo'yicha aylanib faylni mantiqiy bloklarini diskda joylashish jadvalini quradi. YOki har bir almashinuvda bu moslik amalga oshiriladi.

Fayllar tizimini bunday tashkil etilishi bir foydalanuvchi ramkasida birdarajali hisoblanadi, ya`ni barcha fayllar qandaydir foydalanuvchiga tegishli guruhga bog'langan.

Ierarxik fayllar tizimi

Fayl tizimining barcha fayllari daraxt (3-rasm) deb atalgan bir strukturaga qurilgan. Daraxtning ildizida fayl tizimining ildizi joylashgan. Daraxtni bog'langan joyi varaq hisoblansa, bu fayl foydalanuvchining berilganlaridan tashkil topib fayl-

katalog hisoblanadi. Daraxtning varaqdan farqli bog'lanishlari fayl-kataloglar hisoblanadi. Bunday ierarxik fayl tizimida nomlanish turli usullar bilan amalga oshadi.



3-rasm. Fayl tizimining daraxt ko'rinishi

Birinchi tur – faylni eng yaqin katalogga nisbatan nomlash, ya`ni biz F0 katalog uchun yaqin bo'lgan fayllarni qarasak, bu F1 fayl bo'lib, u ham katalogdir va F2 fayl. Bunday tizimda bir darajada ismlar takrorlanmasligi maqsadga muvofiq. Boshqa tomondan, barcha fayllar daraxt bilan bog'langanliklari uchun biz faylni to'liq nomi, ya`ni fayl tizimi ildizidan aniq bir faylgacha bo'lgan yo'l, haqida gapira olamiz. F3 faylning to'liq ismi quyidagicha belgilanadi: /F0/F1/F3. Bunday tashkil etish faylning qisqa ismi bilan ham, to'liq ismi bilan ham ishlash imkonini beradi. Fayllarning to'liq ismi bu yo'ldir. Ixtiyoriy daraxtda uning ildizidan ixtiyoriy bog'lamigacha bitta yo'l mavjud, shunday qilib ismlarni unifikasiya qilish muammosi hal etiladi. Birinchi marta bunday usul Berkli universitetida 60-yillarning oxirida ishlab chiqilgan Multix operasion tizimida foydalanilgan. Keyinchalik bu chiroyli echim ko'pgina operasion tizimlarda qo'llanila boshladi. Bu ierarxiyaga mos ravishda har bir faylga qandaydir murojaat xuquqiga ega atributlarni bog'lab qo'yish mumkin. Murojaat xuquqlariga foydalanuvchilar fayllari bilan birgalikda kataloglar ham egadirlar. Bu tizimning strukturasi ko'pfoydalanuvchilik ishni tashkil etishda, ismlar muammosini yo'qligi hisobiga unumdordir.

Operasion tizimda shaxsiylashtirish va berilganlarni himoyalash

Bu nyuans, ham sodda, ham murakkab. Soddaligi shundaki biz u haqida bir necha og'iz gapiramiz xolos, murakkabligi shundaki, shunday muammolar mavjudki ular haqida uzoq gapirish mumkin.

SHaxsiylashtirish – bu aniq foydalanuvchini identifikasiyalash va shu bilan mos ravishda berilganlarni himoyalash bo'yicha u yoki bu xarakatlarni qabul qilish imkoniyatidir.

Agar biz ixtiyoriy MS-DOS operasion tizimini qarasak, u bir foydalanuvchilik.

Operasion tizimlarning ikkinchi darajasi – foydalanuvchilarni ro'yxatdan o'tkazadigan, lekin barcha foydalanuvchilar yagona sub`ekt to'plami ko'rinishida va bir-biri bilan bog'liq emaslar. Bunday operasion tizimlarga misol sifatida IBM firmasining mainframe-kompyuterlari uchun operasion tizimlarini ko'rsatish

mumkin. Misol uchun ma`ruzachi o`zining eshituvchi talabalarning qaysi biri qanday guruxga tegishli ekanligini bilmaydi, lekin ularning shu kurs talabalari ekanligini biladi. Bu ham yaxshi, ham yomon. Bu kursni eshitish uchun yaxshi, lekin ma`ruzachi tomonidan savol javob qilish masalasida yomon, chunki bir kun ichida u hamma talabalar bilan savol-javob qilishga ulgura olmaydi. SHuning uchun u barcha eshituvchilarni qandaydir bo'laklashi kerak, lekin qanday bu noaniq.

Mos ravishda bunday bir o'lchamli shaxsiylashtirishda barcha biz aytib o'tgan funksiyalar (xususan, himoyalash) ta`minlanadi, lekin foydalanuvchilarni bunday tashkil etish foydalanuvchilar guruhini anglatmaydi. Menga esa bizning fakultet serverida mening laboratoriyam ajratilsa va bu laboratoriya ramkasida fayllarga murojaat xuquqini bir biriga berish imkoni berilsa maqsadga muvofiq bo'lar edi.

Mos ravishda fayllar tizimidagi kabi foydalanuvchilarning ierarxik tashkil etish paydo bo'ladi. YA`ni bizda «barcha foydalanuvchilar» va «foydalanuvchilar guruhi» degan tushuncha mavjud. Guruhda real foydalanuvchilar mavjud. Bunday shaxsiylashtirishni ierarxik tashkil etish quyidagilarni keltirib chiqaradi. Qandaydir foydalanuvchini ro'yxatdan o'tkazish uchun uni avval qandaydir guruhga kiritish kerak, - bu laboratoriya, kafedra yoki o'quv bo'limi bo'lishi mumkin. Foydalanuvchilar guruhlarga birlashganliklari uchun, foydalanuvchilarning zahiralariga murojaat xuquqini bo'linish imkoniyati yuzaga keladi. YA`ni, masalan foydalanuvchi uning zahiralaridan barcha guruhdagi foydalanuvchilar foydalanishlari mumkin ekanligini e`lon qilishi mumkin. Bunday chizma ko'pdarajali (guruhlar guruhchalarga bo'linadilar va x.k.) mos xuquq va imkoniyatlardan kelib chiqqan holda bo'lishi mumkin. Hozirgi kunda shunday operasion tizimlar yaratilmoqdaki, ularda murojaat xuquqi faqatgina ierarxik strukturaga bog'liq bo'lib qolmay, balki murakkabroqdir, ya`ni murojaat xuquqini ierarxiyani buzgan holda qo'shish mumkin.

Sinov savollari

1.Hisoblash tizimini qanday bo'laklar tashkil etadi?

2.Amaliy dasturiy ta`minotning vazifasi nimalardan iborat?

3.Tizimli dasturiy ta`minotning vazifasi haqida tushuncha bering.

4.Dastur interfeysi nima uchun kerak?

5.Foydalanuvchi interfeysi nima uchun kerak?

6.Dastur qobiqlari nima?

7.Utilitalar nima?

8.Tizimli qayta ishlovchi dasturlarning asosiy funksiyalari va tarkibi haqida ma`lumot bering.

9.Operasion tizim nima?

10.Operasion tizimlarni ishlab chiqarishdagi uch echish yo'nalishlari haqida ma`lumot bering.

11.Operasion tizimlarning rivojlanish yo'nalishlari qanday sabablarga bog'liq?

12.Jarayon tushunchasi haqida ma`lumot bering.

13.Jarayonlarni boshqarishdagi muammolar nimalardan iborat?

14.Markaziy prosessor vaqtidan foydalanishni boshqarish deganda nimalarni tushunasiz?

15.Kiritish buferi va real xotirani boshqa masalalar uchun bo'shatishni boshqarish jarayoni haqida so'zlab bering.

16.Bo'laklangan zahiralarni boshqarish jarayoni haqida so'zlab bering.

17.Fayl nima?

18.Fayllarning asosiy xususiyatlari haqida ma`lumot bering.

19.Fayl ko'rsatkichi nima?

20.Fayllar tizimi nima?

21.Fayllar tizimining asosiy xususiyatlari haqida ma`lumot bering.

22.Uzluksiz segmentli fayllarni birdarajali tashkil etishning yutuq va kamchiliklari haqida ma`lumot bering.

23.Fayllar blokli tashkil etilgan fayllar tizimining yutuq va kamchiliklari haqida ma`lumot bering.

2.BO'LIM

Berilganlar strukturasi va ularni tasvirlash va ulardan foydalanish usullari

2.1.Berilganlar strukturasi tushunchasi va ularni umumiy ifodalash

Hisoblash jarayoni berilganlar dasturi yordamida amalga oshiriladi, dasturning o'zi esa spesifik tabiatga ega berilganlarni ifodalaydi. Berilganlar ikki xil darajada ifodalanadi.

1.mantiqiy darajada;

2.fizik darajada (mashina darajasida), berilganlar fizik xotirada ifodalanadi.

Ikkala darajada ham berilganlarga aniq ta`rif berish mumkin.

Mantiqiy darajada berilganlar qandaydir informasion ob`ektlardan tashkil topgan bo'lib, ular qandaydir faktlarni qandaydir algoritm yordamida qayta ishlash maqsadida formal ko'rinishda ifodalashga xizmat qiladilar.

Fizik darajada murakkabligi va mazmunidan qatiy nazar berilganlar ikkilik razryadlari ketma-ketligi ko'rinishida ifodalanadi. Bu darajada berilganlar o'zining ichki tabiatiga ega emas, ya`ni ular strukturalanmagan.

Birgalikda qaraladigan va qayta ishlanadigan elementar berilganlarning to'plami **berilganlar strukturasi** deb ataladi.

Umumiy holda berilganlar strukturasi – **berilganlar elementlari** o'rtasida mavjud munosabatlarni ifodalovchi qoida va chegaralar to'plamidir.

Bir berilganlar elementining boshqa berilganlar elementi barcha aloqalari qandaydir berilganlar strukturasida **«munosabat» elementini** tashkil etadi.

Berilganlar strukturasi bir-biri bilan o'zarobog'langan berilganlar elementlaridan tashkil topadi. Strukturaning har bir elementi umumiy holda ikki bo'lakdan tashkil topadi:

1.Berilganlar elementi;

2.Munosabat elementi.

O'z navbatida berilganlar elementi qandaydir turdagi **bir qiymatli berilganlar** bo'lishi mumkin, yoki bir turdagi qandaydir **berilganlar to'plami** bo'lishi mumkin.

Berilganlar elementi turlari tushunchasiga quyidagilar kiradi:

1.mumkin bo'lgan qiymatlar diapazoni (sohasi);

2.mumkin bo'lgan amallar to'plami;

3.mashinada ifodalash usuli.

Berilganlar strukturasi elementlari orasidagi o'zaro aloqa graf ko'rinishida ifodalanishi mumkin, bu erda cho'qqilar berilganlarning alohida elementlari, qobiqlar esa ular orasidagi aloqalar.

Mantiqiy darajada berilganlar strukturasi **mantiqiy (abstrakt) struktura** deb ataladi, chunki berilganlar xotirada qanday joylashganligiga bog'liq emas.

Fizik darajada berilganlar strukturasi **fizik (saqlash strukturasi) deb ataladi.**

Mantiqiy va fizik berilganlar strukturasi o'rtasida juda katta farq mavjud, masalan, dasturchi nuqtai nazaridan, to'plam (ikki o'lchamli) jadval ko'rinishiga

ega bo'lib, elektron hisoblash mashinasi (EHM) xotirasida berilganlar elementlarining chiziqli ketma-ketligini ifodalaydi.

Berilganlarni qayta ishlash tizimida har doim maxsus proseduralar ishlab chiqiladi va ular mantiqiy daraja berilganlarini fizik daraja berilganlariga aylantiradilar. Bu proseduralarning bosh maqsadi – mantiqiy darajadan kelib chiqqan holda, fizik darajadagi berilganlarga murojaatni ta`minlash. Masalan, ikki o'lchamli to'plam uchun shunday aks ettirish funksiyasi mavjud bo'lishi kerakki, u to'plam elementining jadvaldagi koordinatasidan kelib chiqqan holda aniq manzilini olish imkonini bersin.

$A_{ij} = A_0 + a(i,j)$; $a(i,j)$ – функция

Mantiqiy va fizik ko'rinishdan tashqari ixtiyoriy struktura mantiqiy va fizik strukturalar amallari to'plami bilan xarakterlanadi. Fizik strukturalar ustidagi amallar to'plami qanday xotirada amalga oshirilayotganiga ko'ra aniqlanadi: operativ xotiradami yoki tashqi xotiradami ?

Operativ xotiradagi holatning xususiyati shundaki, u xotira yacheykalarining tartiblangan ketma-ketligidan tashkil topgan bo'lib, xotiraning ixtiyoriy yacheykasiga to'g'ridan-to'g'ri murojaat etish imkoniyatiga ega, ketma-ket raqamlangandir.

Tashqi xotirada esa berilganlar strukturasi turli turda tashkil etilgan hisoblanadi (ketma-ket, to'g'ridan -to'g'ri, indeksli-ketma-ket).

2.2.Berilganlar strukturasining sinflanishi

Berilganlar strukturasi quyidagi to'rt belgi bo'yicha sinflanadi:

1.Struktura elementlari orasidagi aloqalarning borligi yoki yo'qligi. Bu belgi bo'yicha ikki ko'rinish ajratiladi:

- bog'langan strukturalar;
- bog'lanmagan strukturalar.

Bog'langan strukturalarga vektorlar, to'plamlar, qatorlar kiradi.

Bog'lanmaganlariga barcha mumkin bo'lgan ro'yxatlar kiradi.

2.Strukturaning o'zgaruvchanligi nuqtai –nazaridan ajratiladi:

- statik. Statik – strukturalarda elementlar soni va elementlar o'rtasidagi aloqalar soni o'zgarmaydi (qatorlar, vektorlar, to'plamlar);
- yarimstatik. YArimstatik – strukturaning ba`zi bo'laklari o'zgarishi mumkin (steklar);
- dinamik. Dinamik – strukturalarda elementlar soni va elementlar o'rtasidani aloqalar soni ixtiyoriy vaqtda o'zgarishi mumkin (ro'yxatlar).

3.Tartiblanish darajasi bo'yicha farqlanadi:

- chiziqli-tartiblangan – bir element keyingisi ketidan qa`tiy keladi;
- chiziqli-tartiblanmagan – chiziqsiz.

4.Elementlarning xotirada bir biriga nisbatan o'zaro joylashish xarakteri bo'yicha farqlanadi:

- elementlari ketma-ket joylashgan strukturalar (vektorlar, to'plamlar, qatorlar)
- elementlari ixtiyoriy joylashgan strukturalar. Tartibi ixtiyoriy, lekin elementlari bir biri bilan bog'langan.

2.3.Berilganlarning oddiy statik strukturalari

Ularga vektorlar, to'plamlar, yozuvlar, jadvallar kiradi.

1. Vektor – bir turdagi berilganlarning tartiblangan to'plamidir.

Vektorga bir o'lchamli to'plam mos keladi. Uning elementlari xotirada qat'i biri-biridan keyin joylashadi. Bunday tartiblanish elementlarni raqamlash imkoniyatini beradi (raqamlar-indekslar).

Eng ko'p qo'llaniladigan amallardan biri – berilgan indeks bo'yicha elementga murojaat amalidir.

Vektorlar ustidagi yana boshqa amallardan vektorning pastki va yuqorigi chegaralarini aniqlash, vektor elementi turini aniqlash kabilardan foydalaniladi.

Vektorning fizik ko'rinishi xotiraning bir xil o'lchamdagi bo'laklari (maydon, slotlar) ketma-ketligi ko'rinishida amalga oshiriladi. Har bir maydonda bitta element saqlanadi. Bu slotlar xotirada element indekslari oshib borishi tartibida joylashadi.

Vektorning fizik strukturasi xotirada joylashgan va quyidagi tabiatga ega diskreptor bilan kuzatiladi. Deskriptor quyidagi maydonlardan tashkil topadi:

- berilganlar strukturasining ichki kodi
- vektorning ismi
- birinchi elementning manzili
- indeksning pastki qiymati
- indeksning yuqori qiymati
- elementning toifasi
- slot o'lchami

Birinchi maydon diskreptor bilan bog'langan berilganlar strukturasini aniqlashga imkon beradi.

Manzil funksiyasi (aks ettirish funksiyasi) vektor xolatida quyidagi ko'rinishga ega bo'ladi: $adr(i) = Adr(p) + (i - p) * I$

Boshlang'ich berilganlar:

- indeksning minimal qiymati: p
- i manzil qidirilayotgan indeks
- maksimal qiymat: q
- slotning o'lchami: I

2. To'plam, umumiy holda - bu har bir elementi vektor bo'lishi mumkin bo'lgan berilganlar strukturasidir.

Ikki o'lchamli, uch o'lchamli, n-o'lchamli to'plamlar bo'lishi mumkin. To'plamning har bir elementini aniqlash uchun indeksdan foydalaniladi (har bir koordinata bo'yicha): $A(j_1,...,j_n)$.

To'plamning elementiga murojaat qilish uchun n indeksdan foydalaniladi

To'plamlar bilan ishlaganda eng ko'p foydalaniladigan amallardan biri manzil funksiyasini amalga oshirishni talab etadigan murojaat amalidir. Bu amal vektordan farqli ravishda netrivial bajariladi.

Misol qaraymiz.

Faraz qilaylik n-o'lchamli to'plam berilgan bo'lsin:

$B(i_1K_1,...,i_nK_n)$

$B(i_1,i_2...,i_n)$ – boshlang'ich element

$B(j_1,...,j_n)$ – xotirada manzilini topish kerak bo'lgan ixtiyoriy element.

Manzil funksiyasi quyidagicha hisoblanadi:

$$Adr(B(j_1,...,j_n)) = Adr(B(i_1,i_2,...,i_n)) + (\sum(j_m * D_m)) * L - (\sum(i_m * D_m)) * L,$$

bu erda:

L – slot o'lchami

D – to'plamni xotirada (qatorlar yoki ustunlar bo'yicha) joylashishiga qarab aniqlanadi. Agar qatorlar bo'yicha joylashgan bo'lsa, u holda avval birinchi qator joylashadi va x.k., agar ustunlar bo'yicha joylashsa- avval birinchi ustun joylashadi va x.k.

Elementlar qator bo'yicha joylashgan hol uchun, u holda:

$D_m = (K_{m+1} - i_{m+1} + 1) * D_{m+1}$

$m = n-1,...,1$

$D_n = 1$

YAna shu kabi uchrab turadi:

- simmetrik kvadrat matrisa (elementlari bosh diogonalga nisbatan simmetrik va teng)
- kesilgan matrisa (razrejennaya matrisa)- (ko'pgina elementlari 0 ga teng). Bunday matrisalar ro'yxat ko'rinishida saqlanadi.

3. YOzuv – umumiy holda turli turdagi tartiblangan elementlarning to'plamidir (maydonlar deb ataladi).

YOzuvlar oddiy va bir-biriga kirishgan bo'lishi mumkin (maydon o'rniga boshqa yozuv joylashishi mumkin). Biri-biriga kirishish darajasi yozuvlar deskriptorida aks etadi.

4. Jadval – bir turdagi yozuvlarning tartiblangan ketma-ketligidir (yozuvlar to'plami).

Jadvallar tizimning ichki ifodalanishida keng qo'llaniladi (jadvallar xizmati, amallar kodlari jadvallari va x.k.).

Jadval yozuvlardan tashkil topadi, yozuv esa maydonlardan.

YOzuvlarning bir turdalilik xususiyati barcha bir ismli maydonlar bir xil turga ega ekanligini anglatadi.

Har bir yozuvda jadval chegarasida qidiruvni osonlashtirish uchun maydonlardan biri kalitli hisoblanadi (barcha yozuvlar kalitli maydonning turli qiymatlariga ega). YOzuv xotira yacheykasining chiziqli ketma-ketligi ko'rinishida amalga oshiriladi. Xotira sohasi o'lchovi yozuvlar soni va har bir yozuv registri bilan aniqlanadi.

Jadvallar bilan ishlashdagi mavjud tipik amallar:

- yozuvni qidirish (barcha yozuv qidiriladi);
- elementni qo'shish;
- elementni o'chirish;
- tartiblash.

Jadvallar bilan ishlashda ushbu jadvallar bilan amallar bajarilish tezligi katta ahamiyatga ega.

Kichikrok jadvallar bo'lgan holida muammolar yuzaga kelmaydi, katta jadvallar bilan ishlashda esa qidiruvning turli usullaridan foydalaniladi.

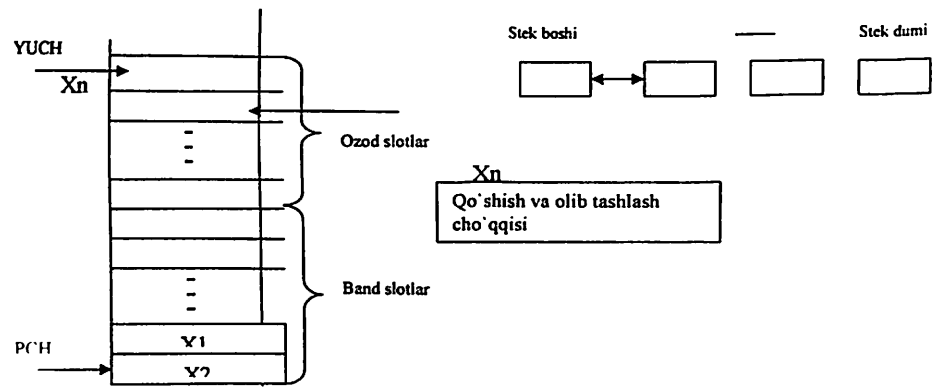
2.4.Ro`yxat strukturalari

CHiziqli ro'yxat deb berilgan elementlarlarning chiziqli tartiblangan ketma-ketligiga aytiladi. Ularni $X_1, X_2, ..., X_n$ deb nomlash mumkin.

Ketma-ketlik indekslariga va xotiradagi navbatiga qarab aniq va noaniq o`rnatilishi mumkin. Ko'rsatkichlar elementlar bilan aloqani yoki oldindan yoki orqadan o'rnatishlari mumkin.

Agar bunday struktura bilan ishlashda **n** (elementlar) soni o'zgarmasa, u holda ularni statik berilganlar strukturasiga kiritish mumkin. Agar **n** elementlar soni o'zgarsa, va aloqa strukturasi noaniq bo'lsa, - u holda bu yarimstatik berilganlar strukturasidir; **n** o'zgaradi va aloqa strukturasi aniq bo'lsa, - bu dinamik berilganlar strukturasidir.

Steklar (4-rasm) – o'zgaruvchan o'lchamli ketma-ket ro'yxatlar bo'lib, ularda elementlarni qo'shish va olib tashlash faqatgina boshidan yoki stek chuqqisidan amalga oshiriladi.



4-rasm. Stekning tuzilishi

Stekning boshqacha nomlanishi – magazin xotirasidir. LIFO strukturasi.

Stek bilan amallar:

- elementni olib tashlash;
- elementni qo'shish.

Manzili yacheykada saqlanadigan ko'rsatkich yordamida bosh manzilni manzili ko'rsatiladi.

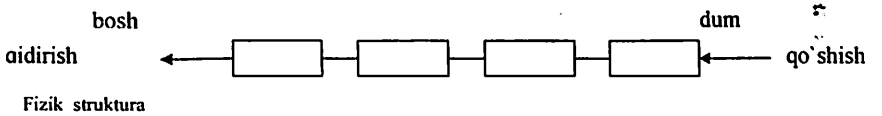
Elementni olib tashlashda Xn yacheykining qiymati o'qiladi, stekning boshi pastga keyingi band elementga tomon xarakatlanadi. Jarayon stekning boshi X1 dan past pozisiyani egallamagunicha davom etadi.

Elementni qo'shishda stekni cho'qqisi bir pozisiya yuqoriga xarakatlanadi. SHunday qilib, boshni ko'rsatkichi qaerda bo'lsa, o'sha erga yoziladi. Stekni yuqori chegarasi 0 bo'lganida jarayon chegaralangan.

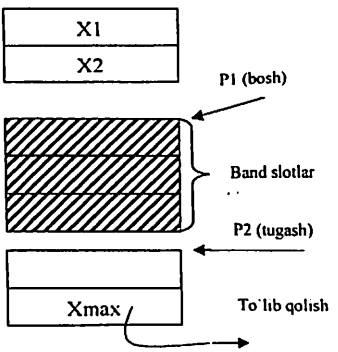
Stekni deskriptor quyidagi elementlarni o'z ichiga oladi:

- strukturaning kodi;
- ismi;
- yuqori chegara manzili (YUCH);
- pastki chegara manzili (PCH);
- boshni ko'rsatkichi;
- elementni ifodalash (tur, o`lchov) .

Navbat (5-rasm) – elementlarni qo'shish va olib tashlash turli pozisiyalardan bajariluvchi o'zgaruvchan o'lchovli ketma-ket ro'yxat. Ikkita ko'rsatkichdan foydalaniladi (boshiga va dumiga).



Fizik struktura



5-rasm. Navbatning tuzilishi

Har bir element uchun – bitta slot.

Elementlarni turi bir xil.

R2 – dumdan keyingi birinchi ozod slotning manzili.

Amallar:

- olib tashlash;
- qo'shish.

YAngi element R2 ga yoziladi, keyin R2 kupayadi.

Olib tashlashda: element olib tashlangandan so'ng R1 keyingi elementga suriladi.

Navbatning to'lib qolishi: bu xolat echimsiz bo'lib qolmasligi uchun, R2 Xmax qiymatdan oshib ketsa, R2 X1 ga agar u bo'sh bo'lsa o'tkaziladi.

Navbat deskriptorining elementlari:

- struktura kodi;
- ismi;
- yuqori chegara manzili;
- pastki chegara manzili;
- bo'sh o'rinni ko'rsatkichi;
- elementlarni ifodalovchi.

Navbatlardan EHM zahiralarini ifodalashda foydalaniladi.

Dek – elementlarni qo'shish va olib tashlash uning ixtiyoriy tomonidan amalga oshirilishi mumkin bo'lgan navbatdir.

Deklarni turlari:

- kirishlari cheklangan dek – qo'shish faqat bir tomondan, olib tashlash esa ikki tomondan amalga oshiriladi;
- chiqishlari cheklangan dek – qo'shish ikki tomondan, olib tashlash esa bir tomondan amalga oshiriladi.

Dekning mantiqiy va fizik strukturasi navbatnikiga o'xshash bo'lib amallari bir xil, amallarning bitta parametri – bu tugash belgisi bo'lib, elementni qo'shish va olib tashlash shu belgidan bajariladi.

2.5.CHiziqli -dinamik strukturalar
Birbog'lamli chiziqli ro'yxatlar

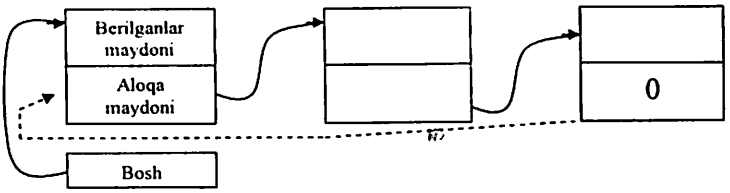
CHiziqli -dinamik strukturalar strukturasining elementlari sonining turg'un emasligi, elementlarni tarqatishdagi fizik bir-biriga o'tishning mavjud emasligi bilan xarakterlanadilar. Bu ro'yxatlarda elementlarning ketma-ketligi bir yoki bir necha ko'rsatkich yordamida aniq beriladi. SHunga ko'ra murojaat qilish murakkabroqdir.

Bog'langan ro'yxat (6-rasm) – elementlari bir xil formatga ega yozuvlardan tashkil topgan berilganlar strukturasidir. Bu elementlar ularda joylashgan ko'rsatkichlar yordamida bog'lanadi. Bunday ro'yxatning elementlari oddiy holda – bu ikki maydon:

- ma`lumotli;

- aloqa maydoni.

Mantiqiy struktura



6-rasm. Bog'langan ro'yxatning tuzilishi

Bog'langan ro'yxatlardan eng soddasi – bu birbog'lamli chiziqli ro'yxatlardir CHiziqliligi shundaki, har bir element uchun yagona avvalgi va yagona keyingi element mavjud (birinchi va oxirgi elementlardan tashqari).

Berilganlar maydoni ixtiyoriy murakkablikdagi strukturadan tuzilgan bo'lishi mumkin. Zanjirning oxirgi elementida, bog'liqlik maydonida, tugatish belgisi (0,Null) saqlanadi. Bosh birinchi elementga ko'rsatadi.

Fizik struktura:

- deskriptor;
- yozuvlar uchun xotira sohasi to'plamidan tashkil topadi.

YOzuvlar soni noaniq. Bu esa deskriptordagi shunday elementlar tarkibini aniqlaydi:

- struktura kodi;
- ismi;
- boshlanish manzili;
- joriy o'lchami;
- alohida element ifodalovchisi.

Amallar: ro'yxatni boshidan oxirigacha qarab chiqish (elementni qo'shish va olib tashlash uchun). Kerakli element qiymatiga ko'ra qidiriladi. Qarab chiqishni osonlashtirish uchun, ro'yxatdan aylana ro'yxat tashkil etiladi, bu erda Null belgisi o'rniga birinchi elementni manzili joylashadi. O'shanda qarab chiqishni ixtiyoriy elementdan boshlash mumkin bo'ladi.

CHiziqli ikki bog'lamli ro'yxatlar.

Keyingi va avvalgi elementni manzilini saqlovchi ikki ko'rsatkichdan foydalanilmokda (7-rasm). To'g'ri ko'rsatkich (keyingi elementga) va teskari ko'rsatkich (avvalgi elementga).

Qulayligi: ixtiyoriy yo'nalishda xarakatlanish mumkinligi.

Amallar:

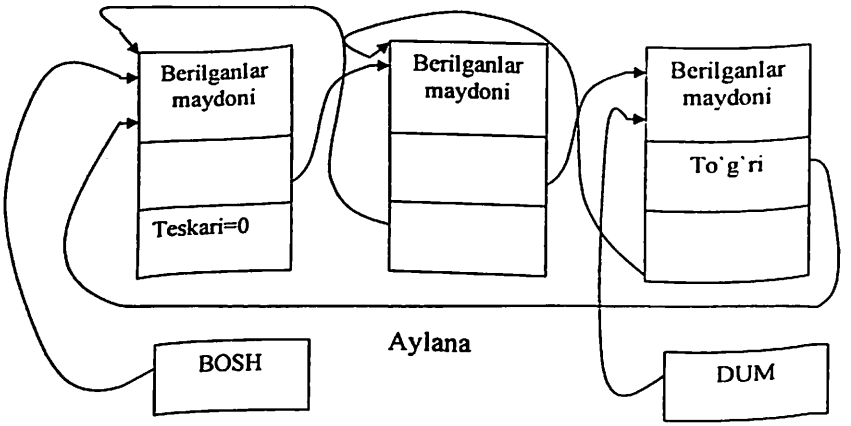
- elementni qo'shish
- elementni olib tashlash

Barcha elementlarni ikki guruhga bo'lish mumkin:

1 – qayta ishlanuvchi (funksional ro'yxat);

2 – bo'sh (bo'sh bloklar ro'yxati).

Ularning formatlari mos tushadi.



7-rasm. Chiziqli ikki bog'lamli ro'yxatlar

Boshlang'ich vaqtda funksional ro'yxat bo'sh, har bir ro'yxat uchun xotirada joy ajratilgan. Bo'sh ro'yxat maksimal sohani band etadi. Ish jarayonida bo'sh ro'yxatdan funksional ro'yxatga elementlar uzatish amalga oshiriladi. Ro'yxatlar strukturasi bir turli emas.

Keyingi misollar uchun berilganlarning belgilanishini kiritamiz:

- FREE – yacheykadagi bo'sh ro'yxatning boshlanishiga ko'rsatkich.
- NEW – qo'shish mumkin bo'lgan yangi element.
- START – funksional ro'yxatning boshlanish manzili.
- DATA – ma`lumotlar maydoni.
- LINK – aloqa maydoni.

2.6.CHiziqsiz bog'langan strukturalar (Daraxt ko'rinishli va tarmoqli)

Ko'pbog'lamli ro'yxat– bu ro'yxatning har bir elementi boshqa elementlari bilan ixtiyoriy aloqalar soniga ega bo'lishi mumkin.

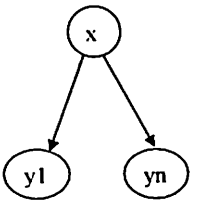
Daraxt – bu shunday strukturaki u quyidagilar bilan xarakterlanadi:

- yagona element bo'lib unga boshqa hech qanday boshqa elementdan ko'rsatkich yo'q.

• ildizdan boshlab, ko'rsatkichlar zanjiri bo'ylab ixtiyoriy elementga murojaat qilish mumkin.

• Ixtiyoriy elementga (ildizdan tashqari) yagona ko'rsatkich mavjud.

Bunday daraxt ildizli daraxt (8-rasm) deb ataladi. Aloqalar esa shoxchalar yoki qobiqlar deb ataladi. O'zidan hech qanday aloqa chiqmayotgan bog'lamlar barglar deb ataladi. Ularning o'rtasida oraliq bog'lamlar joylashadi.



8-rasm. Ildizli daraxt

Agar juda bo'lmaganda ikki cho'qqi mavjud bo'lsa: x va y, u holda x bog'lam onao'zak (roditelьskiy) deb ataladi, u – o'zakka tegishli, agar u bog'lamlar bir nechta bo'lsalar, u holda ularning barchasi x ning avlodlari hisoblanadilar, o'zaro ular aka yoki singillar hisoblanadilar. Agar qandaydir onao'zakdan bir qancha avlodlar soni chiqayotgan bo'lsa, u holda bu avlodlar soni chiqish darajasi deb ataladi.

Ko'pincha qaraladigan daraxtlar tartiblangan hisoblanadilar (9-rasm).

Daraxtning tartiblanganligi bu avlod-bog'lamlar o'rtasidagi tartiblanishni anglatadi, bu esa chapdan o'ngga tomon avlodlar o'rtasida kattalik munosabati mavjud: chapdan- katta, o'ngdan – kichik. Agar har bir

bog'lamning chiqish darajasi m dan oshmasa, u holda daraxt m-lik daraxt deb ataladi.

m-lik daraxtlar uchun: agar chiqish darajasi yoki m, yoki 0 bo'lsa, u holda bu to'liq m-lik daraxtdir.

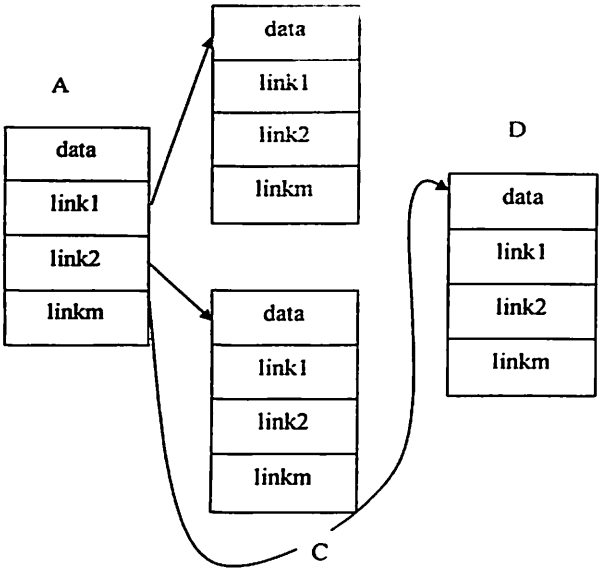
Har bir daraxt balandlik bilan xarakterlanadi – bu daraxtning ildizidan ixtiyoriy bargigacha bo'lgan yo'lning maksimal uzunligidir (qobiqlarda o'lchanadi)

Daraxt ko'pbog'lamli ro'yxat ko'rinishida ifodalanadi, agar daraxt m-lik bo'lsa, u holda ko'rsatkichlar soni = m.

Har bir bog'lam uchun xotirada quyidagi ko'rinishdagi strukturadan foydalaniladi :

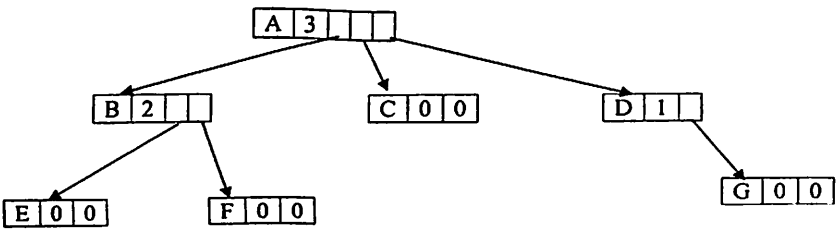
data	chiqish daraja	link1	link2	...	linkm

Misol qaraymiz (10-rasm):

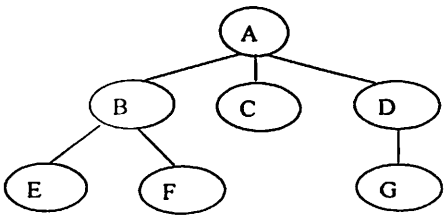


Daraxt ildizi bo'lib A bog'lam hisoblanadi. To'la to'kislik nuqtai nazaridan har bir bog'lamda 5 slotdan foydalanish kerak bo'ladi, lekin bunga zaruriyat yuk. Nullar oxirgi bog'lamni anglatadi. Birinchi nul – chiqish darajasi, ikkinchi nul – zanjir oxiri.

9-rasm. Tartiblangan daraxtlar



10-rasm. Daraxt ko'pbog'lamli ro'yxat ko'rinishi



11-rasm. Binar daraxtlar

Binar daraxtlar (11-rasm) nam mavjudki, ularda chiqish darajasi <=2. Agar ikkilik daraxtda chiqish darajasi =2 yoki 0 bo'lsa, u holda bu to'liq daraxtdir. Ularga qiziqish juda katta, chunki mashina xotirasida har bir element uchun ikki ko'rsatkich saqlash zarur: birinchisi- chapdan aloqa, ikkinchisi – o'ngdan aloqa.

Ixtiyoriy binar daraxtni ikki bog'lamli ro'yxat sifatida qurish mumkin.

Bu daraxt to'liq hisoblanmaydi. Ixtiyoriy m-lik daraxt o'ziga ekvivalent binar daraxtga almashtirilishi mumkin.

Binar daraxt ustida bajariladigan asosiy amallar quyidagilardir:

- 1. bog'lamlarni aylanish amali
- 2. qo'shish
- 3. olib tashlash

Bog'lamlarni aylanish amali boshqa amallarni tarkibiy qismi sifatida qidiruvni bajarishda foydalaniladi.

Qo'shish amali parametrlar sifatida: qo'shilayotgan daraxtcha, nisbatan qo'shimcha kiritilayotgan kirish daraxtining bog'lami, qo'shilayotgan daraxtchaga belgilangan raqam.

Olib tashlash amali esa parametr sifatida olib tashlanayotgan daraxtcha uchun ildiz rolini o'ynaydigan kirish daraxtining bog'lamini talab qiladi.

Sinov savollari

1.Berilganlar strukturasi deganda nima tushuniladi?
2.Berilganlarni mantiqiy darajada ifodalash tushunchasi nimani anglatadi?
3.Berilganlarni fizik darajada ifodalash nimani anglatadi?
4.Berilganlar elementlari nima?
5.Berilganlar strukturasining sinflanishi haqida ma`lumot bering.
6.Bog'langan strukturalar nima?
7.Bog'lanmagan strukturalar nima?
8.Statik, yarimstatik va dinamik strukturalar haqida ma`lumot bering.
9.Berilganlarning oddiy statik strukturalari haqida ma`lumot bering.
10.Ro'yxat strukturalar haqida ma`lumot bering.
11.CHiziqli ro'yxat nima?
12.Stek nima?
13.Navbat nima?
14.Dek nima?
15.CHiziqli -dinamik strukturalar haqida ma`lumot bering.
16.Birbog'lamli va ikki bog'lamli ro'yxatlarning farqi nimalarda ko'rinadi?
17.Bog'langan ro'yxat nima?
18.Deskriptor nima?
19.CHiziqli ikki bog'lamli ro'yxatlar nima?
20.CHiziqli bog'langan strukturalar haqida ma`lumot bering.
21.Ko'pbog'lamli ro'yxat nima?
22.Daraxt nima?
23.Daraxt ildizi haqida ma`lumot bering.
24.Binar daraxt nima?
25.Binar daraxt ustida bajariladigan asosiy amallar haqida ma`lumot bering.

3.BO'LIM

Formal til va formal grammatika tushunchasi

Belgilar zanjiri va tillarni berilish usullari. Belgilar zanjirlari ustida bajariladigan amallar. Til tushunchasi. Tillarni berilish usullari. Tilning sintaksis va semantikasi. Dasturlash tillarining xususiyatlari

3.1.Belgilar zanjiri va ular ustida bajariladigan amallar

Belgilar zanjiri – bu biri biridan keyin yozilgan belgilarning ixtiyoriy ketma-ketligidir.

Belgi (yoki harf) tushunchasi formal tillar nazariyasida baza elementi hisoblanadi va shuning uchun ta`rifga muhtoj emas.

Demak, zanjir — bu ixtiyoriy mumkin bo'lgan belgilar ketma-ketligidir.

Quyidagi qator, rus avfavitining katta va kichik harflaridan, ajratish belgilaridan va bo'sh belgidan iborat bo'lib, zanjirga misol bo'la oladi.

«AVrov, `»

Lekin zanjir belgilarning ma`noga ega ketma-ketligi bo'lishi shart emas. Quyidagi ketma-ketlik *«avvv.lagrь`,.ll»* — belgilar zanjiriga doir misol.

Belgilar zanjiri uchun tarkib, belgilar soni va tartib muhimdir.

Bitta belgi zanjirga ixtiyoriy son marta kirishi mumkin. SHuning uchun *«a»* va *«aa»*, yana *«ab»* va *«ba»* — bu belgilarning turli zanjiridir.

Belgilar zanjiri α va β teng α=β yoki mos tushadilar, agar ular bitta belgilar tarkibiga ega bo'lsalar, bir xil belgilar soniga va belgilarning zanjir bo'ylab bir xil kelish tartibiga ega bo'lsalar.

Zanjirdagi belgilar soni zanjir uzunligi deyiladi. a belgilar zanjirining uzunligi |a| kabi belgilanadi. Ko'rinib turibdiki, agar a = r, bo'lsa, u holda | a | = | r|.

Belgilar zanjiri ustida bajariladigan asosiy amallar- bu zanjirlar konkatenasiyasidir (birlashtirish yoki qo'shish) .

Ikkita belgilar zanjirini konkatenasiyasi (qo'shish, birlashtirish) — bu ikkinchi zanjirni birinchisini oxiriga qo'shib yozishdir. a va r zanjirlarni konkatenasiyasi ar bo'ladi.

Belgilar zanjiri quyidagi xususiyatlarga egadirlar:

1)Konkatenasiya – ikkita ta zanjirni yig'indisi yoki ko'paytmasi αβ

α="BA"

β="CL" => αβ="BACL"

Zanjirlarda belgilar tartibi muhim bo'lganligi sababli, konkatenasiya amali kommutativlik xususiyatiga ega emas, ya`ni αβ≠βα. Assosiativlik xususiyatiga egadir (αβ)γ=α(βγ).

Zanjirlar ustida bajariladigan yana ikkita amalni keltirish mumkin.

2)Zanjirga murojaat – zanjir belgilarini teskari tartibda yozish α^R, α="VASYA" => α^R="YASAV" Ushbu amal uchun $(\alpha\beta)^R=\alpha^R\beta^R$ haqiqat.

3) YAqinlashuv (takrorlash) zanjirni p marta takrorlash, bu erda neN, n>0 — bu zanjirni o'z-o'zi bilan p marta konkatenasiyasidir. a zanjirni n marta yaqinlashuvi a^n kabi belgilanadi. Takrorlash amali uchun quyidagi tengliklar haqiqat V a: $a^1 = a$, $a^2 = aa$, $a^3 = aaa$, ... va x.k.

4)belgilarning bo'sh zanjiri – bu bitta ham belgiga ega bo'lmagan zanjirdir, λ -bo'sh zanjir uchun quyidagilar haqiqat:

a) |λ|=0;

b) ixtiyoriy α: λα=αλ=α;

s)$\lambda^R=\lambda$;

d) ixtiyoriy n≥0:$\lambda^n=\lambda$;

e)ixtiyoriy α: $\alpha^0=1$

3.2.Til tushunchasi. Formal til haqida umumiy tushunchalar

Umumiy xolda til — bu belgilar va qoidalarning berilgan to'plami bo'lib, belgilarning o'zaro kombinasiyalari usullarini ma`noga ega matnlarni yozish uchun o'rnatadi.

Ixtiyoriy tabiiy yoki sun`iy tilning asosini tilning mumkin bo'lgan belgilar to'plamini aniqlovchi alfavit tashkil etadi.

Alfavit – tilning mumkin bo'lgan belgilaridan tuzilgan sanoqli to'plam. Ushbu to'plamni V harfi bilan belgilaymiz.

Qiziq, formal ta`rifga asosan, alfavit chekli (sanab chiqiladigan) to'plam bo'lishi shart emas, lekin haqiqatda barcha mavjud tillar chekli alfavit asosida quriladi.

Belgilar zanjiri α V:α(V) alfavit ustidagi zanjir hisoblanadi, agar unga V belgilar to'plamiga tegishli belgilar kirsalar. Ixtiyoriy V alfavit uchun λ bo'sh zanjir λ(V) ni zanjiri hisoblanishi ham, hisoblanmasligi ham mumkin.

Agar V qandaydir alfavit bo'lsa, u xolda:

V^+ - V alfavitdagi barcha zanjirlar to'plami, λ siz

V^* - barcha zanjirlar to'plami λ bilan

Bu erda $V^*=V^+ U \{\lambda\}$ tenglik o'rinli.

V alfavitli L til V: L(V) - bu qandaydir sanoqli chekli uzunlikdagi V alfavitni barcha zanjirlari to'plamidan olingan qism to'plamdir.

Ushbu ta`rifdan ikkita xulosa chiqarish mumkin: birinchidan, tilning zanjirlar to'plami chekli bo`lishi shart emas; ikkinchidan, tilning ixtiyoriy belgilar zanjiri chekli uzunlikka ega bo'lishi shart bo'lsa ham, bu uzunlik juda uzun bo'lishi mumkin, formal ravishda hech qanday chegara qo'yilmagan.

Berilgan tilga tegishli belgilar zanjirini gaplar deb ataladi.

Ixtiyoriy L(V) til uchun L(V) s V* haqiqat.

L(V) til o'ziga L'(V): L'(V)єL(V), tilni oladi, agar V aєL(V): aєL'(V) bo'lsa. L'(V) tilni zanjirlar to'plami L(V) tilni zanjirlar to'plami hisoblanadilar (yoki to'plamlar mos tushadilar). Ko'rinib turibdiki, ikkita til ham bitta alfavit asosida qurilishi shart.

Ikkita L(V) va L'(V) mos tushadilar (ekvivalentlar):

L'(V) = L(V), agar L'(V)єL va L(V) єL'(V);

yoki, xuddi shunday: V aєL'(V): aєL(V) va V aєL'(V): aєL(V).

Ekvivalent tillar uchun mumkin bo'lgan belgilar zanjirining to'plami teng bo'lishi shart.

Ikkita L(V) va L'(V) til ekvivalentdek deyiladi: L'(V) = L(V), agar L'(V)u{/ = L(V)u{A.} bo'lsa. Ekvivalentdek tillarning mumkin bo'lgan zanjirlar to'plami faqatgina bo'sh belgilar zanjiri bilan farqlanadilar.

3.3.Tillarni berish usullari

SHunday qilib, har bir til –bu qandaydir alfavitning belgilar zanjirlari to'plamidir. Lekin alfavitdan tashqari til yana mumkin bo'lgan zanjirlarni qurish qoidalarini berishni ham ko'zda tutadi, chunki ko'pincha berilgan alfavitning hamma zanjirlari ham tilga tegishli bo'lavermaydi. Belgilar so'zlar yoki tilning elementar konstruksiyasi bo'lgan leksemalarga birlashishlari mumkin, va ular asosida gaplar, ya`ni murakkab konstruksiyalar quriladi. Ikkalasi ham umumiy holda belgilar zanjiri hisoblanadilar, lekin ba`zi qurilish qoidalariga amal qiladilar. SHunday qilib, ushbu qoidalarni ko'rsatish yoki tilni berish zarur.

Tilni uch xil usul bilan berish mumkin:

1)barcha mumkin bo'lgan zanjirlarni sanab o'tish.

2)tilning zanjirlarini tug'ilish usullarini ko'rsatish (tilning grammatikasini berish bilan).

3)tilning zanjirlarini anglash usullarini aniqlash (anglovchi, avtomat).

Birinchi usul, formal usul bo'lib, amaliyotda qo'llanilmaydi, chunki ko'pgina tillar cheksiz sonli zanjirlarga ega bo'lib ularni sanab o'tishning iloji yo'q.

Ikkinchi usul tilning zanjirlarini qurishda foydalaniladigan qoidalarni ifodalashga mo'ljallangan. U xolda, ushbu qoidalar yordamida til alfavitining belgilaridan qurilgan ixtiyoriy zanjir berilgan tilga tegishli bo'ladi.

Uchinchi usul qandaydir mantiqiy qurilmani (anglovchini) –avtomatni qurishni nazarda tutadi,. Avtomat kirishida belgilar zanjirini oladi, chiqishida esa ushbu zanjir berilgan tilga tegishli yoki yo'qligi haqida ma`lumot beradi. Masalan, ushbu matnni o'qib, siz hozir qandaydir ma`noda anglovchi sifatida ishtirok etayapsiz (ushbu matnni o'zbek tiliga tegishli ekanligi aniq).

3.4.Tilning sintaksis va semantikasi

Ixtiyoriy til haqida gapirganda, uning sintaksis va semantikasini ko'rsatish mumkin. Bundan tashqari, translyatorlar tilning leksikasi bilan beriladigan, leksik konstruksiyalar (leksemalar) bilan ishlaydilar. Quyida ushbu barcha tushunchalarga ta`riflar keltirilgan.

Tilni sintaksisi – tilni mumkin bo'lgan konstruksiyalarini aniqlovchi qoidalar to'plamidir. Sintaksis tilning ko'rinishini aniqlaydi – ya`ni tilga tegishli belgilar zanjirlarini to'plamini beradi. Ko'p xollarda til sintaksisi qat`iy qoidalar to'plami ko'rinishida berilishi mumkin, lekin, bu ta`kid faqat formal tillar uchun taaluqlidir. Xattoki ko'pgina dasturlash tillari uchun berilgan sintaksis konstruksiyalar to'plami qo'shimcha izohlarga muhtojlik sezadi.

Tilni semantikasi – tilning gaplarini qiymatini aniqlaydi. Semantika tilning ma`nosini aniqlaydi - ya`ni tilning barcha mumkin bo'lgan zanjirlari uchun qiymat beradi. Ko'pgina tillar uchun semantika noformal usullar bilan aniqlanadi (ishoralar o'rtasidagi munosabat, ular nimalarni anglatishi bilan semantikada o'rganiladi). Ko'pincha formal tillar ma`noga ega emas.

Leksika – tilning so'zlari to'plamidir (so'zlar zahirasi). So'z yoki tilning leksik birligi (leksema) – bu tilning alfaviti elementlaridan tuziladi va o'zida boshqa konstruksiyalarni jamlamaydi. Boshqacha aytganda, leksik birlik faqat elementar belgilardan tashkil topadi va boshqa leksik birliklarga ega emas.

3.5.Dasturlash tillarining xususiyatlari

Dasturlash tillari tabiiy va formal tillar o'rtasidagi oraliq joyni egallaydilar. Formal tillar bilan ularni, asosida til gaplari quriladigan, qat`iy sintaksis qoidalar birlashtiradilar. Tabiiy muloqot tillaridan dasturlash tillariga, asosiy kalit so'zlarni ifodalovchi, leksik birliklar (ko'pincha ular ingliz tili so'zlari, lekin shunday dasturlash tillari ham mavjudki, ularning kalit so'zlari rus va boshqa tillardan olingan). Bundan tashqari, dasturlash tillarining algebrasi matematik amallarning asosiy belgilanishlarini qabul qilganlar, bu esa ularni insonlarga yanada tushunarli bo'lishini ta`minlaydi.

Dasturlash tillarini berish uchun quyidagi masalalarni echish zarur:
1)tilning mumkin bo'lgan belgilari to'plamini aniqlash;
2)tilning to'g'ri dasturlari to'plamini aniqlash;
3)har bir to'g'ri dasturning ma`nosini berish

Ushbu masalalarni birinchi ikkitasini formal tillar nazariyasi yordamida to'liq yoki qandaydir bo'laklarini echish imkoniyati mavjud. Birinchi masala oson echiladi. Alfavitni aniqlash orqali, biz avtomatik ravishda, mumkin bo'lgan belgilar to'plamini aniqlaymiz. Dasturlash tillari uchun alfavit — bu ko'pincha klaviatura yordamida kiritiladigan belgilar to'plamidir. Uning asosini xalqaro belgilarni kodlashning (ASCII jadvallari) kichik yarmi, ya`ni milliy alfavitning belgilari qo'shiluvchi bo'lagi tashkil etadi.

Formal tillar nazariyasida ikkinchi masalaning faqat qandaydir bo'lagi echiladi, chunki barcha dasturlash tillari uchun tilning sintaksisini aniqlovchi qoidalar mavjud, lekin avval aytib o'tganimizdek, ular barcha mavjud sintaksis konstruksiyalarni qat`iy aniqlash uchun etarli emas. Qo'shimcha chegaralar til semantikasi tomonidan qo'yiladi. Bu chegaralar har bir alohida dasturlash tili uchun noformal ko'rinishda beriladi. Bunday chegaralarga o'zgaruvchilar va funksiyalarni avvaldan ifodalash zaruriyatini, ya`ni ifodalardagi o'zgaruvchilar va konstantalarni toifalari, formal va faktik parametrlarni funksiyalarni chaqirish vaqtidagi mosligini va x.k., misol sifatida keltirish mumkin. Bundan kelib chiqadiki, deyarli barcha dasturlash tillari formal tillar hisoblanmaydilar. Va shu sababli barcha translyatorlarda, sintaksis tahlil (ajratish) va til gaplarini tahlilidan tashqari yana qo'shimcha semantik tahlil ko'zda tutilgan.

Uchinchi masala formal tillar nazariyasiga tegishli emas, chunki, avval aytib o'tilganidek, bunday tillar ma`noga ega emas. Bu savolga javob berish uchun boshqa yo'nalishlardan foydalanish kerak. Bunday yo'nalishlar sifatida quyidagilarni ko'rsatish mumkin:

- dasturlash tilida yoki boshqa dastur yo'naltirilgan kishilarga tushunarli tilda yozilgan dasturning ma`nosini bayon qilish;
- ushbu tilda yozilgan «ideal mashina» dan, bajarilishga mo'ljallangan dasturning, ma`nosini tekshirish uchun foydalanish.

Barcha dastur yozgan kishilar, u yoki bu xolda albatta birinchi yo'nalishdan foydalanganlar. YAxshi dasturdagi izohlar –bu dasturning ma`nosini bayon etish demakdir. Blok-chizmaning qurilishi, dastur algoritmining yana boshqa shunga o'xshash ifodalanishi — bu ham dastur ma`nosini boshqa tilda bayon etishning yaxshi usulidir (masalan, ma`nosi o'z navbatida mos GOSTga asosan bayon etilgan blok-chizma algoritmlarini grafik belgilari tilida). Dasturni xujjatlashtirish — bu ham dasturni bayon etishning usulidir. Bu barcha usullar insonlarga mo'ljallangan va ularga tushunarliroqdir. Lekin dastur ifodalashga qanchalik mosligini tekshiruvchi umumiy algoritm hozircha mavjud emas.

Mashina faqat bitta tilni tushunadi — bu mashina komandalari tilidir. Ammo, dasturni mashina komandalari tilida bayon etish inson uchun juda mashaqqatli masala va shuning uchun, bu masalani echishga translyatorlar yaratiladi.

Ikkinchi yo'nalishdan dasturlarni qayta ishlash jarayonida foydalaniladi. Dasturlarni qayta ishlash jarayonidagi dastur natijalarini baholash vazifasini inson bajaradi. Ushbu xarakatlarni mashinaga yuklashning xar qanday urinishlari ma`noga ega emas va hal etilayotgan masaladan tashqaridir. Masalan, Pascal tilidagi dasturning quyidagi ko'rinishdagi gapi: i:=0; while i=0 do i:=0; ixtiyoriy mashina tomonidan, ma`noga ega emas deb, oson baholanishi mumkin. Ammo ushbu masalaga boshqa paralel bajarilayotgan dasturni o'zaroxarakatlarini ta`minlash, yoki, masalan, prosessorning ishonchliligi va chidamliligini yoki xotiraning qandaydir yacheykasini tekshirish, masalasi kiritilsa, u xolda ushbu gap juda ham ma`nosizdek tuyulmaydi. Dasturlarni ma`nosini tekshirish jarayonida tizimli dasturiy ta`minotni ishlab chiqarishni avtomatlashtirish (CASE-tizimlar) ramkasida ba`zi bir yutuqlarga erishilgan. Ularning yo'nalishi, insonga tushunarli bo'lgan tilda masalani bayon etishdan, to uni dasturlash tillarining operatorlariga o'girishgacha bo'lgan, yuqoridan pastga loyixalashtirishga asoslangan, Lekin bunday yo'nalish translyatorlarning imkoniyatlaridan chetga chiqadi, shuning uchun bu yo'nalishni qarab o'tirmaymiz.

Lekin kompilyatorlarni ishlab chiqaruvchilar uchun u yoki bu holda dasturlarni ma`nosi haqidagi masalani echish talab etiladi. Birinchidan, kompilyator boshlang'ich dasturni mashina komadalari ketma-ketligiga aylantirishi kerak, buning uchun unga boshlang'ich dasturning qanday bo'lagi uchun qanday komandalar ketma-ketligi mos kelishi haqidagi taassurotlar zarur. Ko'pincha bunday ketma-ketliklar kiruvchi tilning baza konstruksiyalariga taqqoslanadi. Bu erda dasturlarni bayon etishning birinchi yo'nalishidan foydalaniladi. Birinchidan, ko'pgina

zamonaviy kompilyatorlar boshlang'ich dasturdagi ikkilanish holatlarini, ma`no nuqtai nazaridan, aniqlash imkonini beradilar, bular — foydalanilmagan o'zgaruvchilar, funksiyalarning aniqlanmagan qiymatlari va x.k.. Ko'pincha kompilyator bunday joylarni qo'shimcha ogohlantirishlar ko'rinishida ko'rsatadi, ularni ishlab chiqaruvchilar e`tiborga olishlari yoki olmasliklari mumkin. Ushbu maqsadni amalga oshirish uchun kompilyator dasturning qanday bajarilishi haqida tasavvurga ega bo'lishi kerak — ya`ni ikkinchi yo'nalishdan foydalaniladi. Ammo, ikki holda ham boshlang'ich dastur ma`nosini kompilyatorga uni ishlab chiqaruvchisi tomonidan kiritiladi (yoki ishlab chiqaruvchilar guruhi) — ya`ni noformal usullarni boshqaruvchi inson kiritadi (ko'pincha, bu ushbu tilni ifodalash orqali bajariladi). Formal tillar nazariyasida dasturni ma`nosi masalasi qaralmaydi.

SHunday qilib, yuqorida aytilganlardan kelib chiqqan holda aytish mumkinki, translyatorlarning kiruvchi til gaplarini tekshirish bo'yicha imkoniyatlari chegaralangan, xattoki amaliy ravishda nolga teng. Xuddi shu sababli, ularning ko'pchiligi eng yaxshi holda ishlab chiqaruvchilarga, boshlang'ich dastur matnining ma`no nuqtai nazaridan ikkilanishlar yuzaga kelgan joylari bo'yicha, ko'rsatmalar berish bilan cheklanadilar. SHuning uchun ko'pgina translyatorlar ma`noviy (semantik) xatoliklarning umumiy sonidan juda kam foizini topa oladilar, bunday ko'rinadiki, juda kup sonli bunday xatoliklar, har doim, baxtga qarshi dastur avtorining vijdoniga xavola bo'lib qoladi.

Sinov savollari

1.Formal tilni o'rganish nima uchun kerak?

2.Tabiiy, formal va dasturlash tillarining farqlarini qanday izohlaysiz?

3.Formal grammatika nima?

4.Belgilar zanjiri nima?

5.Belgilar zanjiri ustida qanday amallarni bajarish mumkin ?

6.Ekvivalent grammatika nima?

7.Bir qiymatli bo'lmagan grammatika haqida ma`lumot bering.

8.Grammatikaning berilish usullari haqida ma`lumot bering.

9.Tillarning berilish usullari haqida ma`lumot bering.

10.Tilni berilish usullaridan barcha mumkin bo'lgan zanjirlarni sanab o'tish usuli nima sababdan amaliyotda qo'llanilmaydi?

11.Leksema nima?

12. Leksika nima?

13.Til sintaksisini nimalar aniqlaydi?

14.Tilning sintaksis va semantikasi orasidagi farqni ayting.

15.Dasturlash tillarini berish uchun echilishi kerak bo'lgan masalalarni ayting.

4.BO'LIM
Grammatikaning ta`rifi. Bekus –Naur formasi

Grammatika qoidalarida rekursiya tamoyillari. Grammatika berilishining boshqa usullari

4.1.Grammatikaning ta`rifi. Bekus-Naur formasi
Til grammatikasi tushunchasi

Grammatika – qandaydir tilning gaplarini qurish usullarini ifodalashdir. Boshqacha aytganda, grammatika — bu tilni aniqlovchi matematik tizimdir. Tilni grammatikasini aniqlash orqali biz ushbu tilga tegishli belgilar zanjirlarini tug'ilish qoidalarini ko'rsatamiz. SHunday qilib, grammatika – bu til zanjirlarini generatoridir. U tilni ifodalashning ikkinchi usuli, ya`ni belgilar zanjirlarini tug'ilishiga, tegishlidir. Til grammatikasini turli usullar orqali ifodalash mumkin. Masalan, rus tili grammatikasi, boshlang'ich maktabda o'rganiladigan, etarli darajada murakkab qoidalar to'plami sifatida ifodalanadi. Ammo, ko'pgina tillar uchun (dasturlash tillarining sintaksis bo'lagi uchun ham) grammatikaning, qoidalar yoki (maxsulotlar) tizimi asosida qurilgan, formal ifodalanishi mavjud.

Qoida yoki mahsulot – bu (α,β) tartiblangan belgilarning juftligi bo'lib, qoidalarda (α→β α tug'diradi β) zanjir tartibi muhimdir, shuning uchun ularni ko'pincha quyidagi ko'rinishda yoziladi x->3 (yoki a::=R). Bu yozuv quyidagicha o'qiladi: «a r ni keltirib chiqaradi» a yoki « 3 bu a».

Dasturlash tilining grammatikasi ikki turdagi qoidalarni o'z ichiga oladi: birinchisi (tilning sintaksis konstruksiyalarini aniqlovchi) formal ifodalashlarga oson keltiriladi; ikkinchisi (tilning semantik chegaralarini aniqlovchi) ko'pincha noformal ko'rinishda bayon qilinadi. SHuning uchun dasturlash tilining ixtiyoriy ifodalanishi yoki umumiy qabul qilingan standarti ko'pincha ikki bo'lakdan tashkil topadi: avval sintaksis konstruksiyalarning qurish qoidalari formal ravishda bayon etiladi, so'ngra tabiiy tilda semantik qoidalarning ifodasi beriladi. Tabiiy til dasturlash tilida dasturni yozishi kerak bo'lgan insonga, foydalanuvchiga tushunarlidir; semantik chegaralarni kompilyator uchun dasturlarni to'g'riligini tekshiruvchi algoritmlar ko'rinishida bayon etiladi(gap faqat, yuqorida aytib o'tilganidek, dasturning ma`nosi haqida emas, balki boshlang'ich matnga semantik chegaralar ustida ketayapti). Kompilyatorda bunday tekshiruvni, ushbu ishlar uchun maxsus ishlab chiqarilgan kompilyatorning bo'lagi, semantik tahlilchi bajaradi.

Bunday buyon, dasturlash tillarining grammatikasi haqida gapirilganda, faqat tilning sintaksis konstruksiyalarini qurish qoidalarini nazarda tutamiz. Lekin shuni esda tutish kerakki, ixtiyoriy dasturlash tilining grammatikasi umumiy holda faqat shu qoidalar bilan chegaralanmaydi.

G grammatika bilan berilgan til L(G) bilan belgilanadi.

Ikkita L(V) va L'(V) til ekvivalentdek deyiladi: L'(V) = L(V), agar L'(V)u{/ = L(V)u{A.} bo'lsa. Ekvivalentdek tillarning mumkin bo'lgan zanjirlar to'plami faqatgina bo'sh belgilar zanjiri bilan farqlanadilar.

Ikkita G va G' grammatikalar ekvivalent deyiladi, agar ular bitta yagona tilni aniqlasalar: L(G) = L(G'). Ikkita G va G' grammatikalar ekvivalentdek deyiladi, agar, ular orqali berilgan tillar belgilar zanjirining bo'sh zanjirigagina farq qilsalar: L(G) u {A,} = L(G') u {A,}.

4.2.Grammatikaning formal ta`rifi. Bekus-Naur formasi

Grammatikaning to'liq formal aniqlanishi uchun tilning zanjirlarini tug'ilishi qoidalaridan tashqari yana tilni berish zarur.

Grammatika formal ravishda quyidagi to'rtlik bilan aniqlanadi: $G(V_t,V_n,P,S)$.

bu erda V_t - alfavit bo'lib, uning belgilari terminallar deb ataladi, ulardan grammatika orqali keltiriluvchi zanjirlar quriladi.

V_n - alfavit bo'lib, uning belgilari noterminallar deb ataladi, zanjirlarni qurishda foydalaniladi.

$V_n - V_t$ va V_n umumiy belgilarga ega emaslar, ya`ni $V_t \cup V_n = 0$.

Grammatikaning to'liq alfaviti $V = V_t \cup V_n$ kabi aniqlanadi.

R– tug'iluvchi qoidalar to'plami bo'lib, uning har bir elementi (a, b) juftligidan tashkil topadi.

bu erda a V+ da, va b esa V* da, a qoidaning chap bo'lagi, b esa o'ng bo'lagidir.

Qoida quyidagicha yoziladi:

$a \rightarrow b$.

S V_n ga tegishli va boshlang'ich belgi (aksioma) deb ataladi. Bu belgi tilning ixtiyoriy gapini olish uchun tayanch nuqtadir.

V_t va V_n to'plamlarni qaraymiz. Terminal belgilar to'plami V_t grammatika orqali tug'ilgan va til alfavitiga kiruvchi belgilardan tashkil topadi. Qoida bo'yicha, V_t to'plamdan olingan belgilar zanjirlarda qoidalarning faqat o'ng tomonida uchraydilar, agar ular zanjirda qoidaning chap tomonida uchrasalar, u holda ular uning o'ng tomoni zanjirida ham bo'lishlari shart.

V_n noterminal belgilar to'plami so'zlarni, tushunchalarni, til konstruksiyalarini aniqlovchi belgilardan tashkil topadi.

Bu to'plamning har bir belgisi grammatika qoidalarining ham chap, ham o'ng zanjirlarida uchrashi mumkin, lekin u juda bo'lmaganda bir marta biron qoidaning chap bo'lagida bo'lishi shart. Grammatika qoidalari shunday quriladiki, har bir qoidaning chap bo'lagida juda bo'lmaganda bitta noterminal belgi bo'lishi kerak.

Ushbu ikki to'plamlar kesishmaydilar: har bir belgi yoki terminal belgi, yoki noterminal belgi bo'lishi mumkin. Grammatika alfavitining biron bir belgisi bir

vaqtning o'zida noterminal va terminal bo'lmasligi mumkin emas. Grammatikaning maqsad belgisi — bu har doim noterminal belgidir.

Grammatika qoidalari to'plamida bir xil o'ng tomonga ega qoidalar bo'lishi mumkin, ya`ni: a->r , ach>r_2, ... a->r_p. U xolda bu qoidalarni birlashtirish va quyidagi ko'rinishda yozish mumkin: a—>$p_1|p_2|$ ...| p_n .

Grammatika qoidalarining bunday yozilish ko`rinishi Bekus—Naur formasi deb ataladi. Bekus—Naur formasida qoidalar kabi, noterminal belgilar ham burchak qavslarga olinadi: < >. Goxida «->» belgini grammatika qoidalarida «::=» belgiga almashtiriladi.

Quyida ishorali butun o'nlik sonlar uchun grammatika misoli keltirilgan:

G = {{0,1,2,...,9,-,+,-},{<ishorali son>,<son>,<raqam>}, P,<ishorali son>}
P = {
<ishorali son> ::= +<son> | -<son>
<son> ::= <raqam> | <raqam><son>
<raqam> ::= 0|1|2|3|4|5|6|7|8|9
}

G grammatikaning tashkil etuvchi elementlarini qarab chiqamiz:

- V_t terminal belgilar to'plami o'n ikkita element: o'nta o'nlik raqam va ikkita ishora dan tashkil topadi;
- V_n noterminal belgilar to'plami uch element: belgilar <son>, <ishorali son> va <raqam> dan tashkil topadi;
- Qoidalar to'plami uch qatorga yozilgan 15 qoidadan iborat (ya`ni faqat qoidalar bo'lagining uch turli qoidasi mavjud);
- Grammatikaning maqsad belgisi <son> hisoblanadi..

Noterminal belgilar ismlari ma`noga ega bo'lishlari shart emas, bu esa inson tomonidan grammatika qoidalarini tushunishni oson bo'lishi uchun qilingan. Ixtiyoriy grammatikada, grammatika tomonidan berilgan tilni o'zgartirmagan holda, barcha noterminal belgilar ismlarini to'lik o'zgartirish mumkin. Masalan, Pascal tilidagi dasturda identifikatorlarni ismlarini o'zgartirish mumkin, bu holda dasturning ma`nosi o'zgarmaydi.

Terminal belgilar uchun bu to'g'ri emas. Terminal belgilar to'plami har doim grammatika tomonidan berilgan til alfavitiga qat`iy mos keladi.

Mana masalan, ishorali o'nlik butun sonlar uchun grammatika, bu erda noterminal belgilar katta latin harflari bilan belgilangan:

G'({0,1,2,3,4,5,6,7,8,9,-,+},{S,T,F},P,S)
R:
S -» T | +T | -T
T -> F | TF
F - » 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Bu erda faqat noterminal belgilar to'plami o'zgardi. Endi noterminal belgilar to'plami V_n ={S,T,F}. Grammatika bilan berilgan til o'zgarmadi, G va G' grammatikalar ekvivalentlar.

O'z navbatida qatorlar gaplarni tashkil etadi va x.k. Bunday qatorlar to'plami tilni tashkil etadi. Til qandaydir formal grammatika bilan ifodalanadi deb hisoblasak, grammatika tilni yuzaga keltiradi.

Faraz qilaylik alfavit bu ={0, 1} bo'lsin. Ushbu alfavitdan 01001 qator tashkil etilsin. Zanjirdagi belgilar soni zanjirning uzunligi m deb ataladi (belgilarning mos tushishiga bog'liksiz ravishda). Bo'sh zanjir deb uzunligi m=0 bulgan zanjir ataladi.

Bo'sh to'plam ixtiyoriy to'plamga kiradi.

L alfavitning formal tili deb qandaydir ixtiyoriy qismto'plamga aytiladi. Har bir formal til gaplari qandaydir qoidalardan kelib chiqqan holda tuziladi. Bu qoidalar tilning sintaksisini tashkil etadi. Formal tilning muammolaridan biri til sintaksisini ifodalashdan iboratdir.

Mazmuni: agar tillar ko'p bo'lmagan tugallangan gaplardan tashkil topgan bo'lganlarida edi, u holda ularni sanab chiqish mumkin bo'lar edi va konstruksiyalarning to'g'riligi ushbu to'plamga tegishli yoki tegishli emasligi bilan tekshirilar edi.

Lekin mumkin bo'lgan gaplar soni shunchalik ko'pki, ularni sanab chiqishning sira iloji yuk.

Grammatika – til sintaksisining qoidalari to'plamidir.

Grammatika ikki xil ko'rinishda bo'lishi mumkin:

Tug'iluvchi va anglanuvchi.

Tug'iluvchi grammatika to'g'ri gaplar tashkil etish prosedurasini boshlang'ich belgidan boshlab ifodalaydi.

Anglanuvchi grammatika esa aniq konstruksiyani aniq tilga tegishli ekanligini anglash (ajratish) jarayonini ifodalaydi (zanjirdan boshlang'ich belgigacha).

Bu grammatikalar xarakat yo'nalishi jihatidan farqlanadilar.

SHuning uchun tillar sintaksisini ifodalovchi maxsus –metatillar (tillar ustida tillar) mavjud. Ushbu tilda til konstruksiyasini to'g'riligini aniqlovchi qoidalar tizimini ifodalanadi.

4.3.Grammatika qoidalarida rekursiya tamoyillari

Formal grammatikalarning o'ziga xosligi shundaki, ular chekli qoidalar to'plami yordamida tilni cheksiz zanjirlar to'plamini aniqlash imkoniyatiga egadirlar (albatta, tilning zanjirlari to'plami ham chekli bo'lishi mumkin, lekin, eng oddiy til uchun ham, bu shart ko'pincha bajarilmaydi). YUqoridagi misolda keltirilgan ishorali o'nlik butun sonlar uchun grammatika 15 ta qoida yordamida cheksiz sonli butun sonlar to'plamini aniqlaydi.

Grammatikaning bunday ko'rinishda yozilishida qoidalarning cheksiz to'plamidan foydalanish imkoniyatiga rekursiv qoidalar hisobiga erishiladi. Grammatika qoidalarida rekursivlik bitta noterminalni o'zini-o'zi aniqlashi orqali ifodalanadi. Rekursiya bevosita (aniq) bo'lishi mumkin— bu holda belgi o'z-o'zini bitta qoidada aniqlaydi, yoki bilvosita (noaniq) — bu holda o'z-o'zini aniqlash qoidalar zanjiri orqali o'tadi.

YUqorida ko'rib o'tilgan G grammatikada bevosita rekursiya quyidagi qoidada mavjud: <son>-»<son><raqam>, unga ekvivalent G' grammatikada T-»TF qoidada mavjud.

Rekursiya cheksiz bo'lishi uchun, grammatikaning unda qatnashayotgan noterminal belgisi uchun, yana boshqa qoidalar, ya`ni uni o'zidan boshka aniqlaydigan va shu bilan cheksiz rekursivlakdan qochish imkonini beradigan qoidalar ham, bo'lishi shart (aks holda bu belgi grammatikada kerak bo'lmas edi). Bunday qoidalar bo'lib, G grammatikada <son> -» <raqam>, G' grammatikada T -» F xisoblanadi..

Formal tillar nazariyasida rekursiyalar haqida bundan ko'p ma`lumot berish mumkin emas. Ammo, rekursiyani ma`nosini to'liqroq tushunish uchun, tilning semantikasiga murojaat etish mumkin, yuqorida ko'rilgan misolda bu -ishorali o'nlik butun sonlar tilidir. Uning ma`nosini qarab chiqamiz.

Agar son bo'lib nima hisoblanadi degan savolga ta`rif berishga urinsak, u holda har bir ixtiyoriy raqam o'z navbitida son ekanligini bilamiz. So'ngra yana ixtiyoriy ikkita raqam ham son ekanligini, so'ngra uchta raqam ham son ekanligini va x.k. bilish kiyin emas. Agar sonning ta`rifini shu usulda qurishga boshlasak, u holda jarayon hech qachon tugamaydi (matematikada sonlarning razryadi cheklanmagan). Lekin, biz bilamizki, har safar yangi sonni yaratish uchun kerakli raqamni o'ng tomondan yozilgan raqamlar qatoriga qo'shib yoziladi (chunki chapdan o'ngga qarab yozishga odatlanganmiz). Bu raqamlar qatori esa, bitta raqamdan boshlanib, ular ham o'z navbatida son hisoblanadilar. U holda «son» tushunchasi uchun ta`rifni quyidagicha qurish mumkin: «son — bu ixtiyoriy raqam bo'lib, yoki unga o'ng tomondan ixtiyoriy raqam qo'shilgan boshqa son».

Xuddi ana shu G va G' grammatikalar qoidalari asosini tashkil etadi va bu qoidalar qatorining ikkinchi qatorida aks ettirilgan:

<son>—> <raqam> | <son><raqam> va T -» F | TF.

Bu grammatikalarning boshqa qoidalari songa ishorani qo'shish imkoniyatini (qoidalarning birinchi qatori) va «raqam» tushunchasiga ta`rif (qoidalarning uchinchi qatori) beradi. Ular elementar tushunchalar va shuning uchun aniqlashtirish talab etilmaydi.

Rekursiya tamoyili (ko'pincha uni «yaqinlashuv tamoyili» deb ataladi) — formal grammatikalarning ifodalanishida muhim tushunchadir. Ixtiyoriy dasturlash tillarining grammatikasida u yoki bu holda albatta aniq yoki noaniq rekursiya mavjud bo'ladi. Xuddi uning o'zi tilning cheksiz zanjirlari to'plamini qurish imkonini yaratadi, va ularning yaratilishi haqida rekursiya tamoyilisiz gapirish mumkin emas.

Qoida bo'yicha, aniq dasturlash tili grammatikasida bitta emas, balki rekursiya yordamida qurilgan qoidalar to'plami mavjud bo'ladi.

4.4.Bekus-Naur formasidagi grammatikaning yozilishi

Grammatika qoidalarini berish usuli notasiya deb ataladi. Ko'pincha Bekus Naur formasidan foydalaniladi. Unda quyidagi belgilashlardan foydalaniladi: →; ::= Barcha noterminal belgilar burchak qavslarga olinadi. Agar qandaydir tushuncha uchun chap bo'lakda bir nechta variant bor bo'lsa, u holda "|" belgidan foydalaniladi.

Masalan :

1) <A> ::= a | a<A>

2) $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \ldots, \alpha \rightarrow \beta_n$

3) $\alpha \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$

Bekus Naur formasidan foydalanib tug'iluvchi grammatikaga misol (12-rasm) ko'rib chiqamiz. Bu prosedura rus tilini kesilgan bo'lagi ko'rinishini eslatadi. Grammatika ob`ektlari: gap a`zolari, gap bo'laklari.
Ularning belgilanishi quyidagicha bo'lsin:

PR- predlojenie -gap
P – podlejashee- ega
S – skazuemoe - kesim
IS – imya sushestvitelnoe – ot ismi
M – mestoimenie - olmosh
GF – glagolnaya forma – glagol ko'rinishi
Qoidalar to'plami quyidagi ko'rinishda berilsin:

<PR>--<P><S>
<P>--<IS>
<P>--<M>
<S>--<GF>
<IS>--samolyot
<IS>--dom
<M>--on
<GF>--stoit
<GF>--stroitsya
<GF>--letit

Qoidalar quyidagicha o'qiladi: a→b a b ni keltirib chiqaradi, yoki a dan b chiqadi, yoki b a dan kelib chiqadi. Ko'rsatkich chiqish yo'nalishini ko'rsatadi. O'ngdagi narsa chapdagi narsani aniqlaydi.

<imya> - noterminal belgi
imya – terminal belgi.

Terminal belgilar grammatika qoidalari yordamida ochib berilmaydi, noterminal belgilar esa ochiladi. Grammatikaning qoidalar to'plami P bilan belgilaymiz. Ushbu holda P sanab o'tish ko'rinishida keltirilgan.

YOki chap yoki o'ng qismga kiruvchi, yoki ikkala qismga ham kiruvchi barcha belgilar to'plami V bilan belgilanadi. P qoidada V alfavitning belgilaridan tilning to'g'ri gaplarini tuzish haqidagi ma`lumotlar saqlanadi (ushbu qoidalar bo'yicha tuzilmagan gaplar noto'g'ri hisoblanadilar).

Ushbu holatda G0 grammatika uchun alfavit quyidagicha belgilanadi:

V ={<PR>,<S>, <IS>, <P>, <M>, <GF>, samolyot, dom, on, stoit, stroitsya, letit}.

Umumiy holatda V to'plam ikki qismdan noterminal N va terminal T belgilar to'plamidan tuziladi.

Ixtiyoriy noterminal belgi juda bo'lmaganda bir marotaba qoidaning chap tomoniga kirishi shart.

N = {<PR>, <P>, <S>, <M>, <IS>, <GF>}

T – faqat o'ng bo'lakka kiradigan terminal belgilar to'plami.

T = {samolyot, dom, on, stroitsya, stoit, letit}

Tug'iluvchi grammatiki uchun S – boshlang'ich belgi.

S = {<PR>}.

Kompilyator uchun (Paskal) S sifatida «dastur» tushunchasi turadi.

<Dastur>--<Ifodalashlar bo'lagi><Xarakatlar bo'lagi>.

Agar o'ng bo'lakda bir konstruksiya keyingisidan keyin qat`iy kelsa, bu hol qatorlar konkatenasiyasini anglatadi (zanjirlanish) .

Ixtiyoriy grammatika ikkita masalani hal qilishi kerak:

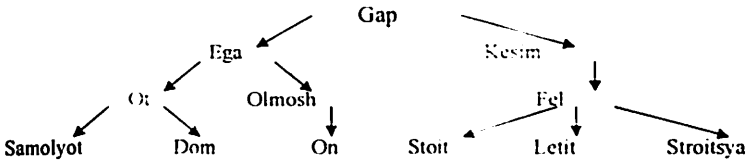
1.Anglash masalasini;

2.Tug'ilish masalasini.

Tug'ilish jarayonida to'g'ri gaplarning chiqishi ifodalanadi. Bu kuyidagicha amalga oshiriladi: boshlang'ich belgidan boshlab chap bo'lakning qoidalarini o'ng bo'lakka almashtirish amalga oshiriladi. Har bir olingan tushuncha o'zining tarifiga almashtiriladi. Bu jarayon o'ng tomonda faqat terminal belgilarning o'zi qolmagunicha davom etadi.

Jarayonni grammatik tahlil daraxti ko'rinishida ifodalash qulay. Ushbu daraxt qanday qoidalarni qanday til konstruksiyalarga qo'llash mumkinligini ko'rsatadi, lekin u aniq tug'ilish jarayonidagi qo'llash tartibini ko'rsatmaydi. Daraxt grammatikaning ushbu qoidalariga asoslangan holda quriladi. YUqorida boshlang'ich belgi joylashadi, pastda – terminal belgilar. Daraxt qoidalarni qo'llash yo'li bilan quriladi.

Gap ega va kesimning konkatenasiyasidan tashkil topadi. Ega bo'lib yoki IS yoki M kelishi mumkin.



12-rasm. Bekus Naur formasidan foydalanib tug'iluvchi grammatikaga misol

Aniq bir gapni chiqishi echim qabul qilishni talab etadi: qaysi yo'l bilan pastga qarab yurish kerak. Daraxt esa qoidalarni ifodalaydi.

Anglash masalasi daraxtdan foydalanib echiladi.

Daraxt bo'yicha pastdan yuqoriga xarakat qilib aniq gapga boshlang'ich belgiga etib borish kerak. Bu erda o'ng tomon bo'laklarini chap tomon bo'laklari qoidasiga almashtiriladi.

Masalan, Dom stroitsya.
IS va GF; --P va S; --PR.
Qoidalar quyidagi ko'rinishga ega: a → b
G1 grammatikaga misol qarab chiqamiz. Bu erda:

Noterminal belgilar → A, B, C, ...
Terminal belgilar → a, b, c, ... berilgan.
Grammatika G1 = {N, T, S, P}, bu erda
N = {A, B, S}
T = {a, b,s}

P = { S → AB (1)
A → aA (2)
A → a (3)
B → Bb (4)
B → b (5)
}

(2) qoidada noterminal belgi A ham chap ham o'ng bo'lakda mavjud. Bu esa A belgi tegishli qatorlar sinfi A belgiga a prefiksni qo'shish orqali qurilishini anglatadi.

Uchinchi qoidada A→ a orqali aniqlanadi. Umumiy holda bunday jarayon qatorlar konkatenasiyasi deb ataladi (A qator chapdan qo'shiladigan a konkatenasiyasidan quriladi).

Qandaydir tushuncha o'zi o'zidan quriladigan holat rekursiya deb ataladi.

To'rtinchi qoidada ham rekursiya mavjud. Zanjir belgini o'ngdan qo'shish yo'li bilan tashkil etiladi.

Bu holatda qoidalar barcha mumkin bo`lgan variantlarni sanab chiqish yo'li bilan beriladi, har bir variant uchun bitta qator.

Bir necha qatorlardan tarkib topgan tilni tassavvur qilishimiz mumkin. Tilni ifodalashda qanday qatorlar ushbu tilga tegishli ekanligi (til sintaksisi) va ushbu qatorlarni qiymati (til semantikasi) aniqlanadi. Sintaksis-formal to'g'ri gaplar to'plamining qoidalari to'plamidir. Tilga tegishli qatorlarni tilning gaplari deb ataladi. Real tillarda cheksiz gaplar soni bo'ladi va ularni sanab o'tishning iloji yo'q.

Eng sodda tilning sintaksisini tabiiy tilda quyidagicha ifodalash mumkin, masalan: «barcha qatorlar, faqat 1 va 0 lardan tashkil topgan» u holda 1111 va 1000110 –tilga tegishli, 1020 esa yo'q.

Masalan, quyidagi gap «mashina yuradi». «Mashina» so'zi ega, «yuradi» kesim. Ushbu gap quyidagi sintaksis qoidalar yordamida ifodalash mumkin bo'lgan tilga tegishli:

<gap>::=<ega><kesim>
<ega>::=mashina | ot
<kesim>::= yuradi | chopadi

Ushbu uchta qatorning ma`nosi quyidagicha: gap ega va kesimdan iborat. Ega yoki mashina degan bir so'zdan yoki ot degan so'zdan tashkil topgan. Kesim ham yoki yuradi degan so'zdan, yoki chopadi degan so'zdan tashkil topgan.

Ixtiyoriy gapni boshlang'ich belgi orkali ketma-ket qo'yish yo'li bilan olish mumkin.

YUqoridagi misolda sintaksis birliklar <gap> <ega> va <kesim> noterminal belgilar deb ataladi, "mashina", "ot", "yuradi", chopadi terminal belgilar deb ataladi, qoidalar esa tug'iluvchi qoidalardir. ::=, | . < > belgilar metabelgilardir. Semantika tilning barcha gaplariga qiymat beradi.

Tilni sintaksisini to'plamlarni tasvirlash orqali aniqlash mumkin, masalan $L=\{0_n\ 1_n\ |n>=0\}$. Ushbu til bir yoki bir necha nullardan, birlardan va bo'sh qatordan tashkil topgan qatorlarni o'z ichiga oladi.

Tilni murakkabrok sintaksisini grammatika yordamida aniqlash yaxshirok. L sintaksisni olamiz va quyidagi qoidalardan foydalanamiz.

1. $S \rightarrow$ 0S1 2.$S \rightarrow \varepsilon$ bu erda ε bo`sh belgi.

Ushbu tilning gaplarini chiqarish uchun quyidagicha ish yuritamiz. S belgidan boshlaymiz va uni 0S1 bilan almashtiramiz yoki ε bilan. Agar S yana olingan qatorda mavjud bulsa, yana almashtiramiz va x.k. SHunday usul bilan olingan S ga ega bo'lmagan qator shu tilning gapi hisoblanadi. Masalan, $S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 000S111 \rightarrow 000111$

Bunday qatorlarning ketma-ketligi 000111 qatorni chiqishi deyiladi, ko'rsatkich belgisi esa chiqish qadamlarini bo'laklash uchun xizmat qiladi. Ushbu tilning barcha gaplarini ikkita qoidadan kelib chiqqan holda keltirib chiqarish

mumkin. Ixtiyoriy keltirib chiqarish mumkin bo'lmagan qator ushbu tilning gapi hisoblanmaydi.

Grammatikani ko'pincha qayta yozish tizimi deb ham ataydilar.

$L=\{0_n 1_n | n>=0\}$.tilni generasiya qiladigan grammatika bo'lib

$G0 =(\{0,1\},\{S\},P,S)$, bu erda $P=\{S \rightarrow 0S1, S \rightarrow \varepsilon\}$ hisoblanadi.

$L=\{a_n b_m | n,m>=0\}$.tilni generasiya qiladigan grammatika bo'lib

$G0 =(\{a,b\},\{S,A,B\},P,S)$, bu erda $P=\{S \rightarrow AB, A \rightarrow aA, A \rightarrow \varepsilon, B \rightarrow bB, B \rightarrow \varepsilon\}$ hisoblanadi.

S belgidan boshlab noterminalni almashtirish qoidasini qo'llab aaabb qatorni generasiya qilish mumkin.

$S \rightarrow AB \rightarrow aAB \rightarrow aaAB \rightarrow aaaAB \rightarrow aaaB \rightarrow aaabB \rightarrow aaabbB \rightarrow aaabb$

Har bir boshlang'ich belgidan keltirib chiqariladigan qator sentensial forma deb ataladi. Sentensial formali gap –bu faqat terminallardan iboratdir. Terminallarni kichkina harflar bilan, noterminallarni katta harflar bilan belgilanadi.

Ikkita bir xil tilni keltirib chiqaradigan grammatikani ekvivalent grammatika deb ataymiz.

4.5.Grammatikani berishning boshqa usullari

Bekus—Naur formasi — formal nuqtai nazaridan qulay, lekin tushunish uchun har doim ham imkoniyati bo'lmagan formal grammatikalarning yozilish usulidir. Rekursiv aniqlashlar til zanjirlarini formal tahlili uchun qulay, ammo inson nuqtai nazaridan noqulaydir. Masalan, quyidagi qoidalarda <son> —> <raqam> | <son><raqam> sonni qurish uchun o'ng tomondan ixtiyoriy sonli raqamni, bittadan boshlab qo'shib yozishini aks ettirish imkoniyati aniq ko'rinib turmaydi va qo'shimcha aniqlashlar talab etiladi.

Lekin, dasturlash tilini yaratishda, tilni grammatikasini ushbu til uchun kompilyatorni yaratuvchilargina emas, balki kelgusida ushbu tildan foydalanib dastur yozuvchi foydalanuvchilar ham tushunishlari muhim. SHuning uchun formal grammatikani ifodalashning boshqa usullari, ya`ni insonlarga tushunarlirok usullari ham, mavjud.

4.6.Grammatika qoidalarini metabelgilar yordamida berilishi

Bunday tashqari grammatika qoidalari metabelgilardan foydalanilgan holda ham beriladi (qavslar):

Grammatika qoidalari metabelgilardan foydalanilgan holda yozish, grammatika qoidalari qatorida maxsus belgilar, ya`ni maxsus ma`noga ega metabelgilar uchrashi mumkinligini anglatadi. Bunday metabelgilar sifatida ko'pincha quyidagi belgilardan foydalaniladi: (dumalok qavslar), (kvadrat qavslar), (figurali qavslar), (vergul) va "" (qo'shtirnoqlar).

Ushbu metabelgilar quyidagi ma`noni anglatadilar:

( ) – dumaloq qavslar, ularning ichidagi barcha sanab o'tilgan belgilar zanjiridan grammatika qoidasining ushbu joyida faqat bittagina zanjirdan foydalanish mumkinligini anglatadi.

[ ] – kvadrat qavslar, ularda keltirilgan zanjir grammatika qoidalarining ushbu joyida uchrashi ham, uchramasligi ham mumkinligini anglatadi (ya`ni unda bir marta yoki bir marta ham bo'lmasligi mumkin).

{ } – figurali qavslar, ularda keltirilgan zanjir grammatika qoidalarining ushbu joyida bir marta ham uchramasligi, bir marta uchrashi yoki juda ko'p marta uchrashi mumkinligini anglatadi.

, - verguldan, dumaloq qavslarning ichida belgilar zanjirini bo'laklash uchun foydalaniladi.

" " - qo'shtirnoqlardan metabelgilardan birontasini zanjirga qo'shib qo'yish kerak bo'lgan holda – ya`ni qavslarning birontasi yoki vergul tilning belgilar zanjirida mavjud bo'lishi kerak bo'lgan holda foydalaniladi (agar qo'shtirnoqning o'zini belgilar zanjiriga qo'shish kerak bo'lsa, u holda uni ikki marta takrorlash kerak)

YUqorida ko'rib o'tilgan G grammatikaning qoidalari ularni metabelgilardan foydalangan holda yozishda quyidagicha bo'ladi:

<son> -» [(+.-)]<raqam>{<raqam>}

<raqam> - > 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

qoidalarning ikkinchi qatori izohga muhtoj emas, lekin, birinchi qoida bunday o'qiladi: «son bu belgilar zanjiri bo'lib, u + yoki - belgilardan boshlanadi, so'ngra bitta raqamga ega bo'lishi kerak, undan so'ng ixtiyoriy sonli raqamlar ketma-ketligi kelishi mumkin». Bekus—Naur formasidan farqli ravishda, bu qoidalarni metabelgilardan foydalangan ko'rinishda yozishda, birinchidan grammatikadan tushunarsiz bo'lgan noterminal belgi «son» olib tashlandi, ikkinchidan esa rekursiyani to'liq yo'qotishga erishildi. Natijada grammatika tushunarlirok bo'lib qoldi.

Metabelgilardan foydalanib qoidalarni yozish ko'rinishi — bu grammatika qoidalarini ifodalash uchun qulay va tushunarli usuldir. U ko'pgina hollarda rekursiyadan, uni yaqinlashuv belgisi bilan almashtirib to'liq qutulish imkonini yaratadi (figurali qavslar). Ushbu ko'rinish ayniqsa regulyar grammatikalar uchun ko'proq qo'llaniladi.

4.7.Grammatika qoidalarini graf ko'rinishida yozish

Grammatika qoidalarini graf ko'rinishida yozishda barcha grammatika maxsus qurilgan diagrammalar to'plami sifatida (2-jadval) ifodalanadi.

Ushbu ko'rinish Pascal tili grammatikasini ifodalashda taklif etilgan bo'lib, keyincha adabiyotlarda keng tarqalgan. U grammatikaning barcha turlari uchun qo'llanila olmaydi, undan faqat grammatikaning kontekst-ozod va regulyar turlari

uchun foydalanish mumkin, bu esa bizga tanish dasturlash tillarini grammatikasini ifodalash uchun etarlidir.

Grammatika qoidalarini graf ko'rinishida yozishda grammatikaning har bir noterminal belgisiga yo'naltirilgan graf ko'rinishidagi diagramma (2-jadval) mos keladi.

Graf cho'qqilarning quyidagi turlariga ega:

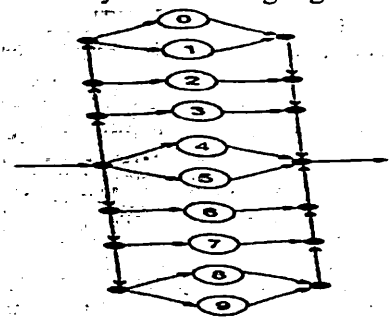
- kirish nuqtasi (diagrammada hech qanday belgilanmaydi, undan grafning kiruvchi qobig'i boshlanadi);
- noterminal belgi (diagrammada ichida noterminal belgi yozilgan to'g'rito'rtburchak bilan belgilanadi);
- terminal belgilar zanjiri (diagrammada ichida belgilar zanjiri yozilgan oval, aylana yoki chetlari aylanali to'g'rito'rtburchak bilan belgilanadi);
- bog'lash nuqtasi (diagrammada bo'yalgan aylana yoki qora nuqta bilan belgilanadi);
- chiqish nuqtasi (diagrammada hech qanday belgilanmaydi, unga grafning chikuvchi qobig'i kiradi);

Har bir diagramma (13-rasmga qarang) faqat bitta kirish va bitta chiqish nuqtasiga ega bo'ladi, lekin boshqa uch toifadagi cho'qqilarning ixtiyoriy soniga ega bo'lishi mumkin. CHo'qqilar o'zaro bir-birlari bilan grafning yo'naltirilgan qobiqlari bilan birlashtiriladi (ko'rsatkichli chiziqlar bilan). Kiruvchi nuqtadan qobiqlar faqat chiqishlari mumkin, kiruvchi nuqtaga - faqat kirishlari mumkin. Qolgan barcha cho'qqilarga qobiqlar kirishi ham, chiqishi ham mumkin (to'g'ri qurilgan grammatikada har bir cho'qqi juda bo'lmaganda bitta kirish va bitta chiqishga ega bo'lishi shart).

Grammatikaning qandaydir noterminal belgiga mos keladigan belgilar zanjirini qurish uchun ushbu belgi uchun diagrammani qarash kerak. U holda, xarakatni kiruvchi nuqtadan boshlab, diagramma grafining qobiqlari bo'ylab, ixtiyoriy cho'qqilar orqali, chiquvchi nuqtagacha davom ettirish kerak. Bu holda, noterminal belgi bilan belgilangan cho'qqidan o'tishda ushbu belgini natijaviy zanjirga joylashtiriladi. Terminal belgilar zanjiri bilan belgilangan cho'qqidan o'tishda, bu belgilarni ham natijaviy zanjirga joylashtiriladi. Diagrammaning bog'lash nuqtalari orqali o'tishda natijaviy zanjir ustida hech qanday xarakat bajarish kerak emas. Graf diagrammasining ixtiyoriy cho'qqisi orqali, xarakatlanishning mumkin bo'lgan yo'llariga bog'liq holda, bir marta o'tish mumkin, bir marta ham o'tmaslik mumkin yoki juda ko'p marta o'tish mumkin. Diagrammaning chiqish nuqtasiga etib kelishimiz bilan natijaviy zanjirni qurish tugatiladi.

Natijaviy zanjir o'z navbatida noterminal belgilarga ega bo'lishi mumkin. Ularni terminal belgilar zanjiriga almashtirish uchun, yana ularga mos diagrammalarni qarab chiqamiz. Jarayoni shu tartibda, belgilar zanjirida faqat terminal belgilargina qolgunicha, davom etadi. Ko'rinib turibdiki, berilgan tilning belgilar zanjirini qurish uchun, ishni grammatikaning maqsad belgisining diagrammasidan boshlash kerak.

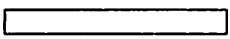
Bu grammatika qoidalarini ifodalashning eng qulay usuli bo'lib, obrazlar bilan ishlashi tufayli faqatgina insonlarga mo'ljallangan. Uning asosiy tamoyillarini sodda ifodalanishi ham juda bahaybat ko'rinishga ega bo'ladi.



13-rasm. Grammatika qoidalarini graf ko'rinishida yozishga doir qurilgan diagramma. Ishorali butun o'nlik sonlar grammatikasini graf ko'rinishida ifodalash

YUqorida aytib o'tganimizdek, ushbu usul asosan adabiyotlarda dasturlash tillarini ifodalashda qo'llaniladi. Dastur ishlab chiqaruvchi- foydalanuvchilar uchun bu usul qulay, lekin ulardan hozircha kompilyatorlarda amalda qo'llanilmayapti.

2-jadval.

Nomlanishi	Belgilanishi	Yo'naltirilishi
Kirish nuqtasi	Hech qanday belgilanmaydi	Undan grafning kiruvchi qobig'i boshlanadi.
Noterminal belgi		Ichida noterminal belgilarning belgilanishi keltirilgan
Terminal belgilar zanjiri		Unda terminal belgilar zanjiri yozilgan
Bog'lash nuqtasi		Chorraxa
Kirish nuqtasi	Hech qanday belgilanmaydi	Unga grafni chiquvchi qobig'i kiradi

4.8.Grammatikalar va tillarning sinflanishi

YUqorida grammatikalarning turli toifalari haqida gap yuritdik, lekin ularning qanday va qaysi tamoyillar asosida toifalarga bo'laklanishi ko'rsatilmadi. Insonlar uchun tillarning murakkab va sodda ko'rinishlari mavjud, lekin bu juda sub'ektiv fikr, chunki bu ko'pincha insonning shaxsiga bog'lik bo'ladi.

Kompilyatorlar uchun ham tillarni sodda va murakkab toifalarga ajratish mumkin, lekin bu holda bunday bo'laklash uchun juda qattiq kriteriylar mavjud bo'ladi. U yoki bu dasturlash tilining qanday toifaga tegishli ekanligidan ushbu tilni anglash murakkabligi bog'liq bo'ladi. Til qanchalik murakkab bo'lsa, kompilyatorning, ushbu tilda yozilgan boshlang'ich dasturni zanjirlarini tahlil qilish, xarajatlari shunchalik kattalashadi, bundan kelib chiqadiki, kompilyatorning o'zi ham

uning strukturasi ham murakkabdir. Tillarning ba'zi bir toifalari uchun ushbu tillardagi boshlang'ich matnlarni chekli hisoblash zahiralari asosida ma'lum vaqtda tahlil qila oladigan kompilyatorni qurish mumkin emas (xuddi shuning uchun hozirgi vaqtgacha tabiiy tillarda dasturlar yaratish mumkin emas /masalan rus yoki ingliz tilida).

4.9.Grammatikalar sinflanishi. Xomskiyning grammatikalari

Formal grammatikalar qoidalarining strukturasi bo'yicha sinflanadilar. Agar grammatika qoidalarining barchasi qandaydir berilgan strukturani qanoatlantirsalar, u holda uni aniq bir toifaga tegishli deb hisoblanadi. Agar, grammatikada qoidalar strukturasining talablarini qanoatlantirmagan bitta qoida bor bo'lsa ham, u grammatika berilgan toifaga tegishli bo'lmasligi uchun etarlidir. Xomskiy darajalanishi bo'yicha to'rt toifa grammatikani ko'rsatish mumkin.

0 toifa : fraza strukturali grammatikalar

Qoidalar strukturasiga hech qanday chegaralar qo'yilmaydi: $G(V_t,V_n,P,S)$, $V= V_tUV_n$ ko'rinishdagi grammatikalar uchun qoidalar quyidagi ko'rinishga ega: $\alpha\rightarrow\beta$, bu erda $\alpha\epsilon V^+$, $\beta\epsilon V^*$. Bu grammatikalarning eng umumiy toifasi. Unga barcha formal grammatikalar kiradilar, lekin ba'zi birlari boshqa toifa sinflariga tegishli bo'ladilar. Gap shundaki, 0 toifa grammatikalari boshqa toifalarga tegishli bo'la olmaydilar va strukturalari bo'yicha eng murakkab hisoblanadilar. Amaliyotda, ushbu toifaga tegishli grammatikalardan qo'llanilmaydi.

1-toifa: kontekst-bog'liq (KB) va qisqartirilmaydigan grammatikalar

Ushbu toifaga grammatikaning ikkita asosiy sinfi kiradi.

Birinchi tur: kontekst-bog'liq grammatikalar $G(V_t,V_n,P,S)$, $V=V_nuV_t$ quyidagi ko'rinishli qoidaga ega:

a) qisqartiruvchi: $\alpha_1A\alpha_2\rightarrow \alpha_1\beta\alpha_2$, bu erda α_1 va $\alpha_2\epsilon V^*$, $A\epsilon V_n$, $\beta\epsilon V^+$.

$G(V_t,V_n,P,S)$.

b) qisqartirilmaydigan grammatika: $G(V_t,V_n,P,S)$, $V=V_nuV_t$ quyidagi ko'rinishli qoidaga ega: $\alpha\rightarrow\beta$, bu erda α va $\beta\epsilon V^+$, $|\beta|>=|\alpha|$.

Kontekst –bog'liq grammatika qoidalari strukturasi ushbu til gaplarini qurish jarayonida bitta noterminal belgi u yoki bu zanjirga kontekstga bog'liq holda qo'shilgan bo'lishi mumkin va unda uchraydi: $\alpha_1A\alpha_2\rightarrow$(yoki α_1, yoki α_2 nolga teng). Xuddi shu sababli ularni kontekst –bog'liq grammatikalar deb ataladi.

Qisqartirilmaydigan grammatika - qoidalarning shunday strukturasiga egaki, berilgan grammatika asosida, til gaplarini qurishda ixtiyoriy belgilar zanjiri o'zining uzunligidan kam bo'lmagan belgilar zanjiriga almashtirilishi mumkin ya'ni A va β zanjirlar uzunligi bir xil. Ushbu ikki sinf grammatikasi ekvivalent ekanligi isbotlangan. Bu esa kontekst-bog'liq grammatika yordamida berilgan ixtiyoriy til uchun qisqartirilmaydigan grammatikani qurish mumkin va bu grammatika ekvivalent tilni beradi, va aksincha ixtiyoriy qisqartirilmaydigan

grammatika yordamida berilgan til uchun kontekst-bog'liq grammatikani qurish mumkin va u ekvivalent tilni beradi. Kompilyatorlarni qurish jarayonida ushbu grammatikadan foydalanilmaydi, chunki dasturlash tillari soddarok strukturaga ega bo'lib, boshqa toifa grammatikalari yordamida qurilishi mumkin.

2 toifa: kontekst-ozod grammatikalar (KO):

Kontekst-ozod grammatikalar $G(V_t,V_n,P,S)$, $V=V_n u V_t$ quyidagi qoidalarga ega: $A\rightarrow\beta$, bu erda $A\epsilon V_n$, $\beta\epsilon V^+$.

Bunday grammatikalarni qisqartirilmaydigan KO grammatikalar deb ham ataladi, chunki, qoidaning o'ng tomonida har doim, juda bo'lmaganda bitta belgi turishi shart.

Ularga deyarli ekvivalent grammatikalar sinfi mavjud: Qisqartiriladigan KO $G(V_t,V_n,P,S)$, $V=V_n u V_t$ quyidagi qoidalarga ega:: $A\rightarrow\beta$, bu erda $A\epsilon V_n$, $\beta\epsilon V^*$.

Ushbu ikki sinf grammatikasining farqi shundan iboratki, qisqartiriladigan KO grammatikada qoidaning o'ng qismida bo'sh belgilar zanjiri bo'lishi mumkin, qisqartirilmaydigan KO grammatikada esa yo'q.

Ushbu ikki sinf grammatikalari ekvivalentdirlar. KO grammatikadan dasturlash tillarining sintaksis konstruksiyalarini ifodalashda foydalaniladi. Ko'pgina dasturlash tillarining sintaksisi KO grammatikalarga asoslangan, shuning uchun ushbu kitobda ularga ko'p e`tibor beramiz.

3-toifa: regulyar grammatikalar

Regulyar grammatikalarga ikkita ekvivalent sinf grammatikasi tegishli: chapchiziqli va o'ngchiziqli.

CHapchiziqli grammatikalar $G(V_t,V_n,P,S)$, $V=V_n u V_t$ ikki ko'rinishdagi qoidalarga ega bo'lishi mumkin:

$A\rightarrow B\gamma$, $A\rightarrow\gamma$,bu erda A va B ϵV_n, $\gamma\epsilon V_t^*$,

O'z navbatida o'ng –chiziqli grammatikalar $G(V_t,V_n,P,S)$, $V=V_n u V_t$ ham ikki ko'rinishdagi qoidalarga ega bo'lishi mumkin::

$A\rightarrow\gamma B$, $A\rightarrow\gamma$, bu erda A va B ϵV_n, $\gamma\epsilon V_t^*$.

Ushbu ikki sinf grammatikalari ekvivalent va regulyar grammatikalar toifasiga tegishli. Regulyar grammatikadan dasturlash tillarining oddiy konstruksiyalari: identifikatorlarni, konstantalarni, qatorlarni, izohlarni va x.k. ifodalashda foydalaniladi. Kompilyatorda uning asosida kiruvchi tilning leksik tahlil funksiyalari quriladi.

Grammatikalar toifalari o'zaro muhim xususiyatga ega. 2 va 3 toifalar ta`rifidan ko'rinib turibdiki, ixtiyoriy regulyar grammatika KO grammatika hisoblanadi, lekin buning aksi emas. YAna shuni ta`kidlash kerakki ixtiyoriy grammatika 0 toifaga tegishli bo'lishi mumkin, chunki u qoidalarga hech qanday chegaralar qo'ymaydi. SHu bilan birga shunday qisqartiriladigan KO grammatikalar mavjudki (2-toifa), ular kontekst-bog'liq ham emas, qisqartirilmaydigan grammatika (1-toifa) ham emas, chunki ular 1-toifada ko'zda tutilmagan $\alpha\rightarrow\beta$ qoidaga egadirlar.

Umuman aytish mumkinki, grammatikaning murakkabligi, o'zi maksimal tegishli bo'lishi mumkin bo'lgan, toifa tartib raqamiga teskari proporsionaldir. Demak, 0-toifaga tegishli bo'lgan grammatikalar eng murakkab hisoblanadilar, 3-toifaga tegishli grammatikalar esa eng sodda grammatikalar hisoblanadilar.

O'ng tomonga to'g'rilangan grammatika holida grammatikaning o'ng qismida bittadan ko'p bo'lmagan noterminal mavjud bo'ladi va u eng o'ng belgi bo'ladi:

$A \rightarrow a$

$A \rightarrow aA$

CHap tomonga to'g'rilangan grammatika holida grammatikaning o'ng qismida bittadan ko'p bo'lmagan noterminal mavjud bo'ladi va u eng chap belgi bo'ladi:

$A \rightarrow a$

$A \rightarrow Va$

3 tur grammatikasining barcha ko'rinishlarini o'z ichiga olgan shajara 2 tur grammatikasi va x.k. hisoblanadi.

4.10.Tillarning sinflanishi

Tillar o'zlari yaratilgan grammatika toifalariga mos ravishda sinflanadilar. Bitta til umumiy holda juda ko'p sonli, turli toifalarga tegishli, grammatikalar yordamida berilishi mumkin bo'lganligi uchun, tilni o'zini sinflashtirish uchun grammatikalar orasidan eng maksimal mumkin bo'lgan grammatikaning sinf toifasi tanlanadi. Masalan, agar L til G_1 va G_2 kontekst-bog'liq toifaga tegishli, G_3 2-toifaga (kontekst-ozod) tegishli, G_4 3-toifaga (regulyar) tegishli grammatikalar yordamida berilgan bo'lsa, u holda tilning o'zi 3-toifaga tegishli bo'lib, regulyar til hisoblanadi.

Tilning sinfiy toifasidan, qanday grammatika yordamida, ushbu tilni gaplarini qurishgina emas, balki ushbu gaplarni anglash murakkabligi ham bog'liq bo'ladi. Gaplarni anglash — bu til uchun anglovchini qurish demakdir (tilni berishning uchinchi usuli). Anglovchilar, ularning strukturasi va sinflanishi keyinchalik alohida mavzu sifatida ko'riladi, hozircha tilni anglash murakkabligi til tegishli bo'lgan toifa sinfidan bog'liq bo'lishini ko'rsatish kifoya. Til murakkabligi til toifasini sinflanish raqamini pasayish tartibida keladi. Demak, eng murakkab til –bu 0-toifa til, eng sodda til 3-toifa til hisoblanadi.

Grammatikalar sinflanishiga mos ravishda to'rt toifa tillar ham mavjud.

0-toifa: fraza strukturali tillar

Bu eng murakkab tillar bo'lib, ular faqatgina 0-toifa grammatika yordamida beriladi. Bunday tillarning zanjirlarini anglash uchun Tьyuring mashinasiga teng kuchli hisoblagichlar talab etiladi. SHuning uchun, agar til 0- toifaga tegishli bo'lsa, u holda til uchun cheklangan hisoblash zahiralari asosida, cheklangan vaqt davomida tilning gaplarini bo'laklash tahlilini bajarishni kafolatlaydigan kompilyatorni qurish mumkin emās deb ayta olamiz. Demak, insonlar orasida muloqotni amalga oshiradigan barcha tabiiy tillar 0-toifaga tegishli va shu sababli, til kompilyatorini qurish mumkin emas. Gap shundaki, tabiiy til frazasining struktura va qiymati faqatgina ushbu fraza kontekstidan bog'liq bo'lib qolmay,

balki ushbu fraza uchraydigan matn mazmunidan ham bogliq bo'ladi. Tabiiy tilda bitta so'z kontekstdan bog'liq holda turli ma`noga ega bo'libgina qolmay, balki gapda turli o`rin egallashi ham mumkin. SHuning uchun, tabiiy tilda yozilgan matnlarni tarjima qilishni avtomatlashtirish murakkabligi yuqoridir, va bunday tillar asosida dasturni tushunuvchi kompilyatorlar ham mavjud emas.

1-toifa: kontekst-bog'lik (KB) tillar

1-toifa til toifalari murakkabligi bo'yicha ikkinchi til. Umumiy holda, 1-toifa grammatikasi orqali beriladi, tilning gaplarini anglash vaqti boshlang'ich belgilar zanjirining uzunligiga eksponensial bog'liq.

1-toifaga tegishli tillar va grammatikalar matnni tabiiy tilga tahlili va o'girishda foydalaniladi. Ular asosida qurilgan anglovchilar kiruvchi til gaplaridagi kontekst –bog'liqlikni hisobga olgan holda matnni tahlil qilish imkoniyatiga egadirlar (lekin, ular matn ma`nosini hisobga olmaydilar, shuning uchun tabiiy tildan aniq tarjima uchun insonning ishtiroki talab etiladi).

Bunday grammatikalar asosida bir tabiiy tildan boshqa tabiiy tilga avtomatlashtirilgan tarjima bajarilishi mumkin, ulardan til prosessorlarida orfografik va to'g'ri yozishni tekshiradigan servis funksiyalarda foydalaniladi.

Kompilyatorlarda KB –tillardan foydalanilmaydi, dasturlash tillari juda sodda strukturaga ega va ularni bu erda to'liq qarab o'tirmaymiz.

2-toifa: kontekst-ozod (KO) tillar

Kontekst-ozod tillar 2-toifa grammatikasi orqali beriladi, zamonaviy dasturlash tillari sintaksis konstruksiyasi asosida yotadi va ular asosida ba`zi bir murakkab buyruqli, takrorlash va shart buyruqlarini boshqaruvchi, prosessorlar xarakatlanadi. YUqoridagi xususiyatlar ushbu sinf tillarini keng tarqalishi sabablarini aniqlaydi. Umuman 1- toifaga tegishli tilni gaplarini anglash vaqti kiruvchi zanjirning uzunligidan polinomial bog'liq (til sinfidan bog'liq holda bu yoki kvadratik, yoki kubik bog'liqlik bo'lishi mumkin).

Lekin KO tillar orasida tillarni ko'p sinflari mavjudki, ular uchun bu bog'liqlik chiziqli bo'ladi. Ko'pgina dasturlash tillarini shunday sinflarning biriga tegishli deb hisoblash mumkin. Kontekst-ozod tillar dasturlash tillarning sintaksis konstruksiyalari asosida yotadi.

KO-tillar ushbu kitobning «Kontekst–ozod tillar» 5.11-bo'limida to'liq o'rganib chiqiladi.

3-toifa: regulyar tillar

Regulyar tillar- tillar eng sodda toifasidir. Va shu sababli, ular hisoblash tizimlari sohasida, eng keng tarqalgan va foydalaniladigan tillardir. Regulyar tilni gaplarini anglash vaqti kiruvchi belgilar zanjirning uzunligidan chiziqli bog'liq. Regulyar tillar dasturlash tillarining oddiy konstruksiyalari (identifikatorlar, konstantalar va x.k.) asosida yotadi. Ular asosida ko'pgina mashina buyruqlarini

mnemokodlari (assembler tillari) quriladi, bundan tashqari komanda prosessorlari, belgili boshqaruvchi buyruqlar va boshqa shunga o'xshash strukturalar quriladi.

Regulyar tillar –bu juda qulay vositadir. Ular bilan ishlash uchun regulyar to'plamlar va ifodalardan, chekli avtomatlardan foydalanish mumkin.

3-toifa grammatikasidan dasturlash tillarining bir necha xususiyatlarini yoki apparaturalarning ifodalashning yuqori daraja tillarni ifodalash uchun foydalaniladi. Masalan, ko'pgina dasturlash tillarining identifikatorlarini generasiyalash uchun quyidagi qoidalardan foydalanish mumkin.

$L \rightarrow l$	$R \rightarrow d$
$L \rightarrow lR$	$R \rightarrow lR$
$R \rightarrow l$	$R \rightarrow dR$

Bu erda (l) va raqam (d) terminal belgilarni anglatadi.

Ko'pincha qoidalarning bir xil o'ng va chap tomonlarini birlashtirish qulay.

YUqorida keltirilgan grammatikani quyidagi ko'rinishda yozish mumkin:

$L \rightarrow l \mid lR$

$R \rightarrow l \mid d \mid lR \mid dR$

Vertikal chiziq «yoki» ma`nosini anglatadi.

Dasturlash tillarining ko'pgina «lokal» qurilmalari, masalan konstantalar, kalit so'zlar va qatorlar 3-toifa grammatika yordamida ifodalanadi. Apparaturaning juda sodda ifodalash tillarini regulyar grammatika yordamida ifodalash mumkin. Lekin 3-toifa grammatikasi faqatgina qat`i cheklangan tillarning turlarini ya`ni regulyar ifodalarni generasiyalaydi.

A alfavitda regulyar ifodalarga quyidagilar kiradi:

1. Element A (yoki bo'sh qator).

Agar P va Q regulyar ifodalar bo'lsa, u holda quyidagilar ham regulyar ifoda hisoblanadilar:

2. PQ (Q P dan keyin keladi)
3. P | Q (P yoki Q)
4. P * (nol yoki P ning ko'p ekzemplyarlari) .

Alfavitda {a,b,c} ab*|ca* - regulyar ifoda va u quyidagi qatorlarni o'z ichiga oluvchi tilni ifodalaydi: abb c caaa ab ca

Regulyar ifodaga misol.

Identifikatorni ifodalovchi regulyar ifoda quyidagi ko'rinishga ega:
L(L\D)*, bu erda L xarfni anglatadi, D raqamni anglatadi.

Regulyar ifodalarda chegaralar mavjud. Masalan, regulyar ifodalar ixtiyoriy uzunlikdagi qavslar shablonini bera olmaydi va mos ravishda ularni 3 tur grammatikasi yordamida generasiya qilib bulmaydi.

Noregulyar ifodaga misol.

Ochilgan va yopilgan qavslardan tashkil topgan tilni qaraymiz (bo'sh qatorni ham qo'shgan holda) u quyidagi xususiyatlarga ega:

1) CHapdan o'ngga o'qish jarayonida uchragan yopilgan qavslar soni hech qachon ochilgan qavslar sonidan oshib ketmaydi.
2) Har bir qatorda ochilgan va yopilgan qavslar soni bir xil bo'ladi.

Masalan: quyidagi qatorlar tilga tegishlidir.

( ) (( ) ( ) (( )))
(( ) ( ) (( ) (( )))) (( ))

Quyidagilar esa tegishli emas:
((( ) ( )) 2 qoidaga mos emas.
( ))) ((( ) 1 qoidaga mos emas.

Ushbu tilning regulyar ifodalar yoki 3- toifa grammatikasi yordamida generasiyalash usuli mavjud emas. Lekin ushbu tilni quyidagi ozod –kontekst grammatikasi yordamida qurish mumkin:

$S \rightarrow (S)$
$S \rightarrow SS$
$S \rightarrow \varepsilon$

Ko'pgina dasturlash tillarida va apparaturani ifodalash tillarida shunday qavslar juftligi borki, ularni kelishi zarur, masalan:

( ), [ ], begin end har bir ochilgan qavsga yopilgan qavs mos kelishi kerak.

Masalan begin ( ) end - to'g'ri
begin (end) - noto'g'ri.

Kontekst-ozod grammatika ushbu chegaralarga yo'l qo'yadi. Dasturlash tillari sintaksisining va maxsus loyixalashni avtomatlashtirish tizimlari tillarining ko'pgina qismi kontekst-ozod grammatika orqali ifodalanadi. Ammo ko'pgina tillarda ba`zi bir xususiyatlar borki, ularni kontekst –ozod grammatika orqali ham ifodalab bo'lmaydi.

Masalan: X:=U mumkin bo'lishi mumkin, agar X va U mos toifaga ega deb e`lon qilingan bo'lsa, yoki mumkin bo'lmasligi mumkin, agar toifalar mos kelmasalar. Bunday shartlarni kontekst-ozod grammatika yordamida amalga oshirib bo'lmaydi, shuning uchun kompilyatorlar ko'pincha tekshiruvni formal sintaksis tahlil fazasida bajara olmaydilar. Lekin kontekst-ozod grammatika g'oyasini, tilning ba`zi bir kontekst-bog'liq xususiyatlarini qo'shgan holda, kengaytirish mumkin.

Regulyar tillar kitobning keyingi 5.6-bo'limida ko'riladi.

Xomskiy darajasi turli tilli translyatorlarni qurish nuqtai nazaridan muhimdir. Grammatikada qanchalik kam chegaralar mavjud bo'lsa, generasiya qilinayotgan tilga qo'yiladigan chegaralar shunchalik murakkabdir. Foydalanilayotgan grammatika sinfi qanchalik universal bo'lsa, biz tilning xususiyatlarini shunchalik

ko'proq ifodalay olamiz. Lekin, grammatika qanchalik universal bo'lsa, ushbu tilni anglovchi dastur shunchalik murakkab bo'ladi.

4.11.Tillar va grammatikalarni sinflanishiga doir misollar

Tillarni sinflanishi soddadan murakkabga qarab yuriladi. Agar biz regulyar tilni qurayotgan bo'lsak, u holda u ham kontekst-ozod, ham kontekst-bog'liq, ham fraza strukturali ekanligini ta`kidlashimiz mumkin. SHu bilan birga, shunday KO-tillar mavjudki ular regulyar til bo'la olmaydilar, va shunday KB –tillar mavjudki ular regulyar til ham, KO-til ham bo`la olmaydilar.

Endi ko'rsatilgan toifalarga doir misollarni ko'rib chiqamiz.

Ishorali butun o'nlik sonlar uchun quyidagi grammatika berilgan bo'lsin.

G({0,1,2,3,4,5,6,7,8,9,-,+},{S,T,F},P,S):

R:

S -> T | +T | -T

T -> F | TF

F - > 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Ushbu G grammatika o'z qoidalari strukturasiga ko'ra kontekst-ozod grammatikalarga (2-toifa) tegishlidir. Albatta uni 0-toifa va 1-toifaga tegishli deyish mumkin, lekin maksimal holda 2-toifaga tegishlilik to'gri bo'ladi. Bu grammatikani 3-toifa deyish mumkin emas, chunki T—»F | TF kator T—»TF qoidaga ega, bu esa, boshqa barcha qoidalar ushbu toifaga mos kelsalar ham, 3-toifa uchun mumkin bo'lmagan holat, ya`ni bittagina mos kelmaslik holati ham etarlidir.

Ushbu (ishorali butun o'nlik sonlar) til uchun yana boshqa grammatikani qurish mumkin.

G'({0,1,2,3,4,5,b,7,8,9,-,+},{S,T},P,S):

R

S->T|+T|-T

T->0|1|2|3|4|5|6|7|8|9|OT|IT|2T|ZT|4T|5T|6T|7T|8T|9T

G' grammatika o'zining qoidalari strukturasi bo'yicha o'ngchiziqlidir va regulyar (3-toifa) grammatika hisoblanishi mumkin.

Xuddi shu grammatika uchun ekvivalent chapchiziqli grammatikani qurish mumkin(3-toifa).

G"({0,1,2,3,4,5,6,7,8,9,-,+},{S, T},P,S):

R

T->+ T|-T|T

S->TO|T1|T2|TZ|T4|T5|T6|T7|T8|T9|SO|S1|S2|S3|S4|S5|S6|S7|S8|S9

Demak, G, G' va G" grammatikalar orqali berilgan ishorali butun o'nlik sonlar regulyar tillarga (3-toifa) tegishlidir.

Ikkinchi misol sifatida quyidagi grammatikani olamiz:

G_2({0,1},{A,S},P,S)

R: S -> 0A1 OA -» 00A1 A -> λ

Bu grammatika 0-toifaga tegishlidir. U gaplari to'plamini quyidagicha yozish mumkin bo'lgan tilni aniqlaydi: $L(G_2) = \{0^n1^n \mid n > 0\}$.

Xuddi shu til uchun kontekst-bog'liq grammatikani qurish mumkin:

$G_2'(\{0,1\},\{A,S\},P',S)$

P':

S -> 0A1 | 01 OA -> 00A1 | 001

Xuddi shu til uchun kontekst-ozod grammatikani ham qurish mumkin:

$G_2''(\{0,1\},\{S\},P'',S)$

R'': S —> 0S1 | 01

Demak, $L = \{0^n1^n \mid n > 0\}$ til kontekst-ozod (2-toifa) til hisoblanadi.

Uchinchi misolda quyidagi R qoidalarga asoslangan, grammatikani qaraymiz.

$G_3(\{a,b,c\},\{B,C,D,S\},P,S)$

R:

S-»BD

V->aVbS|ab Sb -> bS CD -> Dc bDc->bccabD-»abc

Bu grammatika 1-toifaga tegishlidir va u qisqarmaydigan grammatika hisoblanadi. U gaplari to'plamini quyidagicha yozish mumkin bo'lgan tilni aniqlaydi: $L(G_3) = \{a^nb^nc^n \mid n > 0\}$. Ushbu til kontekst-ozod til emasligi aniq, shuning sababli, u uchun 2-yoki 3-toifa grammatikalarini qurish mumkin emas. L - $\{a^nb^nc^n \mid p > 0\}$ til kontekst-bog'liq (1-toifa) hisoblanadi. Albatta, ixtiyoriy, qandaydir grammatika orqali, berilgan til uchun umumiy holda, uning toifasini aniqlash oson ish emas. Ixtiyoriy til uchun maksimal mumkin bo'lgan grammatika toifasini qurish, har doim ham, mumkin emas. Bundan tashqari yana toifani qat`iy aniklashda, ikkita grammatikaning (avval mavjud bo'lgan va yangi qurilgan) ekvivalent ekanligini isbotlash kerak bo'ladi. Bu esa juda ham oson echish mumkin bo'lgan masala emas.

Ko'pgina tillarda, xususan kontekst-ozod va regulyar tillarda ham maxsus ifodalangan ta`kidlar mavjud bo'lib, ular tilni ko'rsatilgan toifaga tegishli yoki yo'qligini tekshirish imkoniyatini beradilar. Bunday ta`kidlar (lemmalar) ushbu kitobning boshqa mos bo'limlarida ko'rib chiqiladi. U holda ixtiyoriy til uchun kerakli ta`kidni isbotlash etarli bo'ladi va undan so'ng ushbu til u yoki bu toifaga tegishli ekanligini aniq aytish mumkin. Grammatikalarni o'zgartirish bu holda talab etilmaydi.

Lekin shunga qaramay, gohida mavjud til grammatikasi uchun berilganidan ko'ra soddaroq grammatikani qurish masalasi yuzaga keladi. Xattoki til toifasi aniq bo'lgan hollarda ham, ushbu masala qiyin masala sifatida qolaveradi.

4.12.CHiqish zanjiri. Sentensial ko'rinish

CHiqish. CHiqish zanjiri

CHiqish deb, til gaplarini tilni aniqlovchi grammatika qoidalari asosida tug'ilish jarayoniga aytiladi. CHiqish jarayoniga formal ta`rif berish uchun yana bir nechta qo'shimcha tushunchalar kiritish zarur. R zanjir a zanjirdan bevosita keltirib

chiqariladi deyiladi, agar $G(V_t,V_n,P,S)$, $V = V_t \cup V_n$, grammatikada quyidagi qoida mavjud bo'lsa: a-»R. R zanjirni a zanjirdan bevosita keltirib chiqarish quyidagicha belgilanadi: a=>r. Boshqacha aytganda, r zanjir a zanjirdan chiqariladi, agar a zanjirdan bir nechta belgini olib, ularni grammatikaning mos qoidasiga asosan boshqa belgilarga o'zgartirish mumkin bo'lsa. Bevosita chiqishning formal ta`rifida zanjirlarning ixtiyoriysi bo'sh bo'lishi mumkin.

R zanjir a zanjirdan chiqariladi deyiladi (belgilanadi a=>*R), agar quyidagi ikki shart bajarilsa:

- r a dan bevosita chiqadi (a=>R);
- u, shundayki, u a va r dan bevosita chiqadi va bevosita yana chiqadi u (a=>*u va u=>R).

Bu zanjirlarni keltirib chiqarishning rekursiv ta`rifidir. Uning mazmuni shundaki, r zanjir a zanjirdan chiqadi, agar a=>r yoki a dan r gacha bo'lgan bevosita chiqarilish zanjirlari ketma-ketligini qurish mumkin bo'lsa. Bunday ketma-ketlikda har bir keyingi chiqariluvchi zanjir o'zidan oldingi zanjirdan bevosita chiqariluvchi bo'ladi. Bevosita chiquvchi zanjirlarning bunday ketma-ketligi chiqish yoki chiqish zanjiri deb ataladi. Bir bevosita chiqariluvchi zanjirning har bir, chiqish zanjiridagi keyingi o'tishi chiqish qadami deb ataladi. Ko'rinib turibdiki, chiqish zanjiridagi chiqish qadamlari har doim oraliq zanjirlardan bittaga ko'p bo'ladi. Agar r zanjir a zanjirdan bevosita chiqariluvchi bo'lsa: a=>r, u holda faqatgina bitta chiqish qadami mavjud bo'ladi.

Agar a zanjirdan r zanjirga chiqish zanjiri bitta yoki undan ko'p oraliq zanjirlarga ega bo'lsa (ikkita yoki ko'proq chiqish qadamlari), u holda u maxsus belgilashga ega bo'ladi a=>+R (r zanjir a zanjirdan netrivial chiqariluvchi deyiladi).

Agar chiqish qadamlari ma`lum bo'lsa, u holda uni bevosita zanjirlarni chiqarilishidagi belgi yonida ko'rsatish mumkin. Masalan, quyidagicha yozish $a=>^4R$ R zanjir a zanjirdan to'rt qadam orqali chiqarilishini anglatadi.

Misol sifatida yana o'sha ishorali butun o'nlik sonlar uchun grammatikani olamiz.

G({0,1,2,3,4,5,6,7,8,9,-,+},{S,T,F},P,S):

R:

S -> T | +T | -T

T -> F | TF

F -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Ushbu grammatikada bir nechta ixtiyoriy chiqish zanjirlarini qurishga xarakat qilamiz:

1.S => -T => -TF => -TFF => -FFF => -4FF => -47F => -479

2.S ==> T => TF => T8 => F8 => 18

3.T=>TF=>TO=>TR0 = T50 => F50 => 350

4.TFT => TFFT => TFFF => FFFF => IFFF => IFF4 => I0F4 = I004

5.F=>5

Quyidagi chiqishlarni oldik:

1. S => * -479 yoki S => + -479 yoki S => 7 -479
2. S => * 18 yoki S => + 18 yoki S => 5 18
3. T => * 350 yoki T => + 350 yoki T => 6 350
4. TFT => * 1004 yoki TFT => + 1004 yoki TFT => 7 1004
5. F => * 5 yoki F =>' 5 (F => + 5 ta`kid notugri!)

Barcha ushbu chiqishlar G grammatika asosida qurilgan. Bu grammatikada (aniq tilning ixtiyoriy boshqa grammatikasida ham) xoxlagan ko'p sonli chiqish zanjirlarini qurish mumkin.

Ikkinchi misol sifatida yana yuqoridagi qurilgan G_3 grammatikani olamiz. $G_3(\{a,b,c\},\{B,C,D,S\},P,S)$.

Adabiyotlarda quyidagi belgilashlar uchraydi: a=>°(a zanjir r zanjirdan 0 qadamda chiqariladi) yoki ushbu zanjirlar teng: a=r.

a=>*r chiqish, o'ziga quyidagi variantni oladi: a=> r. S-»BD

V->aVbS|abSb->bSCD-»DcbDc-*bccabD-»abc

Avval aytilganidek, ushbu grammatika $L(G_3) = \{0^n1^n \mid p > 0\}$ tilni beradi.

G grammatika asosida $L(G_3)$ tilni *«aaaabbbbcccc »* gapini chiqishini qarab chiqamiz.

S => BD => aBbCD => aaBbCbCD => aaaBbCbCbCD => aaaabbCbCbCD => aaaabbbCCbCD = aaaabbbCbCCD => aaaabbbbCCCD => aaaabbbbCCCDc => aaaabbbbCDcc => aaaabbbbDccc s aaaabbbbcccc.

U xolda G_3 grammatika uchun quyidagi chiqishni olamiz:

S => * aaaabbbbcccc.

Gohida, chiqish yurishini aniqlash uchun, ko'rsatkich ustiga, chiqishning har bir qadamini belgilovchi, grammatikaning qanday qoidasi asosida ushbu qadam amalga oshirilganligini bildiruvchi belgilash yoziladi (ushbu maqsad uchun grammatika qoidalarini ularning kelish tartibi bo'yicha raqamlash osonrokdir). Misollarda qurilgan grammatika 15 ta qoidalarga ega va chiqish zanjirlarining har bir qadamida ushbu qadam qanday qoida asosida amalga oshirilganligini tushunish mumkin, lekin murakkab holatlarda chiqish qadamlariga grammatika qoidalari raqamlarini ko'rsatish foydali bo'lishi mumkin.

4.13.Grammatikaning sentensial formasi. Grammatika orqali berilgan til

CHiqish tugallangan hisoblanadi, agar chiqish natijasida olingan r zanjir asosida boshqa bitta ham chiqish qadami qilish mumkin bo'lmasa. Boshqacha aytganda, chiqish tugallangan xisoblanadi, agar chiqish natijasida olingan r zanjir, bo'sh yoki $G(V_t,V_n,P,S)$ $r \epsilon V_t^*$ grammatikaning faqatgina terminal belgilardan tashkil topgan bo'lsa. Tugallangan chiqish natijasida olingan r zanjir chiqishning oxirgi zanjiri deb ataladi.

YUqorida ko'rilgan misolda barcha qurilgan chiqishlar tugallangan hisoblanadi, masalan, S =>* -4FF (misolda birinchi zanjirdan olingan) chiqish esa tugallanmagan hisoblanadi..

asV* belgilar zanjiri G(VG,V_n,P,S), V=V_tuV_n grammatikaning sentensial formasi deyiladi, agar, u grammatikaning S =>* a maqsad belgisidan chiqarilsa. Agar a∈V_t* zanjir tugallangan chiqish natijasida olingan bo'lsa, u holda u tugallangan sentensial forma deb ataladi.

YUqoridagi ko'rilgan misolga ko'ra, quyidagi zanjirlar ishorali o'nlik butun sonlar grammatikasining «—479>> va «18» tugallangan sentensial formasi hisoblanadilar, chunki S =>* -479 va S =>* 1 (1 va 2 misollar) chiqishlar mavjud. 2 – chiqishdan F8 zanjir, masalan, sentensial forma hisoblanadi, chunki S =>* F8, lekin u chiqish zanjiridagi oxirgi qadam emas. SHu bilan birga 3-5 misollarning chiqishlarida sentensial formalar mavjud emas. Haqiqatda esa, «350», «1004» va «5» zanjirlar oxirgi sentensial formalardir. Buni isbotlash uchun, chiqishning boshqa zanjirlarini qurish zarur (masalan, «5» zanjir uchun quramiz: S => T => F => va quyidagini olamiz S => * 5). «TFT» zanjirni esa (4 misol) S grammatikaning maqsad belgisidan keltirib chiqarish mumkin bo'lmaydi, shuning uchun u sentensial forma hisoblanmaydi. G(V_t,V_n,P,S) grammatika orqali berilgan til L — bu G grammatikaning barcha tugallangan sentensial formalari to'plamidir. G grammatika orqali berilgan L til, L(G) kabi belgilanadi. Bunday L(G) tilning alfaviti bo'lib, VT grammatikaning terminal belgilari to'plamini hisoblash mumkin, chunki grammatikaning barcha tugallangan sentensial formalari — bu VT alfavit ustidagi zanjirlardir. Ikkita grammatikalar G(V_t,V_n,P,S) va G'(V_t',V_n',P',S') ekvivalent deyiladi, agar ular orqali berilgan tillar: L(G) = L(G') ekvivalent bo'lsalar. Ko'rinib turibdiki, ekvivalent grammatikalar kesishuvchi terminal belgilar to'plamiga ega bo'lishlari shart, ya`ni V_tnV_t* (ko'pincha, bu to'plamlar xattoki mos tushadilar V_t=VI"), noterminal belgilar to'plamida esa grammatika qoidalari va maqsad belgi farq qilishi mumkin.

4.14.CHaptomonli va o'ngtomonli chiqishlar

CHiqish chaptomonli deyiladi, agar grammatika qoidalari chiqishning har bir qadamida har doim zanjirdagi eng chap noterminal belgiga nisbatan qo'llanilsa. Boshqacha aytganda, chiqish chaptomonli deyiladi, agar chiqishning har bir qadamida boshlangich zanjirdagi eng chetki noterminalni o'rniga grammatika qoidalari asosida zanjir belgilari almashtiriladi.

Xuddi shuningdek, chiqish o'ngtomonli deyiladi, agar grammatika qoidalari chiqishning har bir qadamida har doim zanjirdagi eng o'ng noterminal belgiga nisbatan qo'llanilsa.

Agar avvalgi misolda ko'rilgan chiqish zanjirlarini qarasak, u erda 1 va 5 chiqishlar chaptomonlidir, 2, 3 va 5 — o'ngtomonlidirlar (5 chiqish bir vaqtning o'zida ham chaptomonli, ham o'ngtomonlidir), 4 chiqish esa chaptomonli chiqish ham o'ngtomonli chiqish ham emas.

2 va 3 toifa grammatikalari uchun (KO-grammatikalar va regulyar grammatikalar) ixtiyoriy sentensial forma uchun har doim o'ngtomonli yoki chaptomonli chiqishlarni qurish mumkin. Boshqa toifa grammatikalari uchun bu har doim ham mumkin emas, chunki ularning qoidalari strukturasi bo'yicha eng chetki

chap yoki eng chetki o'ng noterminalni zanjirda almashtirishni bajarish har doim ham mumkin emas.

YUqorida ko'rib o'tilgan, S => * aaaabbbbcccc chiqish, $L(G_3) = \{O^n|^n|n > 0\}$ tilni beruvchi, G_3 grammatika uchun, chaptomonli chiqish ham, o'ngtomonli chiqish ham hisoblanmaydi. Grammatika 1- toifaga tegishli va bu holda u uchun har bir qadamda bitta noterminal belgi belgilar zanjirida almashtiriladigan chiqishni qurish mumkin emas.

4.15.CHiqish daraxti. CHiqish daraxtini qurish usullari

$G(V_t,V_n,P,S)$ grammatikaning chiqish daraxti – bu graf, u qandaydir chiqish zanjiriga mos keladi va quyidagi shartlarni qanoatlantiradi:

- daraxtning har bir cho'qqisi grammatikaning $A\epsilon(V_tuV_n)$ belgisi bilan belgilanadi;

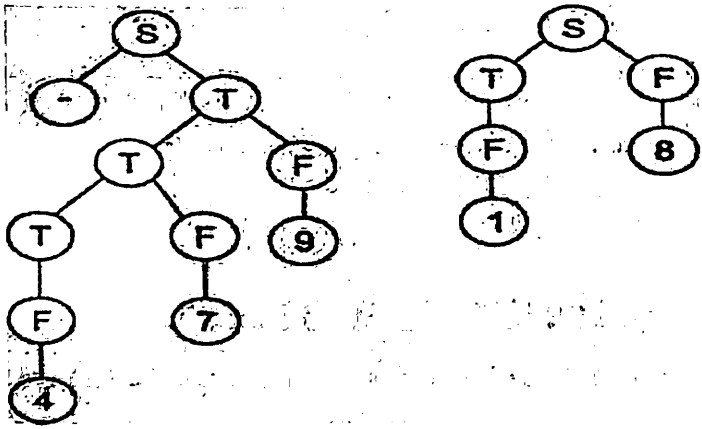
- daraxt ildizi bo'lib grammatikaning maqsad belgisi S hisoblanadi;

- daraxt barglari bo'lib har doim grammatikaning terminal belgilari yoki bo'sh belgilar hisoblanadilar;

- agar qandaydir bog'lam noterminal belgi A bilan belgilangan bo'lsa, unga bog'langan bog'lamlar $b1,b2,\ldots,b_n$ belgilar bilan belgilangan bo'lsalar, u holda G grammatikada $A \to b1,b2,\ldots,b_n$ qoida mavjuddir.

YUqoridagilardan kelib chiqib, chiqish daraxtini ko'rsatilgan ko'rinishda har doim kontekst-ozod (2-toifa) grammatika va regulyar (3-toifa) grammatikalar uchun qurish mumkin. Boshqa toifa grammatikalari uchun chiqish daraxtini ushbu ko'rinishda har doim ham qurish mumkin bo'lavermaydi.

YUqorida ko'rilgan misol uchun 1 va 2 chiqish zanjirlari uchun chiqish daraxtini quramiz. Bu daraxtlar 14-rasmda keltirilgan.



14-rasm. Ishorali butun o'nlik sonlar grammatikasi uchun chiqish daraxtlari misollari

CHiqish daraxtini qurish uchun. faqat chiqish zanjiriga ega bo'lish etarlidir. CHiqish daraxtini ikki usul bilan qurish mumkin: yuqoridan pastga va pastdan yuqoriga. Qat'iy ifodalangan chiqish daraxtini qurish uchun har doim kat'iy aniqlangan chiqishdan foydalanish kerak: yoki chaptomonli chiqish, yoki o'ngtomonli chiqish.

1.CHiqish daraxtini yuqoridan pastga qurishda qurish grammatikaning maqsad belgisidan boshlanadi, maqsad belgi daraxtning ildizida joylashadi. So'ngra grammatikada zarur qoida tanlab olinadi va birinchi chiqish qadamida ildiz belgi birinchi darajaning bir necha belgilariga yoyiladi. Ikkinchi va keyingi qadamlarda barcha oxirgi cho'qqilardan eng chetki (eng chetki chap –chap tomonli chiqish uchun va eng o'ng chetki - o'ng tomonli chiqish uchun) noterminal belgi bilan belgilangan cho'qqi tanlanadi, ushbu cho'qqi uchun grammatikaning kerakli qoidasi tanlanadi va u keyingi darajaning bir necha cho'qqilariga yoyiladi. Qurish barcha oxirgi cho'qqilar terminal belgilar bo'lgan holda tugallanadi, aks holda ikkinchi qadamnga qaytiladi va qurish davom ettiriladi.

2. CHiqish daraxtini qurish pastdan yuqoriga, ya'ni daraxt barglaridan boshlanadi. Barglar sifatida qurishning birinchi qadamida daraxtning oxirgi darajasini tashkil etuvchi tugallangan belgilar zanjirining terminal belgilari tanlanadi.

Daraxtni qurish darajalar bo'yicha olib boriladi. Qurishning ikkinchi qadamida grammatikada, o'ng qismi daraxt darajasida chetki belgilarga mos qoida tanlanadi (eng chetki o'ng –o'ng tomonli chiqish uchun va eng chetki chap – chap tomonli chiqish uchun). Darajaning tanlangan cho'qqilari, qoidaning chap bo'lagidan olingan, yangi cho'qqi bilan birlashtiriladi. YAngi cho'qqi daraxt darajasiga tanlangan cho'qqilar o'rniga tushadi. Daraxtni qurish ildiz cho'qqiga (maqsad belgi bilan belgilangan) etib borilgach tugatiladi, aks holda ikkinchi qadamga qaytiladi va qaytatdan boshlanadi. Bu usulda qurish teskari tartibda olib boriladi, ya'ni daraxt barglaridan boshlanadi.

Biz bilamizki, barcha mashhur dasturlash tillari «chapdan –o'ngga» notasiya yozuviga ega, kompilyator ham kiruvchi dasturni chapdan o'ngga qarab o'qiydi (va yuqoridan pastga, agar dastur bir nechta qatorlarga bo'lingan bo'lsa). SHuning uchun chiqish daraxtini «yuqoridan pastga» usuli bilan qurish uchun, chapchiqishli usuldan foydalaniladi, «pastdan –yuqoriga» usuli bo'yicha esa o'ngtomonli chiqishdan foydalaniladi. Kompilyatorlarning ushbu xususiyatlariga e'tibor qaratish kerak. Dasturlarni «chapdan-o'ngga» o'qish notasiyasi kompilyator tomonidan dasturni ajratish tartibigagina ta'sir ko'rsatmay (foydalanuvchi uchun bu ahamiyatga ega emas), balki amallarni bajarilishi tartibiga ham ta'sir ko'rsatadi, ya'ni qavslarni yo'qligi sababli, ko'pgina tengkuchli amallar chapdan o'ngga tomon tartibda bajariladilar, bu esa muhim ahamiyatga ega.

4.16.Grammatikalarning birqiymatlilik va ekvivalentlik muammolari

Birqiymatli va birqiymatli bo'lmagan grammatikalar

Qandaydir G ({+,-,*,/,(,), a, b}, {S}, R, S) :

R: S -> S+S | S-S | S*S | S/S | (S) | a | b grammatikani ko'rib chiqamiz.

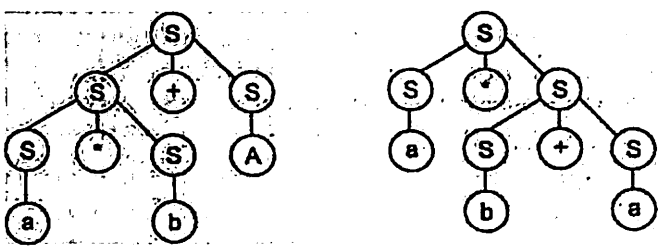
Ko'rinib turibdiki, keltirilgan grammatika to'rt asosiy amallar (qo'shish, ayirish, ko'paytirish va bo'lish), a va b operandlar, va qavslarga ega arifmetik ifodalarning tilini aniqlaydi. Ushbu tilning ilovalari bo'lib quyidagilar xizmat qilishi mumkin:: a*b+a, a*(a+b), a*b +a*a va x.k.

a*b +a zanjirni olamiz va u uchun chaptomonli chiqishni quramiz. Natijada ikkita variant olinadi:

S => S+S => S*S+S => a*S+S => a*b+S => a*b+a

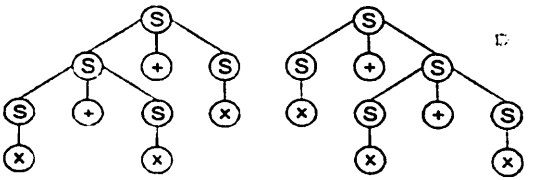
S => S*S => a*S => a*S+S => a*b+S => a*b+a .

Ushbu variantlarning har biriga o'z chiqish daraxti mos keladi. «a*b+a» zanjir uchun chiqish daraxtining ikki varianti 15-rasmda keltirilgan. Grammatika asosida berilgan formal til nuqtai nazaridan, mumkin bo'lgan variantlardan qanday chiqish zanjiri va qanday chiqish daraxti qurilishining ahamiyati yo'q. Ammo formal til hisoblanmagan dasturlash tillari uchun bu holat qandaydir ma`noviy xususiyatga egadir, bu esa ahamiyatga ega. Masalan, agar yuqorida ko'rilgan grammatika qandaydir arifmetik ifodalarning tilini aniqlasa, u holda arifmetika nuqtai nazaridan, chiqish daraxtini qurish tartibi arifmetik xarakatlarning bajarilish tartibiga mos keladi. Arifmetikada, bilamizki, qavslar bo'lmagan holda ko'paytirish amali qo'shish amalidan har doim avval bajariladi (chunki, ko'paytirish amali yuqori ustunlikka ega); lekin yuqorida ko'rilgan grammatikada bu hech qaerdan kelib chiqmaydi, unda barcha amallar tengkuchlidirlar. SHu sababli arifmetik amallar nuqtai nazaridan keltirilgan grammatika to'g'ri bo'lmagan semantikaga ega — unda amallar ustunligi yo'q, bundan tashqari, tengkuchli amallar uchun bajarilish tartibi aniqlanmagan («chapdan o'ngga»), ammo u yordamida qurilgan ifodalarning sintaksis strukturasi to'g'ri bo'ladi.



15-rasm. Arifmetik ifodalarning birqiymatli bo'lmagan grammatikasining «a*b+a» chiqish zanjiri daraxtining ikki varianti

Bunday holat grammatikada birkiymatli bo'lmagan grammatika deb ataladi. YAna bir misol, quyidagi tug'iluvchi qoidali grammatika uchun
S → S+S| x x+x+x gap ikkita sintaksis daraxtga (16-rasm), ikkita chap (o'ng) tomonli tahlilga ega.



16-rasm. Tahlil variantlari

S → S+S	S → S+S
→ S +S+S	→ x+S
→ x+S+S	→ x+S+S
→ x+x+S	→ x+x+S
→ x+x+x	→ x+x+x

Agar grammatikada generasiya qilingan qandaydir gap, bittadan ortiq tahlil daraxtiga ega bo'lsa, bunday grammatika haqida u bir qiymatli emas deyiladi. Ekvivalent shart shunda ko'rinadiki, gap bittadan ortiq chap yoki o'ng tomonli tahlilga ega bo'lishi kerak. Grammatikaning bir qiymatli emasligini o'rnatish masalasi umumiy holda echimi yo'q masaladir, ya`ni kirishda ixtiyoriy grammatikani qabul qiladigan va uni bir qiymatlimi yoki yo'qligini aniqlaydigan universal algoritm mavjud emas. Ba`zi bir bir qiymatli bo'lmagan grammatikalarni o'sha tilni generasiya qiladigan bir qiymatliga aylantirish mumkin. Masalan, quyidagi tug'iluvchi qoidalarga ega grammatika

S →x| S +x bir qiymatli bo'lib, u o'sha tilni xuddi avvalgi bir qiymatli bo'lmagan grammatika kabi generasiyalaydi.

Tabiiyki, kompilyatorlarni va dasturlash tillarini qurish uchun birqiymatli bo'lmagan grammatikalardan foydalanish mumkin emas. SHunday qilib birqiymatli bo'lmagan grammatikaga aniq ta`rif beramiz

Grammatika birqiymatli emas deb ataladi, agar tilning, ushbu grammatika tomonidan berilgan, har bir ixtiyoriy belgilar zanjiri uchun, yagona chaptomonli chiqishni (va yagona o'ngtomonli chiqishni) qurish mumkin bo'lsa. YOki, grammatika birqiymatli deyiladi, agar ushbu grammatika yordamida berilgan tilning ixtiyoriy belgilar zanjiri uchun yagona chiqish daraxti mavjud bo'lsa. Aks holda, grammatika birqiymatli emas deb ataladi.

YUqoridagi, misolda ko'rilgan, arifmetik ifodalarning grammatikasi, ko'rinib turibdiki, birqiymatli hisoblanmaydi.

4.17.Ekvivalentlik va birqiymatli bo'lmagan grammatikani birqiymatli grammatikaga aylantirish

Dasturlash tillarining grammatikalari har doim birqiymatli bo'lishi kerakligi sababli, u holda ixtiyoriy holda albatta echilishi kerak bo'lgan ikki masala mavjud:

1) Berilgan grammatika bir qiymatli yoki yo'qligini qanday tekshirish mumkin?
2) Agar berilgan grammatika birqiymatli bo'lmasa, uni qanday qilib birqiymatli ko'rinishga aylantirish mumkin?

Bir qiymatlilik -bu grammatikaning xususiyati, tilning emas. Ba`zi bir birqiymatli bo'lmagan grammatikalar orqali berilgan tillar uchun goxida birqiymatli ekvivalent grammatikani qurish mumkin bo'ladi (o'sha tilni beradigan birqiymatli grammatikani).

Qandaydir grammatika birqiymatli emasligiga ishonch hosil qilish uchun, ta`rifga asosan berilgan tilda juda bo'lmaganda bitta belgilar zanjirini, ya`ni bittadan ko'p chaptomonli yoki o'ngtomonli chiqishga ega, belgilar zanjirini topish etarli. Ammo har doim ham bunday belgilar zanjirini topish oson emas. Bundan tashqari, agar bunday belgilar zanjiri topilmasa, biz ushbu grammatika birqiymatli deb ishonch bilan ayta olmaymiz, chunki tilning barcha zanjirlarini qarab chiqish mumkin emas, chunki ularning soni cheksizdir. SHunday qilib, grammatikaning birqiymatliligini tekshirishning boshqa usullar zarur.

Agar grammatika bir qiymatli bo'lmasa, u holda uni birqiymatli ko'rinishga aylantirish zarur. Gohida bu mumkin bo'ladi. Masalan, yuqorida ko'rilgan a va b operandli arifmetik ifodalarning birqiymatli bo'lmagan grammatikasi uchun, unga ekvivalent quyidagi ko'rinishdagi birqiymatli grammatika mavjud

G'({+,-,*,/,(,),a,b},{S,T,E},P,S):

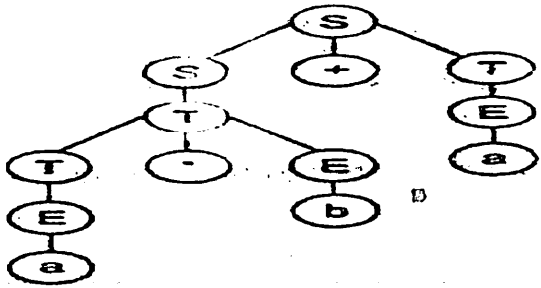
R':

S_>S+T|S-T|T
T->T*E|T/E|E
E->(S)|a|b.

Bu grammatikada avval ko'rilgan tilning a*b+a belgilar zanjiri uchun faqat bitta chaptomonli chiqish mumkin:

S⇒S+T⇒T+T⇒T*E+T⇒ E*E+T=* a*E+T⇒a*b+T⇒a*b+E⇒a*b+a.

Ushbu chiqishga yagona chiqish daraxti mos keladi. U quyidagi 17-rasmda keltirilgan. Ko'rinib turibdiki, chiqish zanjiri anchagina uzaygan bo'lsa ham, lekin bu holda amallar ustunligi yagona mumkin bo'lgan va arifmetikadagi qoidalar tartibiga mos keladi.



17-rasm. Arifmetik ifodalarning birqiymatli grammatikasi uchun chiqish daraxti

Bu holatda ikki muammoni hal qilish zarur: birinchidan, mavjud ikkita grammatikalar ekvivalent ekanligini isbotlash (ya`ni, bitta tilni beradilar); ikkinchidan, yangi qurilgan grammatika birqiymatli ekanligini tekshirish imkoniyatiga ega bo'lish.

Grammatikalarning ekvivalentligi muammosi umumiy holda quyidagicha ifodalanadi: ikkita G va G' grammatikalar mavjud, shunday algoritmni qurish kerakki, u ushbu ikki grammatika ekvivalent yoki yo'qligini tekshirish imkoniyatini bersin.

Grammatikalarning ekvivalentligi muammosi, umumiy holda algoritmik echimi yo'q masaladir. Bu esa hozirgacha ikkita berilgan grammatikalar ekvivalent yoki yo'qligini tekshiradigan algoritm mavjud emasligini va hech qachon tashkil etilmasligini anglatadi. Xuddi shuningdek, grammatikaning birqiymatli emasligini o'rnatish masalasi ham, umumiy holda echimi yo'q masaladir, ya`ni kirishda ixtiyoriy G grammatikani qabul qiladigan va uni birqiymatlimi yoki yo'qligini aniqlaydigan universal algoritm mavjud emas. YAna birqiymatli bo'lmagan G grammatikani unga ekvivalent bo'lgan G' birqiymatli grammatikaga aylantirish algoritmi ham mavjud emas. Ba`zi bir bir qiymatli bo'lmagan grammatikalarni o'sha tilni generasiya qiladigan bir qiymatliga aylantirish mumkin.

Ba`zi xususiy hollar uchun — masalan, grammatikaning aniqlangan toifalari va sinflari uchun (xususiy holda, regulyar grammatikalar uchun) —bu muammolar echilgan.

4.18.Grammatikaning birqiymatli emasligini beruvchi qoidalar

Umumiy holda, berilgan grammatika birqiymatli yoki birqiymatli emasligini tekshirish mumkin emas. Lekin, KO-grammatikalar uchun shunday aniqlangan qoidalar mavjudki, ular bo'yicha $G(V_t,V_n,P,S)$ grammatikaning qoidalari to'plamida ushbu grammatika birqiymatli emasligini ta`kidlash mumkin bo'ladi.

KO tillar uchun bir qiymatlilikni aniqlovchi qoidalar mavjud. Bu qoidalar quyidagi ko'rinishga ega:

1) $A \rightarrow AA|\alpha$

2) $A \to A\alpha A|\beta$

3) $A \to \alpha A|A\beta|\gamma$

4) $A \to \alpha A|\alpha A\beta A|\gamma$ Bu erda $A \in V_n$ $\alpha, \beta, \gamma \in (V_n \cup V_t)$

Agar berilgan grammatikada juda bo'lmaganda bitta shunga o'xshash (keltirilgan qoidalardan ixtiyoriysi) qoida uchrasa, u holda ushbu grammatika bir qiymatli emasligi aniq. Lekin, agar shunga o'xshash qoidalar grammatikaning barcha qoidalari to'plamida yo'q bo'lsa, u holda bu grammatika bir qiymatli ekanligini anglatmaydi. Bunday grammatika bir qiymatli bo'lishi ham mumkin, bo'lmasligi ham. YA`ni ko'rsatilgan ko'rinishdagi qoidalarning mavjud emasligi (barcha variantlar) – bu zaruriy, lekin grammatikaning birqiymatliligi uchun etarli bo'lmagan shartdir.

Boshqa tomondan, shunday shartlar o'rnatilganki, ular qanoatlantirilishi bilan grammatika birqiymatli hisoblanadi. Ular barcha regulyar va ko'pgina kontekst-ozod grammatika sinflari uchun to'g'ridir. Lekin, bu shartlar etarli hisoblanib, grammatikaning birqiymatliligi uchun zaruriy shartlar emas.

YUqorida ko'rilgan misolda arifmetik ifodalarning a va b operandli G({+,-,*,/,(,),a,b},{S},P,S) grammatikasida, R: S -> S+S|S-S|S*S|S/S|(S)|a|b qoidalar to'plamida 2-toifadagi qoidalar uchraydi. SHuning uchun ushbu grammatika birqiymatli bo'lmagan grammatika hisoblanadi bu esa yuqorida ko'rsatilgan edi.

Sinov savollari

1.Grammatika nima va u qanday beriladi?

2.Tilning terminal va noterminal belgilari qanday farqlanadilar?

3.U yoki bu belgining tilning gaplarida uchrashini qanday izohlaysiz?

4.«Boshlang'ich belgi» nima va u tilning boshqa belgilaridan nima bilan farq qiladi?

5.Sentensial qator nima?

6.CHiqish qatoriga ta`rif bering.

7.Grammatika quyidagi tug'iluvchi qoidalar orqali berilgan bo'lsin.

S ->(S) S->ε,
S->SS bu erda S – boshlang'ich belgi, ε bo'sh qatordir.

Quyidagi qatorlar ushbu grammatika bo'yicha generasiya qilingan tilga tegishlimi? Javobingizni isbotlang.

a) qator((1)()), v) qator (()()())

8.Xomskiy ierarxiyasi necha tur grammatikani o'z ichiga olgan? Ularning turlarini ayting.

9.Grammatikalarning darajalanish sabablarini izohlang.

10.Regulyar grammatikaning yutuq va kamchiliklari nimalardan iborat?

11.Identifikatorlar uchun regulyar grammatikani qurish. Identifikator harflar, raqamlar va belgilardan «_» va albatta harfdan boshlanadi.

12.SHunday regulyar grammatikani topish kerakki, u quyidagi tug'iluvchi qoidalarga ega grammatika kabi o'sha tilni generasiya qilsin (S-boshlang'ich belgi):

$S \rightarrow A \mid B$ $Y \rightarrow y \mid yY$
$A \rightarrow X \mid Y$ $B \rightarrow b \mid bB$
$X \rightarrow x \mid xX$

13.(101)* (010)* tilni generasiyalaydigan regulyar grammatikani quring.

5.BO'LIM
Anglovchilar. Tahlil masalasi
5.1.Anglovchining umumiy chizmasi

Har bir dasturlash tili uchun (va boshqa ko'pgina tillar uchun ham) ushbu tilda faqatgina dastur yozish emas, balki mavjud dastur matnini ushbu tilga tegishli ekanligi aniqlash ham muhim. SHuning uchun, ushbu masalani kompilyatorlar boshqa masalalar qatorida hal qiladilar (kompilyator boshlang'ich dastur matnini anglashigina emas, balki unga ekvivalent bo'lgan natijaviy dasturni qurishi ham kerak). Boshlang'ich dasturga nisbatan kompilyator anglovchi sifatida, qandaydir dasturlash tilida dasturni yozgan inson esa ushbu tilni belgilar zanjirini generatori sifatida chiqadilar.

Anglovchi (yoki tahlilchi) —bu maxsus algoritm bo'lib, u belgilar zanjirini qandaydir tilga tegishli yoki yo'qligini aniqlash imkonini beradi.

Anglovchining vazifasi, bu boshlang'ich kiruvchi zanjir asosida zanjirning ushbu tilga tegishli yoki yo'qligini aniqlashdan iboratdir.

Anglovchilar, avval yuqorida aytilganidak, tilni aniqlashning usullaridan birini ifodalaydilar.

Umumiy holda anglovchini shartli chizma ko'rinishida 18–rasm kabi aks ettirish mumkin.

Anglovchilar sinflanishi quyidagi elementlarni tashkil etuvchilardan amalga oshiriladi:

1.o'quvchi qurilmaning ko'rinishidan:
- birtomonlamalik – o'quvchi boshni lenta bo'ylab bir yo'nalish bo'ylab xarakatlanishini ta`minlaydi;

- ikkitomonlamalik.

Anglovchi ishining har bir qadamida o'quvchi bosh yoki berilgan yo'nalish bo'yicha qandaydir sonli o'ringa belgilar lentasi bo'ylab surilishi mumkin, yoki o'z o'rnida qolishi mumkin. Biz bilamizki, barcha dasturlash tillari boshlang'ich dasturni o'qishni «chapdan o'ngga» notasiyasidan foydalanadi, va yana xuddi shunday barcha anglovchilar ham ishlaydilar. SHuning uchun, birtomonli anglovchilar haqida gapirilganda, avvalombor chaptomonli ko'rinishga ega, boshlang'ich zanjirni chapdan o'ngga o'qiydigan va zanjirning o'qib bo'lingan qismiga orkaga qaytmaydigan, anglovchilar tushuniladi.

Ikkitomonli anglovchilar esa, o'quvchi boshni kiruvchi belgilar lentasiga nisbatan ikki yo'nalishda xarakatlanishiga imkon beradilar: lentani boshidan oxirigacha oldinga, va orqaga o`qib bo'lingan belgilarga qaytib.

2.Boshqaruv qurilmasining ko'rinishi bo'yicha determinirlangan va nodeterminirlangan bo'ladi:
- determinirlangan deb ataladi, agar anglovchi ishining, qandaydir qadamda yuzaga kelgan, har bir mumkin bo'lgan konfigurasiya uchun yagona mumkin bo'lgan konfigurasiya mavjud bo'lib, anglovchi unga o'z ishining keyingi qadamida o'tadi;

- nodeterminirlangan – bu anglovchi shunday mumkin bo'lgan konfigurasiyalarga egaki, ular uchun anglovchi ishining keyingi qadamida mumkin bo'lgan chekli konfigurasiyalar to'plami mavjud.

3.tashqi xotira ko'rinishi bo'yicha:
- tashqi xotirasiz; (agar avtomat tashqi xotirasiz bo'lsa, u holda boshqaruv qurilmasining xotirasidan foydalaniladi)

- chekli tashqi xotira;
- cheksiz tashqi xotira.

Tashqi xotirasiz anglovchilarda xotira butunlay mavjud bo'lmaydi. Ularning ish jarayonida boshqaruv qurilmasining xotirasidan foydalaniladi, tashqi xotiraga murojaat bajarilmaydi.

CHekli tashqi xotirali anglovchilar uchun, boshlang'ich belgilar zanjirining uzunligiga bog'lik holda, tashqi xotiraning o'lchami chegaralangan. Bu chegaralar zanjir uzunligidan xotira hajmiga qandaydir bog'liqlik asosida qo'yilishi mumkin — chiziqli, polinomial, eksponensial va x.k. Bundan tashqari, bunday anglovchilar uchun tashqi xotirani tashkil etish usuli ko'rsatilishi mumkin — stek, navbat, ro'yxat va x.k.

CHeksiz tashqi xotirali anglovchilar o'z ishlari uchun cheksiz hajmli tashqi xotira talab qilinadi deb o'ylaydilar (kiruvchi zanjirning uzunligiga bog'liq bo'lmagan holda). Bunday anglovchilarda ixtiyoriy murojaat usuliga ega xotira taklif qilinadi.

Ushbu uch tashkil etuvchilar birgalikda anglovchilarning umumiy sinflanishini tashkil etish imkonini yaratadilar. Masalan, ushbu sinflanishda shunday toifa bo'lishi mumkin: «ikkitomonli nodeterminirlangan chiziqli chegaralangan stek xotirali anglovchi».

Sinflanishdagi anglovchi toifasidan, anglovchini qurish murakkabligi aniqlanadi, demak, kompilyator uchun mos dasturiy ta`minotni ishlab chiqarish murakkabligi aniqlanadi. Sinflanishda anglovchi qanchalik yuqorida tursa, uni ishini ta`minlovchi algoritmni tashkil etish shunchalik murakkab bo'ladi. Ikkitomonli anglovchilarni ishlab chiqarish birtomonli anglovchilarni qurishga qaraganda murakkabroq. YAna, nodeterminirlangan anglovchilar determinirlangan anglovchilarga nisbatan murakkablik darajasi bo'yicha yuqoridirlar deyish mumkin. Tashqi xotira ko'rinishi bo'yicha algoritmni tashkil etish xarajatlarining bog'liqligi ham ko'rinib turibdi.

5.2.Tillar turlari bo'yicha anglovchilarning sinflanishi

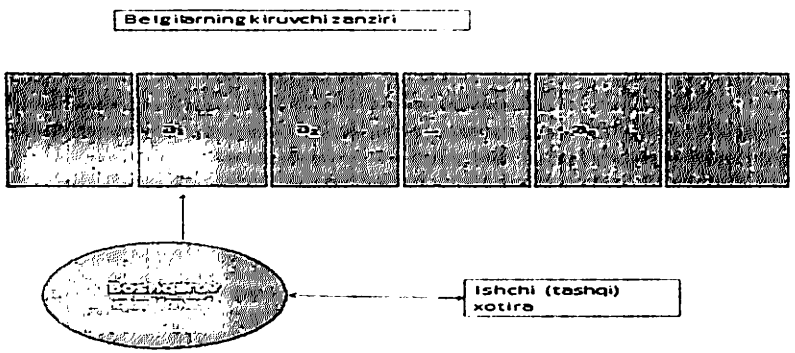
Avvalgi bo'limda aytilganidek, anglovchilarning sinflanishi (anglovchi tarkibiga kiruvchi komponetalar ko'rinishi) anglovchi ishining algoritmini murakkabligini aniqlaydi. Lekin, anglovchining murakkabligi yana to'g'ridan to'g'ri, anglovchi qabul qiladigan kiruvchi zanjir tegishli bo'lgan, til toifasi bilan ham bog'liq.

YUqorida to'rt toifa asosiy til toifalari aniqlangan edi. Ushbu tillarning har bir toifasi uchun, aniq tarkib komponentalariga va nihoyat ish algoritmining berilgan murakkabligiga ega bo'lgan, o'zi anglovchisining toifasi mavjud ekanligi isbotlangan.

1.Frazali strukturali tillar uchun (0 toifa) Tьyuring mashinasi qudratiga tengkuchli, ya`ni ikkitomonlik nodeterminirlangan cheksiz tashqi xotiraga ega avtomat anglovchi zarur.

SHuning uchun ushbu toifa tillari uchun, chekli vaqtda chekli hisoblash zahiralari asosida anglovchi ishini tugatishini va kiruvchi zanjir ushbu tilga tegishli yoki yo'qligini kafolatlash mumkin emas. Bundan shunday xulosa chiqarish mumkinki, fraza strukturali tillar amaliyotga qo'llanilishi mumkin emas.

2.Kontekst-bog'liq tillar (1-toifa) uchun anglovchilar bo'lib, ikkitomonli nodeterminirlangan chiziqli cheklangan tashqi xotirali avtomatlar hisoblanadilar. Bunday avtomatning ishlash algoritmi umumiy holda, eksponensial murakkablikka ega, ya`ni kiruvchi zanjirni anglash uchun avtomatga zarur qadamlar soni (taktlar), ushbu zanjirni uzunligidan eksponensial bog'liq. Va nihoyat, berilgan algoritm bo'yicha kiruvchi zanjirni tahlili uchun kerak bo'ladigan vaqt belgilar zanjirining uzunligidan eksponensial bog'liq bo'ladi. Anglovchining bunday algoritmi kompyuterning dasturiy ta`minotida amalga oshirilgan bo'lishi mumkin, chunki kiruvchi belgilar zanjirining uzunligini bilgan holda, har doim, qanday maksimal mumkin bo'lgan vaqt mobaynida zanjirning berilgan tilga tegishli yoki yo'qligini va qanday hisoblash zahiralari bu ish uchun talab etilishini aniqlash mumkin bo'ladi. Ammo tahlil vaqtining zanjir uzunligidan eksponensial bog'liqligi anglovchilarni konteks-ozod tillar toifasi uchun qo'llanilishini chegaralaydi. Ko'pincha, bunday anglovchilar tabiiy til matnlarini tahlili va avtomatlashtirilgan tarjimasi uchun qo'llaniladi va bu holda tahlil uchun vaqt chegaralari mavjud emas (tabiiy tillar kontekst-ozod tillarga qaraganda murakkabrok bo'lganliklari sababli, ularni avtomatik tarjimasidan so'ng, yana inson aralashuvi talab etiladi).



18- rasm. Anglovchining shartli chizmasi

SHuni ta`kidlash kerakki, keltirilgan 18-rasm anglovchining ishlash algoritmini aks ettiruvchi shartli chizmadir. Kompyuter tarkibida hech qachon bunday qurilmani qidirib yurmang. Anglovchi, kompyuter bo'lagi bo`lib, u kompyuterning dasturiy ta`minoti bo'lagini ifodalaydi.

18- rasmdan ko'rinib turibdiki, anglovchining asosiy komponentalari quyidagilardir:

1.Kiruvchi belgilar zanjiridan tashkil topgan lenta va ushbu zanjirdagi navbatdagi belgini ko'rsatuvchi o'quvchi bosh;

2.Anglovchini ishini koordinasiya qiluvchi, cheklangan holatlar soniga va katta xotiraga ega (o'zining holati va qandaydir oraliq ma`lumotlarni saqlash uchun), boshqaruv qurilmasi;

3.Ma`lumotlarni saqlash uchun foydalaniladigan tashqi xotira.

Anglovchi o'z alfavitining belgilari bilan ishlaydi. Anglovchi alfaviti chekli. U o'ziga kiruvchi zanjirning barcha mumkin bo'lgan belgilarini, shuningdek, qandaydir qo'shimcha belgilar alfavitini (boshqaruv qurilmasida qayta ishlanadigan va anglovchining ishchi xotirasida saqlanadigan) oladi.

Ish jarayonida anglovchi quyidagi elementar amallarni bajaradi:

- kiruvchi oqimdan navbatdagi belgini o'qish;
- kiruvchi zanjirni aniq sonli belgilarga surish;
- ma`lumotlarni o'qish yoki yozish uchun tashqi xotiraga murojaat;
-xotirada ma`lumotlarni aylantirish;
-boshqaruv qurilmasi holatini o'zgartirish.

Anglovchi taktlar yoki qadamlar bo'yicha ishlaydi. Anglovchining barcha ishi taktlar ketma-ketligidan tashkil topadi. Har bir takt boshida anglovchining holati uning konfigurasiyasi bilan aniqlanadi. Ishlash jarayonida anglovchining konfigurasiyasi o'zgaradi. Anglovchining konfigurasiyasi quyidagi parametrlar bilan aniqlanadi:

- kiruvchi belgilar zanjirining to'plami va undagi o'quvchi boshning holati bilan;
- boshqaruv qurilmasining holati bilan;
- tashqi xotiraning ichidagilar bilan.

Anglovchi uchun, har doim aniq boshlang'ich konfigurasiya hisoblangan, konfigurasiya beriladi, ya`ni boshlang'ich holat va chekli holatlar to'plami beriladi.

1. O'quvchi bosh kiruvchi belgilar zanjiridan boshlang'ich belgini ko'rsatadi;
2. Boshqaruv qurilmasi berilgan boshlang'ich holatda turadi;
3. Tashqi xotira yoki bo'sh, yoki qat`iy aniqlangan ma`lumotni saqlaydi.

Boshlang'ich holatdan tashqari anglovchi uchun bitta yoki bir nechta oxirgi konfigurasiyalar beriladi. Oxirgi konfigurasiyada o'quvchi bosh boshlang'ich zanjirning oxiridan tashqarida joylashadi (ko'pincha anglovchilar uchun, kiruvchi zanjir oxirini bildiruvchi, maxsus belgi kiritiladi). Anglovchi kiruvchi belgilar zanjirini anglaydi agar, boshlang'ich konfigurasiyada joylashib va ushbu zanjirni kirishiga olib, u, o'zining oxirgi konfigurasiyalaridan biri bilan tugovchi, qadamlar ketma-ketligini bosib o'tishi mumkin bo'lsa. «Qadamlar ketma-

ketligini bosib o'tishi mumkin bo'lsa» kabi ifodalanish, to'g'ridan-to'g'ri «qadamlar ketma-ketligini bajaradi» degan ifodalanishdan ko'ra aniqroq, chunki ko'pgina anglovchilar uchun boshlang'ich konfigurasiyadan boshlanuvchi bir kiruvchi belgilar zanjiri bo'yicha turli, hammasi ham oxirgi konfigurasiyaga olib keluvchi, qadamlar ketma-ketligi bo'lishi mumkin.

Anglovchi bilan aniqlangan til — bu anglovchi o'tkazishi mumkin bo'lgan barcha zanjirlar to'plamidir.

Keyinchalik ushbu kitobning 5.3-bo'limida anglovchilarning, turli toifa tillari uchun, aniq toifalari qaraladi. Ammo, hozirgacha aytilganlar barcha til toifalari uchun anglovchilar toifalariga tegishlidirlar.

5.3.Anglovchilarning ko'rinishlari

Anglovchilarni ularni tashkil etuvchilaridan bog'liq holda sinflarga bo'lish mumkin: o'quvchi qurilma, boshqaruv qurilmasi va tashqi xotira.

O'quvchi qurilma ko'rinishi bo'yicha anglovchilar ikkitomonli va birtomonli bo'ladilar.

Birtomonli anglovchilar o'quvchi boshni kiruvchi belgilar lentasi bo'ylab faqat bir yo'nalishda joylashtirish imkonini beradilar. Turli dasturlash tillarida leksik tahlil uchun kontekst –bog'liq anglovchilar qo'llanilmaydi, chunki kompilyatorning ishlash tezligi muhim ahamiyat kasb etadi, dastur matnini sintaksis tahlili uchun esa oddiyroq, kontekst-ozod til toifasidan foydalanish mumkin. SHu sababli, biz kontekst-bog'liq tillarni ham qarab o'tirmaymiz.

Kontekst-ozod tillar (2-toifa) uchun anglovchilar bo'lib tashqi magazin xotirali nodeterminirlangan birtomonli anglovchilar, ya`ni MP- avtomatlar hisoblanadilar. Bunday avtomatning ishlash algoritmini tashkil etishda u eksponensial murakkablikka ega, lekin algoritmni rivojlantirish yo'li bilan kiruvchi zanjirni uzunligiga bog'liq bo'lgan tahlil vaqtini polinomial (kub) holatga keltirish mumkin. SHunday qilib, KO-tillar uchun anglovchining polinominal murakkabligi haqida gapirish mumkin.

Barcha KO-tillar ichida determinirlangan KO-tillar sinfini ajratish mumkin, ular uchun anglovchilar bo'lib tashqi magazin xotirali (stek) determinirlangan avtomatlar, ya`ni DMP-avtomatlar hisoblanadilar. Ushbu tillar birqiymatlilik xususiyatiga ega bo'lib, ixtiyoriy determinirlangan KO-til uchun har doim birqiymatli grammatikani qurish mumkin ekanligi isbotlangan. Bundan tashqari, bunday tillar uchun kvadratik murakkablikka ega algoritm asosida ishlovchi anglovchi mavjud. Bu tillar birqiymatli bo'lganliklari sababli, xuddi ular kompilyatorlarni qurishda katta qiziqishga sabab bo'ladilar.

YAna barcha determinirlangan KO-tillar uchun shunday tillar sinfi mavjudki, ular uchun chiziqli anglovchini qurish mumkin, ya`ni bu anglovchining belgilar zanjirini tilga tegishli yoki yo'qligi haqidagi echimni qabul qilish vaqti zanjirning uzunligidan chiziqli bog'liq bo'ladi. Barcha mavjud dasturlash tillarining sintaksis konstruksiyalari shunday tillarning biron sinfiga kiritilishi mumkin. Bu xususiyat zamonaviy tez ishlovchi kompilyatorlarni ishlab chiqarish uchun juda muhimdir.

SHuning uchun, KO-tillarga bag'ishlangan 5.11-bo'limda, ushbu tillarning shunday sinflariga, birinchi navbatda, katta ahamiyat beriladi.

YUqoridagilarga qaramay, dasturlash tillarining faqat sintaksis konstruksiyalari KO-tillar anglovchilari yordamida tahlil qilish imkonini berishlarini esdan chiqarmaslik kerak. Dasturlash tillarining o'zi, avval aytilganidek, KO-tillar toifasiga to'liq kiritilishi mumkin emas, chunki boshlang'ich dastur matnida qandaydir kontekst-bog'liqlikka ega bo'ladilar (masalan, o'zgaruvchilarni avvaldan ifodalash zaruriyati kabi). SHuning uchun deyarli barcha kompilyatorlar sintaksis tahlildan tashqari, boshlang'ich dastur matnini qo`shimcha semantik tahlil qilishni ham ko'zda tutadilar. Bu holatdan, agar kontekst-bog'liq anglovchi asosida kompilyator qura olsak, va bunday kompilyatorning ishlash tezligi juda xam past bo'lmasa, qochib kolish mumkin edi, chunki bu variantda tahlil vaqti boshlang'ich dastur matnidan eksponensial bog'liq bo'ladi. KO-til anglovchisi va qo'shimcha semantik tahlilchi kombinasiyasi boshlang'ich dastur matnini tahlil qilish tezligi nuqtai nazaridan foydaliroq hisoblanadi.

Regulyar tillar (3-toifa) uchun anglovchilar bo'lib, tashqi xotirasiz nodeterminirlangan birtomonli avtomatlar, ya`ni chekli avtomatlar (KA) hisoblanadilar. Bu anglovchilarning eng sodda toifasi bo'lib, belgilar zanjirining uzunligiga bog'liq tahlil uchun ketadigan vaqt chiziqli bog'liq ekanligini ko'zda tutadi. Bunday tashqari, chekli avtomatlar muhim ahamiyatga egadirlar: ixtiyoriy nodeterminirlangan chekli avtomatni har doim determinirlangan ko'rinishga aylantirish mumkin. Bu xususiyat anglovchini xarakatlanishini ta`minlovchi dasturiy ta`minotni ishlab chiqarishni sezilarli soddalashtiradi.

Bunday anglovchilarning oddiyligi va ishlash tezligining yuqoriligi regulyar tillarning qo'llanilishini keng sohasini aniqlaydi.

Kompilyatorlarda regulyar tillar asosidagi anglovchilar boshlang'ich dastur matnini leksik tahlil qilish, ya`ni dasturda tilning oddiy identifikatorlar, konstantalar, qatorlar kabi konstruksiyalarini ajratish, uchun foydalaniladi. Bu esa boshlang'ich dastur matnini hajmini sezilarli qisqartiradi, va dasturni sintaksis tahlilini soddalashtiradi. Dastur matnini leksik va sintaksis tahlilchilarini o'zaroxarakatlari haqida to'liqroq kompilyator strukturasiga bag'ishlangan 6.8-bo'limda qaraladi. Regulyar tillar anglovchilari asosida assemblerlar, ya`ni assembler (mnemokod) tilidan mashina komandalari tiliga kompilyatorlar ishlaydilar.

Kompilyatorlardan tashqari regulyar tillar yana hisoblash tizimlarining dasturiy ta`minotini ishlab chiqarish bilan bog'liq bo'lgan ko'pgina sohalarida ham qo'llaniladi. Ular asosida ko'pgina komanda prosesorlari tizimli va amaliy dasturiy ta`minotda ishlaydilar. Regulyar tillar uchun anglovchilarning tashkil etishni engillashtirish imkonini beradigan rivojlangan, matematika nuqtai nazaridan asoslangan mexanizmlar mavjud. Ular ushbu jarayonni avtomatlashtirish imkonini beradigan, mavjud turli dasturiy vositalar asosiga qurilgan. Regulyar tillar va ular bilan bog'liq matematik usullar kitobning 5.6-bo'limida qaraladi.

5.4.Tahlil masalasi (masalaning qo'yilishi)

Grammatikalar va anglovchilar — bu ikki bir-biriga bog'liq bo'lmagan usul bo'lib, ulardan qandaydir tilni aniqlash uchun foydalaniladi. Ammo qandaydir dasturlash tili uchun kompilyatorni ishlab chiqarishda tillarni berishning ushbu ikki usulini bir-biriga o'zaro bog'lashni talab qiluvchi masala yuzaga keladi. Kompilyatorni ishlab chiqaruvchilar har doim aniqlangan dasturlash tili bilan ish yuritadilar. Ushbu tilning sintaksis konstruksiyalari uchun berilgan grammatika aniq. U qoida bo'yicha til to'liq standartida ifodalanadi, va ifodalash ko'rinishi ixtiyoriy bo'lsada, uni har doim talab etilgan ko'rinishga aylantirish mumkin (masalan, Bekus—Naur formasiga yoki metabelgilar orqali ifodalash ko'rinishiga). Kompilyator ishlab chiqaruvchilarning vazifasi berilgan til uchun anglovchini qurishdan iborat bo'lib, va ushbu anglovchi kompilyatorda keyinchalik sintaksis tahlilchi uchun asos bo'ladi.

SHunday qilib, tahlil masalasi umumiy holda quyidagicha ifodalanadi: qandaydir tilning mavjud grammatikasi asosida ushbu til uchun anglovchini qurish. Berilgan grammatika va anglovchi ekvivalent bo'lishlari shart, ya`ni, bitta tilni aniqlashlari kerak (gohida ularning deyarli ekvivalent bo'lishlari tushuniladi, chunki bo'sh zanjir ko'pincha ahamiyatga ega emas).

Tahlil masalasini umumiy holda barcha tillar uchun echish mumkin bo'lmaydi. Avval aytganimizdek, kompilyatorni ishlab chiqaruvchilarni eng avval kontekst-ozod va regulyar tillar qiziqtiradi. Ushbu toifadagi tillar uchun tahlil masalasini echimi borligi isbotlangan. Bundan tashqari tahlil masalasini echish uchun formal usullar topilgan. Tahlil masalasini echish usullarini ifodalash va asoslashga keyingi bo'limlar bag'ishlanadi.

Dasturlash tillari toza formal til hisoblanmaganliklari uchun va o'zlarida qandaydir ma`noni (semantikani) mujassam qilganliklari uchun, u holda aniq kompilyatorlarni tashkil etish uchun tahlil masalasi, toza formal til uchun ifodalanishdan ko'ra, kengroq tushuniladi. Kompilyator faqatgina kiruvchi belgilar zanjiri ushbu tilga tegishli yoki yo'qligi haqidagi javobni beribgina kolmay, yana uning ma`noviy yuklanishini ham aniqlashi kerak. Buning uchun grammatika qoidalarining asosida ushbu zanjir qurilgan qoidalarini ko'rsatish zarur. Anglovchilarning kompilyator tarkibidagi ishi kiruvchi belgilar zanjirining u yoki bu ko'rinishdagi tahlil daraxtini qurishga keltiriladi. So'ngra ushbu tahlil daraxti kompilyator tomonidan natijaviy kodni sintezi uchun foydalaniladi.

Bundan tashqari, agar kiruvchi belgilar zanjiri berilgan tilga tegishli bo'lmasa, u holda boshlang'ich dastur xatoga ega bo'ladi, ya`ni dastur ishlab chiqaruvchisi uchun esa xatoliklarning borligini bilishning o'zi etarli emas. Ushbu holda tahlil masalasi yanadi kengayadi: kompilyator tarkibidagi anglovchi kiruvchi dasturdagi xatolikni mavjudligini aniqlab qolmasdan, balki, imkon darajasida xatolikning toifasi va belgilar zanjiridagi o'rni haqida ma`lumot berishi kerak.

Demak, tahlil masalasi qandaydir tilning mavjud grammatikasi asosida ushbu tilning anglovchisini qurishni anglatadi.

Tahlil masalasi barcha tillar uchun echilmaydi.

Tahlil masalasi quyidagi masalalar uchun echiladi:

- kiruvchi belgilar zanjirining ma`noviy yuklanishini aniqlash;
- zanjirni qurishda foydalanilgan qoidalarni aniqlash;
- xatoliklarni mavjudligi faktini aniqlash, xatoliklar o'rni va turi.

5.5.Tahlil muammosi

Kompilyator belgilar qatorini tekshirish muammosini, ushbu qator shu tilga tegishli yoki yo'qligini aniqlash uchun va tegishli bo'lsa, u holda tug'iluvchi grammatika qoidalari terminlarida qatorni strukturasini anglashni hal qilishi kerak. Ushbu muammo tahlil muammosi sifatida mashhur.

CHiqish – grammatika qoidalari asosida til gaplarini tug'ilish jarayonidir.

CHiqish zanjiri – bu bevosita chiqariluvchi zanjirlarning ketma-ketligidir. Har bir bevosita chiqariluvchi zanjirdan keyingisiga o'tish chiqish qadami deyiladi. CHiqish tugallangan deyiladi, agar zanjir asosida bitta ham boshqa chiqish qadami qilish mumkin emas bo'lsa (yoki fakat terminal belgilardan tashkil topsa).

Tug'iluvchi qoidalar bilan ishlovchi grammatikani tekshiramiz.

(E-boshlang'ich belgi).

1. E → E+T
2. E→ T
3. T → T*F
4. T → F
5. F→ (E)
6. F→ x
7. F→ y

Ko'rinib turibdiki, (x+y)*x qator ushbu tilga tegishli. Xususiy holda, buni quyidagicha keltirib chiqarish mumkin (har bir keltirib chiqarish qadami uchun qo'llanilayotgan qoida raqami ko'rsatilgan):

2) E → T
3) → T*F
4) → F*F
5) →(E)*F
1) → (E+T)*F
2) →(T+T)*F
4) →(F+T)*F
6) →(x+T)*F
4) → (x+F)*F
7) → (x+y)*F
6) → (x+y)*x

YOki quyidagicha keltirib chiqarish mumkin:

2) E → T
3) → T*F
6) → T*x
4) → F*x
5) → (E)*x
4) → (E+F)*x
7) → (E+y)*x
2) → (T+y)*x
4) → (F+y)*x
6) → (x+y)*x

1) → (E+T)*x

Birinchi chiqishning har bir bosqichida sentensial formaning eng chap noterminali grammatikaning biron bir tug'iluvchi qoidasi yordamida almashtirildi. SHu sababli ushbu chiqish chap tomonli chiqish deyiladi. Har bir bosqichida eng o'ng noterminal almashtirilgan ikkinchi chiqish esa o'ng tomonli chiqish deb ataladi.

SHu kabi boshqa chiqishlar ham mavjudki, ular chap tomonli ham, o'ng tomonli ham hisoblanmaydilar, lekin translyatorlarni qurishda ulardan foydalanilmaydi. Gapni chap tomonli tahlili chap tomonli chiqishni qo'llanilgan holda gapni generasiya qilish uchun tug'iluvchi qoidalarning ketma-ketligi sifatida aniqlanadi. Ushbu holda chap tomonli tahlilni quyidagicha yozish mumkin:

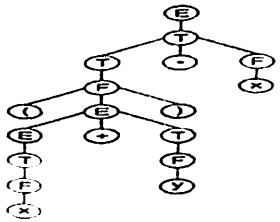
2,3,4,5,1,2,4,6,4,7,6.

Gapni o'ng tomonli tahlili o'ng tomonli chiqishni qo'llanilgan holda gapni generasiya qilish uchun tug'iluvchi qoidalarning teskari ketma-ketligi hisoblanadi; masalan, yuqorida keltirilgan holda o'ng tomonli tahlil quyidagicha yoziladi:

6,4,2,7,4,1,5,4,6,3,2.

Tug'iluvchi qoidalar ketma-ketligining teskari tartibi shu bilan bog'liqki, o'ng tomonli tahlil gapni boshlang'ich belgiga keltirish sifatida qaraladi, gapni boshlang'ich belgidan boshlab generasiya qilish emas (pastdan yuqoriga qarab tahlil). SHuni ta`kidlash kerakki, har bir tug'iluvchi qoidadan ikkala chiqishda ham bir xil son marta foydalaniladi (tahlillarda).

Tahlil daraxti. CHiqish yana sintaksis daraxt (razbor daraxti) nomi bilan mashhur daraxtni qurish terminlarida ham ifodalanishi mumkin. (x+y)*x qator holida sintaksis daraxt quyidagi 19-rasm ko'rinishida bo'ladi:



19-rasm. Tahlil (sintaksis) daraxti ko'rinishiga misol

Tahlil muammosini quyidagi masalalarga keltirish mumkin.

1. chap tomonli tahlilni topish
2. o'ng tomonli tahlilni topish
3. sintaksis taxlil daraxtini qurish.

Kontekst ozod grammatikalar va regulyar grammatikalar uchun har doim chap tomonli va o'ng tomonli chiqishlarni qurish mumkin.

Tahlil usullari ko'pincha pastdan yuruvchidir, ya`ni boshlang'ich belgidan boshlab gapga qarab yuriladi yoki yuqoridan yuruvchi bo'lib, gapdan boshlab boshlang'ich belgiga qarab yuriladi.

Tahlilda echimni rad etish qaytish deb ataladi. Tahlil usullari qaytish bor yoki yo'qligiga qarab determinirlangan va determinirlanmagan bo'lishi mumkin. Determinirlanmagan usullar xotira va vaqt nuqtai nazaridan qimmat bo'lib, kompilyasiya vaqtida bajariluvchi natijalari keyinchalik yo'q qilinishi kerak bo'lgan xarakatlarni sintaksis tahlilchiga qo'shishni qiyinlashtiradi (masalan, belgilar jadvalini qurish va x.k.). Bundan so'ng biz faqat tahlilning determinirlangan usullari haqida so'z yuritamiz.

5.6.Regulyar tillar va grammatikalar

CHapchiziqli va o'ngchiziqli grammatikalar. Avtomat grammatikalar

Regulyar grammatikalarga, avval aytilganidek, ikki toifa grammatikalar tegishli: chapchiziqli va o'ngchiziqli.

CHapchiziqli grammatikalar $G(V_t,V_n,P,S)$, $V=V_n \cup V_t$ ikki ko'rinishdagi qoidalarga ega bo'lishlari mumkin: A-»Vu yoki A-»u, bu erda $A,B \in V_n$, $y \in V_t^*$.

O'z navbatida, o'ngchiziqli grammatikalar $G(V_t,V_n,P,S)$, $V - V_n \cup V_t$ ham ikki ko'rinishdagi qoidalarga ega bo'lishlari mumkin:

A-»uV yoki A-»u, bu erda $A.B \in V_n$, $y \in V_t^*$.

Ushbu ikki sinf grammatiklari ekvivalent ekanliklari isbotlangan. Ixtiyoriy, o'ngchiziqli grammatika yordamida berilgan, regulyar til uchun, ekvivalent tilni aniqlovchi, chapchiziqli grammatikani qurish mumkin; va aksincha — ixtiyoriy chapchiziqli grammatika yordamida berilgan, regulyar til uchun, ekvivalent tilni aniqlovchi, o'ngchiziqli grammatikani qurish mumkin.

CHapchiziqli va o'ngchiziqli grammatikalar o'rtasidagi farq asosan tilni gaplari qanday tartibda tuzilishi bilan bogliq: chapchiziqli grammatika uchun chapdan yo'nalish, yoki o'ngdan chapga o'ngchiziqli grammatika uchun. Dasturlash tillarining gaplari qoida bo'yicha chapdan o'ngga tartibda yo'naltirilgan bo'lganliklari uchun; bundan buyon regulyar grammatikalar bo'limida chapchiziqli grammatikalar ustida to'xtalamiz.

Regulyar grammatikalar orasida shunday alohida sinfni ajratish mumkin, bu avtomat grammatikalardir. Ular ham chapchiziqli va o'ngchiziqli bo'lishlari mumkin.

CHapchiziqli avtomat grammatikalar $G(V_t,V_n,P,S)$, $V- V_n \cup V_t$ ikkita ko'rinishdagi qoidalarga ega bo'lishlari mumkin: A-»Bt yoki A-»t, bu erda $A.B \in V_n$, $t \in V_t$.

o'ngchiziqli avtomat grammatikalar $G(V_t,V_n,P,S)$, $V -V_n \cup V_t$ ikkita ko'rinishdagi qoidalarga ega bo'lishlari mumkin:

A-»tB yoki A-»t, bu erda $A.B \in V_n, t \in V_t$.

Avtomat grammatikalar va oddiy regulyar grammatikalar o'rtasidagi farq quyidagicha: oddiy regulyar grammatikalar qoidalarida terminal belgilar zanjiri bo'ladi, avtomat grammatikalarda esa faqat bitta terminal belgi ishtirok etadi. Ixtiyoriy avtomat grammatika regulyar hisoblanadi, lekin aksincha emas — har qanday regulyar grammatika avtomat grammatika emas.

Oddiy regulyar grammatikalar va avtomat grammatikalar sinflari deyarli ekvivalent ekanligi isbotlangan. Bu esa, ixtiyoriy, regulyar grammatika tomonidan berilgan, til uchun unga deyarli ekvivalent bo'lgan tilni aniqlovchi avtomat grammatikani qurish mumkin ekanligini bildiradi (teskari ta`kid ko'rinib turibdi).

Avtomat va regulyar grammatika sinflari to'liq ekvivalent bo'lishlari uchun, avtomat grammatikalarda quyidagicha qo'shimcha qoidaga ruxsat etiladi: S->^., bu erda S — grammatikaning maqsad belgisi. Bu holda S belgi boshqa grammatika qoidalarining o'ng qismda uchramasligi shart. U holda G avtomat grammatika tomonidan berilgan til o'zida bo'sh zanjirni kiritishi mumkin.

Bunday holatda avtomat chapchiziqli va o'ngchiziqli grammatikalar, xuddi oddiy chapchiziqli va o'ngchiziqli grammatikalar kabi regulyar tillarni beradilar. Amalda foydalanilayotgan tillar bo'sh belgilar zanjiriga ega emaslar, ushbu ikki toifa grammatikalar o'rtasidagi bo'sh zanjirga farq ahamiyatga ega emas va quyidagi ko'rinishli qoida S->X bundan buyon qaralmaydi.

Ixtiyoriy regulyar grammatikani avtomat ko'rinishga aylantiruvchi algoritm mavjud, ya`ni unga ekvivalent avtomat grammatikani qurish mumkin. U juda foydali bo'lib, regulyar grammatikalar uchun anglovchilarni qurish ishini sezilarli ravishda osonlashtiradilar.

5.7.CHekli avtomatlar

CHekli avtomat - bu qandaydir tilni qatorlarini anglash uchun qurilmadir. Uning tarkibida chekli holatlar to'plami bor bo'lib, ularning ba`zilari oxirgilari deb ataladi. Har bir literaning o'qilishi davomida nazorat qatori holatdan holatga berilgan o'tishlar to'plamiga mos ravishda uzatiladi. Agar qatorning oxirgi literasini o'qiganidan so`ng avtomat oxirgi holatlardan birida bo'lsa, qator haqida, u avtomat bilan qabul qilingan tilga tegishli deb aytiladi. Boshqa hollarda qator avtomat bilan qabul qilingan tilga tegishli emasdir.

CHekli avtomat formal ravishda quyidagi beshta xarakteristika bilan aniqlanadi:

- (K) holatlarning chekli to'plami
- (Σ) chekli kirish alfaviti
- (δ) o'tishlar to'plami
- ($S_0 \in K$) boshlang'ich holat
- ($f \in K$) oxirgi holatlar to'plami

$M = (K, \Sigma, \delta, S_0, f)$.

Misol: Avtomatning holatlari A va V, kiruvchi alfavit {0,1}, boshlang'ich xolati –A , oxirgi holatlar to'plami -{A}, o'tishlar

(A,0) = A, (V,0)=V
(A,1) = V, (V,1)=A

Bu o'tishlar A holatda 0 ni o'qishda boshqaruv A holatga uzatilishini anglatadi va x.k. 01001011 qatorni o'qishda boshqaruv ketma-ket ravishda quyidagi tartibda uzatiladi: A,A,V,V,V,A,A,V,A

A so'nggi holat bo'lgani uchun qator chekli avtomat tomonidan qabul qilinadi, lekin 00111 qatorni o'qishda avtomat quyidagi holatlar orqali o'tadi A,A,A,V,A,V.

V oxirgi holat bo'lmaganligi sababli, bu qator qabul qilinmaydi, ya'ni bu qator ushbu avtomat qabul qiladigan tilga tegishli emas. SHu sababli, nullar avtomatning holatiga ta'sir ko'rsatmaydi, har bir 1 esa uning holatini A dan V ga V dan A ga va boshlang'ich holat esa oxirgi holatdek bo'ladi, ya'ni avtomat bilan qabul qilingan til juft birlar soniga ega qatorlardan tuzilgan.

O'tishlarni quyidagi jadval (3-jadval) orqali va chizma ko'rinishini (20-rasm) tasavvur qilish mumkin.

3- jadval

Holatlar			
Kirish		A	V
	0	A	V
	1	V	A

20- rasm. CHekli determinirlangan avtomatning o'tishlari

Bunday avtomatni determinirlangan deb ataladi, chunki o'tishlar jadvalining har bir elementida bitta holat mavjud. Determinirlanmagan chekli avtomat holida esa ushbu shartga amal qilinmaydi.

Quyidagi holat orqali aniqlangan chekli avtomatni qaraymiz:

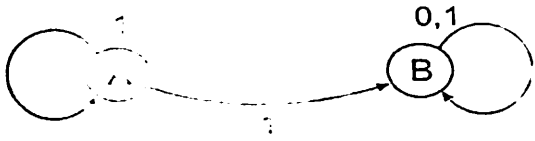
$M_1 = (K_1, \Sigma_1, \delta_1 , S_1 , f_1)$.

Bu erda $K_1=\{A,B\}$, $\Sigma_1=\{1,0\}$, $S_1=\{A\}$, $f_1=\{B\}$

O'tishlar esa 4- jadval va 21- rasmda keltirilgan.

4-jadval

Holatlar			
Kirish		A	V
	0	0	{V}
	1	{A,V}	{B}



21- rasm. M1 chekli determinirlanmagan avtomatning o'tishlari

Birinchi qator qabul qilinadi, chunki qatorni o'qishda oxirgi holatga eltuvchi o'tish mavjud (o'tishlar ketma-ketligi). Xuddi shunday oxirgi bo'lmagan qatorga ham o'tish mavjud, lekin buning qatorning qabul qilinishiga ahamiyati yo'q. SHu sababli, qator determinirlanmagan chekli avtomat tomonidan qabul qilinmasligi mumkin degan xulosaga kelishdan avval, barcha mavjud o'tish imkoniyatlarini ko'rib chiqish zarur.

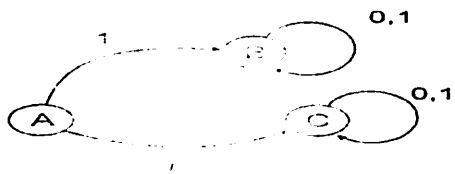
M2 chekli determinirlangan M1 ga mos keluvchi o'sha tilni qabul qiluvchi avtomat mavjud. M2 avtomatning o'tishlari 5- jadval va 22- rasmda keltirilgan.

$M_2 = (K_2, \Sigma_2, \delta_2 , S_2 , f_2)$.

Bu erda $K_2=\{A,B,C\}$, $\Sigma_2=\{1,0\}$, $S_2=\{A\}$, $f_2=\{B\}$

5-jadval

Holatlar				
Kirish		A	B	C
	0	C	B	C
	1	V	B	C



22-rasm. M2 determinirlangan chekli avtomatning o'tishlari

M1 kabi M2 qator ham 0 va 1 lardan tashkil topgan qatorlarni qator 1 dan boshlangan holda qabul qiladi. Lekin qatorni M2 yordamida anglashda qaytish hech qachon talab etilmaydi, chunki aniq bir kiruvchi belgini o'qish jarayonida ixtiyoriy holatdan boshqa holatga aniq bir o'tish amalga oshiriladi. Bu esa M2 dan foydalanishda qatorni anglash vaqti uning uzunligiga proporsional bo'ladi.

Leksik tahlilga moslik shundan iboratki, 3 turdagi har bir tilga determinirlangan chekli avtomat mos kelib u ushbu tilni qatorlarini anglash vazifasini bajaradi. Masalan, G1 grammatikada generasiya qilingan va quyidagi tug'iluvchi qoidalar yordamida aniqlangan qatorlar

$A \rightarrow 1A|\ 1V\ |\ 1$

$V \rightarrow 0V|\ 1V\ |\ 0\ |\ 1$

Bu erda A boshlang'ich belgi bo'lib, M1 va M2 yordamida anglanadi.

Grammatikani determinirlanmagan chekli M1 avtomatdan quyidagicha olinadi:

1. Avtomatning boshlang'ich holati grammatika gapining boshlang'ich belgisi bo'ladi.
2. $\delta(A,1)=A,$ $\delta(V,0)=V$
   $\delta(A,1)=V,$ $\delta(V,1)=B$

o'tishlarga quyidagi tug'iluvchi qoidalar mos keladi:

$A \rightarrow 1A|\ 1V$

$V \rightarrow 0V|\ 1V$

A holatda 1 ni o'qishda oxirgi V holatga o'tish $A \rightarrow 1$ ga mos keladi va analogik ravishda $V \rightarrow 0 | 1$

Xuddi shunday, grammatikadan M1 avtomatni keltirib chiqarish mumkin.

5.8.Regulyar to'plamlar va regulyar ifodalar

Regulyar to'plamlarning aniqlanishi

V alfavitli belgilar zanjirini to'plamlari ustida konkatenasiya va yaqinlashuv amallarini quyidagicha aniqlaymiz:

PQ - konkatenasiya $P \epsilon V^*$ va $Q \epsilon V^*$: $PQ = \{pq\ Vp \epsilon P, Vq \epsilon QJ;$

P^* - yaqinlashuv $P \epsilon V^*$: $P^* = \{p^n\ Vp \epsilon P, Vn \epsilon N\}$.

U holda V alfavit uchun regulyar to'plamlar rekursiv aniqlanadi:

1. 0 — regulyar to'plam.
2. *{k}* — regulyar to'plam.
3. {a} — regulyar to'plam $Va \epsilon V$.
4. Agar R va Q — ixtiyoriy regulyar to'plamlar bo'lsalar, u holda to'plamlar PuQ, PC va R^* ham regulyar to'plamlar hisoblanadilar.
5. Boshqa hech narsa regulyar to'plam hisoblanmaydi.

Regulyar to'plamlar — berilgan alfavit ustidagi belgilar zanjirlarining, aniqlangan ko'rinishda qurilgan, to'plamlaridir (birlashtirish, konkatenasiya va yaqinlashuv amallaridan foydalanilgan holda).

Barcha regulyar tillar regulyar to'plamlarni ifodalaydilar.

Regulyar ifodalar. Regulyar ifodalarning xususiyatlari

Regulyar to'plamlarni regulyar ifodalar yordamida belgilash mumkin. Bu belgilashlar quyidagi ko'rinishda kiritiladi:

1. 0 — 0 ni belgilovchi regulyar ifoda.
2. X —{X} ni belgilovchi regulyar ifoda.
3. a — {a} Va ϵV ni belgilovchi regulyar ifoda.

4. Agar r va q R va Q regulyar to'plamlarni belgilovchi regulyar ifodalar bo'lsa, u holda p+q, pq, p* regulyar ifodalar, mos ravishda PuQ, PQ va R* regulyar to'plamlarni belgilaydi.

Ikkita regulyar ifodalar teng bo'ladilar, agar ular ikkita bir xil to'plamni belgilasalar.

Har bir regulyar ifoda faqat bitta regulyar to'plamni belgilaydi, lekin bitta regulyar to'plam uchun ixtiyoriy ko'p sonli regulyar ifodalar mavjud bo'lishi mumkin.

Regulyar ifodalarni yozish uchun, oddiy arifmetik ifodalar kabi, dumaloq qavslardan foydalaniladi. Qavslar yo'q bo'lgan holda amallar, ustunlik xuquqini hisobga olgan holda, chapdan o'ngga bajariladilar. Amallar uchun ustunlik quyidagicha qabul qilingan: birinchi bo'lib yaqinlashuv bajariladi (yuqori ustunlik), so'ngra konkatenasiya, keyin — to'plamlarni birlashtirish (quyi ustunlik).

Barcha ushbu xususiyatlarni to'plamlar nazariyasiga asoslangan holda oson isbotlash mumkin, chunki regulyar ifodalar — bu mos to'plamlar uchun faqat belgilashlardir. YAna shuni e'tiborga olishimiz kerakki, xususiyatlar ichida tenglik, ya'ni or = ra mavjud emas, chunki konkatenasiya amali kommutativlik xususiyatiga ega emas. Bu amal uchun belgilarning kelish tartibi muhimdir.

Regulyar koeffisientli tenglamalar

Regulyar ifodalar asosida regulyar koeffisientli tenglamalarni qurish mumkin. Regulyar koeffisientli tenglamalarni eng sodda ko'rinishlaridan biri quyidagicha bo'ladi:

$X = aX + r,$

$X = Xa + r,$

Bu erda a, PєV* —regulyar ifoda bo'lib, V alfavit, X o'zgaruvchi Vga tegishli.

Bunday tenglamalarning echimlari regulyar to'plamlar bo'ladilar. Bu agar tenglamani echimi bo'lgan regulyar to'plamni olsak, uni mos regulyar ifodalar ko'rinishida belgilasak va tenglamaga qo'ysak, u holda haqiqiy tenglikni olamiz. Tenglamalarni ikki yozish ko'rinishi (o'ngtomonli va chaptomonli yozuv) shu bilan bog'liqki, regulyar ifodalar uchun konkatenasiya amali kommutativlik xususiyatiga ega emas, shuning uchun koeffisientni o'zgaruvchidan chap tomonda ham, o'ng tomonda ham yozish mumkin va bu holda turli tenglamalar olinadi. Ikkala yozuv tengxuquqlidir.

Regulyar tillarning berilish usullari
Regulyar tillarni berishning uch usuli

Regulyar (o'ngchiziqli va chapchiziqli) grammatikalar, chekli avtomatlar (KA) va regulyar to'plamlar (ularni belgilovchi regulyar ifodalar ham) — bu regulyar tillarni berilishining uch turli usulidir. Regulyar tillarni boshqa usullar bilan ham

aniqlash mumkin, lekin ushbu ko'rsatilgan uch usul ko'proq qiziqishga sabab bo'ladi.

Ushbu uch usuldan regulyar tillarni aniqlashda teng barobar foydalanish mumkinligi isbotlangan. Ular uchun quyidagi ta'kidlarni yozish mumkin:

1-Ta'kid. Til regulyar to'plam hisoblanadi, shu holda va faqat shu holdaki, qachon u chapchiziqli (o'ngchiziqli) grammatika orqali berilgan bo'lsa.

2-Ta'kid. Til chapchiziqli (o'ngchiziqli) grammatika bilan berilishi mumkin, shu holda va faqat shu holdaki, qachonki u regulyar to'plam hisoblansa.

3-Ta'kid. Til regulyar to'plam hisoblanadi, shu holda va faqat shu holdaki, qachonki u chekli avtomat tomonidan berilgan bo'lsa.

4-Ta'kid. Til chekli avtomat tomonidan anglanadi, shu holda va faqat shu holdaki, qachonki u regulyar to'plam hisoblansa.

Regulyar tillarning aniqlashning ushbu uch usuli ekvivalentdirlar. SHunday, ushbu yuqorida keltirilgan usullar orqali berilgan regulyar til uchun, boshqa, xuddi ana shunday tilni aniqlovchi usulni quradigan, algoritmlar mavjud.

Regulyar ifodalar va regulyar grammatikalarning aloqasi

Regulyar ifodalar va regulyar grammatikalar o'zaro bir-biri bilan quyidagicha bog'langan:

1. Regulyar ifoda yordamida berilgan ixtiyoriy regulyar til uchun, xuddi o'sha tilni aniqlovchi, regulyar grammatikani qurish mumkin;
2. Regulyar grammatika orqali berilgan ixtiyoriy regulyar til uchun, xuddi o'sha tilni aniqlaydigan, regulyar ifodani olish mumkin.

Regulyar ifodalar va chekli avtomatlarning aloqasi

Regulyar ifodalar va chekli avtomatlar o'zaro quyidagicha bog'langan:

1. Ixtiyoriy regulyar ifoda bilan berilgan ixtiyoriy regulyar til uchun, o'sha tilni aniqlovchi chekli avtomatni qurish mumkin;
2. CHekli avtomat bilan berilgan ixtiyoriy regulyar til uchun o'sha tilni aniqlovchi regulyar ifodani olish mumkin

Regulyar tillarning xususiyatlari

To'plam qandaydir amalga nisbatan yopiq to'plam deb ataladi, agar ushbu amalni, berilgan to'plamga tegishli ixtiyoriy elementlar ustida, bajarilishi natijasida, ushbu to'plamga tegishli, yangi element olinsa.

Masalan, butun sonlar to'plami ko'paytirish, qo'shish va ayirish amallariga nisbatan yopiq, ammo u bo'lish amaliga nisbatan yopiq emas, ikkita butun sonni bo'lish natijasi har doim ham butun son bo'lmaydi.

Regulyar to'plamlar (va ular bilan bog'lik regulyar tillar) belgilar zanjirlariga qo'llaniladigan ko'pgina amallarga nisbatan yopiqdirlar.

Masalan, regulyar tillar quyidagi amallarga nisbatan yopikdirlar:

- kesishish;
- birlashish;

- qo'shimcha kiritish;
- yaqinlashuv;
- konkatenasiya;
- gomomorfizm (belgilar ismini o'zgartirish va belgilar o'rniga zanjirlarni qo'yish).

Regulyar to'plamlar kesishish, birlashtirish va qo'shimcha kiritish amallariga nisbatan yopiq bo'lganliklari sababli, ular to'plamlarni buleva algebrasini ifodalaydilar. Regulyar to'plamlar nisbatan yopiq bo'lgan boshqa ko'pgina amallar ham mavjud.

Regulyar tillar tillarning juda qulay toifasini ifodalaydilar. Ular uchun boshqa til toifalari uchun echimi bo'lmagan ko'pgina muammolarni echimi mavjud. Masalan, quyidagi muammolarning echilishi mumkinligi isbotlangan.

Ekvivalentlik muammosi. $L_1(V)$ va $L_2(V)$ ikkita regulyar til berilgan. Ushbu ikki tilni ekvivalentmi yoki yo'qligini tekshirish zarur.

Zanjirni tilga tegishliligini aniqlash muammosi. L(V) regulyar til va teV* belgilar zanjiri berilgan. Ushbu zanjirni berilgan tilga tegishli yoki yo'qlini tekshirish zarur.

Tilning bo'shligi (bandmasligi) muammosi. L(V) regulyar til berilgan. Ushbu tilning bo'sh ekanligini tekshirish kerak.

Ushbu muammolarni regulyar til bu usullardan qaysi biri orqali berilganidan qat`iy nazar echish mumkin. Demak, ushbu muammolar regulyar tillarning, regulyar to'plamlarning, regulyar grammatikalarning va chekli avtomatlarning barcha ifodalash usullari uchun echilishi mumkin.

Regulyar grammatikalar uchun yana birqiymatlilik muammosi ham echilishi mumkin — ixtiyoriy regulyar grammatika uchun unga ekvivalent bo'lgan birqiymatli regulyar grammatikani qurish mumkinligi isbotlangan. Ixtiyoriy regulyar grammatika uchun, ushbu grammatika orqali berilgan tilni aniqlovchi, birqiymatli regulyar ifodani qurish mumkin bo'lganligi sababli, bu ko'rinib turibdi.

5.9.Regulyar tillarni anglovchilari

Regulyar tillarning anglovchilari bo'lib $M(Q,V,\delta,q_0,F)$ Q-avtomat holatlari to'plami ko'rinishda berilgan chekli avtomatlar hisoblanadi; V- avtomatning mumkin bo'lgan belgilari to'plami (avtomat alfaviti);
δ- o'tishlar funksiyalari; q_0-avtomatning boshlang'ich holati, $q_0 \in Q$; F oxirgi holatlar to'plami.

CHekli avtomat to'liq aniqlangan deb ataladi, agar uning har bir holatiga barcha mumkin bo'lgan belgilar uchun o'tishlar funksiyasi mavjud bo'lsa. Avtomat qadamlar bo'yicha yoki kartalar bo'yicha ishlaydi.

Avtomatning holati har bir ish taktida uning konfigurasiyasi : (q,ω,n) bilan aniqlanadi. Bu erda q- avtomatning joriy holati, $q \in Q$; ω- kiruvchi belgilar zanjiri; n-kiruvchi belgilar zanjiridagi ko'rsatkich holati. Keyingi taktdagi konfigurasiya quyidagicha aniqlanadi: $(q',\omega,n+1)$, $\delta(*,holat)=holat$ $q'=\delta(sim,q)$. Avtomatning boshlang'ich konfigurasiyasi quyidagi uchlik bilan aniqlanadi $:(q_0,\omega,0)$. Avtomatning so'nggi konfigurasiyasi quyidagicha aniqlanadi: (f,ω,n). CHekli

avtomat belgilar zanjirini qabul qiladi deyiladi, agar belgilar zanjirini kirishda olib boshlang'ich q_0 holatdan f oxirgi holatlardan biriga o'tishi mumkin bo'lsa.

CHekli avtomatning graf o'tishlari.

Bu yo'naltirilgan graf bo'lib chekli avtomatning holatlarini cho'qqilarda aks ettiradi (p,q), p,q-holatlar$\in$Q, a$\in$V belgilar bilan belgilangan, agar avtomatning konfigurasiyasida o'tishlar funksiyasi bo'lsa $\delta(a,p)=q$.

5.10.Kontekst-ozod grammatikalar

3-toifa grammatikalari leksik tahlil vaqtida tashkil etiladigan belgilarni generasiya qilish uchun qulay, lekin ular yuqori daraja tillarini gaplarida bu belgilarni birlashtirish usullarini ifodalash uchun noqulay. Masalan, yuqorida aytilganidek, 3- toifa grammatikasi orqali qavslarni moslashtirishni amalga oshirib bo'lmaydi. Kontekst-ozod grammatikalar (2-toifa) esa tipik dasturlash tillarining barcha xususiyatlarini amalga oshira olmasalar ham, universal hisoblanadilar va kompilyasiyaning sintaksis tahlil fazasi uchun asos sifatida foydalidirlar.

Kontekst-ozod grammatikalar kompilyasiyaning sintaksis tahlil fazasi uchun asos sifatida xizmat qiladilar. Agar qandaydir dasturlash tilini kontekst-ozod grammatika yordamida generasiya qilish mumkin bo'lmasa, u holda shunday kontekst-ozod grammatikani topish mumkinki, u o'ziga berilgan til dasturini kirituvchi til usti to'plamini generasiya qiladi.

Sintaksis tahlil uchun bunday grammatikani qo'llash real tildagi qator chegaralar (masalan, dasturdagi barcha identifikatorlarni e`lon qilinishini shartligi) tahlilchi orqali tekshirilmasligini anglatadi.

Lekin kompilyatorda berilgan dasturdagi zarur tekshiruvlarni bajarilishi bilan bog'liq xarakatlarni qarab chiqish qiyin emas (masalan, belgilar jadvalini qo'llash hisobiga).

Kontekst-ozod grammatika ta`rifidan ko'rinadiki, kontekst-ozod grammatikalar sinfi, regulyar grammatikaga ko'ra, juda qudratli bo'lib (ya`ni ko'p tillarni generasiya qilish mumkin), kontekst-bog'liq grammatikalar sinfiga nisbatan kuchsizroqdir.

$\{a^n b^n \mid n \geq 0\}$ til kontekst-ozod hisoblanadi, lekin regulyar emas. U quyidagi qoidalarni tug'diruvchi grammatika orqali generasiyalanadi:

$S \rightarrow aSb \mid \varepsilon$

Boshqa tomondan $\{a^n b^n c^n \mid n \geq 0\}$ til kontekst-bog'liq hisoblanadi, kontekst-ozod emas.

Kontekst-ozod grammatikalar bir qator xarakteristikalarga ega bo'lib, ular uchun quyidagi xolatlar to'g'ridir:

1.Kanonik forma

a) Har bir kontekst-ozod grammatika normal formada Xomskiy grammatikasiga barcha quyidagi ko'rinishdagi tug'iluvchi qoidalari bilan terminal va noterminallarga tegishli oddiy hollarda ekvivalentdir.

$A \rightarrow VS \mid a$

b) Har bir kontekst-ozod grammatika normal formada Greybax grammatikasiga barcha quyidagi ko'rinishdagi tug'iluvchi qoidalari bilan ekvivalentdir.

$A \rightarrow ba$

Bu erda a noterminallar qatori (bo'sh bo'lishi ham mumkin).

2.O'z-o'ziga yuklash.

Agar G grammatikada A noterminal bor bo'lsa, uning uchun

$A \rightarrow a_1 A a_2$

(bu erda a_1 va a_2 terminallarning yoki noterminallarning bo'sh bo'lmagan qatorlari hisoblanib), u holda bunday grammatika haqida o'z-o'ziga yuklovchi deyiladi.

Masalan, quyida keltirilgan ikkita grammatikalar o'z-o'ziga yuklashni amalga oshiradi:

$G_1 = (\{S\}, \{a,b\}, P, S)$ bu erda R ning elementlari:

$S \rightarrow aSb$

$S \rightarrow \varepsilon$

$G_2 = (\{S,A\},\{begin,end,[,]\},P,S)$ bu erda R ning elementlari:

$S \rightarrow$ begin A end

$S \rightarrow \varepsilon$

$A \rightarrow [S]$

Oxirgi holda A va S haqida ular o'z-o'ziga yuklash xususiyatiga ega deb aytiladi. Nazariy jihatdan ixtiyoriy o'z-o'ziga yuklash xususiyatiga ega bo'lmagan kontekst-ozod grammatika regulyar grammatikaga ekvivalent va regulyar tilni generasiyalaydi. Boshqa tomondan regulyar grammatika o'z-o'ziga yuklashga ega bo'la olmaydi. Haqiqatda o'z-o'ziga yuklash kontekst-ozod grammatikani va regulyar tillarni farqlashga imkon beradi. Ikkinchi misoldan ko'rinib turibdiki, qavslarni moslashuvi va x.k. lar o'z-o'ziga yuklovni talab etadi, shuning uchun ularni regulyar grammatika yordamida qurib bo'lmaydi.

5.11.Kontekst –ozod tillar
Kontekst –ozod tillarning anglovchilari

Kontekst-ozod (KO) tillar deb, $G(V_t, V_n, P, S)$ grammatika toifasi bilan berilgan, R qoidalar quyidagi ko'rinishga ega:

A-»r, bu erda P: $A \rightarrow V_n, A \in V_n, \beta \in V^+ = V_n \cup V_t$ tilga aytiladi.

Kontekst-ozod (KO) tillarning anglovchilari bo'lib magazin-xotirali avtomatlar (MX-avtomatlar) xizmat qiladilar.

Tahlil nuqtai nazaridan kontekst-ozod til chekli avtomatga ekvivalent magazin turidagi avtomatni anglashga haqli bo'lib, unga magazin turidagi xotira

qo'shilganligi muhim. Magazin turidagi avtomatning funksiyalariga quyidagilar kiradi:

a) kiruvchi belgini o'qish, stekning yuqori belgisini belgilar qatoriga o'zgartirish (bo'sh bo'lishi ham mumkin) va holatni o'zgartirish yoki

v) barchasi yuqoridagidek, faqat kiruvchi belgini o'qimasdan.

Kontekst-ozod tillarning anglovchilari bo'lib magazin xotirali avtomatlar hisoblanadilar MX-avtomatlar–bir yoqlamali determinirlanmagan chiziqli cheklangan magazin xotirali anglovchilardir (stek).

Magazin xotirali avtomatni quyidagicha qisqacha tasavvur qilish mumkin.

R(Q, V, Z, d, q_0, z_0, F), bu erda
Q – avtomatning holatlari to'plami.
V – kiruvchi belgilar to'plami.
z – avtomatning magazin belgilarining maxsus chekli avtomati.
d – o'tishlar funksiyasi.
z_0 – magazinning boshlang'ich belgisi.
F – chekli holatlar to'plami.

Bir holatdan boshqa holatga o'tish faqatgina kiruvchi belgidan bog'liq bo'lib qolmay, stekdagi bir va bir necha yuqori belgilardan ham bog'liq.
(q, «lyamda», n)

Ishning birinchi taktida stekdan yuqoridagi belgi ketadi, o'tish shartiga mos ravishda o'tish qoidasiga mos keluvchi zanjir qo'shiladi. Zanjirning 1 belgisi stekning yuqorisiga chiqadi.

Avtomatning konfigurasiyasi har bir qadamda quyidagicha aniqlanadi:
(q,α,ω)α=(β,n). q-joriy holat; ω-stekning qiymati; β-barcha zanjir n-joriy vaqtdagi ko'rsatkich holati.

Magazindan taktni bajarilishi jarayonida o'tish shartiga mos yuqorigi belgi olinadi va o'tish qoidasiga mos zanjir qo'shiladi. Bu zanjirning 1-chi belgisi stekning yuqorisiga chiqadi. λ-o'tishlarga yo'l qo'yiladi, bu holda kiruvchi belgi informasiyalanadi. Avtomat uchun belgilarni magazindan chiqarish shart emas. MX- avtomat determinirlanmagan deb ataladi, agar bittagina konfigurasiyadan bittadan ko'p o'tish mumkin bo'lsa. Boshlang'ich konfigurasiya quyidagicha aniqlanadi. Obraz (q_0,α,z_0) bilan, cheklisi (q,λ,ω). MX avtomat belgilar zanjirini o'tkazadi deyiladi, agar ushbu zanjirning kirishiga u biron bir chekli konfigurasiyadan o'tishi mumkin bo'lsa.MX avtomat magazinni bo'shatish bilan zanjirni o'tkazadi deyiladi, agar, tahlilni tugashida avtomat biron bir chekli konfigurasiyalardan birida joylashsa, stek esa bo'sh. Kengaytirilgan MX avtomat stekning yuqorigi qismida zanjirning chekli o'lchovini boshqa cheklangan o'lchovli belgilar zanjiriga o'zgartirishi mumkin.

Quyidagicha aniqlangan M magazin turidagi avtomatni misol sifatida ko'rib chiqamiz:

K= {A}, S_0 ={A},
S={'(',')'}, Z_0 =I

G= {O,I},

δ δ (A,I,'(')=(A,IO) kabi beriladi.

(bu esa A holatda I bilan stek cho'qqisida '(' o'qishda A xolatga o'tish kerakligini va I ni IO ga o'zgartirishni bildiradi.

δ (A,O,'(')=(A,OO), δ (A,O,')')=(A, ε), δ (A,I, ε)=(A, ε)

M avtomat mos qavslarni anglaydi. Ochilgan qavslar stekka joylanadi va mos yopilgan qavs uchraguncha u erda saqlanib, mos qavs uchragandan so'ng u erdan ketadi. Qavslar qatori, agar barcha qator o'qilgandan so'ng stek bo'sh qolsa, qabul qilinadi. Bu usulni yana qatorlarni oxirgi holatlar bo'yicha qabul qiladigan magazin tipidagi avtomatlar ham aniqlashlari mumkin. Ushbu ikki tur ekvivalentdirlar.

YUqorida ifodalangan magazin turidagi avtomat determinirlangandir, ya'ni har bir mumkin bo'lgan kiruvchi belgiga bir qiymatli o'tish mavjud. CHekli avtomatlar holiga kelsak, biz magazin turidagi berilgan kirish uchun o'tishlar to'plamiga, holatiga va stekka ega determinirlanmagan avtomatlarni xuddi shunday aniqlashimiz mumkin.

Tahlil jarayonida mos magazin turidagi avtomatni effektiv modellashtirish amalga oshiriladi. Ba'zi bir kontekst-ozod tillar determinirlangan holda tahlil qilina olmaydi (qaytishsiz). Determinirlangan tahlilga ega til determinirlangan til deb ataladi, ko'pgina dasturlash tillari determinirlangandir, juda bo'lmasa, shunga yaqinroqdirlar.

Kontekst-ozod grammatika va magazin turidagi avtomat o'rtasida to'liq moslik mavjud va avtomatning determinirlanganligi tilni generasiya qilish uchun qanday grammatikadan foydalanilayotganligiga bog'liq. Tahlil usuli determinirlangan (aniq grammatika uchun) hisoblanadi, agar ushbu grammatikaning tahlilida qaytish talab etilmasa. Ba'zi bir tillarni grammatik tahlilning faqat birgina usuli yordamida determinirlangan tahlil etish mumkin. Xususiy holda, bir qator tillarni pastdan yuqoriga, yuqoridan pastga emas, determinirlangan holda tahlil qilish mumkin. Bundan keyin biz tahlilning determinirlangan usullarini qarab o'tamiz. Nodeterminirlangan usullar Fortran kabi qatorga mo'ljallangan tillarga qo'llanishi mumkin. Kuchli rekursiv tillar (S++, Paskal kabi) da kompilyatorning joriy qatordagina orqaga qaytishi talab qilinmay, balki dasturning kattagina qismida ham talab qilinsa, determinirlanmagan usullarni qo'llash mumkin bo'lmaydi. Qaytishning boshqa kamchiligi shundaki, u kompilyatorning sintaksis tahlil bilan parallel olib boriladigan xarakatlarini rad etishi mumkin.

KO-tillarning xususiyatlari

Ixtiyoriy KO-tillarning xususiyatlari

KO-tillari sinfi o'rniga qo'yish amallariga nisbatan yopiqdir. Bu esa, KO-tilning har bir belgilar zanjirida qandaydir belgining o'rniga boshqa KO—tilning belgilar zanjirini qo'ysak, hosil bo'lgan yangi zanjir ham KO-tillarga tegishli bo'lishini anglatadi. Bu KO-tillarning asosiy xususiyatidir.

Masalan:

$L = \{0^p1^p \mid p > 0\}$, $Lo = \{a\}$, $L_t = \{b^m c^m \mid m > 0\}$ — bu boshlang'ich KO-tillar, u holda o'miga qo'yishlardan so'ng yangi KO-tilni olamiz:

$L' = \{a^n b^m i c^m i b^m 2 c^m 2 ... b^m n c^m n \mid n > 0, Vi{:}mi > 0\}$.

KO-tillari sinfining o'rniga qo'yish amallariga nisbatan yopiqligi asosida KO-tilning boshqa xususiyatlarini isbotlash mumkin. Xususiy holda, KO-tillar sinfi quyidagi to'rtta amalga nisbatan yopiqdir:

- birlashtirish;
- konkatenasiya;
- yaqinlashuv;
- gomomorfizm (belgilar ismini o'zgartirish).

5.12.Keltirilgan grammatikalar

Grammatikalarni kerakli ko'rinishga aylantirish. Aylantirish maqsadi

Avval aytib o'tganimizdek, KO-grammatikalar uchun umumiy holda ularning birqiymatliligi va ekvivalentligini tekshirishni imkoniyati yo'q. Ammo juda ko'p hollarda KO-grammatika qoidalarini avvaldan berilgan qandaydir, yangi kiruvchi grammatikaga ekvivalent bo'lgan, ko'rinishga aylantirish mumkin va zarur. Grammatika qoidalarining avvaldan aniqlangan ko'rinishlari ushbu grammatika asosida berilgan til bilan ishlashni soddalashtiradi va u uchun anglovchini qurishni osonlashtiradi.

SHunday qilib, KO-grammatikalarni kerakli ko'rinishga aylantirishni ikkita asosiy maqsadini ajratish mumkin: grammatika qoidalarini soddalashtirish va til anglovchisini qurishni osonlashtirish. Har doim ham ushbu ikki maqsadni birga bajarib bo'lmaydi. Dasturlash tillari holatida, ya`ni grammatika bilan ishlashning natijasi til kompilyatorini qurish bo'lgan holda, aylantirishning ikkinchi maqsadi asosiy o'rinni egallaydi. SHu sababli til anglovchisini qurishni soddalashtirish bajarilayotgan bo'lsa qoidalarni soddalashtirishga ahamiyat berilmaydi.

Barcha aylantirishlarni shartli ikki guruhga bo'lish mumkin:

- birinchi guruh — bu grammatikadan, grammatikaning ularsiz mavjud bo'lishi mumkin bo'lgan qoida va belgilarini olib tashlash bilan bog'liq bo'lgan aylantirishlardir (xuddi ana shu aylantirishlar qoidalarni asosiy soddalashtirishlarni amalga oshirish imkoniyatini yaratadilar);

- ikkinchi guruh — bu grammatika ko'rinishi va qoidalari tarkibini o'zgartirishga mo'ljallangan aylantirishlar, bu holda grammatika yangi qoidalar bilan to'ldirilishi mumkin, uning noterminal belgilar lug'ati esa yangi belgilar (ya'ni ikkinchi guruh aylantirishlari soddalashtirishlar bilan emaslar) bilan to'ldiriladi.

YAna shuni ta'kidlashimiz kerakki, aylantirishlar natijasida biz, har doim usha tilni aniqlaydigan va kiruvchi grammatikaga ekvivalent, yangi KO-grammatikaga ega bo'lamiz.

U holda formal ravishda aylantirishlarni quyidagi ko'rinishda ifodalashimiz mumkin: $G(V_t, V_n, P, S) \rightarrow G'(V_t', V_n', P', S')$: $L(G) - L(G')$

Keltirilgan grammatikalar

Keltirilgan grammatikalar – bu kontekst-ozod grammatikalar bo'lib, ular etib bo'lmaydigan belgilarni, keraksiz belgilarni, takrorlashlarni, λ-qoidalarni (bo'sh qoidalarni) o'zida jamlamaydilar. Keltirilgan grammatikalarni yana kanonik ko'rinishdagi KO-grammatikalar deb ham ataladi.

Ixtiyoriy kontekst-ozod grammatikani kerakli ko'rinishga aylantirish uchun quyidagi xarakatlarni bajarish zarur:

1.Barcha keraksiz belgilarni yo'qotish.
2.Barcha etib bo'lmaydigan belgilarni yo'qotish.
3.Barcha λ-qoidalarni yo'qotish.
4.Zanjir qoidalarini yo'qotish kerak.

Takrorlovchi chiqish: $A \Rightarrow^* A$

$A \rightarrow V \qquad A, V \in V_n$

YAna shuni ta'kidlash kerakki, aylantirish qadamlari xuddi ko'rsatilgan tartibda bajarilishi shart.

Etib bo'lmaydigan belgilarni yo'qotish

$x \in (V_n \cup V_t)$ belgi etib bo'lmaydigan deb ataladi, agar u $G(V_t, V_n, P, S)$ grammatikaning birorta ham sentensial formasida uchramasa.

Albatta, grammatikadan barcha etib bo'lmaydigan belgilarni olib tashlash uchun, uning barcha sentensial ko'rinishlarini qarab chiqish shart emas (bu bajarish mumkin bo'lmagan ish), etib bo'lmaydigan belgilarni olib tashlaydigan maxsus algortmdan foydalanish etarlidir.

Etib bo'lmaydigan belgilarni olib tashlaydigan maxsus algortm $G(V_t, V_n, P, S)$ — Vj grammatikaning etib bo'ladigan belgilari to'plamini quradi. Avval boshda ushbu to'plamga grammatikaning S boshlang'ich belgisi kiradi, so'ngra to'plam grammatika qoidalari asosida to'ldirib boriladi.

Ushbu to'plamga kirmagan belgilar etib bo'lmaydigan hisoblanadilar va yangi G' grammatika lug'atidan va qoidalaridan olib tashlanadilar.

Etib bo'lmaydigan belgilarni qadamlar bo'yicha olib tashlash algoritmi

1. $V_0 = \{S\}$, i:=l.
2. V j - $\{x \mid x \in (V_t \cup V_n)$ va $(A\text{-xxxr}) \in R$, $A \in V \mid R$, $x, p \in (V_t \cup V_n)^*\} \cup V_{j\_0l}$.

3. Agar Vj $\neq$ Vj.t, u holda i := i+1 va 2 qadamga o'tish, aks holda 4-qadamga o'tish.

4. $V_n' = V_n \cap Vj$, $V_t = V_t \cap Vj$, R' ga R ning, Vj, S' = S to'plamga kiruvchi belgilarini o'zida saqlovchi, qoidalari kiradi.

Barcha keraksiz belgilarni yo'qotish

$G(V_t,V_n,P,S)$ grammatikada $A \in V_n$ A-keraksiz deb ataladi agar, undan birorta ham terminal belgilar zanjirini keltirib chiqarish mumkin bo'lmasa.

Eng oddiy holda belgi barcha qoidalarda chap tomonda ham o'ng tomonda ham uchraydigan hollarda keraksiz hisoblanadi. Murakkabrok hollarda keraksiz belgilar zanjirlari orasidagi bog'liqlik holatlari mavjud bo'lib, ular chiqishning ixtiyoriy ketma-ketligida bir birini keltirib chiqaradilar.

Keraksiz belgilarni yo'qotish uchun keraksiz belgilarni yo'qotishning maxsus algoritmidan foydalaniladi. U maxsus Yj noterminal belgilar to'plami bilan ishlaydi. Avval ushbu to'plamga, o'zlaridan terminal belgilar zanjirlarini keltirib chiqarish mumkin belgilar kiradilar, so'ngra u G grammatika qoidalari asosida to'ldirib boriladi.

Keraksiz belgilarni qadamlar bo'yicha yo'qotish algoritmi

1. $Y_o = 0$, i:=l.
2. Yj = {A | (A-x) $\in$R, a$\in$(Y_Mu V_t)*} u Y_M.
3. Agar Yj $\neq Y_{i-1}$, u holda i := i+1 va 2-qadamga o'tish, aks holda 4-qadamga o'tish.
4. $V_n' = Yj$, $V_t = V_t$, R' ga R ning (V_tuYi), S' = S to'plami belgilarini o'z ichiga oluvchi qoidalari kiradi.

Etib bo'lmaydigan belgilarni va keraksiz belgilarni yo'qotishga doir misol

Grammatika misollarida etib bo'lmaydigan va keraksiz belgilarni yo'qotish algoritmlarini qarab chiqamiz:

G({ab,c},{A,B,C,D,E,F,G,S},P,S)

R

S-»aAV| E

A-» aA| bV

V-» ASb| b

S-» A| bA| sS| aE

E-»sE| aEJ Eb| ED| FG

D» a| s| Fb

F-» VS|. ES| AS

G-». Ca| Cb

SHuni ta`kidlash kerakki, aylantirishlarni to'g'ri bajarilishi uchun avval keraksiz belgilarni yo'qotish zarur, so'ngra etib bo'lmaydigan belgilarni, lekin buning aksini bajarish mumkin emas. YA`ni algoritmlarni qaysi tartibda bajarilishi sezilarli ahamiyatga ega.

Keraksiz belgilarni yo'qotamiz:

1. $Y_o = 0$, i:=l.
2. $Y_1 = \{B,D\}$, Y^Y_0; i:=2.
3. $Y_2 = \{B,D,A\}$, Y_2*Y_1 ; i:=3.

4. $Y_3 = \{B,D,A,S,C\}$, $Y_3 * Y_2$; i:=4.
5. $Y_4 = \{B,D,A,S,C,F\}$, $Y_4 * Y_3$; i:=5.
6. $Y_5 = \{B,D,A,S,C,F\}$, $Y_5 = Y_4$.

$V_n' = \{A,B,C,D,F,S\}$, V_t' - {a,b,c} va R' to'plamlarni quramiz.

Quyidagi grammatikani olamiz:

G'({a,b,c},{A,B,C,D,F,S},P',S)

P':

S->aABA->aA|bBV->ACb | bS->A| bA|cCD->a | S | FbF->VS | AC

Etib bo'lmaydigan belgilarni yo'qotamiz:

1. $V_0 = \{S\}$,i=l.
2. $V_1 = \{S,A,B\}$, V^V_0; i:=2.
3. $V_2 = \{S,A,B,C\}$, y>Vj; i:=2.
4. $V_3 = \{S,A,B,C\}$,$V_3 = V_2$.

V_n'' - {A.B,C,S}, V_t'' = {a,b,c} va R' to'plamlarni quramiz

Natijada quyidagi grammatikani olamiz:

G"({a,b,c},{A,B,C,S},P",S)

R":

Si* aAB

A->aA|bB

V->ACb|b

S->A|bA|cC

Ushbu keraksiz belgilarni va etib bo'lmaydigan belgilarni yo'qotish algoritmlari KO-grammatikalarni kerakli ko'rinishga aylantirishning birinchi guruhiga tegishlidirlar. Ular har doim grammatikaning soddalashishiga, alfavit belgilarining va grammatika qoidalarining sonini qisqartirilishiga olib keladilar.

Barcha λ-qoidalarni yo'qotish

λ-qoidalar (yoki bo'sh zanjirli qoidalar) deb quyidagi ko'rinishdagi grammatikalarga aytiladi: $A->X$, gde $A \in V_n$.

$G(V_t,V_n,P,S)$ grammatika λ-qoidalarsiz grammatika deb ataladi, agar unda quyidagi qoidalar mavjud bo'lmasa (A-»^)ЄR, A*S va faqatgina bitta quyidagi qoida mavjud bo'lsa (S-»^.)ЄP, va S bironta qoidaning o'ng qismida uchramaydi.

L(G) tilning anglovchisini qurishni soddalashtirish uchun ixtiyoriy G grammatikani λ-qoidalarsiz ko'rinishga aylantirish maqsadga muvofiqdir. Ixtiyoriy KO-grammatikani λ-qoidalarsiz ko'rinishga aylantirishning maxsus algoritmi mavjud. U W noterminal belgilar to'plami bilan ishlaydi.

Barcha λ-qoidalarni qadamlar bo'yicha yo'qotish algoritmi

1. $W_0 = \{A:(A->^) \in P\}$, i:=l.
2. Wi - W_Hu {A:(A-kx) ЄR, aЄW,'}.
3. Esli W, F Wi-ii u holda i := i+1 va 2-qadamga o'tish, aks holda 4-qadamga o'tish.

4. $V_n' = V_n$, $V_t' - V_t$, R' ga Rning, A-> λ qoidadan tashqari barcha, qoidalari kiradi.

5.Agar (A-»a)ЄR bo'lsa va a zanjirga Wj ning belgilari kirsa, u holda a zanjir asosida, a dan barcha mumkin bo'lgan belgilar variantlarini yo'qotish yo'li bilan {a'} zanjirlar to'plami quriladi va R' ga A-»a' ko'rinishdagi qoidalar qo'shiladi.

6.Agar SЄWj, u holda XЄL(G), va V_n' ga G grammatikaning maqsad belgisi bo'lib hisoblanadigan yangi S' qo'shiladi, R' ga esa ikki yangi qoida qo'shiladi: S'-»X|S; aks holda S' = S.

Ushbu algoritm ko'pincha grammatika qoidalarini sonini ko'payishiga olib keladi, ammo berilgan til uchun anglovchini qurishni soddalashtirish imkoniyatini yaratadi.

Barcha λ-qoidalarni qadamlar bo'yicha yo'qotish algoritmiga doir misol

Quyidagi grammatikani qaraymiz:

G({a,b,c},{A,B,C,S},P,S)
R:
S -> AaV | aV | sS
A -> AV | a | b | V
V -> Va | X
S -> AV | s

X-qoidalarni yo'qotamiz:

1. $W_0 = \{B\}$, i:=l.
2. $W_1 = \{V,A\}$, W^W_0, i:=2.
3. $W_2 = \{V,A,S\}$, W_2*Wi, i:=3.
4. $W_3 = \{B,A,Q\}$, $W_3 = W_2$.
5. $V_n' = \{A,B,C,S\}$, $V_t' = \{a,b,c\}$ to'plamlarni va R' qoidalar to'plamini quramiz.
6. R' to'plamdagi barcha qoidalarni qaraymiz:

S-»AaB | aV | sS qoidalardan A, V va S ning barcha mumkin bo'lgan variantlarini olib tashlaymiz va yangi quyidagi qoidalarni olamiz S—>Aa | aV j a | a | s, ularni R' ga qo'shamiz, bir xillarini yo'qotib:S->AaB|aV|sS|Aa|aV|a|s olamiz.

A->AV | a | b | V qoidalardan A va V ning barcha mumkin bo'lgan variantlarini olib tashlaymiz va yangi quyidagi qoidalarni olamiz A-> A|B, R' ga ularni qo'shish kerak emas, chunki A-»V koida u erda mavjud, A->A qoida esa ma`noga ega emas..

V—>Va qoidalardan V yo'qotamiz va yangi qoidaga ega bo'lamiz V->a, uni R'ga qo'shamiz, va V->Va|a olamiz.

S->AV | s qoidalardan A va V ni barcha mumkin bo'lgan variantlarini olib tashlaymiz va yangi qoidalarga ega bo'lamiz: S—>A| V, ularni R'ga qo'shamiz, va S—>AV | A | V | s olamiz.

7.SЄW_3, shu sababli S grammatikaga S' yangi maqsad belgini qo'shish shart emas. S' = S.

Grammatikani olamiz:

G'({a,b,c},{A,B,C,S},P',S)
R':

S -> AaV | aV | sS | Aa | a | s
A -> AV | a | b | V
V—>Va|a
S -> AV | A | V | s

Zanjir qoidalarini yo'qotish

$G(V_t,V_n,P,S)$ grammatikada takrorlash (takroriy chiqish) deb $A=>^*A$, $A\epsilon V_n$ ko'rinishdagi chiqishga aytiladi. Ko'rinib turibdiki, bunday chiqish foydasizdir. SHu sababli KO-tillarning anglovchilarida takrorlashlarni paydo bo'lish imkoniyatlaridan qochish maqsadga muvofiqdir.

Takrorlashlar faqat, agar KO-grammatikada quyidagi ko'rinishdagi zanjir qoidalari mavjud bo'lsalargina A-»V, $A,V\epsilon V_n$ yuzaga kelishlari mumkin. CHiqish zanjirlarida takrorlashlarni yuzaga kelishini yo'qotish uchun grammatika qoidalaridan zanjir qoidalarni yo'qotish etarlidir.

$G(V_t,V_n,P,S)$ ko'rinishdagi KO-grammatikadagi zanjir qoidalarni yo'qotish uchun, $X\epsilon V_n$ har bir noterminal belgi uchun N^x maxsus zanjirli belgilar to'plami quriladi, so'ngra qurilgan to'plamlar asosida R qoidalarni aylantirishlari bajariladi. SHu sababli zanjir qoidalarini yo'qotish algoritmini V_n to'plamli grammatikaning barcha noterminal belgilari uchun bajarish kerak.

Ushbu grammatikadan KO-til anglovchilarini qurishda foydalaniladi.

5.13.Qaytishli kontekst-ozod tillarni anglovchilari

Qaytishli anglovchilar eng primitivdir bo'lib, MX nodeterminirlangan avtomatni ishini modellashtiradilar. Determinirlanmagan avtomatning ishi modellashtirilayotganligi sababli, ishning qandaydir qadamida quyidagi mumkin bo'lgan holatlari yuz berishi mumkin. Ikki variantdan foydalanish mumkin:

1. Ishning har bir qadamida algoritm holatning barcha mumkin bo'lgan keyingi holatlarini saqlab qolishi kerak, bittasini tanlashi ikkinchisiga o'tishi va x.k. oxirgi aniqlanmagan konfigurasiyaga o'tmaydi. Algoritmning bajarilish vaqti kiruvchi belgilar zanjiriga eksponensial bog'liqdir. Kerak bo'ladigan xotira hajmi kiruvchi belgilar zanjiri uzunligidan chiziqli bog'liqdir.
2. Bir qiymatlilik yuz bergan har bir holatda har bir holatni qayta ishlash uchun yangi nusxa chiqariladi. Vaqt bu holda chiziqli bog'liq, xotira hajmi esa - eksponensial.

MX avtomatni modellashtirishning asosiy xususiyatlari quyidagilardir: tahlil cheklangan qadamlar sonida tugallanmay qolishi mumkin. Bunday holatlarning oldini olish uchun qaytishli tahlil algoritmlari ixtiyoriy MX avtomatlar uchun qurilmaydi, balki berilgan shartlarni qanoatlantiruvchi MX avtomatlar uchun quriladi.

5.14.Qaytishli pastdan chiquvchi anglovchi (alternativlarni tanlab)

Bunday anglovchi bir holatli MX avtomat ishini modellashtiradi.

q: R({q},V,z,δ,q,S,{q})

Boshlang'ich konfigurasiya (q,α,S) quyidagicha aniqlanadi: avtomat o'zining yagona q holatida joylashadi, o'quvchi bosh kiruvchi belgilar zanjirining boshida joylashadi. Stekda grammatikaning maqsad belgisiga mos keluvchi belgi joylashadi.

Oxirgi konfigurasiya (q,λ,λ) avtomat yagona q holatda joylashadi, bosh o'qib bo'lgandan o'ng belgilar zanjiridan chetga chiqadi, stekda hech narsa qolmaydi.

Psevdokodlardan foydalanuvchi algoritmlar ishi.

agar stek yuqorisida $A \in V_n$ joylashsa

u holda A ni α ($A \rightarrow \alpha \in R$) kiruvchi belgilar zanjiriga almashtirish mumkin va o'quvchi bosh joyidan qimirlamaydi – bu qadam «alternativlarni tanlash» deb ataladi

hammasi agar

agar stek yuqorisida $a \in V_t$: a=kiruvchi belgilar zanjiridagi joriy belgi

u holda a ni stekdan tashlab yuborish mumkin va boshni sg$\rightarrow$1 suramiz, bu qadam tashlab yuborish deb ataladi.

hammasi agar

Qanday qadamni bajarish kerakligi haqidagi echim bir qiymatli qabul qilinadi.

Modellashtiruvchi algoritm mumkin bo'lgan alternativlarning birinchisini tanlovini va qanday alternativlarni qanday qadamda tanlanganligi haqidagi ma`lumotni saqlanishini ta`minlashi kerak.

Berilgan MX avtomat chaptomonli chiqishlarni quradi, kiruvchi belgilar zanjirini chapdan-o'ngga o'qiydi, shuning uchun chiqish daraxtini qurish yuqoridan pastga qarab yuradi, grammatika chaprekursiv bo'lishi kerak.

Quyidagi belgi $A \in V_n$ rekursiv deb ataladi, agar u uchun quyidagi chiqish zanjiri $A =>^{+} \alpha A \beta$ mavjud bo'lsa.

$\alpha=\lambda$ $\beta \neq \lambda$ chap rekursiya
$\alpha \neq \lambda$ $\beta=\lambda$ o'ng rekursiya
$\alpha=\lambda$ $\beta=\lambda$ takrorlashni tashkil etadi.

Kamchiligi: qiymatli vaqt hajmi, chunki hisoblash resurslari algoritmlarini ishlashi uchun zarur bo'lgan eksponensial bog'liqlik mavjud.

YUtuqlari: 1)amalga oshirishning soddaligi 2) universallik

5.15.YUqoridan pastga ishlovchi anglovchining «surilish-hosil qilish» algoritmi bo'yicha ishlash tamoyili

Ushbu anglovchi kengaytirilgan $R(\{q\},V,z,\delta,q,S,\{q_t\})$ MX avtomatning ishini modellashtiradi. Ushbu MX avtomatning boshlang'ich konfigurasiyasi (q,α,λ) aniqlanadi. Oxirgisi (q,λ,S)

Psevdokoddan foydalangan holda algoritmni amalga oshirilishi:

agar stekning yuqorisida γ belgilar zanjiri joylashadi

u holda γ $A\in V_n(A\rightarrow\gamma\in P)$ ga almashtiriladi (ushbu qadam "hosil qilish" deb ataladi)

hammasi agar

agar stekga o'quvchi boshni keltirish va o'quvchi boshni bitta pozisiya o'ngga surish

hammasi agar

Ushbu MX avtomat o'ngtomonli chiqishni quradi va zanjirni chapdan-o'ngga o'qiydi, shuning uchun daraxtni qurish pastdan-yuqoriga yuradi va yuqoriga yuruvchi qaytishli anglovchi deb ataladi.

Kamchiligi: qiymatli vaqt hajmi.

YUtuqlari: a)amalga oshirishning soddaligi b)universallik.

5.16.Kontekst-ozod qaytishsiz til anglovchilari

Ikkita sinf mavjud:

1) yuqoridan pastga yuruvchi anglovchilar chaptomonli chiqishni zanjirini yuqoridan pastga keltirib chiqaradilar, chiqish daraxtini yuqoridan pastga qarab quradi va alternativlarni tanlovchi algoritm modifikasiyasidan foydalanadilar. Ushbu sinfga quyidagilar kiradi: a)rekursiv takrorlash asosida anglovchi b) U(K) anglovchi v) U(S) anglovchi

2)yuqoriga yuruvchi zanjirlar o'ngtomonli chiqishni keltirib chiqaradilar, "surilish-hosil qilish" algoritmi modifikasiyasidan foydalanadilar.

a) 2R(K) asosida anglovchi b) 2R(0) anglovchi v) 2R(1) anglovchi

5.17. YUqoridan pastga yuruvchi LL(1) –grammatikalar

LL(1) –grammatika – bu shunday grammatikaki, uning asosida yuqoridan pastga tamoyili bilan ishlovchi determinirlangan sintaksis tahlilchini olish mumkin. LL(1) –grammatikaga aniq ta`rif berishdan avval s –grammatika tushunchasini kiritamiz.

s –grammatika bu shunday grammatikaki, unda:

1) har bir tug'iluvchi qoidaning o'ng qismi terminaldan boshlanadi.

2) chap bo'lakda bittadan ko'p tug'iluvchi qoidalar bo'lgan holida bittagina noterminal paydo bo'ladi, mos o'ng bo'laklar turli terminallardan boshlanadi.

Birinchi shart, grammatika Greybaxning normal formasiga ega, faqat har bir o'ng bo'lak qoidasining boshlanishida terminaldan keyin noterminallar yoki terminallar kelishi mumkin degan ta`kidga o'xshashdir.

Ikkinchi shart determinirlangan pastki tahlilchini yozish imkonini beradi, chunki tilning gaplarini chiqarishda har doim alьternativ tug'iluvchi qoidalar o'rtasida eng chap noterminal uchun sentensial formada, avvaldan bitta keyingi belgini tekshirib, tanlov qilish mumkin.

Quyidagi tug'iluvchi qoidalar bilan aniqlangan grammatika

S→ pX X→ x
S→ qY Y→ y
X→ aXb Y→ aYd

s –grammatikani tashkil etadi, u holda quyidagi grammatika esa, xuddi o'sha tilni generasiya qiladi va u s –grammatika emas.

S→ R X→ aXb
S→ T X→ x
R→ pX Y→ aYd
T→ qY Y→ y

chunki ikkita qoidaning o'ng bo'laklari terminallardan boshlanmaydi.

s–grammatikadan berilgan grammatika sifatida foydalanilayaptimi yoki yo'qmi aniqlash juda oson, ba`zi hollarda s –grammatika bo'lmagan grammatikani ham s –grammatikaga generasiya qilingan tilni bezovta qilmagan holda aylantirish mumkin.

paaaxbbb qatorni tahlilini yuqoridagi s–grammatikadan foydalanilgan holda qaraymiz. S belgidan boshlab qatorni generasiya qilishga boshlaymiz. CHap tomonli chiqishni qo'llaymiz. Natija esa quyidagicha bo'ladi.

Boshlang'ich qator: paaaxbbb

CHiqish: S→ pX →paXb →paaXbb → paaaXbbb →paaaxbbb

CHiqishda sentensial ko'rinishdagi boshlang'ich terminallar boshlang'ich qatordagi belgilar bilan solishtiriladi. Sentensial ko'rinishdagi eng chap terminallarni bittadan ko'p tug'iluvchi qoidalarni qo'llab almashtirish mumkin bo'lgan hollarda, har doim mos tug'iluvchi qoidani kiruvchi qatordagi keyingi belgina tekshirib tanlash mumkin. Bu shu bilan bog'liqki, biz s –grammatikadan foydalanganligimiz tufayli alternativ tug'iluvchi qoidalarning o'ng qismlari turli terminallardan boshlanadi. SHunday qilib har doim yuqoridan pastga tahlilni amalga oshiruvchi determinirlangan tahlilchini s –grammatikadan generasiya qilinadigan til uchun yozish mumkin.

LL(1) –grammatika s –grammatikani umumlashtirilgani hisoblanib, uning umumlashtirish tamoyili pastdan chiquvchi determinirlangan tahlilchilarni qurish

imkonini beradi. Ikkita LL harfi qatorlar chapdan o'ngga (Left) qaralayotganligini va eng chap chiqishlardan (Left) foydalanilishini bildiradi, 1 raqami esa tug'iluvchi qoidalarning variantlari avvaldan qurilgan bitta belgi yordamida tanlanishini anglatadi. Demak, LL(2) –grammatika haqida gap ketsa, bu qatorlar chapdan o'ngga qarab qaralishini va eng chap chiqishlardan foydalanilishini, va tug'iluvchi qoidalarning variantlari ikkita avvaldan qurilgan belgilar yordamida tanlanishini anglatadi.

Xuddi shuningdek, LR(1) –grammatika qatorlar chapdan o'ngga (Left) qaralayotganligini va eng o'ng chiqishlardan (Right) foydalanilayotganligini bildiradi, tug'iluvchi qoidalar variantlari esa avvaldan qurilgan bitta belgi yordamida tanlanadi.

Agar tug'iluvchi qoidaning o'ng qismi terminaldan boshlanmagan holda ham noterminal belgining har bir varianti aniq bir terminallar to'plamidan boshlangan qatorlarga boshlanish berishi mumkin.

Masalan quyidagi tug'iluvchi qoidalar asosida

$S \rightarrow RY$ $\quad R \rightarrow paXb$

$S \rightarrow TZ$ $\quad T \rightarrow qaYd$

keltirib chiqarish mumkinki, R va T uchun boshqa tug'iluvchi qoidalar bo'lmasalar $S \rightarrow RY$ ni yuqoridan pastga (pastga tomon tahlil- boshlang'ich belgidan qatorgacha) tahlilda qo'llash, faqatgana avvaldan qarab chiqilgan belgi r bo'lgan holda maqsadga muvofiqdir.

Xuddi shunday tug'iluvchi qoida $S \rightarrow TZ$ dan bunday belgi q bo'lgan holda foydalanish tavsiya etiladi.

Bu esa quyidagi hamrox belgilar to'plami g'oyasini keltirib chiqaradi:

$a \in S(A) \Leftrightarrow A \Rightarrow a\alpha$,

bu erda A – noterminal belgi, α – terminallar qatori va/yoki noterminallar qatori, bo'sh bo'lishi mumkin, S(A) esa A hamrox belgilar to'plamini anglatadi. Quyidagi tug'iluvchi qoidalarga ega grammatikada

$P \rightarrow Ac$ $\quad A \rightarrow aA$

$P \rightarrow Bd$ $\quad B \rightarrow b$

$A \rightarrow a$ $\quad B \rightarrow bB$

a va b belgilar R uchun hamrox-belgilardir.

Xuddi shunday

$a \in S(\alpha) \Leftrightarrow \alpha \Rightarrow a\beta$ qator terminal va/yoki noterminallar uchun ham hamrox – belgilarni aniqlashimiz mumkin.

Bu erda α va β terminallar va/yoki noterminallarning qatorlari (β bo'sh qator ham bo'lishi mumkin).

Grammatika LL(1) xususiyatlariga ega bo'lishining zaruriy sharti shuki, o'ng tomon alternativ tug'iluvchi qoidalarining hamrox belgilari to'plamlari bir-birlari bilan kesishmasligidir.

O'ng tomonning boshlanishida noterminal belgi bo'sh qatorni generasiyalayotgan vaqtda ehtiyotkorlik zarur. Masalan, quyidagi grammatikada

P→ AB	A→ ε
P→ BG	B→ bB
A →aA	B→ c

S(AB)= {a,b,c}, ga ega bo'lamiz va bu erda b va c to'plamga kiradi, chunki A bo'sh qatorni generasiya qilishi mumkin; S(BG)= {b,c} – to'plamlar kesishadilar va bundan ko'rinadiki ushbu grammatika LL(1) grammatika emas.

YAna quyidagi tug'iluvchi qoidalarga asoslangan grammatikani qaraymiz:

T→ AB	Q →ε
A→PQ	B→ bB
A→ BC	B→e
P→ pP	C →cC
P →ε	C→ f
Q→ qQ	

Ushbu grammatika uchun S(PQ) = {p, q} va S(BC)={b,e}.

Lekin PQ bo'sh qatorni generasiyalashi mumkin bo'lgani uchun keyingi qaraladigan belgi A→PQ tug'iluvchi qoidani b yoki e bo'lishi mumkin.

SHuning uchun yo'naltiruvchi belgilar tushunchasi kiritiladi, va u quyidagicha aniqlanadi: agar A noterminal bo'lsa, u holda uning yo'naltiruvchi belgilari bo'lib S(A)+ (A dan keyingi barcha belgilar, agar A bo'sh qatorni generasiya qilish mumkin bo'lsa).

Boshqacha aytganda, bu barcha hamrox-belgilar to'plami + A dan keyingi barcha belgilar, agar A bo'sh qatorni generasiya qilishi mumkin bo'lsa. Umumiy holda a noterminalning R(R→ a) berilgan varianti uchun quyidagiga ega bo'lamiz:

DS(P, a)={a| a∈S(a) yoki (a⇒ ε va a∈F(P))}

Bu erda F(P) R dan keyin keluvchi belgilar to'plamidir. YUqorida keltirilgan misolda yo'naltiruvchi belgilar -bu quyidagilardir:

DS(A,PQ)= {p,q,b,e}
DS(A,BC)={b,e}

Ko'rsatilgan to'plamlar kesishganligi sababli, ushbu grammatika pastga tomon determinirlangan tahlilchi uchun asos bo'la olmaydi.

LL(1) grammatika ta`rifini aniqlashtiramiz: Grammatikani LL(1) grammatika deb ataymiz, agar bittadan ko'p tug'iluvchi qoidalarda uchraydigan har bir noterminal uchun, alьternativ tug'iluvchi qoidalarning o'ng qismiga mos yo'naltiruvchi belgilar to'plami kesishmasalar. Barcha LL(1) grammatikalarni yuqoridan pastga determinirlangan holda tahlil qilish mumkin.

5.18.Grammatikalarni LL(1) ko'rinishga aylantirish

Ko'pgina dasturlash tillari uchun «ko'rinib turgan» grammatika bu LL(1) grammatika emas. Lekin, ko'pincha dasturlash tilining juda katta sonli kontekst-ozod vositalarini LL(1) grammatika orqali ifodalash mumkin. Muammo shundaki, LL(1) grammatika belgilariga ega bo'lmagan grammatikaga ega bo'lgan holda, unga teng kuchli bo'lgan LL(1) grammatikani topish kerak. Ixtiyoriy kontekst-ozod grammatikani LL(1) ko'rinishga aylantirishning umumiy algoritmi mavjud emas. Lekin ko'pgina xususiy hollarda bunday aylantirishlarni bajaruvchi qator usullar mavjud.

CHap rekursiyani yo'qotish.

CHap rekursiyaga ega grammatika LL(1) grammatika hisoblanmaydi.

Quyidagi qoidalarni ko'rib chiqamiz:

$A \rightarrow Aa$ (A da chap rekursiya)

$A \rightarrow a$

Bu erda a belgi-hamrox A noterminalning ikkala varianti uchun.

Xuddi shuningdek chap rekursiv takrorlashga ega grammatika ham LL(1) grammatika hisoblanmaydi, masalan

$A \rightarrow VS$

$V \rightarrow SD$

$S \rightarrow AE$

CHap rekursiv takrorlashga ega grammatikani faqat to'g'ri chap rekursiyaga ega grammatikaga aylantirish mumkin va keyinchalik qo'shimcha noterminallar kiritish yo'li bilan chap rekursiyani to'liqligicha olib tashlash mumkin bo'ladi (haqiqatda esa u o'ng rekursiya bilan almashtiriladi. O'ng rekursiya esa LL(1)-xususiyatlariga nisbatan muammo hosil qilmaydi).

Misol sifatida quyidagi tug'iluvchi qoidalarga va chap rekursiv takrorlashga ega bo'lib A,B,C,D larni o'ziga oluvchi grammatikani qarab chiqamiz:

$S \rightarrow Aa$ $\quad C \rightarrow Dd$

$A \rightarrow Bb$ $\quad C \rightarrow e$

$B \rightarrow Cc$ $\quad D \rightarrow Az$

Ushbu takrorlashni to'g'ri chap rekursiyaga o'zgartirish uchun noterminallarni quyidagicha tartibga solamiz: S, A,B,C,D.

Barcha quyidagi ko'rinishdagi tug'iluvchi qoidalarni qaraymiz:

$Xi \rightarrow Xjy$,

Bu erda Xi va Xj noterminallar, y esa terminal va noterminal belgilar qatori.

j >=i bo'lgan qoidalar uchun hech qanday xarakat bajarilmaydi. Lekin bu tengsizlik chap rekursiv takrorlashlarga ega barcha qoidalar uchun ham mos kelavermaydi. YUqoridagi biz tanlagan tartibda quyidagi yagona qoida bilan ishlaymiz.

$D \rightarrow Az$

chunki bu tartibda A D dan keyin kelmoqda. Endi A ni chap tomondagi barcha mavjud qoidalardan foydalanib almashtiramiz. Natijada

$D \rightarrow Bbz$ ni olamiz.

Tartibda B D dan keyin kelayotganligi sababli, jarayon takrorlanadi, buni esa quyidagi qoida beradi:

$D \rightarrow Ccbz$

So'ngra u yana takrorlanadi va ikkita qoidani beradi:

$D \rightarrow ecbz$

$D \rightarrow Ddcbz$

Endi keltirib chiqarilgan grammatika quyidagicha bo'ladi:

$S \rightarrow Aa$ $\quad C \rightarrow e$

$A \rightarrow Bb$ $\quad D \rightarrow Ddcbz$

$B \rightarrow Cc$ $\quad D \rightarrow ecbz$

$C \rightarrow Dd$

Barcha ushbu tug'iluvchi qoidalar talab qilingan ko'rinishga ega, chap rekursiv takrorlanish esa to'g'ri chap rekursiya bilan almashtirilgan. To'g'ri chap rekursiyani yo'qotish uchun yangi noterminal belgi Z kiritamiz va qoidalarni almashtiramiz:

$D \rightarrow ecbz$

$D \rightarrow Ddcbz$

larni

$D \rightarrow ecbz$ $\quad Z \rightarrow dcbz$

$D \rightarrow ecbzZ$ $\quad Z \rightarrow dcbzZ$

D aylantirishgacha va undan so'ng quyidagi regulyar ifodani generasiyalaydi

(ecbz) (dcbz)*

Umumlashtirgan holda, shuni ko'rsatish mumkinki, agar A noterminal r+s tug'iluvchi qoidalarning chap qismida uchrasa, r bulardan to'g'ri chap rekursiyadan foydalansa, s esa foydalanmasa, ya`ni

$A \rightarrow A\alpha_1, A \rightarrow A\alpha_2, \ldots, A \rightarrow A\alpha_r$

$A \rightarrow \beta_1, A \rightarrow \beta_2, \ldots, A \rightarrow \beta_s$

u holda bu qoidalarni quyidagilarga almashtirish mumkin:

$$\left.\begin{matrix} A \rightarrow \beta_i \\ A \rightarrow \beta_i Z \end{matrix}\right\} \; L \le i \le s \qquad \left.\begin{matrix} Z \rightarrow \alpha i \\ Z \rightarrow \alpha i Z \end{matrix}\right\} \; L \le i \le r$$

Noformal isbot shuni ko'rsatadiki, A aylantirishgacha va undan so'ng quyidagi regulyar ifodani generasiyalaydi:

$(\beta_1 | \beta_2 | \ldots | \beta_s)(\alpha_1 | \alpha_2 | \ldots | \alpha_r)^*$

SHuni ta`kidlash lozimki, chap rekursiyani (yoki chap rekursiv takrorlashni) yo'qotish natijasida LL(1) grammatikaga ega bo'lamiz, shunday qilib. olingan grammatika qoidalarining chap qismidagi ba`zi bir noterminallarga alternativ o'ng qismlar mavjud bo'lib, ular bir xil belgilardan boshlanadilar. SHuning uchun chap rekursiyani yo'qotilgandan keyin grammatikani LL(1) ko'rinishgacha aylantirishni davom ettirish kerak.

5.19.Faktorlash

LL(1) grammatika xususiyatlariga ega bo'lmagan ko'pgina grammatika holatlarida grammatikani faktorlash jarayoni yordamida LL(1)grammatikaga aylantirish mumkin bo'ladi. Quyidagicha misolni qaraymiz.

P→ begin D; C end
D→ d, D
D→ d
C→ s; C
C→ s

Faktorlash jarayonida chap qismdagi o'ng tomoni bir xil belgidan (belgilar zanjiridan) boshlanadigan bir noterminalning bir necha qoidalarni bitta qoidaga, ya`ni o'ng qismida umumiy boshlanishda qo'shimcha kiritilgan noterminal keladigan, qoidaga almashtiramiz. YAna shu bilan birga grammatikaga qo'shimcha noterminal uchun qo'shimcha qoidalar qo'shiladi.

YUqorida keltirilgan grammatika uchun bu quyidagi LL(1) grammatikani beradi.

P→ begin D; C end
D → dX (qo'shimcha noterminal X kiritamiz)
X→, D (D uchun boshlang'ich grammatikaning 1-chi qoidasi bo'yicha d dan keyin, D keladi)
X →ε (D uchun boshlang'ich grammatikaning 2-chi qoidasi bo'yicha d dan keyin hech nima yo'q (bo'sh qator))
C→ sY (qo'shimcha Y noterminal kiritamiz)
Y → ; C (S uchun boshlang'ich grammatikaning 1-chi qoidasi bo'yicha s dan keyin, C keladi)
Y→ε (C uchun boshlang'ich grammatikaning 2-chi qoidasi bo'yicha s dan keyin hech nima yo'q (bo'sh qator))
SHunday qilib, quyidagi tug'iluvchi qoidalarni

S → aSb
S →aSc
S →ε

faktorlash yo'li bilan quyidagi qoidalarga aylantirish mumkin:

S → aSX
S → ε
X→ b
X → c

va olingan natija LL(1) grammatika bo'ladi.

Faktorlash jarayonini umumiy holatlarga tadbiq etib, avtomatlashtirish mumkin emas. Keyingi misol bu holatda qanday hol yuz berishi mumkinligini ko'rsatadi. Quyidagi qoidalarni qaraymiz:

1. $P \rightarrow Qx$ 4. $Q \rightarrow q$
2. $P \rightarrow Ry$ 5. $R \rightarrow sRn$
3. $Q \rightarrow sQm$ 6. $R \rightarrow r$

Yo'naltiruvchi belgilarning ikki to'plami R ning ikkala variantida s mavjud, va s ni qavsdan tashqariga chiqarishga urinib, biz Q va R ni 1 va 2 qoidalarni o'ng tomoniga almashtiramiz.

$P \rightarrow sQmx$ $P \rightarrow qx$

$P \rightarrow sRny$ $P \rightarrow ry$

Bu qoidalarni quyidagilar bilan almashtirish mumkin:

$P \rightarrow qx$ $P_1 \rightarrow Qmx$

$P \rightarrow ry$ $P_1 \rightarrow Rny$

$P \rightarrow sP_1$

P_1 uchun qoidalar R uchun qoidalarga o'xshash va yo'naltiruvchi belgilarning kesishuvchi to'plamlariga ega. Biz ushbu qoidalarni xuddi R qoidalar kabi aylantiramiz:

$P_1 \rightarrow sQmmx$ $P_1 \rightarrow sRnny$

$P_1 \rightarrow qmx$ $P_1 \rightarrow rny$

Faktorlash natijasida

$P_1 \rightarrow qmx$ $P_2 \rightarrow Qmmx$

$P_1 \rightarrow rny$ $P_2 \rightarrow Rnny$

$P_1 \rightarrow sP_2$

P_2 uchun qoidalar P_1 va P kabidir, lekin ulardan uzunroq, va endi ko'rinib turibdiki bu cheksiz jarayondir. SHu yo'l bilan faktorlash zarur aylantirishni amalga oshiradi, ba`zi bir grammatikalarni esa LL(1) ko'rinishga aylantirishning iloji yo'q.

5.20.KO-tillarning qaytishsiz pastdan yuqoriga yuruvchi anglovchilari

LR(k) grammatikaning ta`rifi

YUqorilovchi anglovchilar daraxtni qurishni pastdan yuqoriga qarab bajaradilar. Ularning ish natijalari bo'lib o'ngtomonli chiqish hisoblanadi. Bunday anglovchilarni ish xarakatlari «surilish-hosil qilish» algoritmiga asoslangan.

Ushbu algoritmni g'oyasi shundan iboratki, algoritm ishining har bir qadamida quyidagi savollarga birqiymatli javob berish mumkin bo'lsin:

- surilish (o'tkazish) yoki hosil qilishni bajarish mumkin bo'lsin;
- hosil qilishni bajarish uchun stekdan qanday a belgilar zanjirini tanlash kerak;
- hosil qilishni bajarish uchun qanday qoidani tanlash kerak (bir qancha quyidagi ko'rinishdagi qoidalar mavjud bo'lsalar A_1-»a, A_2-»a, ... A_p-»a).

U holda KO-til zanjirlarini anglovchi yuqorilovchi algoritm qaytishlarni talab qilmaydi, chunki tilning o'zi kengaytirilgan MX-avtomat bilan berilgan. Albatta, avval aytib o'tilganidek, umumiy holda buni barcha KO-tillar uchun qo'llash mumkin emas (chunki determinirlangan KO-tillar sinfi KO-tillar sinfiga nisbatan qisqadir). Ammo, KO-tillar orasidan shunday sinfni (yoki sinflarni) ajratib ko'rsatish mumkinki, ular uchun yuqoridagi algoritmni qo'llash mumkin bo'ladi.

Birinchi navbatda LL(k)-grammatikalar asosiga qo'yilgan yo'nalishdan foydalanish mumkin. U holda biz KO-grammatikalarning boshqa, LR(k)-grammatikalar nomi bilan ataluvchi ko'rinishiga ega bo'lamiz.

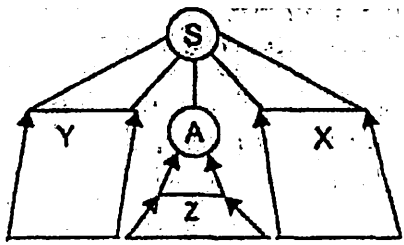
KO-grammatika LR(k), k>0 xususiyatga ega bo'ladi, agar chiqishning har bir qadamida masalani, «surilish-hosil qilish» («perenos-svertka») algoritmi bo'yicha, birqiymatli echimi uchun MX-avtomatga stekning yuqori qismidagi qiymatni bilish va kiruvchi belgilar zanjirining avtomat o'quvchi boshining joriy holatini birinchi k belgilarini qarashi etarli bo'ladi.

Grammatika LR(k)-grammatika deb ataladi, agar u LR(k) uchun k>0 xususiyatga ega bo'lsa.

«LR(k)» atamasi, yuqoridagi qurilgan «LL(k)» kabi aniq ma`noga ega. Birinchi litera «L» kiruvchi belgilar zanjirini o'qish tartibini anglatadi: chapdan — o'ngga. Ikkinchi litera «R» «right» so'zidan kelib chiqqan bo'lib, anglovchi ishining natijasida o'ngtomonli chiqishni olinishini bildiradi. Grammatika atamasidagi «k» o'rniga, «surilish-hosil qilish» algoritmining har bir qadamida, echim qabul qilish uchun, kiruvchi belgilar zanjirining nechta belgisini qarash kerakligini ko'rsatadigan son bo'lishi kerak. SHunday qilib «LR(0)», «LR(1)» va boshqa grammatikalar sinfi mavjud.

Barcha «LR(k)» grammatikalar to'plami birgalikda barcha k>0 uchun LR-grammatikalar sinfini tashkil qiladilar.

23-rasmda qandaydir LR(k) grammatika uchun chiqish daraxtini chizma bo'lagi ko'rsatilgan. Bu erda a kiruvchi belgilar zanjirining daraxtining chap bo'lagi u belgilangan va u ko'rib chiqilgan. Daraxtning o'ng bo'lagi x —bu hali ko'rib chiqilmagan qismdir. A — noterminal belgi bo'lib, unga keyinga qadamda , MX avtomatning stekining yuqorisida joylashgan z belgilar zanjiri jamlangan. Ushbu zanjirga kiruvchi zanjirning avvaldan o'qib bo'lingan, ammo joylashtirilmagan bo'lagi ham kiradi. Daraxtning x o'ng qismi kiruvchi zanjirning asosida quriladi.



23-rasm. LR(k) grammatika uchun chiqish daraxtini chizma bo'lagi

5.21.Hamroxlik (predshestvovaniya) grammatikasi asosida anglovchi

Bunday anglovchilarning ishini tashkil etish grammatikaning har bir tartibli belgilar juftligiga hamroxlik munosabati o'rnatilishiga asoslangan.

Hamroxlik grammatikasi:
1)oddiy hamroxlik grammatikasi
2)kengaytirilgan hamroxlik
3)kuchsiz hamroxlik
4)hamroxlikning aralash strategiyalari
5)aniqlangan hamroxliklar grammatikasi.

Eng ko'p tarqalgan va oddiylari 1 va 5. To'g'ri hamroxlik grammatikasi – bu shunday keltirilgan kontekst-ozod grammatika $G(V_t, V_n,P,S)$ bo'lib unda:

1) har qanday tartibli juftlik V_t va V_n uchun uchta hamroxlikning bittadan ko'p bo'lmagan bittasi bajariladi.

2)turli tug'iluvchi qoidalar turli o'ng bo'laklarga ega bo'lishlari kerak.

Hamroxlik usuli hamroxlik munosabatini anglovchi qatorda 2 ta ?? belgili bo'lsa, quyidagi 3 variantga mos kelishi asoslanadi.

1)$F_i=F_{i+1}$ agar belgilar F_i va F_{i+1} bitta asosga tegishli bo'lsa

2) $F_i<F_{i+1}$ agar F_i qandaydir asosning chetki chap belgisi.

3) $F_i>F_{i+1}$ agar F_i qandaydir asosning chetki o'ng belgisi.

Keyin keltirib chiqarish matrisasi quriladi.

Kamchiligi: har qanday determinirlangan kontekst-ozod til oddiy keltirib chiqarish grammatikasi bilan berilgan bo'lishi mumkin; agar grammatikada terminal va noterminal belgilar ko'p bo'lsa, u holda keltirib chiqarish matrisasi kattalashadi.

Operatorli keltirib chiqarish grammatikasi – bu keltirilgan kontekst-ozod grammatika bo'lib, unda barcha qoidalarning o'ng bo'lagi noterminal belgilarni o'zida jamlamaydi va ular uchun keltirib chiqarish munosabatlari V_t boshlanish va oxirgi belgilarni qo'shgan holda to'plamda beriladi.

Sinov savollari

1.Kontekst-ozod grammatikaga ta`rif bering.

2.Xomskiyning normal formasi nima?

3.Greybaxning normal formasi nima?

4.Grammatikani o'z-o'ziga yuklashiga misol keltiring.

5.Qanday til determinirlangan deb ataladi?

6.Determinirlangan chekli avtomat ko'rinishida KO grammatikani amalga oshirish mumkinmi? Magazin turidagi avtomat ko'rinishida KO grammatikani amalga oshirish mumkinmi?

7.LL(1) –grammatikaga ta`rif bering va uning qanday toifaga tegishli ekanligini aniqlang.

8.Yo'naltiruvchi belgi nima?

9.Quyida keltirilgan grammatika (S- boshlang'ich belgi) LL(1) – grammatikami? Javobingizni asoslang.

S $\rightarrow$ P|P P$\rightarrow$ (S) | o | PBP B$\rightarrow$ + | - | * | /

10.Tahlil muammosi nima?

11.CHap tomonli va o'ng tomonli tahlil nima?

12.Nima uchun asosiy e`tibor, juda ko'p sonli chap yoki o'ng tomonli bo'lmagan tahlillar bo'la turib, chap va o'ng tomonli tahlillarga qaratilgan?

13.Sintaksis daraxtni qurishda tahlil muammosini qanday hal etish mumkin?

14.Bir qiymatli bo'lmagan grammatikaga ta`rif bering.

15.Determinirlangan va determinirlanmagan tahlil usullarini farqini aytib bering

16.Quyidagi tug'iluvchi qoidalarga ega grammatika berilgan:

S $\rightarrow$ S+T F $\rightarrow$ (S)

S $\rightarrow$ T F $\rightarrow$ a

T $\rightarrow$ T*F F $\rightarrow$ b

T $\rightarrow$ F

a) (a+b)*a+a ifoda uchun sintaksis daraxtni quring.

b) (a+b)*a+a ifoda uchun chap tomonli tahlilni quring.

c) (a+b)*(a+b) ifoda uchun o'ng tomonli tahlilni quring.

17.Quyidagi tug'iluvchi qoidalarga ega grammatika bir qiymatli emasligini isbotlang.

S $\rightarrow$ if c then S else S S $\rightarrow$ x

S $\rightarrow$ if c then S

18.CHekli avtomatlar Xomskiy ierarxiyasi bo'yicha qaysi grammatika turiga tegishlidir?

19.CHekli avtomatga ta`rif bering.

20.Determinirlangan chekli avtomat nodeterminirlangan chekli avtomatdan qanday farq qiladi?

6.BO'LIM

Translyatorlarni qurishning asosiy tamoyillari

Translyatorlar. kompilyatorlar va interpretatorlarning umumiy ishlash chizmasi. Zamonaviy kompilyatorlar va interpretatorlar

6.1.Translyator, kompilyator va interpretatorning ta`rifi

Translyator — bu tizimli qayta ishlovchi dastur bo'lib, u yuqori daraja dasturlash tilida yozilgan boshlang'ich (kiruvchi) dasturni unga teng kuchli bo'lgan (chiquvchi) mashina ko'rinishidagi dasturga o'giradi. Ushbu ta`rifda uch marta dastur so'zi ishtirok etmoqda, bu xato emas. Translyatorning ishida haqiqatan ham uchta dastur ishtirok etadi.

Birinchidan, translyatorning o'zi dastur hisoblanadi va u ko'pincha hisoblash tizimining tizimli dasturiy ta`minoti tarkibiga kiradi. YA`ni translyator — bu dasturiy ta`minotning bo'lagi bo'lib, u mashina komandalari va berilganlarning to'plamini ifodalaydi va kompyuter tomonidan boshqa dasturlar kabi operasion tizim ramkasida bajariladi. Translyatorning barcha tarkibiy qismlari o'zining kiruvchi va chiquvchi berilganlariga ega bo'lgan dastur modullari yoki bo'laklarini ifodalaydi.

Ikkinchidan, translyator ishining boshlang'ich berilgani bo'lib kiruvchi dastur matni xizmat qiladi — bu kiruvchi dasturlash tilining qandaydir gaplari ketma-ketligidir. Ko'pincha bu belgili fayl bo'lib, u kiruvchi tilning sintaksis va semantik talablarini qanoatlantiruvchi dastur matni bo'lishi kerak. Bundan tashqari ushbu fayl kiruvchi til semantikasi bilan aniqlangan ma`noga bo'lishi kerak.

Uchinchidan, translyatorning chiquvchi berilganlari bo'lib natijaviy dastur matni hisoblanadi. Natijaviy dastur translyatorning chiquvchi tilida berilgan sintaksis qoidalarga asosan quriladi, uning ma`nosi esa chiquvchi til semantikasi bilan aniqlanadi. Translyatorning ta`rifida eng muhim talab - bu kiruvchi va chiquvchi dasturlarning ekvivalentligi hisoblanadi. Ikki kiruvchi va chiquvchi dasturlarning ekvivalentligi ularning kiruvchi til semantikasi (boshlang'ich dastur uchun) va chiquvchi tilning semantikasi (natijaviy dastur uchun) nuqtai nazaridan mos tushishini anglatadi. Ushbu talab bajarilmasa translyator o'zining barcha amaliy ma`nosini yo'qotadi.

Demak, translyatorni qurish uchun kiruvchi va chiquvchi tilni tashkil etish zarur. Kiruvchi til gaplarini unga ekvivalent chiquvchi til gaplariga aylantirish nuqtai nazaridan translyator xuddi tarjimon sifatida qaraladi. Masalan, S dasturlash tilidan assembler tiliga dasturni translyasiya qilish ma`no jihatidan rus tilidan ingliz tiliga tarjima qilishdan farq qilmaydi, faqat farqi shundaki, tillarning murakkabligi boshqacharok. SHuning uchun ham «translyator» (inglizcha: translator) «tarjimon» degan ma`noni anglatadi. Boshlang'ich dastur matni to'g'ri, ya`ni kiruvchi til nuqtai nazaridan sintaksis va semantikasi xatosiz bo'lsa, translyatorning ish natijasi bu natijaviy dastur bo'ladi. Agar boshlang'ich dastur noto'g'ri bo'lsa (bitta bo'lsa ham xatoga ega bo'lsa), u holda translyator ishining natijasi xato haqidagi ma`lumotni

berish bo'ladi. Bu ma`noda translyator qo'liga noto'g'ri matn berilgan til tarjimoniga o'xshashdir.

«Translyator» tushunchasidan tashqari unga ma`no jihatidan yaqin bo'lgan «kompilyator» tushunchasi qo'llaniladi.

Kompilyator — bu translyator bo'lib, u boshlang'ich dasturni unga ekvivalent bo'lgan mashina komandalari yoki assember tilidagi ob`ekt dasturga o'giradi. SHunday qilib, kompilyator translyatordan faqatgina uning natijaviy dasturi mashina kodlari yoki assember tilida yozilganligi bilan farqlanadi. Translyatorning natijaviy dasturi, umumiy holda ixtiyoriy tilda yozilishi mumkin, masalan, Pascal tilidan S tiliga translyator. Mos ravishda har qanday kompilyator translyator hisoblanadi, lekin har qanday translyator kompilyator bo'lmaydi. Masalan, yuqorida aytib o'tilgan Pascal tilidan S tiliga translyator kompilyator hisoblanmaydi. «Kompilyator» so'zi inglizcha «compiler» («tuzuvchi», «yig'uvchi») terminidan kelib chiqadi. Termin o'zining kelib chiqishiga kompilyatorlarning boshlang'ich dasturlar asosida ob`ekt dasturlarni tuzish qobiliyati tufayli erishgan bo'lsa kerak. Kompilyatorning natijaviy dasturi «ob`ekt dastur» yoki «ob`ekt kod» deb ataladi. U yozilgan fayl ko'pincha «ob`ekt fayl» deb ataladi. Hatto, natijaviy dastur mashina komandalari tilida yaratilsa ham, ob`ekt dastur (ob`ekt fayl) va bajariluvchi dastur (bajariluvchi fayl) o'rtasida sezilarli farq mavjud. Kompilyator tomonidan yaratilgan dastur kompьyuterda bevosita bajarila olmaydi, chunki u o'zining kodi va berilganlari joylashgan aniq xotira sohasiga bog'langan emas. Kompilyatorlar translyatorlarning eng keng tarqalgan ko'rinishidir (ko'pchilik hattoki, yagona ko'rinishi deb hisoblaydilar, lekin aslida bunday emas). Tabiiyki, translyatorlarni va kompilyatorlarni, barcha boshqa dasturlar kabi insonlar ishlab chiqaradilar — ko'pincha bu ishlab chiqaruvchilar guruhidir. Ular kompilyatorlarni bevosita mashina komandalari tilida yaratishlari mumkin edi, lekin zamonaviy kompilyatorlarning kod va berilganlari hajmi shundayki, ularni mashina komandalari tilida ma`lum muddat va xarajatlar bilan yaratish mumkin emas. SHuning uchun barcha zamonaviy kompilyatorlar ham kompilyatorlar yordamida yaratiladi (ko'pincha kompilyatorlarning ishlab-chiqaruvchi firmaning avvalgi variantlaridan foydalaniladi). Demak kompilyator boshqa kompilyator uchun chiquvchi dastur hisoblanadi.

«Translyator» va «kompilyator» kabi o'zaro o'xshash tushunchalardan tashqari, «interpretator» tushunchasi mavjud.

Interpretator — bu kiruvchi dasturni boshlang'ich tilda qabul qiladigan va bajaradigan dasturdir. Translyatorlardan farqli ravishda interpretatorlar natijaviy dasturni yaratmaydilar — va ana shu ular o'rtasidagi asosiy farqlaridan biridir. Interpretator translyator kabi boshlang'ich dastur matnini tahlil qiladi. Lekin u, natijaviy dasturni yaratmasdan, birdaniga kiruvchi til semantikasiga mos ravishda berilgan boshlang'ich dasturni bajaradi. SHunday qilib, interpretator ishining natijasi boshlang'ich dastur ma`nosidan kelib chiqqan holda, agar dastur to'g'ri berilgan bo'lsa, natijaga erishiladi va agar boshlang'ich dasturda xatoliklar bo'lsa, u holda xatoliklar haqidagi ma`lumot chiqariladi. Albatta, boshlang'ich dasturni bajarish

uchun, interpretator u yoki bu holda dasturni mashina kodlari tiliga aylantirishi kerak, chunki aks holda dasturni kompyuterda bajarish mumkin emas. U shunday ham qiladi, lekin olingan mashina kodlarini interpretatorni foydalanuvchisi qura olmaydi.

6.2.Translyatorlar, kompilyatorlar va interpretatorlarning belgilanishi

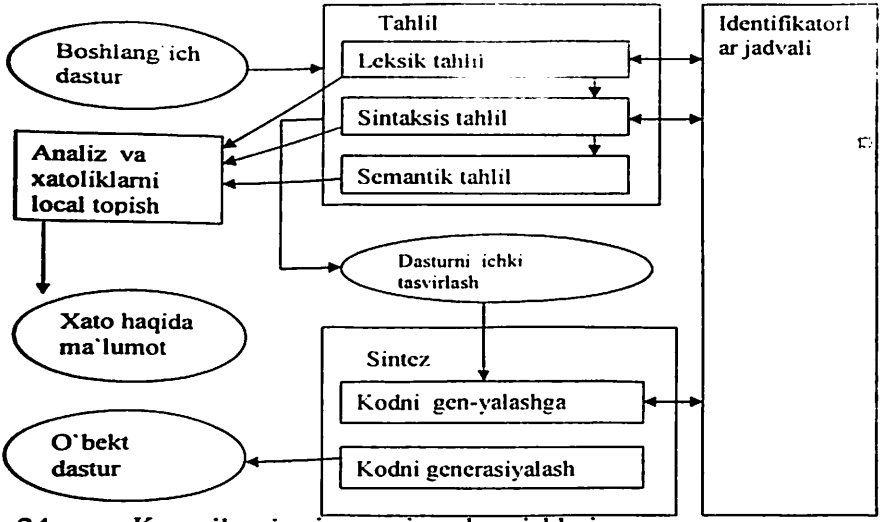
Birinchi avlod EHM uchun yaratilgan birinchi dasturlar bevosita mashina kodlarida yozilgan edi. Bu esa juda mashaqqatli ish edi. Inson, hattoki hisoblash texnikasi sohasida zo'r mutaxassis bo'lsa ham, mashina komandalari tilida gapirishi mumkin emas edi. Lekin kompyuterni inson tilida gapirishga o'rgatishga urinishlar ham samarasiz ketdi. SHundan beri kompyuterlarning barcha dasturiy ta`minoti kompilyatorlarning yuzaga kelishi va rivojlanishi bilan uzviy bog'liq. Birinchi kompilyatorlar bu assembler tili kompilyatorlari, ya`ni mnemokodlar edi. Mnemokodlar mashina komandalari tilini mutaxassis tushunadigan ushbu komandalarning mnemonik belgilashlar tiliga aylantirdilar (dasturlarni berish anchagina sezilarli darajada soddalashdi, lekin mnemokodni o'zini bajarishni esa (assembler tili) bironta ham kompyuter uddalay olmaydi, shuning uchun kompilyatorlarni yaratish zaruriyati yuzaga keldi. Bu kompilyatorlar elementar darajada oddiy, lekin ular dasturlash tizimlarida hozirgi kungacha muhim rol o'ynab kelmokdalar. Keyingi bosqich bu yuqori daraja dasturlash tillarini yaratish bo'ldi. YUqori daraji dasturlash tillari (ularga ko'pgina dasturlash tillari kiradi) insonlarning tabiiy muloqot tili va formal tillar orasidagi oraliq zvenoni ifodalaydilar. YUqori daraja dasturlash tillarining yaratilishi dasturlash jarayonini sezilarli ravishda soddalashtirdi. Bunday yuqori daraja dasturlash tillari hozirgi kunda yuztadan oshib ketgan. Ushbu jarayonning oxiri yo'q. Lekin shunday bo'lsada, hozirda ham tradision «neyman» arxitekturasiga ega, faqat mashina komandalarini tushunadigan, kompyuterlar hali ham nazardan qolmaganlar. SHu sababli, kompilyatorlarni yaratish masalasi hali ham aktual bo'lib qolmoqda.

Hozirgi kunda kompilyatorlar ixtiyoriy hisoblash tizimining ajralmas bo'lagi hisoblanadilar. Mashina kodlari tilida dasturlash hozirgi kunda juda kam holat va uchraganda ham juda tor soha uchun uchraydi. Endi kompilyatorlar va interpretatorlarning qo'llanilishiga doir, shu bilan birga ularning boshqa mavjud dasturiy vositalar bilan moslashuvi haqida gaplashamiz. Kompilyatorlar, keyinchalik ko'rib o'tamiz, qo'llanilishi nuqtai nazaridan interpretatorlarga qaraganda anchagina soddadirlar. Foydalilik bo'yicha ham ular ustunlikka egadirlar, ya`ni kompilyasiya qilingan kod xuddi o'sha boshlang'ich dasturni interpretasiyasiga qaraganda tezroq bajariladi. Bundan tashqari, har bir dasturlash tili ham oddiy interpretatorni qurish imkonini beravermaydi. Lekin interpretatorlarning bitta sezilarli ustunligi mavjud, ya`ni kompilyasiya qilingan kod har doim, o'zi mo'ljallangan hisoblash tizimining arxitekturasiga bog'liq bo'ladi, boshlang'ich dastur esa faqat dasturlash tilining semantikasiga bog'liq bo'ladi. Bu holatni avvalboshda e`tiborga olmaganlar. Birinchi kompilyatorlar mnemokodli kompilyatorlar edi. Ularning avlodlari assembler tilidan o'giruvchi zamonaviy kompilyatorlar barcha tanïqli hisoblash tizimlarida mavjud. Ular arxitekturaga juda qattiq mo'ljallangan. So'ngra FORTRAN, ALGOL-68, PL/1 kabi tillarning kompilyatorlari paydo bo'ldilar. Ular masalalarni paketli qayta ishlovchi katta EHM uchun mo'ljallangan edilar. YUqorida sanab o'tilganlardan

faqat FORTRANdan, hozirgi kungacha foydalanilmokda. chunki u turli belgilashli katta sonli kutubxonalarga ega [7]. ADA, Modula, Simula kabi ko'pgina tillar keng tarqalmadilar, ular faqatgina mutaxassislarning tor doirasi uchun xizmat qildilar. SHu bilan birga dasturiy tizimlar bozorida yorqin kelajak bashorat qilinmagan kompilyator tillari, birinchi navbatda bu C va C++ rivojlanib bormoqda. Ulardan birinchisi UNIX toifasidagi operasion tizimlar bilan birgalikda tug'ildi va o'zining «quyosh ostidagi o'rniga» ega bo'ldi, so'ngra boshqa toifa OTga o'tdi. Ikkinchisi esa o'zida, amaliyotda o'zi haqida yaxshi tassurot qoldirgan, ob`ektlarga yo'naltirilgan dasturlash g'oyalarining qo'llanilishini aks ettirdi. YAna keng tarqalgan Pascal tilini ham eslash mumkin, u ham to'satdan, ko'pchilik uchun, universitet muhitidagi toza o'quv tilidan tashqariga chiqdi.

Interpretatorlarning tarixi juda ham boy emas (hozircha!). Avval aytganimizdek, ularga sezilarli ahamiyat berilmadi, chunki ular deyarli barcha parametrlari bo'yicha kompilyatorlarga joy bo'shatadilar. Taniqli tillardan interpretasiyani taklif qiladigan faqat Basicni eslash mumkin, ammo hozirda ko'pchilikka uning Microsoft firmasi tomonidan bajarilgan Visual Basic toifasi, ma`lum. Bundan tashqari, hozirgi kunda, Internet tarmog'ining rivojlanishi bilan bog'liq holda, dasturlarni ko'chish imkoniyati va ularning apparat-platformaga bog'liq emasligi masalasi aktuallik kasb etmokda. Eng taniqli misol — bu Java dasturlash tili (u o'zida ham kompilyasiyani, ham interpretasiyani jamlaydi), va yana u bilan bog'liq JavaScript dir. Va nihoyat, HTTP bayonnoma tashkil etiladigan, butundunyo tarmog'ining rivojiga turtki bo'lgan HTML tili ham interpretatordir. Kelajakda bizni, yangi interpretatorlarning paydo bo'lishi sohasida yangi syurprizlar kutmokda, va ularning birinchilari paydo bo'ldi ham — masalan, Microsoft. firmasi tomonidan yaratilgan C# («Ci-diez» tili, hamma joyda «Ci sharp» deb ataladi.

6.3.Translyasiya bosqichlari. Translyator ishining umumiy chizmasi



24-rasm.Kompilyasiya jarayoninng bosqichlari

Kompilyator formal tillar nuqtai nazaridan quyidagi 2 asosiy funksiyalarni bajaradi: 1) u kiruvchi dastur matni tili uchun anglovchi hisoblanadi (kiruvchi dastur zanjirlar generatori bo'lib hisoblanadi); 2)natijaviy dastur tili uchun generator hisoblanadi (anglovchi bo'lib hisoblash tizimi hisoblanadi)

24-rasmda kompilyator ishining umumiy chizmasi keltirilgan. Undan ko'rinadiki, kompilyasiya jarayoni ikki asosiy fazadan tashkil topadi.

1. tahlil fazasi
2. sintez fazasi

1.Tahlil fazasi bosqichida boshlang'ich dastur matnini anglash, ya`ni identifikatorlar jadvalini tashkil etish va to'ldirish, leksik tahlilchining ishi bajariladi (skanerlash). Uning ish natijasi bo'lib kompilyatorga tushunarli bo'lgan dasturning ichki ifodalanishi xizmat qiladi.

Sintez bosqichida dasturning ichki ifodalanishi va jadvallardagi ma`lumotlar asosida natijaviy dasturning matni yuzaga keladi. Ushbu bosqichning natijasi ob`ekt kod hisoblanadi.

Bundan tashqari, kompilyator tarkibida xatolarni tahlil etish va to'g'rilashga javob beruvchi bo'lak mavjud bo'lib, boshlang'ich dastur matnidagi xatolik mavjud holida u foydalanuvchini xatolik toifasi va yuz bergan joyi haqida to'liq ma`lumot bilan ta`minlashi kerak. Eng yaxshi holda kompilyator foydalanuvchiga xatolikni tuzatish variantini taklif etadi.

Bu bosqichlar, o'z navbatida kompilyasiya fazalari deb ataluvchi mayda bosqichlardan tashkil topadi. Kompilyasiya fazasi tarkibi umumiy holda aniq amalga oshirilish va o'zaroxarakatlari holda keltirilgan.

Birinchidan, u boshlang'ich dastur tili uchun anglovchi hisoblanadi. U kiruvchi tilning belgilar zanjirini olishi va uni ushbu tilga tegishli yoki yo'qligini tekshirishi, bundan tashqari ushbu zanjir qurilgan qoidalarni keltirib chiqarishi kerak. Kiruvchi til zanjirlarining generatori bo'lib, foydalanuvchi- kiruvchi dastur avtori hisoblanadi.

Ikkinchidan, kompilyator natijaviy dastur tilining generatori hisoblanadi. U chiqishda chiquvchi til zanjirini, assembler tili yoki mashina komandalari taklif qilgan, aniq qoidalar asosida qurishi kerak. Ushbu zanjirning anglovchisi bo'lib, endi, natijaviy dasturni yaratuvchi, hisoblash tizimi chiqadi.

Endi kompilyasiya fazasining asosiy fazalari (bo'laklari) va ularning funksiyalarini qisqacha ta`rifini keltiramiz.

Leksik tahlil – bu kompilyator bo'lagi bo'lib, u kiruvchi tildagi dastur literalarini o'qiydi va ular orqali kiruvchi til leksemalarini quradi. Leksik tahlilchi kirishiga boshlang'ich dastur matni kelib tushadi, chiquvchi ma`lumot esa keyingi ishlov uchun sintaksis tahlilchiga uzatiladi.

Sintaksis tahlil – tahlil bosqichidagi kompilyatorning asosiy bo'lagidir. U tilning leksik tahlilchi tomonidan boshlang'ich dastur matnida qayta ishlangan sistaksis konstruksiyalarini ajratadi. Kompiyasiyaning ushbu fazasida dasturning sintaksis to'g'riligi tekshiriladi. Sintaksis bo'lak kiruvchi til dastur matnini anglashni bajaruvchi asosiy o'rinni egallaydi («Sintaksis tahlilchilar». «Sintaksis boshqariluvchi tarjima» bo'limlarini qarang).

Semantik tahlil – bu kompilyator bo'lagi bo'lib, kiruvchi til semantikasi nuqtai nazaridan dastur matnini tekshiradi. Bundan tashqari, semantik tahlilchi kiruvchi til semantikasi bo'yicha talab qilingan matn aylantirishlarini amalga oshirishi kerak (bu, toifalarni aniq bo'lmagan aylantirishlari funksiyasini qo'shish kabi). Kompilyatorlarni amalga oshirishning turli variantlarida semantik tahlilchi ko'pincha sintaksis tahlilchi fazasiga kirishi mumkin, ko'pincha —kodni generasiyalash fazasiga ham kirishi mumkin.

Kodni generasiyalashga tayyorgarlik – natijaviy dasturning sintezi bilan bog'liq bo'lgan, lekin chiquvchi tildagi matnni tug'ilishiga olib kelmaydigan, xarakatlarga tayyorgarlik bajariladi. Ko'pincha ushbu fazaga til elementlarini identifikasiyalash, xotirani taqsimlash bilan bog'liq xarakatlar kiradi («Semantik tahlil va kodni generasiyalashga tayyorgarlik» bo'limiga qarang).

Kodni generasiyalash – bevosita chiquvchi til gaplarini tashkil etuvchilarini kodlarini tug'ilishi va natijaviy dastur matnini hosil etish bilan bog'liq faza – kodni optimizasiyalashni o'z ichiga olgan asosiy faza.

Ushbu keltirilgan fazalar kompilyatorning versiyasidan bog'liq holda farqlanishi mumkin. Ammo ushbu barcha fazalar u yoki bu ko'rinishda har bir kompilyatorda mavjud bo'ladilar.

Identifikatorlar jadvali (gohida –«belgilar jadvali) –bu maxsus holda tashkil etilgan ma`lumotlar to'plami bo'lib, undi kiruvchi dastur elementlari haqidagi ma`lumotlar saqlanadi, so`ngra ushbu ma`lumotlardan natijaviy dasturning matnini hosil etishda foydalaniladi.

Identifikatorlar jadvali aniq kompilyatorning amalga oshirilishida bitta, yoki bir nechta mavjud bo'lishi mumkin. Kiruvchi dastur elementlari bo'lib, ya`ni kompilyasiya jarayonida saqlash kerak bo'ladigan ma`lumotlar bo'lib, o'zgaruvchilar, konstantalar, funksiyalar va x.k. hisoblanadi, elementlarning aniq to'plami foydalanilayotgan kiruvchi dasturlash tiliga bog'liq. «Jadval» tushunchasi ma`lumotlarni saqlash albatta jadval ko'rinishida tashkil etilishi shart degan fikrdan yiroqdir, uni tashkil etishning mumkin bo'lgan variantlari «Identifikatorlar jadvali. Identifikatorlar jadvalini tashkil etish» bo'limida to'liq ko'rib chiqiladi.

Keyinchalik ushbu kitobning bo'limlarida kompilyasiyaning ko'rib o'tilgan fazalarini tashkil etishning turli variantlarini ko'rib chiqiladi. Ushbu holda ularning bir-biriga qanday bog'liq ekanliklari ko'rsatilgan. Bu erda bunday bog'liqlikning faqat umumiy aspektlarini qarab chiqamiz.

Birinchidan, leksik tahlil fazasida kiruvchi dastur matnidan leksemalar ajratiladi, chunki ular keyingi sintaksis tahlil fazasi uchun zarurdirlar.

Ikkinchidan, quyida ko'rsatilganidak, sintaksis tahlil va kodni generasiyalash bir vaqtda bajarilishi mumkin. SHunday qilib, kompilyasiyaning bu uch fazasi aralash holda ishlashlari mumkin, ular bilan birga kodni generasiyalashga tayyorgarlik ham bajarilishi mumkin.

6.4.O'tish tushunchasi. Ko'po'tishli va biro'tishli kompilyatorlar

Avval aytilganidek, dasturlarni kompilyasiya jarayoni bir necha fazalardan tashkil topadi. Anik kompilyatorlarda ushbu fazalarning tarkibi yuqorida ko'rib o'tilganlardan farqlanishi, ba`zi birlari bir necha tashkil etuvchilarga ajralishi, boshqalari esa, aksincha, bitta fazaga birlashtirilishlari mumkin. Kompilyasiya fazalarini bajarilish tartibi ham turli kompilyatorlarda turlicha o'zgarishi mumkin. Bir holatda kompilyator kiruvchi dastur matnini qarab chiqadi, tezda kompilyasiyaning barcha fazalarini bajaradi va natija- ob`ekt kodni oladi. Boshqa variantda u kiruvchi dastur matni ustida kompilyasiyaning faqat ba`zi bir fazalarini bajaradi va oxirgi natijani olmay, balki oraliq ma`lumotlarning to'plamini oladi. Ushbu ma`lumotlar yana qayta ishlashga qo'yiladi va ushbu jarayon bir necha marta takrorlanishi mumkin.

Aniq kompilyatorlar kiruvchi dastur matnini o'girishni bir necha o'tishlar orqali bajaradilar.

O'tish – bu tashqi xotiradan berilganlarni ketma-ket o'qish jarayoni, ularni qayta ishlash va tashqi xotiraga joylashtirishdir. Kompilyasiyaning bir fazasi - bir o'tishdir. Oraliq o'tishlarning natijasi bo'lib, kiruvchi dastur matnini ichki ifodalanishi hisoblanadi, oxirgi o'tishning natijasi esa natijaviy ob`ekt dastur hisoblanadi.

Tashqi xotira sifatida ixtiyoriy ma`lumot tashuvchilar, kompьyuterning operativ xotirasi, magnit disklari, magnit lentalari va x.k. kelishi mumkin.

Zamonaviy kompilyatorlar, ma`lumotlarni saqlash uchun kompьyuterni operativ xotirasidan maksimal ravishda foydalanishga xarakat qiladilar, va mumkin bo'lgan xotirani etishmagan hollaridagina qattiq magnit disklaridan foydalanadilar. Boshqa ma`lumot tashuvchilar zamonaviy kompilyatorlarda, ma`lumot almashinuvining yuqori bo'lmagan tezligi sababli, foydalanilmaydi.

Har bir o'tishning bajarilishida kompilyatorga barcha avvalgi o'tishlarning natijasida olingan ma`lumotlarga ruxsat mavjud. O'tishlarning bajarilishidagi olinadigan ma`lumotlarga foydalanuvchi uchun ruxsat yo'q. Ular translyasiya jarayoni tugaganidan so'ng kompilyator tomonidan ozod etiladigan operativ xotirada, yoki vaqtinchalik fayllar ko'rinishida disklarda saqlanadi, va ular kompilyator ishining tugatilishi bilan yo'qotiladi. SHu sababli, kompilyatorda ishlovchi inson kompilyator nechta o'tishni bajarishini bilmasligi mumkin, u har doim kiruvchi dastur matnini va natijaviy ob`ekt dasturni quradi. Lekin bajarilgan o'tishlar soni - bu kompilyatorning muhim texnik xarakteristikasi bo'lganligi sabab, taniqli firmalar —kompilyatorlarni ishlab chiqaruvchilar, ko'pincha uni o'zlarining mahsulotlarini ifodalanishida ko'rsatadilar.

Kompilyatorlarni ishlab chiqaruvchilar o'tishlar sonini maksimal darajada qisqartirishga xarakat qiladilar, chunki bu holda kompilyator ishining tezligi oshadi va unga zarur xotira hajmi kamayadi.

Kirishida boshlang'ich dasturni oluvchi va o'sha zaxoti natijaviy ob`ekt kodini tug'diruvchi biro'tishli kompilyator bu ideal variantdir. Ammo o'tishlar sonini qisqartirishni har doim ham amalga oshirish mumkin emas. O'tishlarning zarur soni kiruvchi tilning grammatikasi va semantika qoidalari asosida aniqlanadi. Tilning grammatikasi qanchalik murakkab bo'lsa va semantik qoidalar qanchalik ko'p variantlarni taklif etsalar, kompilyator shunchalik ko'p o'tishlarni bajaradi. Masalan, xuddi shu sababli Pascal tilining kompilyatorlari S tilining kompilyatorlariga nisbatan. tezroq ishlaydi. Pascal tilining grammatikasi oddiyrok, semantik qoidalari esa qattiqroqdir. Biro'tishli kompilyator juda sodda tillar uchun mumkin bo'lib, ular juda kam uchraydilar. Real kompilyatorlar kamida ikkitadan beshtagacha o'tishlarni bajaradilar. SHunday qilib, real kompilyatorlar kupo'tishli hisoblanadilar. Eng ko'p tarqalganlari ikki- va uch o'tishli kompilyatorlar, masalan: birinchi o'tish— leksik tahlil, ikkinchisi — sintaksis tahlil va semantik tahlil, uchinchisi — generasiyalash va kodni optimallash (bajarilish variantlari albatta ishlab chiqaruvchilarga bog'liq).

Zamonaviy dasturlash tizimlarida, ko'p hollarda, kompilyatorni birinchi o'tishi (kodni leksik tahlili) boshlang'ich dastur kodini tahrirlash bilan parallel bajariladi (kompilyatorlarni bunday qurilishi varianti kitobning keyingi bo'limlarida keltiriladi).

6.5.Interpretatorlar. Interpretatorlarni qurishning xususiyatlari

Interpretator — bu boshlang'ich tildagi kiruvchi dastur matnini qabul qiladigan va bajaradigan dasturdir. Avvalda aytilganidek, interpretatorlarning translyatorlar va kompilyatorlardan asosiy farqi shundaki, interpretator natijaviy dasturni keltirib chiqarmaydi, u faqat boshlang'ich dasturni bajaradi xolos.

«Interpretator» (interpreter) termini «translyator» kabi «tarjimon»ni anglatadi. Termin nuqtai nazaridan ushbu tushunchalar o'xshash, lekin formal tillar nazariyasi va kompilyasiya nuqtai nazaridan ular o'rtasida katta prinsipial farq mavjud. Agar «translyator» va «kompilyator» tushunchalarini bir-biridan ajratib bo`lmasa ham, «interpretator» tushunchasi haqida bunday deya olmaymiz.

Interpretatorning eng oddiy usuli sifatida boshlang'ich dastur avval to'liq mashina komandalariga translyasiya qilinadigan, so'ngra bajariladigan variantini hisoblashimiz mumkin. Bunday interpretator kompilyatordan juda kam farq qilib, faqatgina unda natijaviy dasturga foydalanuvchiga ruxsat yo'q bo'ladi. Bunday interpretatorning kamchiligi shundaki, foydalanuvchi boshlang'ich dasturni kompilyasiyasini u bajarilishga qo'yilishidan avval kutishi kerak edi. Haqiqatda esa bunday interpretatorda muhim ma`no yo'q- u analogik kompilyator bilan taqqoslaganda hech qanday ustunlikka ega emas. SHu sababli, ko'pgina interpretatorlar boshlang'ich dasturni ketma-ket ravishda, ya`ni ularning interpretator kirishiga kelib tushishiga qarab bajaradilar. Bu holda foydalanuvchining barcha boshlang'ich dasturni kompilyasiyasini kutib o'tirishga hojat yo'q. Bundan tashqari, u kiruvchi dasturni ketma-ket kiritishi va shu zahoti komandalarning kiritilishiga qarab natijani kuzatib turishi mumkin.

Interpretator ishining bunday tartibida shunday sezilarli xususiyat paydo bo'ladiki, u interpretatorni kompilyatordan farqlaydi, agar interpretator komandalarni kelib tushishi tartibi bo'yicha bajarsa, u boshlang'ich dasturni optimallashni bajara olmaydi. Bundan kelib chiqadiki, optimallash fazasi interpretatorning umumiy strukturasida mavjud bo'lmaydi. Bundan boshqa tomondan interpretatorning strukturasi kompilyatordan juda kam farqlanadi. Faqat shuni e`tiborga olish kerakki, oxirgi kodni generasiyalash bosqichida mashina komandalari ob`ekt faylga yozilmaydilar, ularni kelib chiqishi bo'yicha bajariladilar.

Optimallash qadamining mavjud emasligi ko'pgina interpretatorlar uchun xarakterli bo'lgan yana bir xususiyatni aniqlaydi: dasturlarni ichki tasvirlash uchun ularda juda ko'p hollarda teskari polyak yozuvlaridan foydalaniladi (qarang. «Kodni generasiyalash. Kodni generasiyalash usullari», 11-bo'lim). Bu amallarni qulay ifodalash usuli bo'lib, faqat bitta sezilarli kamchilikka ega — uni optimallashtirish murakkab ish. Lekin interpretatorlarda xuddi ana shu xususiyat talab etilmaydi.

Barcha dasturlash tillari ham boshlang'ich dasturni komandalarning kelib tushishi bo'yicha bajaradigan interpretatorlarni qurishga imkon bera olmaydilar. Buning uchun til boshlang'ich dastur tahlilini bitta o'tish davomida bajaruvchi kompilyatorni mavjud bo'lishiga imkon berishi kerak. Bundan tashqari, til berilganlar funksiyalari va strukturalariga murojaat qilish uchun, ularni bevosita ifodalanishidan avval, imkon bersa, komandalarni kelib tushishi bo'yicha ushbu til interpretasiya qilina olmaydi. SHu sababli ushbu usul bilan C va Pascal kabi tillar interpretasiya qilina olmaydilar.

Optimallash qadamining yo'qligi interpretator yordamida dasturni bajarilishi kompilyator yordamida dasturni bajarilishidan ko'ra foydasizroqdir. Bundan tashqari, interpretasiya jarayonida boshlang'ich dastur har safar bajarilishi mobaynida yangidan tahlil qilinishi kerak bo'ladi, kompilyasiya jarayonida esa u bir marta tahlil qilinadi va boshqa safar esa ob`ekt fayldan foydalanish mumkin bo'ladi. SHunday qilib,

interpretatorlar har doim kompilyatorlarga ishlab chiqarish tezligi bo'yicha yutqazadilar.

Interpretatorlarning ustunligi dasturlarning maqsadli hisoblash tizimining arxitekturasiga bog'liq bo'lmay bajarilishi hisoblanadi. Natijada kompilyasiya jarayonida har doim aniq arxitekturaga yo'naltirilgan ob`ekt kodi olinadi. Maqsadli hisoblash tizimining boshqa arxitekturasiga o'tish uchun dasturni qaytadan kompilyasiya qilinishi talab etiladi. Dasturni interpretasiyasi uchun esa faqat uni boshlang'ich dastur matni va mos til interpretatori zarur bo'ladi.

Interpretatorlar ko'p vaqtlar mobaynida kompilyatorlarga nisbatan keng tarqalmadilar. Interpretatorlar cheklangan nisbatan oddiy dasturlash tillari uchun mavjud bo'ldilar (masalan, Basic). Dasturlashning yuqori ishlab chiqarishni ta`minlovchi vositalari kompilyatorlar asosida qurildilar.

Interpretatorlarning rivojlanishiga yangi global hisoblash tarmoqlarining keng tarqalishi sabab bo'ldi. Bunday tarmoqlar o'z tarkibiga turli arxitekturali EHMlarini kiritishi mumkin va bu holda ularning har birida boshlang'ich dastur matnini bir marta bajarilishini ta`minlash talabi aniqlovchi bo'lib hisoblanadi. SHu sababli, global tarmoqlarning rivojlanishi bilan va Internet dunyo tarmog'ining keng tarqalishi bilan boshlang'ich dastur matnini interpretasiya qiladigan yangi tizimlar paydo bo'ldi. Butundunyo tarmog'ida qo'llaniladigan ko'p dasturlash tillari ob`ekt kodini yaratmasdan dastur matnini interpretasiya qilishni ko'zda tutadilar.

Zamonaviy dasturlash tizimlarida kompilyator va interpretatorlning funksiyalarini o'zida jamlovchi dasturiy ta`minot mavjud, ya`ni foydalanuvchining talabiga ko'ra dastur yoki kompilyasiya qilinadi, yoki interpretasiya qilinadi. Bundan tashqari, ba`zi bir zamonaviy dasturlash tillari ishlab chiqarishni ikki bosqichini ko'zda tutadilar: avval boshlang'ich dastur oraliq kodga kompilyasiya qilinadi (past darajali qandaydir til), so'ngra ushbu kompilsiya natijasi berilgan oraliq til interpretatori yordamida bajariladi.

Interpretasiya qilish tillarining eng ko'p tarqalgan misoli bo'lib, HTML (Hypertext Markup Language) — tili, ya`ni gipermatnlarni ifodalash tili hisoblanadi. U asosida hozirgi vaqtda Internet tarmog'ining barcha strukturasi xarakatlanadi. YAna boshqa misol — Java va JavaScript tillar bo'lib, ular o'zida kompilyasiya va interpretasiyaning funksiyalarni jamlaydilar. Boshlang'ich dastur matni qandaydir, maqsadli hisoblash tizimining arxitekturasidan bog'liq bo'lmagan, oraliq ikkilik kodga kompilyasiya qilinadi, ushbu kod tarmoq bo'yicha tarqatiladi va qabul qilinuvchi tomonda bajariladi, ya`ni interpretasiya qilinadi.

Avval aytganmizdek, interpretatorlar xuddi kompilyasiyaga o'xshash tamoyilni amalga oshiradilar. Kompilyatorlar va interpretatorlar ko'pgina umumiy xususiyatlarga egalar. Interpretator ham avvalombor boshlang'ich dasturni qarab chiqadi va undagi leksemalarni ajratadi. Buning uchun xuddi kompiyasiya qiluvchi dasturlar tarkibiga kiruvchi skanerlash bloki va sistaksis tahlilchilardan foydalaniladi. Lekin interpretator u yoki bu amallarni bajaradigan ob`ekt kodni qurish o'rniga o'zi mos xarakatlarni amalga oshiradi.

Interpretatorni yutuqlari:
- amalga oshirishning nisbatan soddaligi;
- dasturlarni qayta ishlashning qulayligi.

Kompilyatorning yutuqlari:
- bajarilish tezligi;
- bajariluvchi kodning dasturlash tizimlariga bog'liq emasligi;
- buyurtmachilarga dasturlarni boshlang'ich matnsiz berish imkoniyati.

6.6.Assembler tilining translyatorlari («assemblerlar»)

Assember tili bu quyi daraja dasturlash tilidir. Ushbu tilning strukturasi va zanjirlari orasidagi o'zaroxarakatlari maqsadli hisoblash tizimining, natijaviy dastur bajariladigan, mashina komandalariga yaqinroqdir. Assembler tilining qo'llanilishi ishlab chiqaruvchiga maqsadli hisoblash tizimining zahiralarini mashina komandalari darajasida boshqarish imkonini yaratadi (jarayonchini, tezkor xotirani, tashqi qurilmalarni va x.k) Assembler tilidagi boshlang'ich dastur matnining har bir komandasi kompilyasiya jarayoni natijasida bitta mashina komandasiga aylantiriladi.

Assembler tilining translyatori har doim kompilyator bo'lib ham hisoblanadi, chunki natijaviy dastur tili bo'lib mashina kodlari hisoblanadi. Assembler tilining translyatorini ko'pincha oddiygina qilib, «assembler» yoki «assembler dasturi» deb ataladi.

Assembler tili kompilyatorlarining amalga oshirishilishi

Assembler tili, mashina komandalaring mnemonik kodlaridan tashkil topadi. Ko'pincha ingliz tili komandalari mnemonikasidan foydalaniladi, lekin assembler tilining boshqa variantlari ham mavjud (shu jumladan rustili variantlari ham mavjud). Xuddi shuning uchun assembler tili avvalda «mnemokodlar tili» deb atalgan (hozirda ushbu atama amaliyotda umuman qo'llanilmayapti).

Assembler tilining har birida mavjud bo'lishi mumkin bo'lgan komandalarini ikki guruhga bo'lish mumkin: birinchi guruhga tilning oddiy komandalari kiradilar, va ular translyasiya jarayonida mashina komandalariga aylanadilar; ikkinchi guruhga tilning maxsus komandalari kiradilar, ular mashina komandalariga aylantirilmaydilar, lekin ulardan kompilyator tomonidan kompilyasiya masalasini bajarish uchun foydalaniladilar (masalan, xotirani tarqatish masalasi).

Tilni sintaksisi juda sodda. Boshlang'ich dastur matnlari dasturning bitta qatorida bitta komanda yozilishini ko'zda tutadi. Assembler tilining har bir komandasi uchta, biri biridan keyin ketma-ket keluvchi: belgi bo'yicha, amal kodi va operand maydoni kabi, tashkil etuvchilarga bo'linishi mumkin.

Assember tilining kompilyatori ko'pincha kiruvchi dastur matnida, komandalardan berilgan ajratkichlar bilan ajratiluvchi, izohlar bo'lishi imkoniyatini ko'zda tutadi.

Belgi maydoni belgini ifodalovchi identifikatordan tashkil topadi, yoki bo'sh bo'lishi mumkin. Belgining har bir identifikatori assembler tilining dasturida faqat bir marta uchrashi mumkin. Belgi dasturda birinchi uchragan joyida ifodalangan hisoblanadi, ya`ni belgilarni avvaldan ifodalanishi talab qilinadi. Belgidan o'zi olgan

komandaga boshqaruvni uzatish uchun foydalanilishi mumkin. Kam hollarda belgi komandalarning boshqa bo'laklaridan maxsus ajratkich bilan ajratiladi (ko'pincha bu - ikki nuqta «:»).

Amal kodi har doim jarayonchining qat`iy aniqlangan mumkin bo'lgan komandalaridan birini ifodalaydi. Amal kodi kiruvchi til alfaviti belgilari orqali yoziladi. Ularning uzunligi ko'pincha 3-4 ga teng, kam hollarda — 5 yoki 6 belgidan iboratdir.

Operandlar maydoni yoki bo'sh, yoki bitta, ikkita kam hollarda uchta operanddan tashkil topgan ro'yxatni ifodalaydi. Operandlar soni qat`iy aniqlangan va amal kodidan bog'liq bo'ladi, assembler tilining har bir amali o'zining qattiq berilgan operandlarini ko'zda tutadi. Mos ravishda, ushbu variantlarning har biriga manzilsiz, birmanzilli, ikkimanzilli yoki uchmanzilli komandalar mos keladilar (operandlarning ko'pgina soni amalda foydalanilmaydi, zamonaviy EHMlarida hattoki, uchmanzilli komandalar kam uchraydi). Operandlar sifatida identifikatorlar yoki konstantalar kelishi mumkin.

Assembler tilining xususiyatlaridan biri shundaki, qator identifikatorlar jarayonchi registrlarini belgilish uchun maxsus ajratiladi. Bunday identifikatorlar bir tomondan, avvaldan ifodalashni talab qilmaydilar, lekin boshqa tomondan ular foydalanuvchi tomonidan boshqa maqsadlarda ishlatila olmaydilar. Bunday identifikatorlar to'plami har bir assembler tili uchun aniqlangan bo'ladi.

Gohida assembler tili operandlar sifatida registrlarni, identifikatorlarni va konstantalarni belgilashning aniqlangan chegaraviy tarkibidan foydalanishni ko'zda tutadi va ular qandaydir amal ishoralari bilan birlashtiriladilar. Bunday tarkiblar ko'pincha manzillash toifalarini belgilash uchun maqsadli hisoblash tizimining mashina komandalarida foydalaniladi.

Masalan, quyidagi komandalar ketma-ketligi

```
datas db  16 dup(0) loops: mov datas[bx+4],cx dec
bx jnz loops
```

assembler tilining Intel 80x86 jarayonchilarining oilasi komandalari ketma-ketligini ifodalaydi. Bu erda (db) berilganlar to'plamini ifodalash komandasi, (loops) belgisi, (mov, dec va jnz) amal kodlari mavjud. Operandlar bo'lib identifikator berilganlari to'plami (datas) hisoblanadi, jarayonda registrlarni (bx va sx), belgini (loops) va konstanta (4) sifatida belgilangan. Tarkibiy operand datas[bx+4] datas berilganlar to'plamini baza registrlari bx va sx bo'yicha 4 birlikka surgan holda bilvosita manzillashni aks ettiradi.

Tilning bunday sintaksisini regulyar grammatika yordamida qiyinchiliklarsiz ifodalanishi mumkin. SHu sababli, assembler tili uchun anglovchini qurish qiyin emas. SHu sababdan ham assembler tilining kompilyatorlarida leksik va sintaksis tahlilni bitta anglovchiga yuklatilgan.

Assembler tilining semantikasi butunligicha va to'liq, berilgan tilga mo'ljallangan, maqsadli hisoblash tizimi orqali aniqlanadi. Assembler tilining semantikasi assembler tilining har bir komandasiga qanday mashina komandasi mos

kelishini, shuningdek, qanday operandlar va qanday sonda u yoki bu amal kodi uchun ruxsat etilishini aniqlaydi.

SHu sababli kompilyatorda assembler tilidan semantik tahlil sintaksis tahlil kabi oddiy. Uning asosiy masalasi bo'lib, har bir amal kodi uchun operandlarning ruxsat etilishini tekshirish hisoblanadi, shuningdek. kiruvchi dasturda uchraydigan barcha identifikatorlar va belgilar ifodalangan va ularni belgilagan identifikatorlar avval ifodalangan, amal kodlari va jarayonchilarni belgilash uchun foydalanilgan, identifikatorlar bilan mos tushmasligini tekshiradi.

Kompilyatorda assembler tilidan sintaksis va semantik tahlil oddiy chekli avtomatlar asosida amalga oshirilgan bo'lishi mumkin. Xuddi ana shu xususiyat, assembler tilining kompilyatorlari EHM uchun tashkil etilgan birinchi kompilyatorlar sifatida tarixda qolganligini aniqlaydi. YAna bir qancha boshqa xususiyatlar ham mavjudki, ular faqat assembler tiliga tegishli va ular uchun kompilyatorlar qurishni osonlashtiradi.

Birinchidan, assembler tilining kompilyatorlarida o'zgaruvchilarni qo'shimcha identifikasiyalash bajarilmaydi — tilning barcha o'zgaruvchilari foydalanuvchi tomonidan berilgan o'z ismini saqlaydi. Ismlarning yagonaligi uchun kiruvchi dasturda ishlab chiqaruvchi javobgardir, til semantikasi bu jarayonga hech qanday qo'shimcha talablar qo'ymaydi.

Ikkinchidan, assembler tilining kompilyatorlarida xotiraning taqsimlanishi juda soddalashtirilgan. Assembler tilining kompilyatori faqat statik xotira bilan ishlaydi. Agar dinamik xotira ishlatilayotgan bo'lsa, u holda u bilan ishlash uchun mos kutubxona va OT funksiyalaridan foydalanish kerak bo'ladi, uning taqsimlanishi uchun boshlang'ich dastur ishlab chiqaruvchisi javob beradi. Parametrlarni uzatish va prosedura va funksiyalarni displey xotirasini tashkil etish uchun ham boshlang'ich dasturni ishlab chiqaruvchisi javob beradi. U yana berilganlarni dastur kodidan ajratish masalasini ham o'ylashi kerak, assembler tilining kompilyatori, yuqori daraja dasturlash tillarining kompilyatorlaridan farqli ravishda avtomatik tarzda bunday ajratishni bajarmaydi.

Va uchinchidan, assembler tilining kompilyatorida kodni generasiyalash bosqichida optimallash amalga oshirilmaydi, chunki boshlang'ich dasturni ishlab chiqaruvchisi hisoblashlarni, ya`ni mashina komandalarining ketma-ketligi va jarayonchi registrlarini taqsimlashni, tashkil etilishi uchun o'zi javob beradi. Ushbu xususiyatlarni e`tiborga olinmasa assembler tilining kompilyatori oddiy kompilyator hisoblanadi, lekin yuqori daraja dasturlash tilining ixtiyoriy kompilyatoriga nisbatan anchagina soddalashtirilgan bo'ladi. Assembler tilining kompilyatori ko'pincha ikkio'tishli chizma bo'yicha amalga oshiriladilar. Birinchi o'tishda kompilyator boshlang'ich dasturni tahlilini bajaradi, uni mashina kodlariga aylantiradi va bir vaqtning o'zida identifikatorlar jadvalini to'ldiradi. Lekin birinchi o'tishda mashina komandalarida tezkor xotirada joylashgan operandlarning manzillari to'ldirilmay qoladi. Ikkinchi o'tishda kompilyator ushbu manzillarni to'ldiradi va bir vaqtning o'zida ifodalanmagan identifikatorlarni topadi. Bu esa, operand dasturda birinchi marta ishlatilganidan keyin ifodalangan hol bilan bog'liq. U

holda uning manzili mashina komandalarini qurish vaqtida ma`lum bo'lmaydi, va shu sababli ikkinchi o'tish talab etiladi. Bunday operandga misol bo'lib, komandalar ketma-ketligi bo'yicha oldinga o'tishni ko'zlayotgan, belgi hisoblanadi.

6.7.Makroaniqlashlar va makrokomandalar

Assembler tilida dastur ishlab chiqarish bu etarli darajada murakkab, ko'p hollarda ko'p marta uchragan amallarni oddiy takrorlashni talab qiladigan jarayon bo'lib hisoblanadi. Misol sifatida proseduraga yoki funksiyaga kirishda displey xotirasini tashkil etish uchun har safar bajariluvchi komandalar ketma-ketligini ko'rsatish mumkin.

Ishlab chiqaruvchining mehnatini osonlashtirish uchun makrokomandalar tashkil etilgan.

Makrokomanda bu bajarilishi davomida har bir aniq ko'rinishli identifikator qandaydir berilganlar omboridan belgilar zanjiriga almashtiriladigan matnli o'rin almashtirishni ifodalaydi. Makrokomandani bajarilish jarayoni makrogenerasiya deyiladi, makrokomanda bajarilishi jarayonida olingan belgilar zanjiri makrokengaytma deyiladi.

Makrokomandaning bajarilish jarayoni boshlang'ich dastur matnini ketma-ket qarab chiqish, unda aniqlangan identifikatorlarni va ularni almashtiriladigan mos belgilar qatorini topishdan iboratdir. Bu erda bir belgilar zanjirini (identifikatorlar) boshqa belgilar zanjiriga (qatorga) matnli almashtirish bajariladi. Bunday almashtirish makroo'rinalmashtirish deyiladi.

Qanday identifikatorlarni qanday qatorlarga almashtirish zarurligini ko'rsatish uchun makroaniqlashlar xizmat qiladi. Makroaniqlashlar boshlang'ich dastur matnida bevosita mavjud bo'ladilar. Ular maxsus kalit so'zlar orqali yoki dastur matnida boshqa hech qaerda uchramaydigan, ajratkichlar orqali ajratiladi. Qayta ishlash jarayonida barcha makroaniqlashlar kiruvchi dastur matnidan to'liq olib tashlanadi, ulardagi mavjud ma`lumotlar esa makrokomandalarni bajarilishida qayta ishlash uchun saqlab quyiladi.

Makroaniqlashlar parametrlarga ega bo'lishi mumkin. U holda unga mos keluvchi har bir makrokomanda chaqirilganda har bir parametr o'rniga belgilar qatorini o'zida saqlashi kerak. Bu qator makrokomandaning bajarilishi jarayonida har bir o'sha mos parametr uchragan joyga qo'yiladi. Makrokomandaning parametri sifatida boshqa makrokomanda kelishi mumkin, u holda, u har safar parametrni qo'yilishini bajarilishi zaruriyati tug'ilganda rekursiv ravishda chaqiriladi. Makrokomandalar, prosedura va funksiyalarning rekursiv chaqiriklariga o'xshash, rekursiv chaqiriqlar ketma-ketligini tashkil etishlari mumkin, lekin hisoblashlar va parametrlar uzatilishi o'rniga faqat matnli o'rinalmashtirishlar bajariladi.

Makrokomandalar va makroaniqlashlar maxsus makrojarayonchi yoki makrogenerator deb atalgan modul tomonidan qayta ishlanadi. Makrogenerator kirishida makroaniqlashlar va makrokomandalarga ega boshlang'ich dastur matnini oladi, va chiqishida boshlang'ich dastur matnining, makroaniqlashlarsiz va

makrokomandalarsiz, makrokengaymasi paydo bo'ladi. Ikkala matn ham dastur matni hisoblanadi va boshqa hech qanday qayta ishlashlar bajarilmaydi. Boshlang'ich matnning xuddi ana shu makrokengaytmasi kompilyatorning kirishiga tushadi.

Makrokomandalar va makroaniqlashlar sintaksisi qat'iy berilgan emas. U assembler tili kompilyatorining amalga oshirilishiga bog'liq holda farqlanishi mumkin. Lekin dastur matnida makroo'rinalmashtirishlarni bajarilish tamoyili o'zgarmaydi va ularning sintaksisidan bog'liq emas.

Makrogenerator ko'pincha alohida dastur moduli ko'rinishida mavjud bo'lmaydi, balki assembler tili kompilyatori tarkibiga kiradi. Boshlang'ich dastur makrokengaytmasi uchun ko'pincha uni ishlab chiqaruvchisiga ruxsat yo'q. Bundan tashqari, makroo'rinalmashtirishlar boshlang'ich dastur matnini tahlil qilishda kompilyatorni birinchi o'tishida barcha dastur matnini tahlili bilan birgalikda bajarilishi mumkin, va u holda boshlang'ich dastur makrokengaytmasi butunlay mavjud bo'lmasligi ham mumkin.

Masalan, quyidagi matn assembler tilida Intel 8086 jarayonchi toifasini push_0 makrokomandasini aniqlaydi.

```
push_O macro
xog     ax, ax  push
 ax endm
```

Ushbu makrokomandani semantikasi «0» sonini stekka ax jarayonchini registri orqali yozishdan iboratdir. U holda dastur matnining barcha joyida makrokomanda uchraganda

```
push_O
```

u makroo'rinalmashtirishlar natijasida komandalar ketma-ketligiga aylanadi:

```
xog     ax, ax  push    ax
```

Bu makroaniqlashning eng oddiy variantidir. Parametrlar bilan yanada murakkabroq makroaniqlashlarni tashkil etish imkoniyatlari mavjud. SHunday makroaniqlashlardan biri quyida keltirilgan:

Bunday rekursiyaning chuqurligi, kuchli cheklangan bo'ladi. Makrokomandalarni rekursiv chaqiriqlari ketma-ketligiga ko'pincha, displey xotirasini stekli tashkil etishda faqat parametrlarni uzatish steki o'lchami cheklangan, prosedura va funksiyalarning rekursiv chaqiriqlari ketma-ketligiga nisbatan, sezilarli qattiq cheklashlar yuklatiladi.

```
 add_abx macro  x1,x2
add             ax,x1
add             bx,x1
add             cx,x2
push              ax
endm
```

U holda dastur matnida makrokomanda ham mos parametrlar soni bilan ko'rsatilishi shart. Quyidagi misolda makrokomanda add_abx4,8

makroo'rinalmashtirish natijasida quyidagi komandalar ketma-ketligiga almashtiriladi:

```
add ax,4 add bx.4 add sx,8
push ax
```

Assembler tilining ko'pgina kompilyatorlarida yanada murakkab, ya`ni lokal o'zgaruvchilar va belgilardan tashkil topgan, konstruksiyalar ham mumkin.

Bunday konstruksiyaga misol bo'lib quyidagi makroaniqlash xizmat qiladi:

```
loop_ax macro xl,x2,yl
    local loopax
    mov ax,xl
    xog bx.bx loopax: add bx,yl
    sub ax,x2
    jge loopax
    endm
```

Bu erda loopax belgi lokal hisoblanib, faqatgina ushbu makroaniqlashni ichida aniqlangan. Bu holda makrokomandani dastur matniga oddiy matnli o'rinalmashtirishni bajarib bo'lmaydi, chunki berilgan makrokomandani ikki marta bajarilsa, u holda bu dastur matnida ikkita bir xil loopax belgini paydo bo'lishiga olib keladi. Bunday holatlarda makrogenerator, ya`ni kompilyatorlarda kiruvchi dastur matnining leksik elementlarini identifikasiyalashda, barcha mumkin bo'lgan lokal o'zgaruvchi va makrokomanda belgilariga dastur chegarasida yagona ism berish uchun, matnli o'rinalmashtirishlarning murakkabrok usullaridan foydalanishi kerak. Makroaniqlashlar va makrokomandalar faqatgina assembler tillarida emas, balki ko'pgina yuqori daraja dasturlash tillarida ham keng qo'llanilmoqda. U erda ularni maxsus til preprosessori deb atalgan modul qayta ishlaydi (masalan, S tili preprosessori keng tarqalgan). Qayta ishlash tamoyili bu erda ham, xuddi assembler dasturlash tilidagi kabi preprosessor tomonidan, boshlang'ich dasturni qatorlari ustida bevosita matnni o'rin almashtirish orqali bajariladi.

YUqori daraja dasturlash tillarida makroaniqlashlar boshlang'ich dastur matnidan alohida ajratilgan bo'lishlari shart, chunki preprosessor ularni kiruvchi tilning sintaksis konstruksiyalari bilan adashtirib yubormasligi kerak. Buning uchun yoki, hech qachon boshlang'ich dastur matnida uchramaydigan, maxsus belgilar va komandalardan foydalaniladi (preprosessor komandalari), yoki makroaniqlashlar boshlang'ich dastur matnining ichki ma`noga ega bo'lmagan qismida uchraydilar — masalan, izohlar tarkibiga kirishi mumkin (bundan amalga oshirishni, masalan, Borland firmasi tomonidan yaratilgan Pascal tilining kompilyatorlarida ko'rishimiz mumkin). Makrokomandalar esa, boshlang'ich dasturning ixtiyoriy joyida uchrashi mumkin, va ularning sintaksis chaqirish kiruvchi tildagi funksiyalarni chaqirishdan farq qilmasligi mumkin.

Lekin bu o'xshashliklarga qaramay, makrokomandalar prosedura va funksiyalardan farqlanadilar, chunki natijaviy kodni keltirib chiqarmaydilar, faqatgina to'g'ridan-to'g'ri boshlang'ich dastur matnidagi matnli o'rinalmashtirishlarni

bajarishni ifodalaydilar xolos. Funksiyalar va makrokomandalarning chaqirish natijalari shu sababli juda katta farqlashishlari mumkin.

S dasturlash tilidagi misolni qaraymiz. Agar quyidagi

```
int f1 (int a) { return a + a: }
```

funksiya va unga analogik quyidagi makrokomanda berilgan bo'lsin:

```
#define f2(a) ((a) + (a))
```

u holda ularni chaqirish natijalari har doim ham bir xil bo'lmaydi.

Haqiqatda, j=f1(i) va j=f2(i) chaqiriqlar (bu erda i va j — qandaydir butunsonli o'zgaruvchilar) bitta natijaga olib keladilar. Lekin, j=f1(++i) va j=f2(++i) chaqiriqlar j o'zgaruvchini turli qiymatlarini beradilar. Bu erda gap shundaki, f2 —bu makroaniqlash bo'lgani sababli, ikkinchi holatda,

j =((++i) + (++i)) matnli o'rinalmashtirishlar bajariladi. Ko'rinib turibdiki, bu ketma-ketlikda, f1(++i) funksiyaning chaqirishda bir marta bajarilishdan farqli ravishda, ++ i amali ikki marta bajariladi.

6.8.Kompilyatorning strukturasi. Ko'po'tishli va biro'tishli kompilyatorlar

Har bir kompilyator oddiy komandalarning cheklangan to'plamini bajarish qobiliyatiga ega. Ixtiyoriy murakkab xarakatlar ushbu komandalarning ketma-ketligi ko'rinishida tasvirlanadi. YUqori daraja dasturlash tilida yozilgan dasturni bajarish uchun, uni avvalombor mashina kodlaridagi komandalar ketma-ketligiga o'tkaziladi.

Boshlang'ich berilgan dastur (qandaydir dasturlash tilida yozilgan) belgilar ketma-ketligidan iborat bo'lib, ular kompьyuterga kiritiladi va bajarilish uchun kerak bo'lgan ko'rinishga aylantiriladi.

Kompilyator bu shunday, aniq bir ko'rinishdagi belgilar qatorini (ya`ni berilgan dasturlash tilidagi dastur matnini) qabul qiladigan va boshqa belgilar qatorini (mashina tilidagi dasturni) chiqaradigan dasturdir. Kompilyatorlarga bir qator umumiy xususiyatlar xoski, ular kompilyasiya qilinayotgan dasturlarni tashkil etish jarayonini soddalashtiradilar. Ixtiyoriy kompilyator tarkibiga quyidagi uchta asosiy komponenta kiradi:

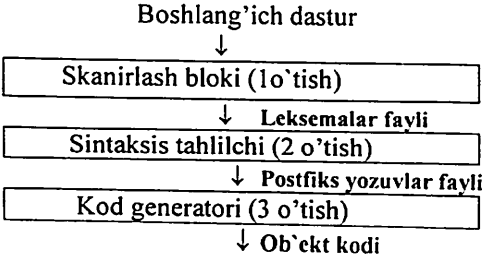
- leksik tahlilchi (skanirlash bloki);
-sintaksis tahlilchi;
-mashina komandalari kodlari generatori.

Tahlilchilarning xarakat tamoyillarini formal modellar yordamida ifodalash mumkin bo'lgan holda, kodlar generatori uchun umumiy formal tasavvurlar mavjud emas. Leksik tahlil fazasida dasturning boshlang'ich berilgan matni leksemalar deb atalgan bir biri bilan bog'lanmagan zanjirlar ko'rinishidagi belgilarga (birliklarga) ajratiladi. Bunday matnli birliklar kalit so'zlar (IF, DO va boshqalar), o'zgaruvchilar ismlari, konstantalar va amallarning ishoralari (+,- yoki *). Bunday so'ng bu so'zlar alohida belgilar guruhi emas, bo'linmaslar sifatida qaraladi. Dasturni leksemalarga bo'laklangandan so'ng, grammatik tahlil deb ataluvchi va operatorlarning ketma-ketligining to'g'riligini tekshiradigan, sintaksis

tahlil fazasi keladi. Misol uchun, «IF ifoda THEN gap» ko'rinishiga ega IF gapi uchun grammatik tahlil quyidagicha: IF leksemasidan keyin to'g'ri ifoda keladi, ushbu ifodadan keyin THEN leksemasi keladi, undan so'ng yana to'g'ri ifoda keladi va u «;» belgisi bilan tugaydi. Oxirida kodni generasiyalash jarayoni bajariladi va sintaksis tahlil natijalaridan foydalanilib bajarishga tayyor mashina tilidagi dastur tashkil etiladi. Ixtiyoriy kompilyator tarkibiga yuqorida aytib o'tilgan uchta komponenta kirsa ham, ularning o'zaro xarakati turli usullar orqali amalga oshiriladi. Ushbu komponentalar orasidagi o'zaro xarakatlarning keng tarqalgan variantlarini qarab chiqamiz.

Skanerlash bloki boshlang'ich dasturni o'qiydi va leksemalar fayli sifatida ifodalaydi. Sintaksis tahlilchi esa bu faylni o'qiydi va dasturni yangi ifodasini chiqaradi, masalan postfiks ko'rinishida. Va nihoyat, bu fayl kod generatori orqali o'qiladi va dasturni ob`ekt kodi tashkil etiladi.

Bunday ko'rinishdagi kompilyator uch o'tishli kompilyator (25-rasm) deb ataladi, chunki dastur bu jarayonda uch marta o'qiladi (boshlang'ich dastur matni, leksemalar fayl va postfiks ko'rinishidagi fayl).



25-rasm. Uch o'tishli translyator chizma ko'rinishi

Kamchiliklari: Bajarilishning juda yuqori bo'lmagan tezligi, chunki ko'pgina operasion tizimlarda fayllarga murojaat amallari nisbatan sekin bajariladi.

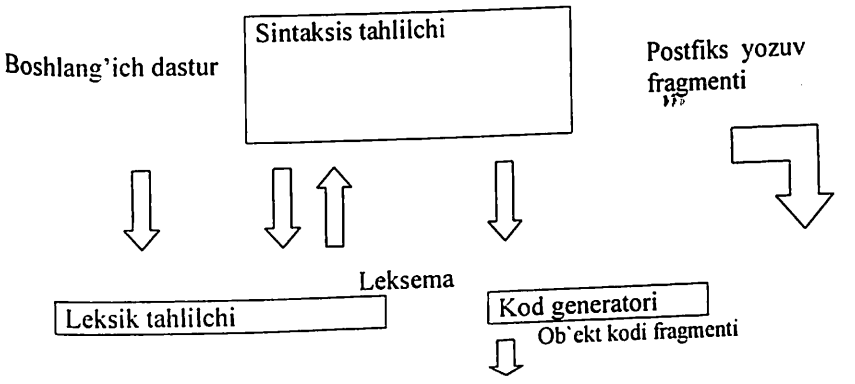
YUtuqlari: Kompiyasiyalash jarayonining har bir fazasining nisbatan biri – biriga bog'liq emasligi. CHunki qayta ishlanayotgan bloklar o'rtasidagi aloqa faqat berilganlarning fayllari orqali amalga oshiriladi, ixtiyoriy o'tish qolganlariga bog'liq bo'lmagan holda amalga oshirilishi mumkin. Bu esa:

1.Kompilyatorning turli bloklarini turli ishlab chiqaruvchilar avtonom ravishda ishlashlari imkoniyatini yaratadi va faqatgina oraliq fayllarni formalarini moslashtirish zarur bo'ladi.

2.Kompilyatorning egiluvchanligi. Masalan, turli turdagi kompyuterlar uchun bir tildan foydalanish, skanerlash va sintaksis tahlilchilarning bir xil bloklaridan foydalanish, lekin har bir tur kompyuter uchun maxsus kod generatorlarini yozishni anglatadi. Bir turdagi kompyuterlar uchun turli tillardan foydalanishda skanerlash va sintaksis tahlilchilarning turli bloklari talab qilinadi, lekin ular uchun umumiy kod generatoridan foydalanish mumkin.

3.Operativ xotira hajmiga minimal talablar (kompilyasiyaning turli faza modullarini navbatma-navbat, avvalgisini chiqarib tashlagan holda, operativ xotiraga yuklash).

Kompilyasiyaning yuqori tezligiga erishish uchun bir o'tishli strukturaga ega kompilyator (26-rasm) qo'llaniladi.



26-rasm. Bir o'tishli translyator chizma ko'rinishi

Bu holda sintaksis tahlilchi asosiy boshqaruvchi dastur sifatida qism dasturlar ko'rinishida tashkil etilgan skaner bloki va kod generatorini chaqiradi. Sintaksis tahlilchi doimiy ravishda skanerlash blokiga qaralayotgan dasturdan bir leksemadan keyin boshqa leksemani olgan holda postfiks yozuvli yangi element uchraguniga qadar murojaat etadi, undan keyin kod generatoriga ushbu dastur fragmenti uchun ob`ekt kodini tashkil etuvchi kod generatoriga murojaat qiladi.

YUtuqlari: Maksimal unumdorlik va bajarilish tezligi. CHunki dastur bir marta qaraladi va faylga murojaat amallari minimal bo'ladi (faqat berilgan fayldan o'qish va ob`ekt fayllariga yozish).

Kamchiliklari:

1.Oldinga o'tishni tashkil etishdagi muammolar. Masalan: quyidagi gapni qarasak,

GO TO metka;

«metka» hali dastur matnida uchramaganligi uchun ba`zi qiyinchiliklar yuzaga kelishi mumkin.

2.Tashkil etilayotgan ob`ekt dasturni optimal emasligi. Masalan, agar quyidagicha matn uchrasa:

A=(V+S);
R=(V+S)+(E+M); kompilyator dasturni quyidagi ko'rinishga keltirib, undan foydaliroq ob`ekt kodini tashkil etishi mumkin edi.

A=(V+S);

R=A+(E+M);

Lekin bir o'tishli kompilyator juda ko`p kerakli ma`lumotni matnda (E+M) formula uchraguniga qadar yo'qotishi mumkin.

3.Bir o`tishli kompilyator (26-rasm) xotirada to'liq joylashganligi sababli, uni amalga oshirish xotira zahiralariga yuqori talablarni qo'yadi.

Ob`ekt dastur bajarilishi unumdorligini oshirish uchun kompilyasiya jarayoniga optimizasiya bosqichini qo'shish mumkin. Optimallash bloki uch o'tishli kompilyatorga sintaksis tahlilchi va kod generatori o'rtasiga oson joylashtiriladi. Bu bosqichda postfiks fayl kiruvchi berilganlar sifatida foydalaniladi va yangilangan xarakteristikali dasturga ekvivalent postfiks yozuvli yangi fayl tashkil etiladi. Optimallash bloki o'zining chiquvchi natijalarini postfiks fayl formatida yozganligi sababli, kodlar generatoriga ularni o'zgartirish shart bo'lmaydi. Amaliyotda optimallash imkoniyati foydalanuvchi hohishiga ko'ra amalga oshiriladi, agar kompilyasiya vaqti juda ko'p vaqt olmasin desak, u holda optimallash blokini bajarmaymiz, agar bajarilish tezligi yuqori bo'lgan dastur kerak bo'lsa, u holda sintaksis tahlilchi ishidan so'ng optimallash bloki chaqiriladi.

YUqorida ko'rib o'tilgan ikki variantdan oraliq holatni egallovchi ikki o'tishli kompilyatorni qarab chiqamiz. Bu holda sintaksis tahlilchi skanerlash blokini chaqiradi, leksemalarni ketma-ket o'qiydi va dasturni postfiks yozuvlar faylini tashkil etadi. Kod generatori ushbu faylni o'qiydi va dasturni ob`ekt kodini tashkil etadi. Bunday strukturaga nisbatan kam bajarilish vaqti sarf bo'ladi, chunki dastur ikki marta o'qiladi (boshlang'ich matn va postfiks yozuvlar fayli). Bu holatda oldinga metka bo'yicha o'tish muammosi hal etiladi, chunki ushbu metka birinchi o'tish bosqichida kod generatorini chaqirishdan avval o'qiladi. Bunday kompilyatorga zaruriyat bo'lganda optimallash blokini qo'shish oson amalga oshiriladi. Barcha ko'rib o'tilgan kompilyatorlar ma`lum ish sharoitlarida o'zining yutuq va kamchiliklariga ega.

Sinov savollari

1. Translyator tushunchasiga ta`rif bering.
2. Translyator va kompilyatorning farqini ayting.
3. Interpretatorning yutuqlari va kamchiliklari haqida ma`lumot bering.
4. Kompilyatorning yutuqlari va kamchiliklari haqida ma`lumot bering.
5. Assember tilining translyatorlari haqida ma`lumot bering.
6. Nima sababdan Assember tili quyi daraja tili bo'lib hisoblanadi?
7. Assembler tilining qanday xususiyatlarini bilasiz?
8. Makrokomandalardan foydalanish nima uchun kerak?
9. Makrogeneratorning vazifasi nimalardan iborat?
10. Makroaniqlashlar nima?
11. Makrochaqiriqlar nima?
12. Makrokengaytma tushunchasiga ta`rif bering.
13. O'tish tushunchasi haqida ma`lumot bering.
14. Bir-ikki va uch o'tishli kompilyatorlarning yutuqlari nimalarda ko'rinadi?
15. Kompilyator va interpretatorning farqi nimada?
16. Bir-ikki va uch o'tishli kompilyatorlarning kamchiliklari nimalarda ko'rinadi?

7.BO'LIM
Identifikatorlar jadvalini tashkil etish

Identifikatorlar jadvalini tashkil etishning maqsadi va xususiyatlari. Identifikatorlar jadvalini tashkil etishning oddiy usullari. Xesh funksiya va xesh manzillash

7.1.Identifikatorlar jadvalini tashkil etishning belgilanishi va xususiyatlari

Semantikaning to'g'riligini tekshirish va kodni generasiyalash boshlang'ich tildagi dasturda uchraydigan o'zgaruvchilar, konstantalar, funksiyalar va boshqa elementlarning xususiyatlarini bilishni talab qiladi. Boshlang'ich dasturda ushbu elementlar identifikatorlar sifatida belgilanadilar. Boshlang'ich dasturda identifikatorlar va boshka elementlarni ajratish leksik tahlil fazasida amalga oshiriladi. Ularning xususiyatlari sintaksis ajratish, semantik tahlil va kodni generasiyaga tayyorlash fazasida aniqlanadi. Mumkin bo'lgan xususiyatlar va ularni aniqlash usullari kiruvchi til semantikasiga bog'liq. Har qanday ixtiyoriy holda ham kompilyator barcha topilgan identifikatorlarni va ularga bog'liq xususiyatlarni butun kompilyasiya jarayoni davomida, kompilyasiya jarayonining turli fazalarida foydalanish imkoniyatiga ega bo'lish uchun, saqlab turish imkoniyatiga ega bo'lishi kerak. Ushbu maqsadlar uchun, kompilyatorlarda maxsus berilganlarni saqlash joylari, ya`ni belgilar jadvali yoki identifikatorlar jadvali deb atalgan, jadvallardan foydalaniladi.

Ixtiyoriy identifikatorlar jadvali maydonlar to'plamidan iborat bo'lib, ularning soni boshlang'ich dasturda topilgan turli identifikatorlar soniga teng. Har bir maydon jadvalning ushbu elementi haqida to'liq ma`lumotni saqlaydi. Kompilyator bitta yoki bir nechta identifikatorlar jadvali bilan ishlashi mumkin, ularning soni kompilyatorni amalga oshirilishiga bog'liq. Masalan, boshlang'ich dasturning turli modullari uchun turli identifikatorlar jadvalini, yoki kiruvchi tilning turli toifali elementlari uchun tashkil etish mumkin.

Boshlang'ich dasturning identifikatorlar jadvalida saqlanayotgan har bir elementi uchun ma`lumotlar tarkibi kiruvchi tilning semantikasi va element toifasiga bog'liq. Masalan, identifikatorlar jadvalida quyidagi ma`lumotlar saqlanishi mumkin:

- o'zgaruvchilar uchun:
  - o'zgaruvchi ismi;
  - o'zgaruvchining qabul qiluvchi qiymatining toifasi;
  - o'zgaruvchi bilan bog'liq xotira sohasi;
- konstantalar uchun:
  - konstanta nomi (agar bor bo'lsa);
  - konstanta qiymati
  - konstanta toifasi (agar talab qilinsa);
- funksiyalar uchun:
  - funksiya ismi;
  - funksiyaning formal argumentlari soni va toifasi;
  - qaytariluvchi natija toifasi;

O funksiya kodi manzili.

Demak, identifikatorlar jadvali – kiruvchi dastur elementlari haqidagi ma'lumotlarni saqlovchi berilganlar to'plami. Bir necha xil identifikatorlar jadvali mavjud bo'lishi mumkin.

Identifikatorlar jadvali leksik tahlil fazasida tashkil etiladi va ketma-ket to'ldiriladi. Leksik tahlil fazasida identifikatorlar turlari yoziladi. Kodni generasiyalashga tayyorgarlik fazasida – xotira ajratiladi.

7.2.Identifikatorlar jadvalini tashkil etishning oddiy usullari

Identifikatorlar jadvalini tashkil etishning eng oddiy usuli elementlarni jadvalga ularning kelib tushishi bo'yicha joylashtirishdir. U holda identifikatorlar jadvali ma'lumotlarning tartibsiz, har bir yacheykasi jadvalning mos elementi haqidagi ma'lumotlarni saqlovchi, to'plamidan tashkil topadi.

Jadvalda kerakli elementni qidirish bu holda, kerakli element topilguniga qadar qidirilayotgan elementni jadvalning har bir elementi bilan ketma-ket solishtirishdan iborat bo'ladi.

Identifikatorlar jadvaliini tashkil etishning asosiy kriteriysi bo'lib qidiruv vaqti hisoblanadi, chunki asosiy funksiyalar yozuvlarni qidiruv va qo'shishdan iborat (ko'proq qidiruv amalga oshiriladi).

U holda, agar ikki elementni (ikkita qatorni taqqoslash) solishtirish uchun kompilyator tomonidan sarf kilingan vaqtni birlik sifatida qabul qilsak, u holda, N elementni o'zida ifodalovchi jadval uchun o'rta hisobda N/2 ta taqqoslashlar bajariladi. Bunday jadvalni to'ldirish juda sodda quyidagicha amalga oshiriladi: jadvalning oxiriga yangi elementni (T_3) qo'shish va ushbu elementni qo'shish uchun talab qilinadigan vaqt, jadvaldagi N elementlar sonidan bog'liq emas. Ammo, agar N katta bo'lsa, u holda qidirish jarayoni ko'pgina vaqt sarflanishini talab etadi. Bunday jadvalda qidirish vaqti (T_p) ni quyidagicha baholash mumkin: $T_p = O(N)$.

Identifikatorlar jadvalida qidirish ko'pincha, kompilyator bajarishi kerak bo'lgan amal hisoblanganligi, va boshlang'ich dasturdagi turli identifikatorlar soni etarli darajada ko'p bo'lishi sababli (bir necha yuzdan bir necha ming elementgacha), identifikatorlar jadvalini tashkil etishning bunday usuli foydasiz usul hisoblanadi.

Agar jadvalning elementlari, qandaydir tabiiy tartibga mos ravishda, tartiblangan bo'lsa qidirishni foydalirok amalga oshirilishi mumkin bo'ladi

Bizning holatda qidirish identifikator ismi bo'yicha, ya'ni alfavit tartibida to'g'ri yoki teskari yo'nalishda, amalga oshiriladi. Tartiblangan N elementli ro'yxatda foydali qidirish usuli bo'lib, binar yoki logarifmik qidirish usuli hisoblanadi. Topish kerak bo'lgan belgini jadval o'rtasida (N+1)/2 element bilan solishtiriladi. Agar ushbu element qidirilayotgan element hisoblanmasa, u holda biz faqat 1 dan (N+1)/2-1 gacha raqamlangan elementlar blokini, yoki (N+1)/2+1 dan N gacha bo'lgan elementlar blokini qidirilayotgan element solishtirilgan elementdan katta yoki kichikligiga bog'liq holda, qarab chiqishimiz shart. So'ngra jarayon ikki marta kichik

o'lchamli kerakli blok ustida takrorlanadi. Bu holat. yoki element topilguniga qadar, yoki algoritm bitta yoki ikkita elementli navbatdagi blokga etguniga qadar davom etadi (qidirilayotgan elementni ushbu elementlar bilan to'g'ridan to'g'ri taqqoslash mumkin bo'ladi). Har bir qadamda bosh elementni saqlashi mumkin bo'lgan elementlar soni, ikki marta qisqarganligi sababli, maksimal taqqoslashlar soni l+log2(N) ga teng bo'ladi. U holda identifikatorlar jadvalidagi elementni qidirish vaqtini quyidagicha baholash mumkin: $T_p = O(\log_2 N)$. N=128 bo'lgan holda, taqqoslash uchun binar qidirish usulida, eng ko'pi bilan 8 ta taqqoslash talab etilishi mumkin, tartiblanmagan jadvaldagi qidirish esa, o'rtacha 64 ta taqqoslashni talab etadi. Usulni, har bir qadamda qaralayotgan ma`lumotlar soni ikki marta qisqarganligi sababli, «binar qidirish», to'plamdagi kerakli elementni topish vaqti elementlarning umumiy sonidan logarifmik bog'liqlikka ega bo'lganligi sababli, «logarifmik» deb ataladi.

Usulning kamchiligi identifikatorlar jadvalining tartiblanishini talab etilishidir. Qidirish olib boriladigan ma`lumotlar to'plami tartiblangan bo'lishi kerak bo'lganligi sababli, uning to'ldirilish vaqti to'plamdagi elementlar soniga bog'liq bo'ladi. Identifikatorlar jadvali to'ldirilguniga, to'liq to'ldirilganligi, qadar qarab chiqiladi, shu sababli tartiblanish sharti unga murojaatning barcha bosqichlarida bajarilishi talab etiladi. Natijada, jadvalni qurish uchun faqat elementlarni to'g'ri tartibli qo'shish algoritmidan foydalanish mumkin.

Jadvalga har bir yangi elementni qo'shish uchun avval ushbu yangi elementni qaerga joylashtirish joyini aniqlash zarur, so'ngra ma`lumotni bo'lagini jadvalga ko'chirishni, agar element uning oxiriga qo'shilmayotgan bo'lsa, bajarish kerak bo'ladi. Agar tartiblangan to'plamlarni tashkil etish uchun qo'llanilayotgan standart algoritmlardan foydalansak, qo'shish joyini qidirish uchun binar qidirish algoritmidan foydalansak, u holda jadvalga barcha elementlarni joylashtirish uchun zarur vaqtni quyidagicha baholash mumkin:

$T_3 = O(N*\log_2 N) + k*O(№)$.

Bu erda k — berilganlarni ko'chirish va kompьyuter tomonidan taqqoslashlarni bajarish amallarini vaqti o'rtasidagi moslikni aks ettiruvchi qandaydir koeffisientdir.

Natijada identifikatorlar jadvalida logarifmik qidirishni tashkil etishda kerakli elementni qidirish vaqtini, jadvalga yangi elementni joylashtirish vaqtini ko'paytirish hisobiga, sezilarli ravishda qisqartirishga erishamiz. Identifikatorlar jadvaligi yangi elementlarni qo'shish, ularga murojaat etishga nisbatan kam amalga oshirilishini e`tiborga olsak, ushbu usulni tartiblanmagan usulga nisbatan foydaliroq deb hisoblashga haqlimiz.

7.3.Binar daraxt usuli bo'yicha identifikatorlar jadvalini qurish

Identifikatorlar jadvalida qidirilayotgan elementni qidirish vaqtini, uni to'ldirishga sarflanadigan vaqtni ko'paytirmasdan, qisqartirish mumkin. Buning uchun identifikatorlar jadvalini berilganlarning uzluksiz to'plami ko'rinishida tashkil etishni rad qilish kerak bo'ladi. Kompilyator kamida yangi identifikatorni jadvalga qo'shishda identifikator mavjudmi yoki yo'qligini tekshirishi kerak

bo'ladi, chunki ko'pgina dasturlash tillarida bironta ham identifikator bir martadan ortiq ifodalanishi mumkin emas. SHunday qilib, har bir yangi elementni qo'shish amali uchun bittadan kam bo'lmagan qidirish amalini bajarilishi kerak bo'ladi. Jadvalni tashkil etishning shunday usuli mavjudki, unda jadval binar daraxt ko'rinishini oladi. Daraxtning har bir tuguni jadvalning elementi bo'lib, ildiz tugun esa, jadvalning to'ldirishda uchraydigan, birinchi element hisoblanadi. Daraxt binar deb ataladi, chunki undagi har bir cho'qqi ikkitadan ortiq shohga ega bo'lishi mumkin emas (va, natijada, pastda yotuvchi ikkita cho'qqidan ortiq cho'qqiga ega emas). Aniqlik uchun ushbu ikki shohni «o'ng» va «chap» deb ataladi.

Binar daraxtni to'ldirish algoritmini qaraymiz. Algoritm, identifikatorlarga ega bo'lgan kiruvchi ma`lumotlar oqimi bilan ishlaydi deb hisoblaymiz (kompilyatorda ushbu ma`lumotlar oqimi kiruvchi dastur matnini tahlili jarayonida keltirib chiqariladi). Birinchi identifikator, daraxtning cho'qqisiga joylanadi. Barcha qolgan identifikatorlar daraxtga quyidagicha kelib tushadilar. Daraxtni qurish jarayonida elementlar bir biri bilan solishtiriladi va natijalarga qarab daraxtdagi keyingi yo'l tanlanadi. Faraz qilaylik, yangi identifikatorni yozish kerak bo'lsin. Buning uchun chap qismdaraxt tanlanadi, agar ushbu identifikator joriy identifikatordan kichik bo'lsa, agar katta bo'lsa u holda o'ng qismdaraxt tanlanadi. Agar joriy tugun oxirgi bo'lmasa, u holda keyingi tugunga o'tiladi (tanlangan yo'nalishga asosan) va yana qo'shilgan identifikatorni joriy identifikator bilan solishtiramiz.

Ushbu usul uchun talab qilingan solishtirishlar soni va olinadigan daraxt ko'rinishi identifikatorlarni qanday tartibda kelib tushishiga bog'liq.

Binar daraxtni qurish algoritmini quyida keltiramiz:

1-qadam. Kiruvchi ma`lumotlar oqimidan navbatdagi identifikatorni tanlash. Agar navbatdagi identifikator yo'q bo'lsa, u holda daraxtni qurish tugadi.

2-qadam. Ildiz cho'qqini daraxtning joriy tuguni deb olish.

3-qadam. Navbatdagi identifikatorni daraxtning joriy tugunidagi identifikator bilan solishtirish.

4-qadam. Agar navbatdagi identifikator kichik bo'lsa, u holda 5-qadamga o'tish, agar teng bo'lsa xatolik haqida ma`lumot berish va algoritmni bajarilishini tugatish (ikkita bir xil identifikatorning mavjud bo'lishi mumkin emas), aks holda 7-qadamga o'tish.

5-qadam. Agar joriy tugunda chap cho'qqi mavjud bo'lsa, u holda uni joriy tugun etish va 3-qadamga qaytish, aks holda 6-qadamga o'tish.

6-qadam. YAngi cho'qqini tashkil etish, unga navbatdagi identifikatorni joylashtirish, ushbu yangi cho'qqini joriy tugunni chap cho'qqisi deb belgilash va 1-qadamga qaytish.

7-qadam. Agar joriy tugunda o'ng cho'qqi mavjud bo'lsa, u holda uni joriy tugun etish va 3-qadamga qaytish, aks holda 8-qadamga o'tish.

8-qadam. YAngi cho'qqini tashkil etish, unga navbatdagi identifikatorni joylashtirish, ushbu yangi cho'qqini joriy tugunni o'ng cho'qqisi deb belgilash va 1-qadamga qaytish.

Misol tariqasida quyidagi identifikatorlar ketma –ketligini qaraymiz: GA, Dl, M22, E, A12, VS, F. Pastdagi 27–rasmda ushbu identifikatorlar ketma-ketligi uchun binar daraxtni qurishning barcha bosqichlari keltirilgan.

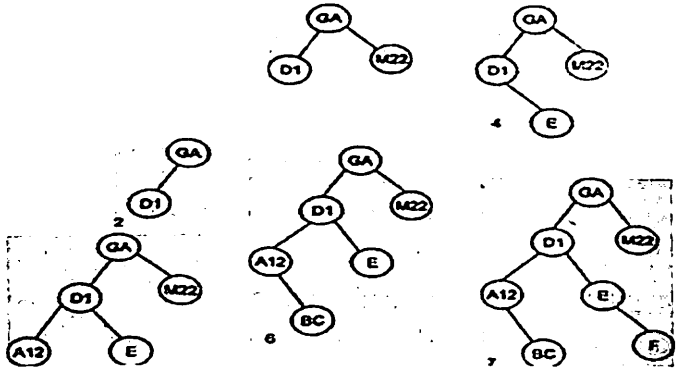
Daraxtda kerakli elementni qidirish daraxtni to'ldirish algoritmiga o'xshash ravishda bajariladi.

1-qadam. Ildiz cho'qqini daraxtni joriy tuguni sifatida olish.

2-qadam. Qidirilayotgan identifikatorni daraxtning joriy tugunidagi identifikator bilan taqqoslash.

4-qadam. Agar identifikatorlar mos tushsalar, u holda qidirilayotgan identifikator topildi, algoritm tugadi, aks holda 5-qadamga o'tish.

5-qadam. Agar navbatdagi identifikator kichik bo'lsa, u holda 6-qadamga o'tish, aks holda 7-qadamga o'tish.



27-rasm. Binar daraxtni, GA, Dl, M22, E, A12, VS, F identifikatorlar ketma-ketligi uchun, qadamlar bo'yicha to'ldirish

6-qadam. Agar joriy tugunda chap cho'qqi mavjud bo'lsa, u holda uni joriy tugun etish va 2-qadamga qaytish, aks holda qidirilayotgan identifikator topilmadi, algoritm tugadi.

7-qadam. Agar joriy tugunda o'ng cho'qqi mavjud bo'lsa, u holda uni joriy tugun etish va 2-qadamga qaytish, aks holda qidirilayotgan identifikator topilmadi, algoritm tugadi.

Masalan, 27-rasmda keltirilgan daraxtda A12 identifikatorni qidirishni amalga oshiramiz. Ildiz cho'qqini olamiz (u joriy tugun bo'lib qoladi), GA va A12 identifikatorlarni taqqoslaymiz. Qidirilayotgan identifikator kichik — joriy tugun bo'lib chap cho'qqi D1 hisoblanadi. YAna identifikatorlarni taqqoslaymiz. Qidirilayotgan identifikator kichik — u holda, joriy tugun bo'lib chap cho'qqi A12 qoladi. Keyingi taqqoslashda qidirilayotgan identifikator topildi.

Agar mavjud bo'lmagan identifikatorni qidirsak — masalan, AN, — u holda qidirish yana ildiz cho'qqidan boshlanadi. GA va AN identifikatorlarni taqqoslaymiz. Qidirilayotgan identifikator kichik — joriy tugun bo'lib chap cho'qqi D1 qoladi. YAna identifikatorlarni taqqoslaymiz. Qidirilayotgan identifikator kichik — joriy tugun bo'lib chap cho'qqi A12 qoladi. Qidirilayotgan identifikator kichik, lekin A12 tugunda chap cho'qqi mavjud emas, shuning uchun bu holda qidirilayotgan identifikator topilmadi.

Avval aytilganidek, ushbu usul uchun talab qilingan solishtirishlar soni va olinadigan daraxt ko'rinishi identifikatorlarni qanday tartibda kelib tushishiga bog'liq.

Masalan, agar qaralayotgan misolda GA, Dl, M22, E, A12, VS, F identifikatorlar ketma-ketligi o'rniga A12, GA, Dl, M22, E, VS, F ketma-ketlikni olsak, u holda olingan daraxt boshqa ko'rinishga ega bo'ladi. Agar misol sifatida A, V, S, D, E, F identifikatorlar ketma-ketligini olsak, u holda daraxt biryo'nalishli tartiblangan bog'langan ro'yxatni tashkil etadi. Bu xususiyat identifikatorlarni tashkil etishning ushbu usulining kamchiligi bo'lib hisoblanadi. YAna bir kamchiligi bu daraxtni qurishda dinamik xotira bilan ishlash zaruriyatidir.

Agar kiruvchi dasturdagi identifikatorlar ketma-ketligini statik tartibsiz deb faraz qilsak, u holda binar daraxtni qurish uchun sarflanadigan o'rtacha vaqt (T_3) va unda elementni qidirishni (T_n) quyidagicha baholash mumkin:

$T_3 = N*O(\log_2 N)$. $T_n - O(\log_2)$.

Umuman olganda binar daraxt usuli identifikatorlar jadvalini tashkil etishning qulay mexanizmi hisoblanadi. U bir qator kompilyatorlarda qo'llanilgan. Gohida kompilyatorlar turli daraxtlarni identifikatorlarning turli toifalari va turli uzunliklari uchun quradilar.

7.4.Xesh-funksiyalar va xesh-manzillash

Xesh-funksiyalarning ish tamoyillari

Identifikatorlar jadvalini qidirish va to'ldirish vaqtining logarifmik bog'liqligi – bu identifikatorlar jadvalini turli usullarni qo'llagan holda tashkil etib erishilgan eng yaxshi natijadir. Lekin aniq dasturlarda identifikatorlar soni shu darajada ko'pki, hattoki qidirish vaqtining ularning sonidan logarifmik bog'liqligini qonikarli deb bo'lmaydi.

Xesh funksiyalar va xesh –manzillashdan foydalanilgan usullardan foydalanib yanada yaxshiroq natijalarni olish imkoni mavjud.

Xesh-funksiya F deb R kiruvchi elementlarning to'plamini qandaydir butun manfiy bo'lmagan Z: $F(r) = n$, $r \in R$, $n \in Z$ sonlarni akslantirishga aytiladi. «Xesh-funksiya» termini inglizcha «hash function» (hash — «meshatь», «smeshivatь», «putatь») terminidan kelib chiqadi, «xeshlash» termini o'rniga ko'pincha «randomizasiya», «qayta tartiblash» terminidan foydalaniladi.

R kiruvchi elementlarning mumkin bo'lgan to'plami xesh –funksiyaning aniqlanish sohasi deb ataladi. F xesh-funksiyaning qiymatlar to'plami deb Z: $M \in Z$

butun manfiy bo'lmagan sonlar to'plamining barcha F: Vr∈R: F(r) ∈M funksiya qaytargan qiymatlarni o'zida jamlovchi M qism to'plamiga aytiladi.

Xesh-funksiyaning aniqlanish sohasining qiymatlar to'plamiga aks ettirilishi jarayoni «xeshlash» deb ataladi.

Identifikatorlar jadvali bilash ishlashda xesh-funksiya identifikatorlar ismlarini butun manfiy bo'lmagan sonlar to'plamiga aks ettirishni bajarishi kerak.

Xesh-funksiyalarning aniqlanish sohasi identifikatorlar ismlarining barcha mumkin bo'lgan to'plamidir.

Xesh-manzillash xesh funksiya tomonidan qaytarilgan qiymatni qandaydir ma`lumotlar to'plamidan olingan yacheyka manzili sifatida foydalanishni anglatadi. U holda ma`lumotlar to'plamining o'lchami xesh funksiya foydalanayotgan qiymatlar sohasiga mos kelishi kerak.

SHunday qilib, aniq kompilyatorda xesh funksiyaning qiymatlari sohasi kompьyuterning mumkin bo'lgan manzillar fazosi o'lchamidan oshmasligi shart.

Xesh- manzillashdan foydalanib identifikatorlarni tashkil etish usuli ushbu element uchun hisoblangan xesh-funksiya tomonidan qaytariladigan manzil bo'yicha jadvalning har bir elementini yacheykaga joylashtirishdan iboratdir.

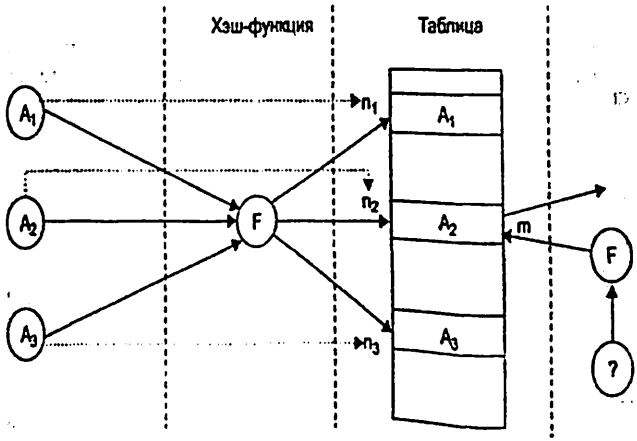
U holda, ideal holatda identifikatorlar jadvalining ixtiyoriy elementini joylashtirish uchun uning xesh –funksiyasini hisoblash va ma`lumotlar to'plamining kerakli yacheykasiga murojaat etish etarli bo'ladi.

Jadvalda elementni qidirish uchun element uchun xesh-funksiyani hisoblash va to'plamning berilgan yacheykasi bo'sh emasligini tekshirish zarur (agar u bo'sh bo'lmasa-element topildi, agar bo'sh bo'lsa-element topilmadi).

Quyidagi 28-rasmda xesh-manzillashdan foydalangan holda identifikatorlarni jadvalini tashkil etish usuli keltirilgan. Rasmda uchta turli A_1, A_2, A_3 identifikatorlarga xesh-funksiyaning uchta n_1 n_2, p_3 qiymatlari mos keladi. n_1 n_2, p_3 manzillarda joylashgan yacheykalarga A_1, A_2, A_3 identifikatorlar haqidagi ma`lumotlar joylashtiriladi. $A_3$ identifikatorni qidirishda $p_3$ manzilning qiymati hisoblanadi va jadvalning mos yacheykasidan ma`lumotlar olinadi.

Ushbu usul, elementni jadvalga joylashtirish vaqti va elementni qidirishga ketadigan vaqt xesh-funksiyani hisoblashga ketadigan vaqt bilan aniqlanib, elementni ko'p marta qidirishga ketadigan vaqtga nisbatan kam bo'lganligi sababli, foydaliroqdir. Usulning ikkita ko'rinib turgan kamchiligi mavjud. Ulardan biri- xotira hajmidan identifikatorlar jadvalini tashkil etishda foydasiz foydalanish: identifikatorlar jadvalini saqlash uchun kerak bo'lgan to'plam o'lchami, jadvaldagi identifikatorlar soni juda kam bo'lsa ham, xesh-funksiyaning qiymatlar sohasiga mos kelishi shart.

Ikkinchi kamchiligi -- xesh-funksiyaning aniq mos tanlovini amalga oshirish zaruriyati.



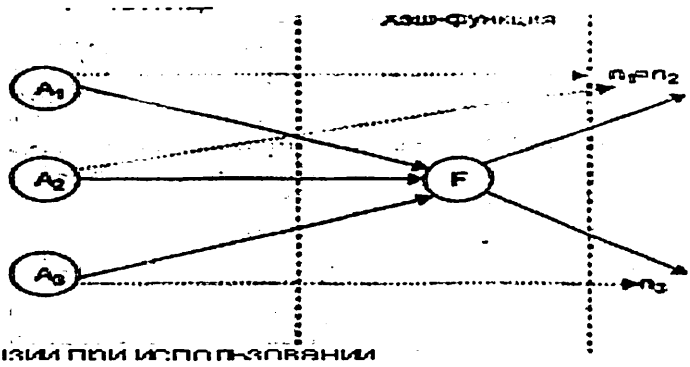
28-rasm. Xesh –manzillash yordamida identifikatorlar jadvalini tashkil etish

7.5.Xesh –funksiya asosida identifikatorlar jadvalini qurish

Xesh-funksiyani turli variantlari mavjud. Xesh-funksiyani natijasini olish ya`ni «xeshlash» - ko'pincha belgilar zanjiri ustida qandaydir sodda arifmetik yoki mantiqiy amallarni bajarish hisobiga erishiladi. Belgi uchun eng sodda xesh-funksiya bo'lib belgi literasining EHM dagi ichki ifodalanish kodi hisoblanadi.

Ushbu xesh-funksiyani zanjirdagi birinchi belgini tanlagan holda belgilar zanjiri uchun foydalanishimiz mumkin. SHunday qilib, agar A belgini ikkilik ASCII tasvirlanishi 00100001 bo'lsa, u holda identifikatorni xeshlash natijasi ATable 00100001 kodi bo'ladi.

YUqorida taklif etilgan xesh-funksiya qonikarli emas: bunday xesh-funksiyadan foydalanishda yangi muammo yuzaga keladi, ya`ni — ikkita turli bir xil harfdan boshlanuvchi identifikatorga bitta xesh-funksiyaning qiymati mos qo'yiladi. U holda, xesh-manzillashda bitta yacheykaga ikkita turli identifikatorni joylashtirish kerak bo'ladi. Bu esa mumkin emas. Bunday holat, ya`ni ikkita yoki undan kup identifikatorga xesh-funksiyaning bitta qiymati mos qo'yiladigan holat ***kolliziya*** deb ataladi. Kolliziyani yuzaga kelgan holati quyidagi 29-rasmda keltirilgan. Ikkita turli identifikatorlar A_1 va A_2 ga xesh-funksiyaning ikkita bir xil qiymati mos keladi, $p_1 = p_2$.



29-rasm. Xesh-manzillashdan foydalanishda kolliziyani yuzaga kelishi

Tabiiyki, kolliziyani yuzaga keltiradigan xesh-funksiya identifikatorlar jadvalida xesh-manzillash uchun foydalanilishi mumkin emas. Bunday xesh-funksiyadan foydalanmaslik uchun identifikatorlar to'plamida bittagina kolliziya holatining uchrashi etarlidir.

YUqoridagi misolda esa juda sodda xesh-funksiya keltirilgan. Kolliziyani butunlay yuzaga keltirmaydigan xesh-funksiyani qurish mumkinmi?

Kolliziyani to'liq yo'qotilishi uchun xesh-funksiya o'zaro bir qiymatli bo'lishi kerak: xesh-funksiyaning aniqlanish sohasidagi har bir elementga uning qiymatlar to'plamidan bitta qiymat mos qo'yilishi kerak, lekin ushbu funksiyaning qiymatlar to'plamidagi har bir qiymatiga uning aniqlanish sohasidan bitta element mos kelishi kerak. U holda, xesh-funksiyaning aniqlanish sohasidagi ixtiyoriy ikkita elementga har doim ikkita turli qiymat mos keladi.

Nazariy tomondan identifikatorlar uchun bunday xesh-funksiyani qurish mumkin, chunki xesh-funksiyaning aniqlanish sohasi ham (identifikatorlarning barcha mumkin bo'lgan ismlari), va uning qiymatlari sohasi (butun manfiy bo'lmagan sonlar) ham cheksiz sanoqli to'plamlar hisoblanadilar. Nazariy jihatdan o'zaro birqiymatli bir sanoqli to'plamni boshqasiga aks ettirish mumkin, lekin buni amalga oshirish juda ham qiyin masala.

Identifikatorlar uchun o'zaro birqiymatli xesh-funksiyani qurish mumkin emasligini belgilovchi chegirmalar mavjud. Gap shundaki, haqiqatda ixtiyoriy xesh-funksiyaning qiymatlar sohasi kompyuterning berilgan arxitekturasining mumkin bo'lgan manzillar sohasining o'lchami bilan chegaralanadi.

Xesh-manzillashni tashkil etishda identifikatorlar jadvalining manzili sifatida foydalanilgan qiymat kompyuterning berilgan manzillar setkasi chegarasidan chiqib ketmasligi shart. Tradision arxitekturali ixtiyoriy kompyuterning manzillar to'plami juda katta, lekin har doim tugallangan, ya`ni chegaralangandir.

CHeksiz to'plamni chekli to'plamga o'zaro birqiymatli aks ettirishni tashkil etish nazariy jihatdan ham mumkin emas. Aniq kompilyatorlarda identifikatorlarning qator

uzunliklari amaliy jihatdan chegaralangandir – ko'pincha ular 32 dan 128 ta belgigacha bo'ladi (ya'ni xesh-funksiyaning aniqlanish sohasi ham cheklidir). Lekin, shu holda ham funksiyaning aniqlanish sohasini tashkil etuvchi chekli to'plamning elementlari soni funksiya qiymatlari sohasining chekli to'plamdagi sonidan ko'proqdir (identifikatorlarning barcha mumkin bo'lgan soni zamonaviy kompyuterlarning mumkin bo'lgan manzillari sonidan ko'proqdir). SHunday qilib, o'zaro bir qiymatli xesh-funksiyani qurish hech bir ko'rinishda mumkin emas. Demak, kolliziyalarni yuzaga kelishini butunlay yo'qotishni iloji yo'q.

Ikkita turli element uchun xeshlash natijalari turlicha ekan, qidirish vaqti xeshlashga ketgan vaqt bilan mos tushadi. Lekin ikkita turli elementni xeshlash natijalari mos tushgan holda qiyinchilik yuzaga keladi.

Kolliziya muammosini echish uchun ko'pgina usullardan foydalanish mumkin. Ulardan biri «rexeshlash» (yoki «qaytataxlashdir») dir. Ushbu usulga ko'ra, agar A element uchun xesh-funksiya tomonidan hisoblangan manzil h(A) bo'lsa, va avvaldan band bo'lgan yacheykaga ko'rsatsa, u holda ni=h_1(A) funksiyani qiymatini hisoblash zarur va yacheykani bandligini ni manzil bo'yicha tekshirish kerak. Agar u ham band bo'lsa, u holda h_2(A) qiymat hisoblanadi, shu holda bo'sh yacheyka topilguniga qadar, yoki h_i(A) ning navbatdagi qiymati h(A) bilan mos tushguniga qadar amalga oshiriladi. Oxirgi holatda, identifikatorlar jadvali to'lgan va unda boshqa joy yo'q deb — jadvalda identifikatorni joylashtirish bo'yicha xatolik ma'lumoti chiqariladi.

Usulning xususiyati shundaki, identifikatorlar jadvali uning barcha yacheykalari bo'sh ekanligi haqidagi ma'lumot bilan to'ldirilgan bo'lishi kerak (ma'lumotlar yo'q). Masalan, agar identifikatorlarning ismlarini saqlash uchun ko'rsatkichlardan foydalanilayotgan bo'lsa, jadvalni avvaldan bo'sh ko'rsatkichlar bilan to'ldirish kerak.

Identifikatorlarni bunday jadvalini elementni joylashtirishning quyidagi algoritmi bo'yicha tashkil etish mumkin..

1 qadam. YAngi A element uchun n = h(A) xesh-funksiyani qiymatini hisoblash.

2 qadam. Agar p manzil bo'yicha yacheyka bo'sh bo'lsa, u holda A elementni unga joylashtirish va algoritmni tugatish, aks holda i:=l va 3 qadamga o'tish.

3 qadam. n_i = hj(A) ni hisoblash. Agar hi manzil bo'yicha yacheyka bo'sh bo'lsa, u holda unga A elementni joylashtirish va algoritmni tugatish, aks holda 4 qadamga o'tish.

4 qadam. Agar p = ni, u holda xatolik haqida ma'lumot berilsin va algoritmni tugatish, aks holda i:-i+l va 3 qadamga qaytish.

U holda A elementni identifikatorlar jadvalidan qidirish quyidagi algoritm bo'yicha amalga oshiriladi.

1 qadam. YAngi A element uchun n = h(A) xesh-funksiyani qiymatini hisoblash.

2 qadam. Agar p manzil bo'yicha yacheyka bo'sh bo'lsa, u holda element topilmadi va algoritm tugadi, aks holda p yacheykadagi element ismini A elementni ismi bilan

solishtirish, Agar ular mos tushsalar, u holda element topildi va algoritm tugadi, aks holda i:=l va 3 qadamga o'tish.

3 qadam. n_i = hj(A) ni hisoblash. Agar p manzil bo'yicha yacheyka bo'sh bo'lsa yoki p = ni, u holda element topilmadi va algoritmni tugatish, aks holda p yacheykadagi element ismini A element ismi bilan solishtirish, Agar ular mos kelsalar, u holda element topildi va algoritm tugadi, aks holda i:=i+l va 3 qadamga o'tish.

Elementni joylashtirish va qidirish algoritmlari bajarilish nuqtai nazaridan o'xshashdirlar. SHuning uchun ular ularni bajarish uchun ketadigan vaqtni baholash bo'yicha ham mos keladilar. Identifikatorlar jadvalini bunday tashkil etishda kolliziya yuz bergan holatlarda algoritm jadvalning bo'sh yacheykalariga elementlarni qandaydir holda tanlagan holda joylashtiradi. Bunday holatlarda elementlar keyinchalik xesh-funksiyaning qiymatlari bilan mos keluvchi manzilli yacheykalarga tushadilar, bu esa qo'shimcha yangi kolliziyalarni yuzaga keltiradi. SHunday qilib, elementni jadvalda joylashtirish va qidirish uchun zarur amallar soni jadvalning to'ldirilishi bilan bog'liq.

Tabiiy ravishda hj(A) funksiyalar elementni joylashtirish va qidirish bosqichlarida hisoblanishlari kerak. Endi masala ushbu hj(A) funksiyalarni hisoblashlarini qanday tashkil etilishi to'g'risida?

hj(A) funksiyalarni hisoblashning eng sodda usuli ularni hi(A) = (h(A)+ pj) mod Nm, bu erda pj —qandaydir hisoblanadigan butun son, N_m – identifikatorlar jadvalining eng maksimal elementlari soni, ko'rinishida tashkil etishdir. O'z navbatida eng sodda usul bu pj = i. U holda hi(A)= (h(A)+i) mod N_m formulani olamiz. Bu holatda xesh-funksiyaning qiymatlarining qandaydir elementlar uchun mos tushganida jadvaldagi bo'sh yacheykani qidirish h(A) xesh-funksiya tomonidan berilgan joriy o'rindan ketma-ket boshlanadi. Ushbu usulni juda qulay usul deb bo'lmaydi, chunki xesh-manzillarni mos tushishida jadval elementlari ular yaqinida guruhlanishi boshlanadi va bu esa joylashtirish va qidirishni amalga oshirishda zaruriy taqqoslashlarni sonini ko'payishiga olib keladi. Bunday jadvalda elementni qidirishning o'rtacha vaqti taqqoslash amallari sonidan bog'liq ravishda quyidagicha baholanishi mumkin :

T_p-O((l-Lf/2)/(l-LQ).

Bu erda Lf — (load factor) identifikatorlar jadvalining to'ldirilish darajasi — jadvalning band yacheykalari sonini undagi maksimal mumkin bo'lgan elementlar soniga munosabati: Lf = N/N_m.

Misol sifatida jadvalning qator ketma-ket p_1, p_2, p_3, p_4, p_5 yacheykalarini va ushbu yacheykalarga joylashtirish kerak bo'lgan qator A_1, A_2, A_3, A_4, A_5 identifikatorlarni qaraymiz. Ushbu identifikatorlar ushbu shartlarni qanoatlantirishlari kerak: $h(A_1) = h(A_2) = h(A_5) = n_1$ $h(A_3) = n_2$; $h(A_4) = n_4$. Eng sodda rexeshlash usulidan foydalanib jadvalda identifikatorlarni joylashtirish ketma-ketligi natijasida jadvalda joylashtirishdan so'ng A_1 identifikatorni qidirish uchun 1 ta taqqoslash, A_2 — uchun 2 ta taqqoslash, A_3 — uchun 2 ta taqqoslash, A_4 — uchun 1 ta taqqoslash va A_5 — uchun 5 ta taqqoslash kerak bo'ladi.

Rexeshlashning ushbu oddiy usuli ham, jadvalni to'liq to'ldirmaslik holida, identifikatorlar jadvalining tashkil etishda etarli darajada foydali bo'ladi. Masalan, 1024 identifikator uchun jadvalni 90 % ga to'ldirilsa, bitta identifikatorni qidirish uchun o'rtacha 5,5 ta taqqoslashlarni amalga oshirish kerak bo'ladi, xuddi shu vaqtda logarifmik qidirish ham o'rtacha 9 tadan 10 tagacha taqqoslashni beradi.

Usulning taqqoslash foydaliligi identifikatorlar sonining oshishi va jadvalning to'ldirilishini kamayishi holida yanada ko'payadi.

Jadvalda bitta elementni joylashtirish va qidirishni o'rtacha vaqtini rexeshlashning mukammalrok usulini qo'llash yo'li bilan kamaytirish mumkin. SHunday usullardan biri r ni h,(A) - (h(A)+pi)modN_m funksiyalar uchun r_i r_2, ..., r_k butun sonlarning psevdotasodifiy ketma-ketligi sifatida foydalanilishi hisoblanadi. Psevdotasodifiy sonlarning generatorini yaxshi tanlangan holida ketma-ketlik uzunligi k $k=N_m$ ga teng. U holda jadvaldagi bitta elementni topishning o'rtacha vaqtini quyidagicha baholash mumkin :

$$E_n = O((1/Lf)*\log_2(1-Lf)).$$

Rexeshlash funksiyalarining yana boshqa tashkil etish usullari ham mavjud bo'lib, ular kvadratik hisoblashlarga asoslangan, yoki masalan, quyidagi formula bo'yicha hisoblangan: hj(A) = (h(A)*i) mod N_m, agar N_m — oddiy son. Umuman olganda rexeshlash jadvalda elementni qidirish bo'yicha yaxshi natijalarga erishish imkoniyatini beradi (binar qidirish va binar daraxt usuliga nisbatan yaxshiroq), lekin usulning foydaliligi identifikatorlar jadvalini to'ldirilganligiga va foydalanilayotgan xesh-funksiyaning sifatiga juda kuchli bog'liq, va kolliziyalar kanchalik kam yuzaga kelsa, usulning foydaliligi shunchalik yuqori bo'ladi. Jadvalning to'liq to'ldirmaslik talabi mavjud xotiraning foydasiz ishlatilishiga olib keladi.

7.6.Zanjirlar usuli bo'yicha identifikatorlar jadvalini qurish

Xesh-funksiyadan foydalangan holda identifikatorlar jadvalini to'liq to'ldirmaslik kompilyatorga tegishli bo'lgan xotira hajmini foydasiz ishlatilishiga olib keladi. Foydalanilmagan xotira hajmi qanchalik ko'p bo'lsa, har bir identifikator uchun shunchalik ko'p ma`lumot saqlanadi. Ushbu kamchilikni identifikatorlar jadvaliga qandaydir oraliq xesh-jadvalni qo'shib to'ldirish yo'li bilan tuzatish mumkin.

Xesh-jadvalni yacheykalarida yoki bo'sh qiymat, yoki asosiy identifikatorlar jadvalidagi qandaydir xotira sohasiga ko'rsatuvchi ko'rsatkich qiymati saqlanishi mumkin.

U holda xesh-funksiya, avval xesh-jadvalga murojaatni amalga oshiruvchi, so'ngra u orqali topilgan manzil bo'yicha identifikatorlar jadvalining o'ziga murojaat etuvchi, manzilni hisoblaydi.

Agar identifikatorlar jadvalining mos yacheykasi bo'sh bo'lsa, u holda xesh-jadval yacheykasi bo'sh qiymatni o'zida jamlaydi. U holda identifikatorlar jadvalining o'zida xesh-funksiyaning har bir mumkin bo'lgan qiymati uchun

yacheykaga ega bo'lish shart emas, ya'ni jadvalni dinamik ravishda ya'ni identifikatorlar jadvalini to'ldirilishga qarab hajmini oshirish mumkin (boshlanishida identifikatorlar jadvali bironta ham yacheykaga ega bo'lmaydi, xesh-jadvalning barcha yacheykalari bo'sh qiymatga ega bo'ladi). Bunday yondashish ikkita yaxshi natijaga erishish imkonini beradi: birinchidan. identifikatorlar jadvalini bo'sh qiymatlar bilan to'ldirish zaruriyati yo'q- buni esa faqatgina xesh-jadval uchun bajariladi; ikkinchidan, har bir identifikatorga identifikatorlar jadvalida qat'iy bittadan yacheyka mos keladi (unda bo'sh foydalanilmagan yacheykalar bo'lmaydi). Bu holatda bo'sh foydalanilmagan yacheykalar faqatgina xesh-jadvalda mavjud bo'ladi va foydalanilmagan xotira hajmi har bir identifikator uchun saqlanayotgan ma'lumotlar hajmidan bog'liq bo'lmaydi – xesh-funksiyaning har bir qiymati uchun asosiy identifikatorlar jadvaliga ko'rsatuvchi bitta ko'rsatkichni saqlash uchun zarur bo'lgan xotira sarflanadi. Ushbu chizma asosida xesh-funksiya yordamida identifikatorlar jadvalini tashkil etishning yana bir «zanjirlar usuli» deb ataluvchi usulini amalga oshirish mumkin. Zanjirlar usuli uchun identifikatorlar jadvaliga har bir element uchun yana bitta, jadvalning ixtiyoriy elementiga ko'rsatkichni saqlovchi, maydon qo'shiladi. Avval boshda bu maydon har doim bo'sh (hech qaerga ko'rsatmaydi). Xuddi shunday yana ushbu usul uchun yana bitta maxsus, har doim asosiy identifikatorlar jadvalining birinchi bo'sh yacheykasiga ko'rsatuvchi, o'zgaruvchiga ega bo'lish zarur (avval boshda — jadvalning boshlanishiga ko'rsatadi).

Zanjirlar usuli quyidagi algoritm asosida ishlaydi.

1-qadam. Xesh-jadvalning barcha yacheykalariga bo'sh qiymatni joylashtirish, identifikatorlar jadvali bitta ham yacheykaga ega bo'lmasligi kerak, *FreePtr* o'zgaruvchi (birinchi bo'sh yacheykaning ko'rsatkichi) identifikatorlar jadvalning boshiga ko'rsatadi; i:=l.

2-qadam. A_i yangi element uchun n_i xesh-funksiyaning qiymatini hisoblash. Agar n_i manzil bo'yicha xesh-jadvalning yacheykasi bo'sh bo'lsa (bo'sh, u holda unga *FreePtr* o'zgaruvchini qiymatini joylashtirish va 5 qadamga o'tish; aks holda — 3 qadamga o'tish.

3-qadam. j:=l ni qo'yish, xesh-jadvaldan m_i identifikatorlar jadvalining yacheykasi manzilini tanlash va 4-qadamga o'tish.

4-qadam. m_i manzil bo'yicha identifikatorlar jadvalining yacheykasi uchun maydon ko'rsatkichi qiymatini tekshirish. Agar u bo'sh bo'lsa, u holda unga *FreePtr* o'zgaruvchidan manzilni yozish va 5-qadamga o'tish; aks holda j:=j+l, ko'rsatkich maydonidan m_i manzilni tanlash va 4-qadamni takrorlash.

5-qadam. Identifikatorlar jadvaliga yangi yacheykani qo'shish, unga A element uchun ma'lumotni yozish, (ko'rsatkich maydoni bo'sh bo'lishi shart), *FreePtr* o'zgaruvchiga qo'shilgan yacheykani oxiridan boshlab manzilni joylashtirish. Agar jadvalga joylashtiriladigan identifikatorlar qolmagan bo'lsa, u holda algoritm ishi tugadi, aks holda i:=i+l va 2 qadamga o'tish.

Identifikatorlar jadvalida shunday tashkil etilgan elementni qidirish quyidagi algoritm bo'yicha bajariladi.

1-qadam. A berilgan element uchun p xesh-funksiyani qiymatini hisoblash. Agar xesh-jadvalning p manzil bo'yicha yacheykasi bo'sh bo'lsa, u holda element topilmadi va algoritm tugadi, aks holda j:=l, xesh-jadvaldan m_i = n. identifikatorlar jadvalining yacheykasi manzilini tanlash.

Usulning kamchiligi xotiraning dinamik taqsimlangan sohalari bilan ishlash zaruriyati hisoblanadi.

Usulning yutug'i, birinchi navbatda foydalanilayotgan xesh-funksiyaning sifatiga bog'liqligi, ikkinchidan- ma`lumotlar uchun qo'shimcha saqlanish joylarini tashkil etish usuliga bog'liqligida ko'rinadi.

Xesh-manzillash –bu usul faqatgina kompilyatorlarda identifikatorlar jadvalini tashkil etish uchungina qo'llanilmaydi. Ushbu usul operasion tizimlarda va ma`lumotlar omborini boshqarish tizimlarida ham qo'llaniladi.

Sinov savollari

1. Identifikatorlar jadvalini tashkil etish nima uchun kerak?
2. Identifikatorlar jadvalida qanday ma`lumotlar saqlanadi?
3. Identifikatorlar jadvalini tashkil etishning usullari haqida ma`lumot bering.
4. Identifikatorlar jadvaliini tashkil etishning asosiy kriteriysi nima?
5. Xesh funksiya nima?
6. Xesh manzillash nima?
7. Kolliziya qanday paydo buladi?
8. Kolliziyani butunlay yuzaga keltirmaydigan xesh-funksiyani qurish mumkinmi?
9. Kolliziya muammosini echish usullari haqida ma`lumot bering.
10. Rexeshlash nima?

8.BO'LIM

Leksik tahlilchilar. Skanerlarni qurish tamoyillari

Leksik tahlilchilar (skanerlar). Skanerlarni qurishning tamoyillari. Leksik tahlilchilarning xizmatchi jadvallari. Leksik tahlilchilarni qurishni avtomatlashtirish

8.1.Leksik tahlilchining belgilanishi

Leksik tahlilchilarni qarab chiqishdan avval, leksema tushunchasiga aniq ta`rif berishimiz kerak.

Leksema (tilning leksik birligi)- bu tilning struktura o'lchovi bo'lib, tilning elementar belgilaridan tashkil topadi va o'z tarkibiga tilning boshqa unga tegishli bo'lmagan belgilarini olmaydi. Tabiiy murojaat tillarining leksemalari bu so'zlardir. Dasturlash tilining leksemalari bo'lib identifikatorlar, konstantalar, tilning kalit so'zlari, amal ishoralari va x.k. hisoblanadilar. Har bir tilning leksemalari to'plami ushbu tilning sintaksisi orqali aniqlanadi.

Leksik tahlilchi (yoki skaner) — bu kompilyator bo'lagi bo'lib, u kiruvchi dastur matnini o'qiydi va matndagi kiruvchi til leksemalarini ajratadi. Leksik tahlilchining kirishiga boshlang'ich dastur matni kelib tushadi, chiqishidagi ma`lumot esa keyingi qayta ishlashlar uchun sintaksis tahlil va ajratish bosqichiga uzatiladi.

Leksema majburiy bo'lak emas, lekin kompilyatorning tarkibiga qo'shiladi. Leksik tahlilchini ixtiyoriy kompilyator tarkibida bo'lishining asosiy sabablari quyidagilar:

1) Tahlil bosqichida matnlar bilan ishlashni soddalashtiradi va qayta ishlangan ma`lumot hajmini qisqartiradi.

2) Leksemalarni matnda ajratish va tahlil uchun sodda va foydali tahlil texnikasi qo'llaniladi. Sintaksis tahlil bosqichida esa tahlilning etarli murakkab tizimlaridan foydalaniladi.

3) Leksik tahlil, konstruksiya bo'yicha kiruvchi dastur matni bilan ishlash jarayonida strukturasi tilning versiyasidan bog'liq bo'lgan, murakkab sintaksis tahlilchini aniqlaydi.

Ko'pgina kompilyatorlarda leksik va sintaksis tahlilchilar –bu o'zarobog'liq bo'laklar. Leksik tahlilchi va sintaksis ajratgichni o'zaro aloqasini tashkil etishni ikkita turli varianti mavjud:

1) ketma-ket;
2) parallel.

1.Ketma-ket. Leksik tahlilchi kiruvchi dastur matnini boshidan oxirigacha qarab chiqadi va leksemalarning strukturalangan jadvalini hosil etadi, bu jadvalda identifikatorning kalit so'zlarini maxsus kelishilgan kodlarga o'zgartiriladi va bundan tashqari identifikatorlar jadvali bilân aloqa o'rnatiladi.

2.Parallel. Kiruvchi dastur matnining leksik tahlili bosqichma –bosqich bajariladiki, sintaksis tahlilchi tilning navbatdagi konstruksiyasini tahlilini bajarishda leksik tahlilchiga keyingi belgini o'qish uchun murojaat qiladi va shu bilan birga har birini kutib turishini ko'rsatadi. Ushbu holatda xatoliklar yuz

berganda «rad etish» bo'lishi mumkin va leksemalarni tahlilini to'g'ri bajarilganida leksema leksemalar jadvaliga yoziladi.

Leksik tahlil o'z ichiga quyidagi masalalarni echishni oladi:

1. dasturning berilish matnini qarab chiqish (skanirlash):
2. izohlar va keraksiz bo'sh belgilarni yo'qotish;
3. leksik o'lchovlarni ajratish (leksik o'lchovlarni chegaralarini topish);
4. leksemalarni ichki tasvirlash ko'rinishlariga aylantirish;
5. leksik o'lchovlarni anglash;
6. leksik tahlilchining chiquvchi jadvallarini tashkil etish.

Ko'pgina dasturlash tillarida leksemalarni chegaralari bo'lib quyidagilar hisoblanadilar:

- bo'sh belgilar;
- ajratkichlar (. , : ;);
- amal ishoralari (+, -, :, *, :=);

Oddiy holda leksema – bu bo'laklovchilar orasida joylashganlardir.

Konstanta va ismlar kabi leksemalarni anglash uchun 3 sinf regulyar grammatikasidan foydalaniladi va anglovchi tugallangan avtomatni tashkil etadi.

Skanirlash natijasida dastur uzun qatordan tashkil topadi.

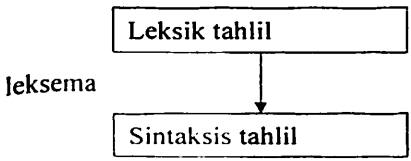
Leksik tahlilchini amalga oshirish uchun ikki tomon mavjud.

Birinchi tomon – leksik va sintaksis tahlilchilarning o'zaro xarakatlarini amalga oshirishning ikki yo'nalishi.

Ikkinchi tomon – leksik tahlilning qurilishi.

Birinchi yo'nalish: va sintaksis tahlilchilarning o'zaro xarakatlarini amalga oshirishning oxirgi chizmasi. Leksik tahlilchi barcha dastur matnini qarab chiqadi va barcha leksemlar uchun kodlar jadvalini tuzadi va natijalarni sintaksis tahlilchiga uzatadi. Bu xolatda leksik va sintaksis tahlilchilar bo'laklangan hisoblanadilar va kompilyator bosqichida bir biridan keyin bajariladilar.

Parallel yo'nalishda leksik tahlilchi navbatdagi leksemani topadi va uni sintaksis tahlilchiga (30-rasm) uzatadi.



30-rasm. Parallel yo'nalishda leksik tahlilchining ish chizmasi

Bu holatda leksik tahlilchi navbatdagi leksemani ajratish uchun sintaksis tahlilchidan chaqiriladigan qism dasturdan tashkil topadi.

Ketma-ket chizmaning ustunligi uning tezroq ishlashidadir.

Ketma-ket chizmaning kamchiligi xotiraning ko'p sarflanishidir.

Parallel chizmaning ustunligi: amalga oshirishning oddiyligi, kam xotira sarflanishidir.

Parallel chizmaning kamchiligi: kompilyasiya vaqti 1,5 – 2 marotaba oshadi.

Ikkinchi tomon.

Birinchi yo'nalish: leksemalar sifatida alfavitning alohida belgilari olinadi: harflar, raqamlar, bo'laklovchilar, amallarning ishoralari va x.k.

Bu holda kalit so'z, masalan, BEGIN beshta leksemaning to'plami sifatida qaraladi: 'B', 'E', 'G', 'I', 'N'.

Bunday yo'nalishda ko'pbelgilik leksemalarni qayta ishlashda (sintaksis tahlil bosqichida) kompilyasiya vaqti sintaksis tahlilning murakkabligi hisobiga ko'payib ketadi.

Ikkinchi yo'nalish: alohida kalit so'zlar, amallarning ishoralari, ismlar, konstantalar alohida leksema ko'rinishida qaraladi va leksik tahlil masalasi ularni anglashdan iborat bo'ladi. Bunday yo'nalishdan ko'p foydalaniladi.

8.2.Leksik tahlilning xizmatchi jadvallari

Leksik tahlil uchun boshlang'ich ma`lumotlar bo'lib belgilar qatori ko'rinishida ifodalangan dastur hisoblanadi. Leksik tahlilni amalga oshirish uchun dastur matni qarab chiqiladi. So'ngra dastur elementlari bo'yicha oralik jadvallar quriladi. Jadvallar quyidagi guruhlarga bo'linadi:

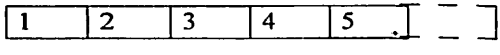
1.kiruvchi;

2.oraliq;

3.chiquvchi;

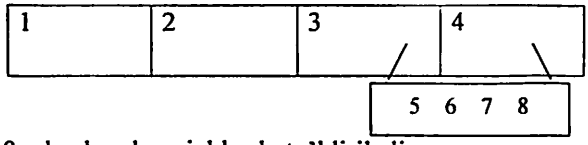
1.Kiruvchi – terminal belgilar jadvali. U yozuvlardan tashkil topadi:
Amal qilish nuqtai nazaridan - bu yozuvlar to'plami yoki ro'yxat.

1.Maydonlar tarkibi bo'yicha:
2.Tartib raqami;
3.Belgining o'zi;
4.Terminal belgining turi (kalit so'z, ajratuvchi, amal ishorasi);
5.Arifmetik va mantiqiy amallar uchun ustunlik belgisi. (leksik tahlilchidan keyin boshqa bosqichlarda to'ldiriladi);
2.Konstantalar jadvali quyidagi yozuvlar to'plamini ifodalaydi:



1 –tartib raqami;
2 –konstantalar qiymati;

3 –asos (sonli konstantalar uchun);
4 –aniqlik (xotira o'lchami);
5 –konstanta turi;
6 –qo'shimcha ma`lumot, masalan, literal uchun – xotiradagi joylashuvning boshlang'ich manzili.



3,4,5,6,7,8 – boshqa bosqichlarda to'ldiriladi;

3.Identifikatorlar jadvaliga barcha kalit so'z hisoblanmagan ismlar kiritiladi. U yuqoridagi ko'rinishga ega:

1 –tartib raqami;
2 –ism uzunligi baytlarda;
3 –xotiradigi ism belgilarini saqlash manzili;
4 –identifikator atributlari;
Atributlar sifatida:

5–ko'rinish (yoki oddiy o'zgaruvchi ismi, yoki to'plam, yoki prosedura (funksiya), belgi);

6 –tur (xaqiqiy, butun, qatorli, belgili, mantiqiy);
7 –xotira sinfi (statik, avtomatik, dinamik, tashqi);
8 –o'lchovi (masalan, qator uchun – uzunlik, to'plam uchun – indekslar soni, intervyulashgan turlar uchun – chap va o'ng chegaralar qiymati);

Ma`lumotlarni ismlarda tasvirlashning shunday varianti mavjudki, unda barcha ismni tashkil etuvchi belgilar zanjirni tashkil etadilar va har bir yangi ism ushbu zanjirning oxiriga ulanadilar.

CHiquvchi jadval – leksemalar kodlari jadvali (sintaksis tahlil uchun boshlang'ich).

Jadval strukturasi:

7-jadval

Leksema raqami (dasturda)	Turi	Leksema raqami (jadvalda)
1		
2		

Birinchi maydonda – dasturdagi leksemani paydo bo'lish raqami.Uchinchi maydonda mos jadvaldagi raqam. Agar ism bir necha marta qaytarilsa, u holda leksemalar kodlari jadvalida unga shuncha marta qator mos keladi. Ikkinchi maydonga ism haqidagi ma`lumot kiritiladi: ism – I, terminal belgilar – T, konstantalar – C.

Jadvalni tashkil etishga misol qaraymiz.

Program authm;

```
Var
  I, J, Sum: Integer;
Begin
  Sum:=0;
  For i:=1 to 100 do;
  Begin
  Read(J);
  Sum:=Sum+J;
  End
    Sum:=Sum div 100;
    Write(Sum);
    End
```

Terminal belgilar jadvali(8-jadval)

8-jadval

№	Belgi	Turi		
		Ajratuvchi	Amal ishorasi	Kalit so'z
1	;	1	0	0
2	,	1	0	0
3	:	1	0	0
4	Bo'sh belgi	1	0	0
5	:=	0	1	0
6	(	1	0	0
7	)	1	0	0
8	+	0	1	0
9	Div	0	1	0
10	Program	0	0	1
11	Var	0	0	1
12	Integer	0	0	1
13	Begin	0	0	1
14	For	0	0	1
15	To	0	0	1
16	Do	0	0	1
17	End	0	0	1
18	Read	0	0	1
19	Write	0	0	1
20	.	1	0	0
21	-	0	1	0
22	*	0	1	0

Ismlar jadvali (9-jadval)

№	Ism
1	Arithm
2	I
3	J
4	Sum

Konstantalar jadvali (10-jadval)

№	Konstanta	Asos0	Turi	Aniqlik

1	0	10	Butun	2
2	1	10	Butun	2
3	100	10	Butun	2

Leksemalar kodlari jadvali (11-jadval)

№	Turi	№ jadvaldagi raqami	Leksemalar
1	T	10	Program
2	T	4	Bo'sh belgi
3	I	1	Arithm
4	T	1	;
5	T	11	Var
6	T	4	Bo'sh belgi
7	I	2	I
8	T	2	,
9	I	3	J
10	T	1	;
11	...	...	...

8.3.Leksik tahlilchining qurilishi

Endi skanerni qurishni amalga oshirishni qarab chiqamiz. Kompilyator o'z tarkibida bir emas, balki bir nechta har biri aniq bir toifa leksemalarini tanlash va tekshirishga mo'ljallangan leksik tahlilchiga ega bo'lishi mumkin.

SHunday qilib, kompilyatorda oddiy umumlashgan leksik tahlilchini ishlash algoritmini quyidagicha ifodalash mumkin:

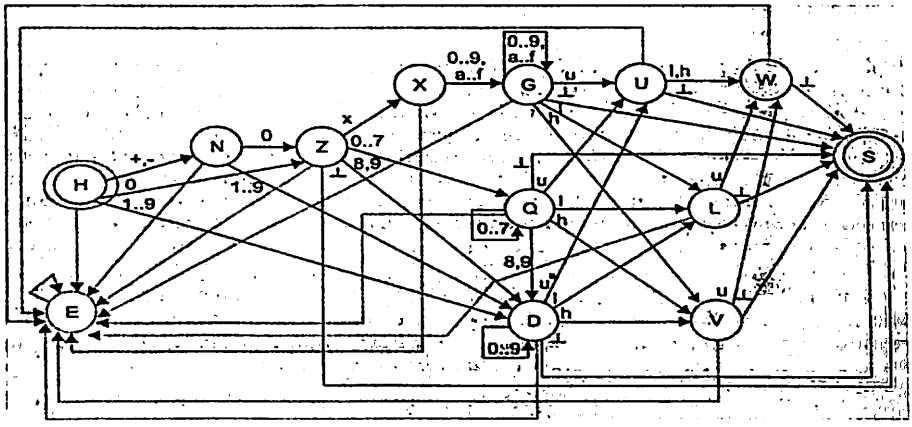
- Kiruvchi oqimdan navbatdagi belgi tanlanadi, va shu belgiga qarab u yoki bu skaner ishga tushadi (belgi e`tiborga olinmasligi yoki xato hisoblanishi mumkin);
- Ishga tushirilgan skaner boshlang'ich dastur matnidagi belgilar oqimini, talab qilingan leksemaga kiradigan belgini navbatdagi belgini topgunicha ajratib, qarab chiqadi, bu belgi esa leksemani chegaralashi yoki xato belgini topishi mumkin;
- Ma`lumotlarni muvaffaqiyatli anglash natijasida ajratilgan leksema haqidagi ma`lumotlar leksemalar jadvali va identifikatorlar jadvaliga joylashadi, algoritm boshlang'ich bosqichga qaytadi va kiruvchi oqimni belgilarini skaner to'xtagan joydan qarashni davom ettiradi;

□ Ma`lumotlarni muvaffaqiyatsiz anglash natijasida xatolik haqida ma`lumot chiqariladi, keyingi xarakatlar amalga oshirilgan skanerdan bog'liq holda — yoki uni bajarilishi to'xtatiladi, yoki navbatdagi leksemani anglashga xarakat qilinadi (algoritmni birinchi bosqichiga qaytish amalga oshadi).

S tili formatidagi butunsonli konstantalarni ifodalovchi leksemalarni tahlil etish misolini qarab chiqamiz. Til talablariga mos ravishda bunday konstantalar o'nlik, sakkizlik yoki o'n oltilik toifalarda bo'lishi mumkin. Sakkizlik konstanta deb 0 raqamidan boshlanuvchi va 0 dan 7 gacha raqamlarni o'z ichiga oluvchi son

hisoblanadi; o'n oltilik konstanta belgilar ketma-ketligidan boshlanib, raqamlar va a dan f gacha harflardan tashkil topishi mumkin. O'nlik konstantalar haqida gapirib o'tirish shart emas. Konstanta yana quyidagi belgilar +, yoki – dan boshlanib oxirida konstanta qiymatini belgilovchi raqamdan iborat bo'lishi mumkin, S tilida bitta harf yoki ikkita uni toifasini bildiruvchi harf kelishi mumkin (u, U — unsigned; h, H — short; l, L — long).

Skanerni qurishda konstantalar S tilidagi dasturni umumiy matniga kiradi deb hisoblaymiz. YUqoridagi qoidalar S tili butunsonli konstantalar grammatikasida Bekus—Naur formasida yozilishi mumkin. Bu grammatika chapchiziqli avtomat regulyar grammatikasi hisoblanadi. U bo'yicha kiruvchi til zanjirlarini anglovchi chekli avtomatni qurish mumkin. Avtomatning graf o'tishlari 31-rasmda keltirilgan. Ko'rinib turibdiki, qurilgan avtomat determinirlangan chekli avtomat hisoblanadi, shuning uchun keyingi aylantirishlar shart emas. Avtomatning boshlang'ich holati bo'lib N holat hisoblanadi.



31-rasm. S tilining butunsonli konstantalarini anglash uchun chekli avtomat grafi

Avtomatni to'liq aniqlangan ko'rinishga keltiramiz — uni yangi E holat bilan to'ldiramiz, va unga barcha noaniq o'tishlarning qobiqlarini birlashtiramiz. Avtomat grafida bu holatga boruvchi qobiqlar, belgilar bilan yuklanmagan – ular ixtiyoriy belgi bo'yicha, grafning o'sha cho'qqisidan chiquvchi boshqa qobiqlar belgilangan belgilardan tashqari, o'tish funksiyasini bildiradilar, (bunday belgilash avtomat grafini belgilashlar bilan to'ldirib tashlamaslik uchun qabul qilingan). 29-rasmda bu holat va u bilan bog'liq o'tish qobiqlari punktir chiziqlar orqali ifodalangan.

CHekli avtomatni loyixalashtirishda S tilining butun konstantasi kiruvchi tilning qanday aniq belgilari bilan tugatilishi kerakligini e'tiborga olish kerak.

S tilida konstantaning chegarasi bo'lib quyidagilar kelishi mumkin:

- Bo'sh belgilar, tabulyasiya belgilari, qatorni o'tkazish belgisi;
- ajratkichlar (,), [,]. {. } , , , : , ; ;
- amal ishoralari +, -, *, /, &, |, ?, ~, <, >, ^, =, %.

Endi ko'rsatilgan avtomatni ishini modellashtiruvchi dasturni yozish mumkin. Berilgan anglovchini amalga oshiruvchi Pascal tilidagi funksiyaning matni keltiriladi. Funksiyaning natijasi bo'lib, tahlilchining kiruvchi ma`lumotlar oqimidagi o'quvchi boshining joriy holati hisoblanadi, agar leksemaning tahlili muvaffaqiyatli amalga oshirilgan bo'lsa, yoki 0 muvaffaqiyatsiz holda (agar leksema xatoga ega bo'lsa).

Dasturda iState o'zgaruvchi avtomatning joriy holatini aks ettiradi, i o'zgaruvchi kiruvchi qatorning belgilarini sanashni amalga oshiradi. Dastur matnida funksiyaning kerakli joylarida yozilishi mumkin bo'lgan izohlar yozib qo'yilgan, masalan topilgan leksemani leksemalar jadvaliga va belgilar jadvaliga yozish, yoki xatoliklar haqidagi ma`lumotlarni yozish.

8.4.Leksik tahlilchini loyixalashtirishga doir misol

Kompilyasiya jarayonining birinchi fazasi bu –leksik tahlil bosqichi bo'lib, unda identifikatorlarni, konstantalarni va til so'zlarini yagona belgilarga (leksemalarga) ifodalovchi belgilar qatorlarini guruhlash bajariladi. Bu jarayon kompilyasiyaning boshqa fazalari bilan parralel borishi (masalan, bir o'tishli kompilyatorlarda) mumkin. Ammo, har qanday holda ham kompilyatorni konstruksiyasini ifodalash va qurishda leksik tahlilni mustaqil faza sifatida tassavur qilish mumkin.

Skanerlash bloki har bir leksemani u yoki bu sinfga tegishli ekanligini aniqlagan holda berishi kerak. Sinflarni tanlovi translyasiya qilinayotgan tilning xususiyatlaridan bog'liq bo'ladi. Ko'pincha o'zgaruvchilar ismi, konstantalar, kalit so'zlar, arifmetik amallar va mantiqiy amallar, maxsus belgilarni ajratib kursatish mumkin.

Anglanayotgan qatorlarni xarakteri leksik tahlil jarayonini anchagina soddalashtirishi mumkin. Masalan:

Identifikatorlar:

a1 one

Sonlar:

100 10.54

Tilning kalit so'zlari:

begin if end

Barcha ushbu qatorlarni regulyar ifodalar yordamida generasiyalash mumkin. Masalan, haqiqiy sonlarni quyidagi regulyar ifodalar yordamida generasiyalash mumkin: (+| -) d d*.d , bu erda d ixtiyoriy raqamni anglatadi. Ifodadan ko'rinib turibdiki, haqiqiy sonlar quyidagi tartibda joylashgan komponentlardan tuziladi:

1. bo'lishi mumkin bo'lgan amal ishorasi;
2. bitta yoki bir nechta raqamlar ketma-ketligi;
3. o'nlik nuqta;
4. nol yoki bir nechta raqamlar ketma-ketligi.

Regulyar ifodalar 3 tur grammatikalariga ekvivalentdir. Masalan, haqiqiy son uchun regulyar ifodaga mos keluvchi 3 tur grammatikasi, quyidagi tug'iluvchi qoidalardan iborat:

R +S| -S| dQ
S dQ
Q dQ|. F|.
F d|dF

Bu erda R –boshlang'ich belgi, d-raqam.

Ba`zi leksemalarni bitta belgini o'qish orqali (*), ba`zilarini ikkita belgini o'qish orqali (:=), boshqalarini esa avvaldan noma`lum son belgini o'qish orqali anglash mumkin.

Faraz qilaylik amalga oshirilayotgan til faqatgina o'zlashtirish operatoridan iborat bo'lsin. Tilning BNFsi ko'rinishi quyidagi ko'rinishda bo'ladi:

<O'zlashtirish > :: = <Identifikator>=<Ifoda>

Qoidadan ko'rinib turibdiki, o'zlashtirish operatorining chap qismida identifikator joylashgan, so'ngra o'zlashtirish belgisi(=),o'ng qismida ifoda joylashgan.

<Ifoda > :: = <Operand>| <Operand> <Binar amal> <Ifoda>

<Binar amal > :: = " –" | " +" | " * " | " /"

<Operand > :: = <Identifikator>| <Const>

<Identifikator > :: = <Harf>
< Const > :: = <Raqam> <Const>| < Raqam >

Leksik tahlilchi kiruvchi dastur matnini quyidagi tartibdagi belgilar ketma-ketligiga aylantiradi.

1- identifikator;
2-konstanta;
3-o'zlashtirish belgisi;
4-ko'paytirish va bo'lish amali belgisi;
5-qo'shish va ayirish amali belgisi.

Leksik tahlilchini yaratish dasturining C++ dasturlash tilidagi bo'lagini keltiramiz.

```
/*prog -  translyasiya qilinayotgan dastur fayli, lex- leksemalarning
chiquvchi fayli */
while(!feof(prog))
{
ch= readsym() ;
/* navbatdagi ch belgini bo'sh belgilarni tashlagan holda o'qish */
 if(isAlpha(ch))
    fprintf(lex . "%c %d", ch, 1);
 else
if(isDigit(ch))
   digit();
/* sonni o'qish va uni faylga chiqarish prosedurasi */
else
if(ch== '=')
fprintf(lex . "%c %d", ch , 3);
else
if(ch== '*'|| ch== '/')
fprintf(lex . "%c %d", ch, 4);
else
if(ch== '+'|| ch== '-')
fprintf(lex . "%c %d", ch , 5);
else
printf("Tilning belgisi emas-  %c \n", ch);
}
```

8.5.Leksik tahlilchilarni avtomatlashtirish (LEX dasturi)

Leksik anglovchilar (skanerlar) —bu kompilyatorning faqatgina muhim bo'lagigina emas. Leksik tahlil kompьyuterda matnli ma`lumotlarni qayta ishlash bilan bog'liq bo'lgan ko'pgina boshqa sohalarda ham qo'llaniladi.

Avvalombor, leksik tahlilni barcha, foydalanuvchi tomonidan komanda qatori va parametrlarni kiritilishini talab etadigan, mumkin bo'lgan komanda jarayonchilari va utilitalar talab qiladilar. Bundan tashqari, kiruvchi matnni eng oddiy leksik tahlilini barcha zamonaviy matn tahrirchilari va matn jarayonchilari qo'llaydilar. Ixtiyoriy dasturiy ta`minotni ishlab chiqaruvchisi, ertami-kechmi, albatta, leksik tahlilni amalga oshirish masalasini echish zaruriyatiga duch keladi. Bir tomondan leksik tahlil masalalari keng tarqalgan. boshqa tomondan esa chekli avtomatlar ishining loyixalashtirish texnikasi asosida aniq formallashtirishni talab etadi.

Bularning barchasi birgalikda ushbu masalaning avtomatlashtirish, va uni echimini olishga mo'ljallangan dasturlarni yaratish, imkoniyatini aniqladi. CHekli avtomatning matematik apparati ko'p vaqtlardan beri ma`lum va leksik tahlil masalalari bilan dasturchilar ancha vaqtdan beri tanish bo'lganliklari sababli, ularni echishga mo'ljallangan dasturlar ham juda ko'p sonlidirlar. Leksik tahlilchilarni qurish masalasini echish uchun turli dasturlar mavjud.

Ulardan eng mashhuri LEX dasturi hisoblanadi. LEX — dasturi skanerlarni (leksik tahlilchilarni) qurishga mo'ljallangan. Kiruvchi til regulyar ifodalar terminidagi leksemalarni ifodalaridan tashkil topadi. LEX dasturning natijasi bo'lib, kiruvchi faylni o'quvchi (ko'pincha bu standart kirish) va undan belgilar ketma-ketligini ajratuvchi, mos regulyar ifodalarda berilgan, qandaydir tildagi dastur hisoblanadi.

LEX dasturining tarixi UNIX operasion tizimlarining tarixi bilan chambarchas bog'liq. Bu dastur OS UNIX utilitalari tarkibida paydo bo'lib, hozirgi kunda ushbu toifadagi, har bir OTning qo'yilishida mavjud. Lekin hozirda ixtiyoriy OT uchun LEX dasturining variantlari mavjud. LEX dasturi OS UNIX muhitida paydo bo'lganligi sababli, birinchi galda u tomonidan qurilgan leksik tahlilchilar S tilida yozilgan. Lekin vaqt o'tishi bilan LEX dasturining, boshqa dasturlash tillari asosidagi skanerlarni keltirib chiqaruvchi variantlari paydo bo'ldilar (masalan, Pascal tili uchun varianti mashhur).

LEX dasturining ish tamoyili juda sodda: uning kirishiga kerakli leksemalar regulyar ifodalar terminlarida beriladi, chiqishida esa, berilgan dasturlash tilidagi skaner dasturining boshlang'ich dastur matnini fayli olinadi (ko'pincha bu - S). Skanerning boshlang'ich dastur matni, ushbu til va dasturlash tizimi qo'llaydigan, ixtiyoriy kutubxonadan olingan funksiyalarni chaqiriqlari bilan to'ldirilishi mumkin. SHunday qilib, LEX dasturi leksik tahlilchilarni ishlab chiqarishni, kerakli leksemalarni regulyar ifodalar terminlarida ifodalash orqali, soddalashtirish imkonini yaratadi.

```
/*prog -  translyasiya qilinayotgan dastur fayli, lex- leksemalarning
chiquvchi fayli */
    while(!feof(prog))
    {
    ch= readsym() ;
    /* navbatdagi ch belgini bo'sh belgilarni tashlagan holda o'qish */
     if(isAlpha(ch))
        fprintf(lex . "%c %d", ch, 1);
     else
    if(isDigit(ch))
       digit();
    /* sonni o'qish va uni faylga chiqarish prosedurasi */
    else
    if(ch== '=')
    fprintf(lex . "%c %d", ch , 3);
    else
    if(ch== '*'|| ch== '/')
    fprintf(lex . "%c %d", ch, 4);
    else
    if(ch== '+'|| ch== '-')
    fprintf(lex . "%c %d", ch , 5);
    else
    printf("Tilning belgisi emas-  %c \n", ch);
    }
```

8.5.Leksik tahlilchilarni avtomatlashtirish (LEX dasturi)

Leksik anglovchilar (skanerlar) —bu kompilyatorning faqatgina muhim bo'lagigina emas. Leksik tahlil kompьyuterda matnli ma`lumotlarni qayta ishlash bilan bog'liq bo'lgan ko'pgina boshqa sohalarda ham qo'llaniladi.

Avvalombor, leksik tahlilni barcha, foydalanuvchi tomonidan komanda qatori va parametrlarni kiritilishini talab etadigan, mumkin bo'lgan komanda jarayonchilari va utilitalar talab qiladilar. Bundan tashqari, kiruvchi matnni eng oddiy leksik tahlilini barcha zamonaviy matn tahrirchilari va matn jarayonchilari qo'llaydilar. Ixtiyoriy dasturiy ta`minotni ishlab chiqaruvchisi, ertami-kechmi, albatta, leksik tahlilni amalga oshirish masalasini echish zaruriyatiga duch keladi. Bir tomondan leksik tahlil masalalari keng tarqalgan, boshqa tomondan esa chekli avtomatlar ishining loyixalashtirish texnikasi asosida aniq formallashtirishni talab etadi.

Bularning barchasi birgalikda ushbu masalaning avtomatlashtirish, va uni echimini olishga mo'ljallangan dasturlarni yaratish, imkoniyatini aniqladi. CHekli avtomatning matematik apparati ko'p vaqtlardan beri ma`lum va leksik tahlil masalalari bilan dasturchilar ancha vaqtdan beri tanish bo'lganliklari sababli, ularni echishga mo'ljallangan dasturlar ham juda ko'p sonlidirlar. Leksik tahlilchilarni qurish masalasini echish uchun turli dasturlar mavjud.

Ulardan eng mashhuri LEX dasturi hisoblanadi. LEX — dasturi skanerlarni (leksik tahlilchilarni) qurishga mo'ljallangan. Kiruvchi til regulyar ifodalar terminidagi leksemalarni ifodalaridan tashkil topadi. LEX dasturning natijasi bo'lib, kiruvchi faylni o'quvchi (ko'pincha bu standart kirish) va undan belgilar ketma-ketligini ajratuvchi, mos regulyar ifodalarda berilgan, qandaydir tildagi dastur hisoblanadi.

LEX dasturining tarixi UNIX operasion tizimlarining tarixi bilan chambarchas bog'liq. Bu dastur OS UNIX utilitalari tarkibida paydo bo'lib, hozirgi kunda ushbu toifadagi, har bir OTning qo'yilishida mavjud. Lekin hozirda ixtiyoriy OT uchun LEX dasturining variantlari mavjud. LEX dasturi OS UNIX muhitida paydo bo'lganligi sababli, birinchi galda u tomonidan qurilgan leksik tahlilchilar S tilida yozilgan. Lekin vaqt o'tishi bilan LEX dasturining, boshqa dasturlash tillari asosidagi skanerlarni keltirib chiqaruvchi variantlari paydo bo'ldilar (masalan, Pascal tili uchun varianti mashhur).

LEX dasturining ish tamoyili juda sodda: uning kirishiga kerakli leksemalar regulyar ifodalar terminlarida beriladi, chiqishida esa, berilgan dasturlash tilidagi skaner dasturining boshlang'ich dastur matnini fayli olinadi (ko'pincha bu - S). Skanerning boshlang'ich dastur matni, ushbu til va dasturlash tizimi qo'llaydigan, ixtiyoriy kutubxonadan olingan funksiyalarni chaqiriqlari bilan to'ldirilishi mumkin. SHunday qilib, LEX dasturi leksik tahlilchilarni ishlab chiqarishni, kerakli leksemalarni regulyar ifodalar terminlarida ifodalash orqali, soddalashtirish imkonini yaratadi.

Sinov savollari

1. Leksema nima?
2. Leksik tahlilning kompilyasiya jarayonidagi o'rnini qanday baholaysiz?
3. Leksik tahlilning xizmatchi jadvallari nima uchun kerak?
4. Identifikatorlar jadvali nima?
5. Ismlar jadvali nima?
6. Terminallar jadvali nima?
7. Konstantalar jadvali nima?
8. Leksik tahlilchining bajaradigan vazifalari haqida ma`lumot bering.
9. Leksik tahlilchining kirishida qanday ma`lumotlar beriladi?
10. Leksik tahlilchining chiqishida qanday ma`lumotlarni olamiz?

9.BO'LIM

Sintaksis tahlilchini belgilanishi. Sintaksis tahlilchilar

Sintaksis tahlilchilarning asosiy ishlash tamoyillari. Tahlil daraxti. Tahlil daraxtini amallar daraxtiga aylantirish. Sintaksis tahlilchilarni qurishni avtomatlashtirish

9.1.Sintaksis tahlilchini belgilanishi

Asosiy funksiyalar: 1) asosiy sintaksis konstruksiyalarni topish va ajratish 2) har bir konstruksiyani turini o'rnatish va tekshirish. 3)matnni generasiya qilinishi uchun qulay sintaksis konstruksiya ko'rinishida tasvirlash.

Sintaksis tahlilchilar. Sintaksis boshqariluvchi tarjima

Sintaksis tahlilchilarning asosiy ishlash tamoyillari

Sintaksis tahlilchi (sintaksis tahlil) – bu kompilyator bo'lagi bo'lib, kiruvchi tilning sintaksis konstruksiyalarini aniqlash uchun javobgardir. Sintaksis tahlilning vazifasiga kiruvchi dastur matnidagi asosiy sintaksis konstruksiyalarni topish va ajratish, ularning toifasini o'rnatish, har bir sintaksis konstruksiyani to'g'riligini tekshirish va nihoyat, sintaksis konstruksiyalarni natijaviy dastur matnini keyingi generasiyalash uchun qulay ko'rinishda ifodalashdan iboratdir. Sintaksis tahlilchining asosida kiruvchi tilning grammatikasi orqali berilgan kiruvchi dastur matnining anglovchisi yotadi. Qoida bo'yicha, dasturlash tillarining sintaksis konstruksiyalari KO-grammatikalar yordamida ifodalanishi mumkin («Kontekst-ozod tillar» bo'limiga qarang), regulyar grammatikalar yordamida ifodalangan tillar kamrok uchraydi («Regulyar tillar» bo'limiga qarang). Ko'p hollarda, regulyar grammatikalar assembler tillariga qo'llaniladi, yuqori daraja tillari esa KO-tillar sintaksisi asosida quriladi. Anglovchi kiruvchi til zanjirlari ushbu berilgan tilga tegishli yoki yo'q degan savolga javob beradi. Ammo, leksik tahlil holatidagi kabi, sintaksis tahlil masalasi ham faqat kiruvchi zanjirning tilga tegishli yoki yo'qligni tekshirish bilangina chegaralanmaydi. Sintaksis tahlilchi bajarishi kerak bo'lgan yuqoridagi sanab o'tilgan barcha masalalarni bajarishi zarur. Bu variantdi tahlilchi MX avtomatning turli ko'rinishi sifatida hisoblanmaydi, uning funksiyalarini kengroq ifodalash mumkin. Sintaksis tahlilchi, barcha topilgan va tahlil qilingan sintaksis konstruksiyalar haqidagi ma`lumotlarni, kompilyasiyaning keyingi fazalariga uzatish uchun kerak bo'ladigan, chiquvchi tilga ega bo'lishi kerak. Bu holatda, u endi magazin xotirali MX-aylantiruvchi hisoblanadi.

Sintaksis tahlil — bu kompilyatorning tahlil bosqichidagi asosiy qismidir. Sintaksis tahlilsiz kompilyatorning ishi ma`noga ega emas, bu vaqtda leksik tahlil esa juda ham shart bo'lmagan faza hisoblanadi. Kiruvchi til sintaksisini tekshirish bo'yicha barcha masalalar sintaksisi tahlil bosqichida hal etilishi mumkin. Skaner, faqat strukturasi murakkab bo'lgan sintaksis tahlilchini, kiruvchi dastur leksemalarini topish va saqlash kabi oddiy masaladan qutqarish imkonini yaratadi.

Leksik tahlilchini chiqishi bo'lib leksemalar jadvali (yoki leksemalar zanjiri) hisoblanadi. Bu jadval sintaksis tahlilchi uchun kirishni tashkil etadi, va har bir

leksemani bitta komponentasini, ya`ni toifasini tekshiradi. Leksema haqidagi qolgan ma`lumotlardan kompilyasiyaning keyingi, semantik tahlil, generasiyaga tayyorgarlik va natijaviy dastur kodini generasiyalash fazalarida foydalaniladi. Sintaksis tahlil (yoki ajratish) — bu leksemalar jadvali tekshiriladigan va u til sintaksisining aniqlanishida ifodalangan struktura shartlari qanoatlantirilganmi yoki yo`q aniqlanadigan jarayondir.

Sintaksis tahlilchi leksik tahlilchini, kiruvchi til grammatikasiga mos ravishda, chiqishini qabul qiladi. Lekin kiruvchi dasturlash tili dasturida ko'pincha qanday konstruksiyalarni leksemalar deb hisoblash kerakligini aniqlashtirilmaydi. Leksik tahlil jarayonida anglanadigan konstruksiyalarga misol sifatida kalit so'zlar, konstantalar va identifikatorlarni keltirish mumkin. Ammo ana shu konstruksiyalar sintaksis tahlilchi tomonidan ham anglanishi mumkin. Amaliyotda qanday konstruksiyalarni leksik tahlil darajasida anglash, yoki qanday konstruksiyalarni sintaksis tahlil uchun qoldirish haqida qattiq qoidalar mavjud emas. Ko'pincha buni kompilyatorni ishlab chiqaruvchisi dasturlashning texnologik aspektlari, kiruvchi til sintaksis va semantikasidan kelib chiqqan holda aniqlaydi. Leksik va sintaksis tahlilchilarning o'zaroxarakatlari tamoyillari «Leksik tahlilchilar» (skanerlar) bo'limida ko'rib o'tilgan.

Quyida sintaksis tahlilchilarning natijalarini kodni generasiyalash bosqichida foydalanish uchun amalga oshirish bilan bog'liq texnik aspektlari qaraladi. Ixtiyoriy sintaksis tahlilchining asosini har doim, KO-grammatika sinfi asosida qurilgan, anglovchi tashkil etadi. SHu sababli, bosh rolni ya`ni sintaksis tahlilchi qanday xarakatlanadi va uning asosida qanday algoritm yotadi degan masalada «Kontekst-ozod grammatikalar» bo'limida qaralgan KO-tillarni qurish tamoyillari o'ynaydi.

9.2.Tahlil daraxti. Tahlil daraxtini amallar daraxtiga aylantirish

Kiruvchi til KO-grammatikasining anglovchisining ish natijasi bo'lib, kiruvchi zanjirni qurish uchun qo'llanilgan, grammatikalar qoidalarining ketma-ketligi hisoblanadi. Topilgan ketma-ketlik bo'yicha, anglovchini toifasini bilgan holda, chiqish zanjiri yoki chiqish daraxtini qurish mumkin. Bu holda chiqish daraxti sintaksis tahlil daraxti sifatida chiqadi va kompilyatorda sintaksis tahlilchining ishi natijasini ifodalaydi. Ammo chiqish zanjiri ham, sintaksis tahlil daraxti ham kompilyator ishining maqsadi hisoblanmaydi. CHiqish daraxti juda ko'p keraksiz ma`lumotlarga ega bo'lib, kompilyator keyingi ishlarida ular kerak bo'lmaydi. Bu ma`lumot o'z ichiga barcha daraxt bo'g'inlaridagi noterminal belgilarni oladi, ular daraxt qurilgandan so'ng, hech qanday yuklov ma`nosiga ega bo'lmaydilar va keyingi ishlar uchun ularning qizig'i yo'q. Kiruvchi tilning topilgan va ajratilgan sintaksis konstruksiyalarining toifasi va strukturasi haqidagi to'liq ma`lumotlarga ega bo'lish uchun, haqiqatda, ularni qurishda qo'llanilgan grammatikalar qoidalarining raqamlari ketma-ketligini bilish etarlidir. Ammo ushbu etarli ma`lumotlarni ifodalashning ko'rinishi kompilyatorning amalga oshirilishidan va kompilyasiya fazasidan bog'liq holda bo'lishi mumkin. Bu ko'rinish dasturni ichki ifodalash deb ataladi («oraliq ifodalash» yoki «oraliq dastur» terminlari xam foydalaniladi).

Dasturni ichki ifodalashning turli ko'rinishlari «Kodni generasiyasi. Kodni generasiyalash usullari» bo'limida qaraladi. Bu erda esa faqat, daraxtni ifodalovchi, bog'liq ro'yxat strukturalari haqida gap boradi, chunki xuddi ana shu strukturalarni sintaksis tahlil bosqichida, sintaksis anglovchi tomonidan keltirib chiqarilgan, chiqish qoidalari va toifasiga tayangan holda qurish oddiy va foydaliroqdir. Sintaksis daraxtda ichki bog'lamlar (cho'qqilar) amallarga mos keladilar, barglar esa operandlarni ifodalaydilar. Sintaksis daraxt barglari qoida bo'yicha, identifikatorlar jadvalining yozuvlari bilan bog'langanlar. Sintaksis daraxt strukturasi, boshlang'ich dastur yozilgan, dasturlash tilining sintaksisini aks ettiradi. Sintaksis daraxtlar kompilyator tomonidan kiruvchi dasturning ixtiyoriy qismi uchun qurilishi mumkin. Sintaksis daraxtga har doim ham natijaviy dastur kodi mos kelishi shart emas— masalan, sintaksis daraxtni tilning deklarativ qismi uchun qurish mumkin. Bu holatda daraxtdagi amallar ob`ekt kodini tug'ilishini talab qilmaydilar, lekin mos elementlar ustidagi kompilyator bajarishi kerak bo'lgan xarakatlar haqidagi ma`lumotlarni saqlaydilar. Bu holda, sintaksis tahlilchiga qandaydir amallar ketma-ketligi mos kelganda, ob`ekt kodni yaratishga olib keladigan, amallar daraxti haqida gapirish mumkin.

9.3.Amallar daraxti

Sintaksis daraxtda barcha cho'qqilar amallarga mos keladi, barglar operandlarga mos keladi va sintaksis daraxt barglari identifikatorlar jadvalining yozuvlari bilan bog'liqdir.

Amallar daraxtini sintaksis tahlilchidan tug'ilgan chiqish daraxtidan bevosita qurish mumkin. Buning uchun chiqish daraxtidan noterminal belgilarning zanjirlarini va kodni generasiyalashga ta`sir ko'rsatmaydigan, semantik ma`noga ega bo'lmagan bog'lamlarni olib tashlash etarli. Bunday bog'lamlarga misol sifatida turli, amallarni bajarilish tartibiga o'zgartirish kirituvchi, qavslar va operatorlar, xizmat qilishi mumkin, lekin daraxt qurilgandan so'ng ular hech qanday ma`noviy yuklovga ega bo'lmaydilar, chunki ularga hech qanday ob`ekt kod mos kelmaydi. U holda, qaysi bog'lam daraxtda amal hisoblanadi, qaysinisi operand ekanligini kiruvchi til sintaksisini ifodalavchi grammatikadan aniqlash mumkin emas. Xuddi shunday, qanday amallarga natijaviy dasturda ob`ekt kod mos kelishi yoki yo'qligi ham aniqlash mumkin bo'lmaydi. Bularning barchasi faqat kiruvchi dastur semantikasidan, ya`ni «ma`nosi»dan aniqlanadi. SHu sababli, faqat kompilyatorni ishlab chiqaruvchisi amallar daraxtini qurishda operandlar va amallarning qanday farqlanishini va qanday amallar ob`ekt kodni tug'ilishida semantika nuqtai nazaridan ahamiyat kasb etmasligni, aniq berishi mumkin. Sintaksis tahlil daraxtini amallar daraxtiga aylantirish algoritmini quyidagicha ifodalash mumkin.

1-qadam. Agar daraxtda boshqa noterminal belgilar bilan belgilangan bog'lamlar bo'lmasa, u holda algoritmni bajarilishi tugatiladi; aks holda — 2-qadamga o'tish.

2-qadam. Daraxtning grammatikani noterminal belgisi bilan belgilangan eng chetki chap bog'lamini tanlanadi va uni joriy qilib olinadi. 3-qadamga o'tish.

3-qadam. Agar joriy bog'lam faqat bitta pastki bog'lamga ega bo'lsa, u holda joriy bog'lamni daraxtdan olib tashlanadi, unga bog'langan bog'lamni esa yuqori

daraja bog'lamga bog'lanadi (daraxtdan zanjirni olib tashlanadi) va 1-qadamga qaytish; aks holda — 4-qadamga o'tish.

4-qadam. Agar joriy bog'lam, noterminal belgi bilan belgilangan va semantik yuklovga ega bo'lmagan, pastki bog'lamga (daraxt bargi) ega bo'lsa, u holda bu bargni daraxtdan olib tashlash va 3 –qadamga qaytish; aks holda — 5- qadamga o'tish.

5-qadam. Agar joriy bog'lam bitta terminal belgi bilan belgilangan va amal ishorasini bildiruvchi, pastki bog'lamga (daraxt bargi) ega bo'lsa, qolgan bog'lamlar esa operandlar sifatida belgilangan, u holda barg-amal ishorasi kabi belgilangan bo'lib, daraxtdan olib tashlanadi, joriy bog'lamni ushbu ishora bilan belgilash va – 3-qadamga o'tish, aks holda 6-qadamga o'tish.

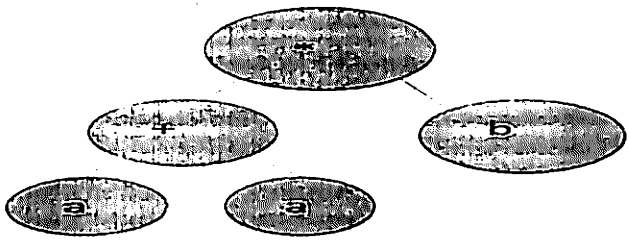
6-qadam. Agar pastki bog'lamlar orasida joriy bog'lam uchun grammatikaning noterminal belgilari bilan belgilangan bog'lamlar bo'lsa, u holda ushbu bog'lamlar orasidan eng chetki chap bog'lamni tanlash zarur, va uni joriy bog'lam qilish va 3-qadamga o'tish; aks holda algoritmni bajarilishi tugatiladi.

Bu algoritm har doim joriy hisoblangan, daraxt bog'lami bilan ishlaydi, va daraxtdan barcha noterminal belgilar bilan belgilangan bog'lamlarni olib tashlashga intiladi. Qanday belgilarni semantika nuqtai nazaridan ma`noga ega emasligini, qaysilarini amal ishoralari ekanligini kompilyatorni ishlab chiqaruvchisi hal qiladi. Agar til semantikasi to'g'ri berilgan bulsa, u holda algoritm ishi natijasida daraxtdan barcha noterminal belgilar olib tashlanadi.

Sintaksis daraxtlarga misol sifatida (a+a)*v zanjir uchun arifmetik ifodalarning grammatikasining turli variantlarini qurishni qaraymiz. Bu daraxtlarga yuqoridagi misol 14-rasmda keltirilgan. Bu grammatikada semantika nuqtai nazaridan ma`noga ega bo'lmagan belgilar bo'lib qavslar (ular amallarni tartibini beradilar va sintaksis tahlilga ta`sir ko'rsatadilar, natijaviy kodni keltirib chiqarmaydilar) va bo'sh qatorlar hisoblanadi. Amal ishoralari +, -, / va * belgilar bilan, qolgan belgilar (a va v) operandlar hisoblanadilar.

Sintaksis tahlil daraxtini amallar daraxtiga aylantirish algoritmini qo'llanilishi natijasida, 32- rasmda ko'rsatilgan amallar daraxtini olamiz. Boshlang'ich sintaksis daraxtlar, foydalanilayotgan grammatikaga bog'liq ravishda, turli strukturaga ega ekanligiga qaramay natijaviy amallar daraxti natijada har doim, kiruvchi til semantikasidan bog'liq bo'lgan, bitta strukturaga ega bo'ladilar.

Amallar daraxti dasturni ichki tasvirlashlarning ko`rinishlaridan biri hisoblanadi va undan sintaksis tahlil, semantik tahlil va kodni generasiyalashga tayyorgarlik bosqichlarida, hali natijaviy dastur komandalari kodlari bilan bevosita ishlash zaruriyati bo`lmagan vaqtda foydalanish qulaydir. Amallar daraxti barcha amallar o'rtasidagi o`zaroaloqani aniq aks ettiradi, shuning uchun undan oxirgi natijani o'zgartirmagan holda amallarni o'rinlarini almashtirish va qayta tartiblash aylantirishlarini amalga oshirish uchun foydalanish qulay (masalan, arifmetik aylantirishlar).



32-rasm. Arifmetik ifodalar tili uchun amallar daraxtining misoli

9.4.Sintaksis tahlilchilarni qurishni avtomatlashtirish (YACC dasturi)

Turli amaliy masalalarni ishlab chiqarishda ko'pincha qandaydir kiruvchi matnni sintaksis tahlili masalasi yuzaga keladi. Albatta, uni har doim mos tahlilchini qurish orqali mustaqil to'liq echish mumkin. Sintaksis tahlilni bajarish masalasi, leksik tahlil masalasi kabi, juda ko'p uchramasa ham, uni echimi uchun mos dasturlash vositalari taklif qilingan. Sintaksis tahlilchilarni qurishni avtomatlashtirish YACC dasturi yordamida bajarilishi mumkin. YACC (Yet Another Compiler Compiler) dasturi kontekst-ozod tilni sintaksis tahlilchisini qurish uchun mo'ljallangan. Tahlil qilingan til Bekus-Naur formasiga yaqin ko'rinishdagi grammatika yordamida ifodalanadi (Bekus—Naurning normal formasi). YACC dasturining ish natijasi bo'lib sintaksis tahlilchi dasturining boshlang'ich matni hisoblanadi. YACC dastur tomonidan tug'iluvchi tahlilchi LALR(l) yuqorilovchi tahlilni amalga oshiruvchi anglovchidir.

LEX, leksik tahlilchini qurishni amalga oshiruvchi, dasturi kabi, YACC dasturi UNIX toifasidagi operasion tizimlar tarixi bilan juda qattiq bog'liqdir. Bu dastur OS UNIX yoki Linux operasion tizimlarining ko'pgina versiyalarida tarkibga qo'shiladi. SHu sababli, YACC dasturi ish natijasi bo'lib, S dasturlash tilining sintaksis anglovchisining boshlang'ich matni hisoblanadi. Ammo YACC dasturining UNIX toifasidagi operasion tizimlardan farqli OTlar boshqaruvida bajariluvchi va boshlang'ich kodni boshqa dasturlash tillarida tashkil etuvchi variantlari ham mavjud (masalan, Pascal). YACC dasturining ish tamoyili LEX dasturi ish tamoyiliga o'xshash: kirishga berilgan KO-tildagi grammatikaning ifodasiga ega, fayl kelib tushadi, chiqishida esa, albatta qo'shimcha kiritish va tahrirlashni amalga oshiradigan, sintaksis anglovchini dastur matnini olamiz.

YACC dasturi uchun boshlang'ich grammatika %% belgilar bilan ajratiluvchi uch bo'lakdan tashkil topadi, ifodalar bo'lagi, grammatika ifodalanuvchi qoidalar bo'lagi, va tashkil etuvchilari chiquvchi faylga ko'chiriluvchi, dasturlar bo'lagi. Masalan, quyida YACC dasturi uchun oddiy grammatikani ifodalanishi keltirilgan bo'lib, u ko'p marta misol sifatida foydalanilayotgan arifmetik ifodalarning grammatikasiga mos keladi:

```
token a *start e
```

Ifodalar bo'lagi a identifikator leksema ekanligi haqidagi ma`lumotni saqlaydi, grammatikaning (terminal belgi), e belgi esa uning boshlang'ich terminal belgisidir. Grammatika shunday yozilganki, bu erda identifikatorlar terminal va noterminal belgilarni bildiradi; '+' va '-' toifali konstantalar belgilari terminal belgilari hisoblanadi. :, |, ; belgilar YACC dasturini metatillariga tegishli va Bekus-Naur formasiga asosan, quyidagicha o'qiladi «ta`rif bo'yicha bor», «yoki» va «qoida oxiri» mos ravishda.

LEX dan, ya`ni har doim leksik anglovchini quradigan, dasturdan farqli ravishda, agar kiruvchi fayl to'g'ri regulyar ifodaga ega bo'lsa, YAC dasturi har doim ham anglovchini qura olmaydi, xatto kiruvchi til to'g'ri KO-grammatika orqali berilgan bo'lsa ham. Axir berilgan grammatika LALR(l) sinfiga tegishli bo'lmasligi ham mumkin. Bu holatda YACC dasturi, sintaksis tahlilchini qurishda, xatolik yuz berganligi haqida ma`lumot beradi (ya`ni grammatikada LALR(l) echimi yo'q to'qnashuv yuz berganligini). U holda foydalanuvchi grammatikani yoki kerakli ko'rinishga aylantirishi, yoki YACC dasturida qandaydir qo'shimcha qoidalarni, ya`ni sintaksis tahlilchini qurishni osonlashtiradigan, qoidalarni berishi kerak. Masalan, YACC dasturi amallar ustunligini berish va ularning bajarilish tartibini ko'rsatish imkoniyatiga ega (chapdan o'ngga va o'ngdan chapga).

Grammatikaning har bir qoidasi bilan berilgan qoida bo'yicha bajariluvchi xarakatlar bog'liq. U figurali qavslar ichiga olingan til operatorlari ketma-ketligi ko'rinishida yoziladi, bu holda anglovchi boshlang'ich dasturi matni tug'iladi (ko'pincha bu S tili). Ketma-ketlik mos qoidaning o'ng qismidan so'ng joylashishi shart. SHuningdek YACC dasturi agar kiruvchi zanjir berilgan tilga tegishli bo'lmagan holda ham anglovchi xarakatlarini boshqarish imkoniga ega. Anglovchi xatolik haqida ma`lumot berish imkoniyatiga ega, tahlilni tugatishi yoki kiruvchi zanjir xatoligini tuzatib, jarayonni davom ettirishi mumkin.

9.5.Sintaksis grafni qurish

Sintaksis tahlilchini qurish uchun ikkita turli usullardan foydalanish mumkin. Ulardan biri – bu grammatik tahlilning barcha mumkin bo'lgan grammatikalar uchun kerak bo'ladigan universal dasturini yozishdir. Bu holda aniq grammatikalar ushbu dasturga uni ishini boshqaruvchi qandaydir struktura ko'rinishida beriladi. SHuning uchun bunday dastur **jadvalli-boshqaruvchi dastur** deb ataladi.

Boshka usul – aniq berilgan til uchun maxsus grammatik tahlil uchun ishlab chiqilgan dasturdir; bu holda uni sintaksisi aniq qoidalar bo'yicha operatorlar ketma-ketligida ifodalanadi, ya`ni dasturda. Tahlilning bunday amalga oshirilishini **dasturli** –**boshqariluvchi dastur** deb ataymiz.

Ushbu usullarning har biri o'zining yutuq va kamchiliklariga ega. Aniq bir til uchun translyatorni qurishda universal dasturga xos bo'lgan yuqori egiluvchanlik va parametrlash talab etilmaydi. Berilgan til uchun maxsus yozilgan grammatik tahlilning dasturi ko`pincha foydaliroq bo'ladi va u bilan ishlash osonroq kechadi. Ixtiyoriy holda ham berilgan sintaksisni **sintaksis graf ko'rinishida** ifodalash

foydalidir. Bunday graf gaplarni grammatik tahlil ishida boshqaruv jarayonini ifodalaydi.

Pastdan chiquvchi grammatik tahlil uchun tahlil maqsadi avvaldan ma`lum bo'ladi. Bu maqsad- gapni anglash, ya`ni boshlang'ich belgidan tug'iluvchi belgilar ketma-ketligini anglashdir. Bu usulni yana **maqsadga yo'naltirilgan grammatik tahlil** deb ham ataladi. Grammatik tahlilning dasturini qurishda noterminal belgilar va maqsadlarning mosligidan foydalanish maqsadga muvofiqdir, har bir noterminal belgi uchun o'zining grammatik tahlili prosedurasi quriladi. Har bir bunday proseduraning maqsadi gapning mos noterminal belgidan tug'ilishi mumkin bo'lgan bo'lagini anglash. Biz grammatik tahlilning barcha dasturini grafini qurish maqsadida ekanmiz, u holda har bir noterminal belgi qism graf ko'rinishida ifodalanadi.

Sintaksis grafni qurish qoidalari:

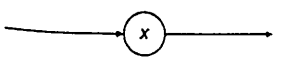
A1.Har bir noterminal belgi A mos tug'iluvchi qoidalari to'plami bilan

$A::=\beta_1|\ \beta_2|\ ....|\ \beta_n$

A sintaksis grafda o'ng qismi A2-A6 qoidalarga mos ravishda aniqlangan struktura asosida ifodalanadi.

A2. β_i qoidadagi har bir x terminal belgining paydo bo'lishi ushbu belgini kiruvchi gapda anglash operatoriga mos keladi.

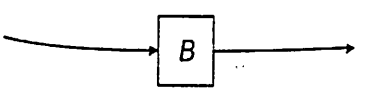
Grafda bu aylana ichiga olingan x belgi ko'rinishidagi qobiq (33-rasm) sifatida aks etadi.



33-rasm. β_i qoidadagi har bir x terminal belgining paydo bo'lishi

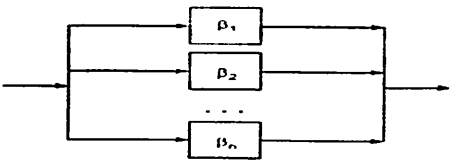
A3. β_i qoidadagi har bir V noterminal belgining paydo bo'lishi V anglash prosedurasiga murojaatga mos keladi.

Grafda bu to'g'rito'rtburchak ichiga olingan V belgi ko'rinishidagi qobiq (34 -rasm) sifatida aks etadi.



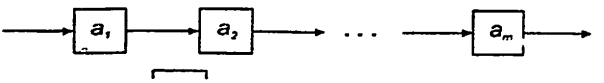
34-rasm. β_i qoidadagi har bir V'noterminal belgining paydo bo'lishi

A4: Quyidagi ko'rinishdagi tug'iluvchi qoidalar A::=β_1| β_2| ....| β_n grafda (35-rasm) quyidagicha aks etiriladi:



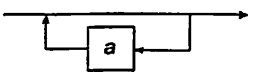
35-rasm. A::=β_1| β_2| ....| β_n ko'rinishdagi tug'iluvchi qoidalarning grafda aks etishi

A5: $\beta = \alpha_1, \alpha_2, \ldots, \alpha_m$ quyidagi ko'rinishga ega β qator grafda (36-rasm) quyidagicha ifodalanadi.



36-rasm. $\beta = \alpha_1, \alpha_2, \ldots, \alpha_m$ ko'rinishga ega β qatorning grafda aks etishi

A6: $\beta = \{\alpha\}^*$ quyidagi ko'rinishga ega β qator grafda (37–rasm) quyidagicha ifodalanadi.



37-rasm. $\beta = \{\alpha\}^*$ ko'rinishga ega β qatorning grafda aks etishi

Misol.

A::=x| (B),

B::=AC

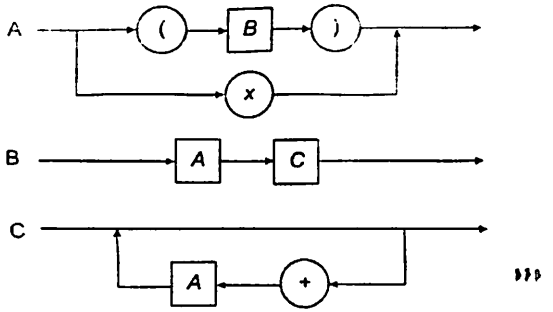
C::={+A}*

Bu erda "+", "x","(" va ")" terminal belgilar bo'lib, { va } metabelgilardir. A dan tug'iluvchi til x operandli ifodalardan, "+" amal ishorasida va qavslardan tashkil topadi.

Gaplarga misollar.

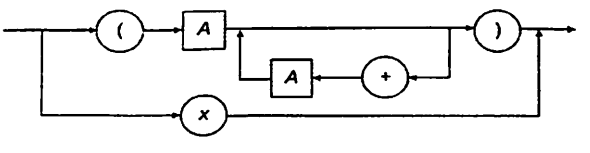
x (x) (x+x) ((x))

oltita qoidani qo'llangan holda olingan graflar quyidagi 38-rasmda keltirilgan.



38-rasm. YUqoridagi misolning sintaksis graflari ko'rinishlari

Ushbu graflar tizimini S, V va V ni A ga mos ravishda almashtirib bitta grafga (39-rasm) keltirish mumkin.



39-rasm.YUqoridagi misolning keltirib chiqarilgan sintaksis graflari ko'rinishlari

Sintaksis graf til grammatikasini ekvivalent tasvirlash hisoblanadi, undan Bekus Naur Formasi (BNF) qoidalari o'rniga foydalanish mumkin. Bu juda qulay ko'rinish bo'lib, ba`zi hollarda BNF sidan ham qulayrokdir.

Graf tilni ishlab chiqaruvchisi uchun tayanch nuqta sifatida xizmat qilib, juda qulay tasvirlash hisoblanadi.

Graf til strukturasi haqida to'liq va aniq tassavur beradi va grammatik tahlil jarayonini yaxshiroq tassavur qilishga imkon yaratadi.

Bitta belgini avvaldan qurishni ta`minlaydigan determinirlangan grammatik tahlilni ta`minlash uchun LL(1) chegaralar o'rnatilgan. Sintaksisni graf ko'rinishida ifodalashda ular quyidagicha ifodalanadi:

1. Har bir tarmoqlanishda shunday shohni tanlash kerakki, u bo'yicha navbatdagi belgi bo'yicha shu shohda tahlil ketsin. Bu ikkita bir xil shoh bitta bir xil belgidan boshlanmasligi kerakligini bildiradi.

2. Agar qandaydir A grafni hech qanday kiruvchi belgini o`qimasdan o'tish mumkin bo'lsa, u holda bunday «nollik shoh» barcha A dan keyin keladigan belgilar bilan belgilanishi kerak. (Bu shu shohga o'tish echimiga ta`sir ko`rsatadi).

9.6.Berilgan sintaksis uchun grammatik tahlil dasturini qurish

Qandaydir tilni anglovchi dasturni uning determinirlangan sintaksis grafi asosida qurish oson. Ushbu graf dasturning blok –chizmasini aks ettiradi, uni ishlab chiqarishda aylantirish qoidalariga ya`ni ular yordamida BNFdan sintaksisning graf tasavvurini olish mumkin bo'lgan qoidalarga qattik amal qilish talab qilinadi. Ushbu qoidalarning qo'llanilishi turli maqsadlarga mos keluvchi proseduralarni o'zida jamlovchi asosiy dasturning mavjud bo'lishini ta`kidlaydi.

Soddalik uchun tahlil qilishimiz kerak bo'lgan gap input kiruvchi fayl bilan berilgan va terminal belgilar esa char turidagi alohida qiymatlardir. Faraz qilaylik char ch o'zgaruvchi har doim navbatdagi o'qilishi kerak bo'lgan belgini saqlaydi. U holda keyingi belgiga o'tish quyidagi operator orqali ifodalanadi:

ch= fgetc(input);

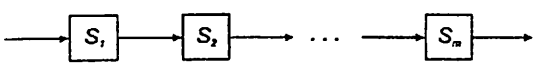
SHuni ta`kidlashimiz kerakki, funksiya char fgetc(FILE *fp) SI dasturlash tilining standart funksiyasi bo'lib, fayldan belgini o'qish uchun foydalaniladi va undan foydalanish uchun dasturning boshlanishida mos sarlavha fayl #include <stdio.h> fayl direktivani joylashtirish zarur.

Asosiy dastur birinchi belgini o'qish uchun o'qish operatoridan tashkil topadi va undan keyin grammatik tahlilning asosiy maqsadini aktivlashtiradigan aktivlik operatori keladi. Alohida grammatik tahlilning maqsadlariga yoki graflariga mos keluvchi proseduralar quyidagi qoidalar asosida olinadi. S grafni aylantirish yordamida olingan operator T(S) bilan belgilansin. Grafni dasturga aylantirish qoidasi:

B1. Mos o'rniga qo'yishlar orqali graflar tizimini mumkin qadar kichik sonli alohida graflarga keltirish.

B2. B3-B7 pastda keltirilgan qoidalar yordamida har bir grafni prosedura ifodasiga aylantirish.

B3. Elementlar ketma-ketligi (40-rasm)



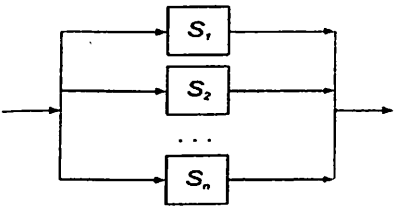
40-rasm. Elementlar ketma-ketligining graf ko'rinishi

quyidagi tarkibli operatorga o'tkaziladi.

{

```
T(S1);
T(S2);
.....;
T(Sn)
}
```

B4. Quyidagi elementlar tanlovi (41-rasm)



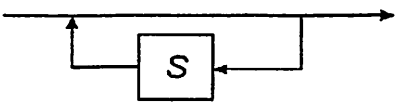
41-rasm. SHartli operatorga o'girish

shartli operatorga o'giriladi.

```
if(belongsTo (ch, L1)) T(S1);
      else if(belongsTo (ch, L2)) T(S2);
else ....
      if(belongsTo (ch, Ln)) T(Sn);
else error();
```

bu erda Li Si(Li=first(Si)) konstruksiyaning boshlang'ich belgilar to'plamini bildiradi, belongsTo (ch,L1) funksiya esa xaqiqiy qiymatni qaytaradi, agar ch belgi Li boshlangich belgilar to'plamiga tegishli bo'lsa, va yolg'on qiymat qaytaradi aks holda.

B5. Quyidagi ko'rinishdagi takrorlash (42-rasm)



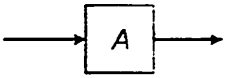
42-rasm. Takrorlash operatorga o'giriladi

quyidagi operatorga o'giriladi.

```
while( belongsTo (ch, L)) T(S);
```

bu erda T(S) B3-B7 qoidalarga mos ravishda S ni aks ettirilishi, a L esa bu L=first(S) to'plamdir.

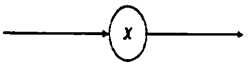
B6. Boshqa A grafni belgilovchi elementni (43-rasm)



43-rasm. Boshqa A grafni belgilovchi element

A() funksiyaga murojaat qiluvchi operatorga o'giriladi.

B7. x terminal belgini belgilovchi graf elementi (44-rasm)



44-rasm. x terminal belgini belgilovchi graf elementi

quyidagi operatorga o'giriladi.

if(ch = = 'x') ch = fgetc(input);

else error();

bu erda error() funksiyaga noto'g'ri konstruksiyaga duch kelinganda murojaat etiladi.

Endi ushbu qoidalarni redusiv grafni grammatik tahlil dasturiga aylantirish misolida ko'rib chiqamiz.

```
char ch;
void A( )
{
 if( ch= = 'x') ch= fgetc(input);
 else
   if(ch= ='('
        {
   ch= fgetc(input);
A( );
   while(ch= ='+')
          {
   ch= fgetc(input);
A( );
          }
 if( ch= = ')') ch= fgetc(input);
 else error( );
          }
   else error( );
  }
```

```
void main(int argc. char **argv)
{
 ch= fgetc(input);
A();
 }
```

Ushbu aylantirishda dasturlashning bir necha dasturni soddalashtiruvchi qoidalari qo'llanildi. Masalan, birga bir aylantirishda to'rtinchi qator quyidagi ko'rinishga ega bo'lar edi.

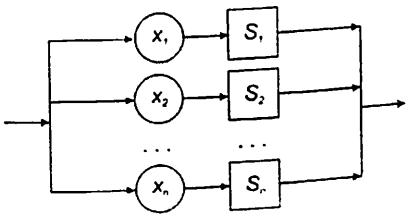
```
if( ch= = 'x')
   if( ch= = 'x') ch= fgetc(input); else error( );
else....
```

Ushbu holni xuddi shu dasturda ko'rib o'tilgandek soddalashtirish mumkin. Sakkizinchi va o'n ikkinchi qatordagi o'qish operatorlari ham xuddi shunday soddalashtirish asosida olingan.

Bunday soddalashtirishlar mumkin yoki yo'qligini aniqlash va ularni graflar ko'rinishida ko'rsatish foydaliroqdir.

Ikkita asosiy holat quyidagi qo'shimcha qoidalar bilan yopiladi:
B4a. (45-rasm)

B4a.

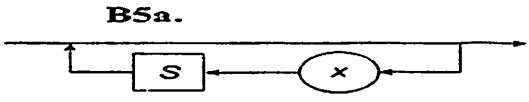


45-rasm. Ikkita asosiy holatning qo'shimcha qoidalar bilan yopilishi

```
if( ch= = 'x1')
    {
   ch= fgetc(input);
T(S1);
    }
   else
   if( ch= = 'x2')
       {
   ch= fgetc(input);
```

```
T(S2);
        }
    else
        ....
    if( ch= = 'xn')
            {
  ch= fgetc(input);
  T(Sn);
            }
    else error( );
```

B5a.(46-rasm)



46-rasm. Ikkita asosiy holatning qo'shimcha qoidalar bilan yopilishi

```
while(ch= ='x')
            {
ch= fgetc(input); T(S );
            }
```

Bundan tashqari ko'pincha uchraydigan quyidagi konstruksiyani

```
ch= fgetc(input); T(S );
while(W)
            {
  ch= fgetc(input); T(S );
            }
```

qisqacharoq quyidagicha ifodalash mumkin:

```
do {
      ch= fgetc(input);
T(S );
} while(W);
```

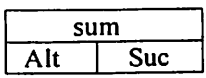
Biz bu erda error («xatolik») funksiyasini ifodalamaymiz, chunki bizni hozircha kiruvchi gapni to'g'ri ekanligini aniqlash qiziqtiradi xolos.

9.7.Grammatik tahlilning jadvalli-boshqaruv dasturini qurish

Jadvalli –boshqariluvchi universal dasturda aniq grammatikalar tahlil qilinishi kerak bo'lgan gaplarda kiruvchi berilganlar ko'rinishida beriladi. Universal dastur

grammatik tahlilning oddiy pastdan chiquvchi usuliga mos ravishda ishlaydi, shu sababli, agar determinirlangan sintaksis grafga asoslangan bo'lsa, u soddadir, ya`ni gapni qaytishsiz bitta belgiga avvaldan qarab tahlil etish mumkin.

Berilgan holda grammatika dasturning strukturasiga emas, balki berilganlarning mos keluvchi strukturasiga aylantiriladi. Grafni tasvirlashning tabiiy usuli -bu har bir belgi uchun tugun kiritish va ushbu tugunlarni ko'rsatkichlar orqali bog'lashdir. Bundan ko'rinadiki, «jadval» bu oddiy to'plam emas. Ushbu strukturaning tugunlari birlashmalarga (union) birlashtirilgan strukturalardan (struct) tashkil topadi. Birlashma tugunning toifasini identifikasiyalaydi. Birinchisi terminal belgi bilan identifikasiyalanadi va tsum bilan belgilanadi, ikkinchisi berilganlar strukturasiga ko'rsatkich bo'lib noterminal nsum ga mos keladi. Birlashmaning ikkala varianti ikkita ko'rsatkichga ega: bittasi keyingi belgiga ko'rsatadi, ko'rsatkich bo'yicha (suc) , ikkinchisi esa mumkin bo'lgan alternativlar ro'yxati bilan bog'liq (alt). Graf ko'rinishida tugunni quyidagicha (47-rasm) ifodalash mumkin.



47-rasm. Graf ko'rinishida tugunni ifodalanishi

Xuddi bunday yana bo'sh ketma-ketlikni ifodalovchi element kerak bo'ladi, «bo'sh» belgi. Uni quyidagi empty terminal element yordamida belgilaymiz.

```
struct node;
tupedef node *pointer;
struct node
{
  pointer suc;
  pointer alt;
  int isTerminal;
 union
 {
  char tsum;
  hpointer nsum;
       };
          };
```

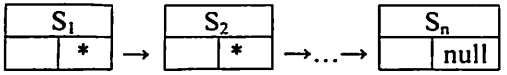
Grafni berilganlar strukturasiga aylantirish qoidalari V1-V7 qoidalar bilan bir xil.

Grafni berilganlar strukturasiga aylantirish qoidalari:

C1. Mos o'rniga qo'yishlar yordamida graflar tizimini mumkin qadar kam sonli alohida graflarga keltirish.

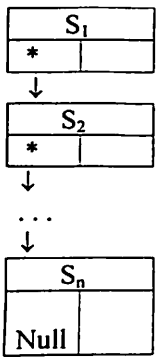
C2. Quyida keltirilgan C3-C5 qoidalarga mos ravishda har bir grafni berilganlar strukturasiga aylantirish.

C3. B3 qoidadagi elementlar ketma-ketligi quyidagi (48-rasm) tugunlar ro'yxatiga aylantiriladi:



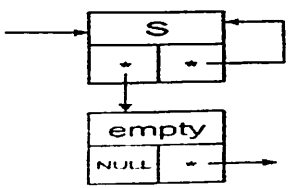
48-rasm. B3 qoidadagi elementlar ketma-ketligi quyidagining tugunlar ro'yxatiga aylantirilishi

C4.B4 qoidadagi alternativlar ro'yxati quyidagi (49-rasm) berilganlar strukturasiga aylantiriladi.



49-rasm. B4 qoidadagi alьternativlar ro'yxatining berilganlar strukturasiga aylantirilishi

C5. B5 qoidadagi takrorlashlar (sikl) quyidagi (50-rasm) strukturaga aylantiriladi.



50-rasm. B5 qoidadagi takrorlashlar (sikl)ning strukturaga aylantirilishi

Misol sifatida yuqoridagi 39-rasmda keltirilgan keltirib chiqarilgan sintaksis graf uchun berilganlar strukturasini quramiz.

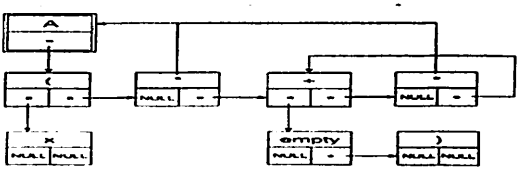
Berilganlar strukturasi shu struktura tegishli bo'lgan noterminal belgi (maqsadni) ismi sarlavha-tugun bilan identifikasiyalanadi. Umuman olganda sarlavha zarur emas, balki uning o'rniga maqsad maydoni o'rniga mos strukturaga «kirish» ni ko'rsatish mumkin. Lekin sarlavhadan chop etilayotgan struktura ismini saqlash uchun foydalanish mumkin:

```
struct header;
tupedef header * hpointer;
struct header {
  pointer entry;
    char sym;
          };
```

Kiruvchi faylda belgilar ketma-ketligi ko'rinishida tasvirlangan gapni grammatik tahlilini amalga oshirayotgan dastur bir tugundan keyingi tugunga o'tishni ifodalovchi takrorlanuvchi operatordan tashkil topadi.

Terminal belgi uchun ajratilgan sum maydoniga belgini o'zi joylanadi, noterminal belgi uchun esa mos berilganlar strukturasiga ko'rsatkich joylashadi. U graf interpretasiyasini beruvchi prosedura kabi ifodalanadi, agar noterminal belgini ifodalovchi tugun uchrab qolsa, u holda ushbu berilgan tugun joriy grafni interpretasiyasini tugashiga olib keladi. SHunday qilib interpretasiya prosedurasi rekursiv chaqiriladi. Agar kiruvchi fayl joriy belgisi (sum) berilganlar strukturasining joriy tugundagi belgisi bilan mos tushsa, u holda prosedura sus maydoni ko'rsatayotgan tugunga o'tadi, aks xolda alt maydoni ko'rsatayotgan tugunga o'tadi.

Olingan struktura 51- rasmda keltirilgan.



51-rasm. Sintaksis grafni berilganlar strukturasi ko'rinishida ifodalash

```
void parse (hpointer goal, int &match)
{
  pointer s;
  s= goal      entry;
  do
    {
    if (s    isTerminal)
      {
```

```
    if(s    tsym  = = sym)
        {
  match= 1;
  sym = fgetc(input);
        }
    else
    match =(s    tsym   = =  empty )? 1: 0;
        }
    else
  parse(s       nsym, match);
  if(match)
     s= s   suc;
     else
              s= s    alt;
            }  while(s!=NULL);
          }
```

Grammatik tahlilning bunday tashkil etilishida dastur ko'pincha tilning aniq bir formatda berilgan grammatikasini o'qiydi va berilganlarning mos strukturalarini to'ldiradi va faqat shundan keyingina berilgan tildagi matnni o'qiydi va uni sintaksis tahlil qiladi.

9.8.YUqorilovchi sintaksis tahlil

Ushbu bo'limda yuqorilovchi sintaksis tahlilning «o'tkazish-almashish» turidagi sintaksis tahlilining asosiy usullari qaraladi. Bundan buyogiga qisqacha PS-tahlil(«perenos-svertka») deb ataladi.

PS-tahlil barglardan boshlab va ildizdan daraxt yuqorisiga qarab ishlaydi va kiruvchi qatorning tahlil daraxtini qurishga harakat qiladi. Bu jarayonni w qatorni grammatikaning boshlang'ich belgisiga "svertka" almashish sifatida qarash mumkin. Almashuvning har bir qadamida qandaydir mahsulotning o'ng bo'lagidagi qismqator mahsulotning chap bo'lagidagi belgi bilan almashtiriladi va agar har bir qadamda qismqatorlar to'g'ri va aniq tanlansa, u holda biz o'ngdan keltirib chiqariluvchi murojatni olamiz.

Misol: Quyidagi grammatikani qaraymiz.

S → aABe
A → Abc| b
B → d

abbcde gap S ga quyidagi qadamlar yordamida keltiriladi:

abbcde
aAbcde
aA de
aABe
S

Biz qandaydir mahsulotni o'ng bo'lagiga mos keluvchi abbcde qatorni qismqatorni qidirish maqsadida skanerlaymiz. Bunday qismqatorlar bo'lib b va d hisoblanadi. Eng chetki chapki b ni olamiz va uni A→b; mahsulotning chap bo'lagi bo'lgan noterminal A bilan almashtiramiz, shunday qilib aAbcde qatorni olamiz. Endi mahsulotning o'ng bo'lagiga Abc, b va d qismqatorlar mos keladi. mahsulotning o'ng bo'lagiga mos keluvchi b eng chetki chap qismqator bo'lsa ham qismqatorni almashtirish uchun Abc qismqatorni tanlaymiz va uni A → Abc mahsulotga mos noterminal A bilan almashtiramiz. Natijada aAde qatorni olamiz. Mahsulotning chap bo'lagini V → d da dni V ga almashtirib aABe ni olamiz va u birinchi mahsulotga mos ravishda boshlang'ich belgi S bilan almashtiriladi. SHunday qilib, to'rtta almashtirishdan kelib chiqadigan ketma-ketlik abbcde qatorni boshlang'ich belgi S ga keltirishga imkon beradi. Bu qisqartirishlar o'ngdan keltiriladigan almashtirishlar (ya`ni teskari tartibda yozilgan) ni ifodalaydi.

S → aABe → aA de → aAbcde → abbcde

Asoslar

Noformal ravishda aytganda, qatorning asosi yoki deskriptori (handle) – bu shunday qismqatorki, u mahsulotning o'ng qismi bilan mos tushadi va u shunday o'girishki mahsulotning chap qismiga o'ng tug'ilishli bitta qadamni ifodalaydi. Ko'pgina hollarda A→β mahsulotning o'ng qismiga mos keluvchi chapdan eng chetki β qismqator asos bo'lmaydi, chunki A→β mahsulotga mos ravishda "svertka" almashish boshlang'ich belgiga o'giriladigan qatorga olib keladi. Agar avval qarab o'tgan misolimizda ikkinchi qatorda aAbcde b belgini A noterminal bilan almashtirsak, u holda aAAcde qatorni olamiz va bu qator S ga o'girilishi mumkin. SHu sababli asos tushunchasiga to'liq ta`rif berishimiz kerak.

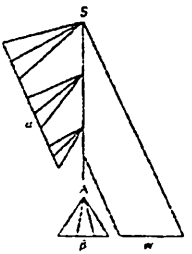
Formal ravishda aytganda, γ ning ung sentensial ko'rinishi asosi A→β mahsulot bo'lib, β va γ qatorning shunday pozisiyasiki, avvalgi o'ng sentensial ko'rinishni γ ning o'ng tug'ilishida olinishi uchun β A noterminal bilan almashtirilishi mumkin. SHunday qilib, agar S⇒ α Aw⇒ $\alpha\beta$ w, u holda A→β a dan so'ng $\alpha\beta$ w qatorning asosini ifodalaydi. w qator asosdan ungda faqat terminal belgilarga ega. SHu o'rinda grammatika, $\alpha\beta$ w ning bir necha o'ng tug'ilishlar bilan, bir qiymatli bo'lmasligini eslatib o'tamiz.

Agar grammatika birqiymatli bo'lsa, u holda grammatikaning har bir o'ngsentensial ko'rinishi bitta asosga ega bo'ladi.

YUqorida keltirilgan misolda abbcde o'ngsentensial ko'rinishni ifodalaydi va uning asosi esa 2 pozisiyada A→β bo'ladi. Xuddi shunga o'xshash aAbcde o'ngsentensial ko'rinishni ifodalaydi va deskriptori 2 pozisiyada - A→Abc. Gohida biz «β qismqator $\alpha\beta$ w asosni ifodalaydi» deb aytamiz, agar β pozisiya va A→β mahsulot bir qiymatli aniqlangan bo'lsa.

Quyidagi 52-rasmda $\alpha\beta$ w o'ng sentensial ko'rinishli ajratish daraxtining $A \rightarrow \beta$ asosi keltirilgan.

Asos eng chap tugallangan qismdaraxtni ifodalab, u tugunlar va barcha avlodlarni o'z ichiga oladi. 52-rasmda A tugun – ichki eng chetki pastki chap tugun bo'lib uning barcha avlodlari daraxtda joylashgan. $\alpha\beta$ w dagi A ga β o'girishni «asosni kesish» deb tasavvur qilish mumkin, ya`ni tahlil daraxtidan A ning barcha avlodlarini olib tashlashdir.



52-rasm. $\alpha\beta$ w tahlili daraxtidagi $A \rightarrow \beta$ asos

9.a Misol.

Quyidagi grammatikani

(1) $E \rightarrow E+E$

(2) $E \rightarrow E*E$

$E \rightarrow (E)$ (9.1)

(3) $E \rightarrow id$

va o'ng tug'ilishni ko'rib chiqamiz:

$$E \underset{rm}{\Rightarrow} \underline{E+E}$$

$$\underset{rm}{\Rightarrow} E+E*E$$

$$\underset{rm}{\Rightarrow} E+E*\underline{id_3}$$

$$\underset{rm}{\Rightarrow} E+\underline{id_2}*id_3$$

$$\underset{rm}{\Rightarrow} id_1+id_2*id_3$$

Qulaylik uchun id indeks bilan va har bir o'ngsentensial ko'rinishni asosini tagiga chizib belgiladik. Masalan, id_1 o'ng sentensial ko'rinishli $id_1+id_2*id_3$ ni asosini ifodalaydi, chunki id $E \rightarrow id$ mahsulotning o'ng qismidir va id_1 ni E ga almashtirish avvalgi o'ngsentensial ko'rinish $E+id_2*id_3$ ga olib keladi. Asosdan o'ngdagi qator faqat terminal belgilardan tashkil topganligiga ahamiyat bering.

52-rasmdagi grammatika birqiymatli bo'lmaganligi sabablli, ushbu tizimning yana bir o'ng tug'ilishi mavjud:

$$E \underset{rm}{\Rightarrow} \underline{E*E}$$
$$\underset{rm}{\Rightarrow} E* id_3$$
$$\underset{rm}{\Rightarrow} E+\underline{E* id_3}$$
$$\underset{rm}{\Rightarrow} E+ \underline{id_2} * id_3$$
$$\underset{rm}{\Rightarrow} \underline{id_1} + id_2 * id_3$$

E+E* id_3 o'ngsentensial ko'rinishni qaraymiz. Bu erda E+E E+E* id_3 ni asosi, avvalgi ko'rgan variantimizda esa uning asosi id_3 edi.

Birinchi tug'ilish * operatoriga + ga nisbatan katta ustunlik beradi, ikkinchi tug'ilishda esa + operatorining ustunligi katta.

Asoslarni qirqish

Murojat qilinayotgan o'ng tug'ilish «asoslarni qirqish» yo'li bilan olinishi mumkin. Biz jarayonni tahlil qilishimiz kerak bo'lgan w terminallar qatoridan boshlaymiz. Agar w ko'rilayotgan grammatikaining gapi bo'lsa, u holda $w=\gamma_n$,bu erda γ_n hali noma`lum bo'lgan

$$S \underset{rm}{\Rightarrow} y_0 \underset{rm}{\Rightarrow} y_1 \underset{rm}{\Rightarrow} y_2 \underset{rm}{\Rightarrow} \quad \underset{rm}{\Rightarrow} y_{n-1} \underset{rm}{\Rightarrow} y_n = w$$

o'ng tug'ilishning n –chi o'ng sentensial ko'rinishidir.

Ushbu tug'ilishni teskari tartibda tashkil etish uchun biz β_n va γ_n asosni topamiz va uni $A_n \rightarrow \beta_n$ mahsulotni , (n-1) γ_{n-1} sentensial ko'rinishni olish uchun, chap qismi bilan almashtiramiz,

So'ngra ifodalangan jarayonni qaytaramiz, ya`ni γ_{n-1} da β_{n-1} asosni topamiz va uni γ_{n-2} o'ngsentensial ko'rinishni olish uchun aylantiramiz. Agar navbatdagi qadamdan keyin o'ngsentensial ko'rinish faqat bitta boshlang'ich S belgiga ega bo'lsa, biz jarayonni tugatamiz va tahlilni muvaffaqqiyatli tugatganimiz haqida ma`lum qilamiz. Aytantirishlarda ishtirok etgan mahsulotlar ketma-ketligi kiruvchi qatorning o'ng tug'ilishlarini ifodalaydi. 9.b misol.

9.a misolda ko'rilgan (9.1)grammatikani va kiruvchi qator $id_1 + id_2 * id_3$ ni qaraymiz. Kiruvchi qatorni boshlang'ich E belgiga olib keluvchi aylantirishlar ketma-ketligi 12- jadvalda keltirilgan. YAna shuni aytib o'tishimiz kerakki, ushbu misoldagi o'ngsentensial ko'rinishlar ketma-ketligi 6.a misoldagi o'ng tug'ilishlarning birinchi kelma-ketligini ifodalaydi.

12-jadval. PS-tahlilchi bajaruvchi aylantirishlar

O'ngsentensial ko'rinish	Asos	Mahsulot
$id_1 + id_2 * id_3$	id_1	E→ id
$E+ id_2 * id_3$	id_2	E→ id
$E+E* id_3$	id_3	E→ id
E+E*E	E*E	E→ E*E
E+E	E+E	E→ E+E
E		

PS-tahlilning stekli amalga oshirilishi.

Asoslarni qirqish usuli bilan sintaksis tahlil qilishda ikkita muammo mavjud. Birinchi muammo o'ngsentensial ko'rinishdagi qismqatorni aylantirish uchun topishdadir, ikkinchisi esa o'ng qismda mos qismqatorda bir nechta mahsulot bo'lgan holda qanday mahsulotni tanlash kerakligini aniqlashdir. Ushbu savollarga javob berishdan avval PS tahlilchida foydalaniladigan berilganlar strukturasini qarab chiqamiz.

PS tahlilchini amalga oshirishning eng qulay yo'li bu tahlil qilinayotgan qatorning kiruvchi buferi va grammatika belgilarini saqlash uchun stekdan foydalanishdir. Stek uchun marker sifatida biz S dan foydalanamiz va bu belgi kiruvchi qatorning o'ng tugashining markeri bo'lib hisoblanadi. Boshlang'ich holda stek bo'sh bo'ladi, kiruvchi bufer esa w qatorga ega:

Stek Kirish
S Sw

Ushbu konfigurasiyaga tushgach sintaksis tahlilchi o'z ishini tugatadi va kiruvchi qatorning muvaffaqiyatli tugaganligi haqida ma`lum qiladi.

9.s. Misol.

9.a misolning birinchi tug'ilishidan foydalanib (9.1) grammatikaning tahlil qilishdagi sintaksis tahlilchining bajaradigan barcha xarakatlarini qadamlab o'tib chiqamiz.

Xarakatlar ketma-ketligi 13- jadvalda keltirilgan.

(9.1) grammatika kiruvchi qator uchun ikkita o'ng tug'ilishga ega bo'lganligi sababli, tahlilchi bajarishi mumkin bo'lgan yana bitta o'tishlar va aylantirishlar ketma-ketligi mavjud.

13-Jadval.$id_1 + id_2 * id_3$ kiruvchi qator uchun PS tahlilchining konfigurasiyasi

Stek	Kirish	Xarakat
(1) S	$id_1 + id_2 * id_3$ S	O'tish
(2) S id_1	$+id_2 * id_3$ S	E→ id bo'yicha aylantirish
(3) S E	$+id_2 * id_3$ S	O'tish
(4) S E+	$id_2 * id_3$ S	O'tish
(5) S E+ id_2	$* id_3$ S	E→ id bo'yicha aylantirish

(6) SE+E	* id_3 S	O'tish
(7) SE+E*	S	O'tish
(8) SE+E* id_3	S	E→ id bo'yicha aylantirish
(9) SE+E*E	S	E→ E*E bo'yicha aylantirish
(10)SE+E	S	E→ E+E bo'yicha aylantirish
(11)SE	S	Ruxsat

PS tahlilchining asosiy amallari bo'lib o'tish va aylantirish hisoblanadi, lekin PS tahlilchi quyidagi to'rtta amalni bajarishi mumkin: (1) o'tish, (2) aylantirish, (30 ruxsat, (4) xatolik.

1. O'tishda navbatdagi kiruvchi belgi stekning cho'qqisiga o'tkaziladi.
2. Aylantirishda sintaksis tahlilchi stekning o'ng oxiri asosini anglaydi, so'ngra asosning chap oxirini topadi va asosni qanday noterminal bilan almashtirganligi haqida echim qabul qiladi.
3. Ruxsatda sintaksis tahlilchi kiruvchi qatorni muvaffaqiyatli tahlil qilinganligi haqida ma`lum qiladi.
4. Xatolik holatida sintaksis tahlilchi kiruvchi oqimdagi xatolikni topadi va xatolikni tiklash dasturini chaqiradi.

PS tahlilchida stekdan foydalanishning juda muhim faktini qaraymiz: asos har doim stekning cho'qqisida joylashadi, hech qachon ichida emas.

Bu ixtiyoriy yo'nalishdagi o'ng tug'ilishlarning ikkita ketma-ket qadamida ko'rinadi. Ushbu ikkita qadam quyidagicha bo'lishi mumkin:

a. $S \underset{rm}{\Rightarrow} \alpha Az \underset{rm}{\Rightarrow} \alpha\beta Byz \underset{rm}{\Rightarrow} \alpha\beta yyz$

b. $S \underset{rm}{\Rightarrow} \alpha BxAz \underset{rm}{\Rightarrow} \alpha Bxyz \underset{rm}{\Rightarrow} \alpha yxyz$

(1) holatda A β By ga almashtiriladi, so'ngra o'ng qismidagi o'ngdan chetki noterminal V y ga almashtiriladi. (2) holatida esa A yana birinchisi bilan almashtiriladi, lekin bu holda o'ng qism faqat terminallardan iborat y ni ifodalaydi. Keyingi chetki o'ng noterminal V y dan chapda joylashadi.

(1) holatni PS tahlilchi quyidagi konfigurasiyaga erishgan vaqtidan boshlab teskari tartibda qaraymiz.

Stek	Kirish
S $\alpha\beta$ y	yzS

Endi sintaksis tahlilchi y asosni V ga aylantiradi va quyidagi konfigurasiyaga o'tadi:

Stek	Kirish
S $\alpha\beta$ V	yzS

Bu erda V $\alpha\beta$ Byz noterminalda o'ngdan chetki hisoblangani sababli, $\alpha\beta$ Byz qator asosining o'ng oxirgi stek ichida joylashishi mumkin emas.

SHunday qilib, sintaksis tahlilchi u qatorni stekka quyidagi konfigurasiyani olish uchun o'tkazishi mumkin:

Stek	Kirish
S$\alpha\beta$ Vu	zS

Bu erda β Vy asos hisoblanadi va A ga aylantiriladi.

(2) holatida quyidagi konfigurasiyalarda

Stek	Kirish
Sα u	xuzS

u asos stekning cho'qqisida joylashadi. u ni V ga aylantirilgandan so'ng sintaksis tahlilchi xu qatorni keyingi asosni olish uchun stek cho'qqisiga o'tkazishi mumkin:

Endi sintaksis tahlilchi u ni Aga aylantiradi.

Ikkala holatda ham aylantirishlardan so'ng sintaksis tahlilchi keyingi navbatdagi asosni olish uchun nol va bir qancha belgilarni stekka o'tkazadi. Sintaksis tahlilchi hech hachon stek ichiga asosning o'ng chegarasini qidirish uchun qaramaydi. Bularning barchasi PS tahlilchini ishlatilishi uchun qulaylik yaratadi.

Aktiv prefikslar

PS sintaksis tahlilchi stekida uchraydigan o'ngsentensial ko'rinishli prefikslar aktiv prefikslar (viable prefixes) deb ataladi. Aktiv prefikslarning ekvivalent aniklanishi – bu ushbu sentensial ko'rinishli chetki o'ng asosning chetki o'ng oxiriga kirmaydigan o'ngsentensial ko'rinishli prefikslardir. Bu ta`rifga asosan, aktiv prefiksning oxiriga har doim terminal belgilarni o'ngsentensial ko'rinishni olish uchun qo'shish mumkin bo'ladi. Bundan kelib chiqadiki, kiruvchi oqimning skanerlangan qismi, agar u aktiv prefiksga aylantirilgan bo'lsa, xatoga ega bo'lmaydi.

PS-taxlil jarayonidagi qarama –qarshiliklar

Kontekst-ozod grammatikalar uchun PS tahlilni qo'llab bo'lmaydi. Ixtiyoriy PS tahlilchi bunday grammatika uchun shunday konfigurasiyaga ega bo'lishi mumkinki, unda stekdagi mavjud ma`lumotlar bo'yicha va navbatdagi kiruvchi belgi bo'yicha o'tkazish yoki aylantirishdan foydalanish kerakligini (**konflikt perenos/svertka**,-**shift/reduce conflict**). yoki bir nechta aylantirishlardan qaysi biri qo'llanilishi kerakligini (**konflikt svertka/svertka , reduce/reduce conflict**) hal qila olmaydi. Bunday grammatikalarni qurilishiga olib keluvchi sintaksis konstruksiyalarga misol qaraymiz. Barcha foydalanilayotgan grammatikalar LR(1) sinfiga tegishlidir.

9.d misol.

Bir qiymatli bo'lmagan grammatika LR grammatika bo'la olmaydi. Grammatikaning «else» ishlatiluvchi holini qaraymiz.

stmt → if expr then stmt
| if expr **then** stmt **else** stmt
| other

Agar PS tahlilchi quyidagi konfigurasiyada joylashsa,

Stek	Kirish
S ... **if** expr **then** stmt	**else** ... $

u holda **if** expr **then** stmt qator asos bo'la oladimi yo'qmi, stekda nima joylashganidan qat`i nazar, ayta olmaymiz. Bu erda **konflikt perenos/svertka** kiruvchi oqimda **else** dan keyin nima kelishiga qarab yuzaga keladi, to'g'ri echim bo'lib **if** expr **then** stmt ni stmt ga aylantirish bo'lishi mumkin, yoki **else** ni o'tkazish va yana bitta stmt ni **if** expr **then** stmt **else** stmt alьternativani yakunlash uchun qidirish bo'lishi mumkin. SHunday qilib, biz bu holda aylantirishni yoki o'tkazishni ishlatish kerak deb ayta olmaymiz, demak bu grammatika LR(1) grammatika hisoblanmaydi. Bundan tashqari xech qanday bir qiymatli bo'lmagan grammatika k ning xech qanday qiymatida LR(k) grammatika bo'la olmaydi.

YAna shuni ta`kidlashimiz kerakki, yuqorida keltirilgan ba`zi birqiymatli bo'lmagan grammatikalar holida PS tahlilni oson o'zlashtirish mumkin. yuqoridagi mahsulotga ega grammatika uchun sintaksis tahlilchini qurishda biz **else** o'tkazishning **perenos/svertka** yoki stmt→ if expr then stmt aylantirishning qarama qarshiligiga ega bo'lamiz.

Agar biz ushbu qarama-qarshilikni o'tkazish foydasiga hal etsak sintaksis tahlilchi normal ishlaydi.

Grammatikaning LR grammatika bo'lmasligining yana bir sababi, asos bo'lgan holda, stekning qiymati va navbatdagi kiruvchi belgi mahsulotni aylantirishda aniqlash uchun etarli bo'lmasa yuzaga keladi.

Quyidagi misol ushbu holatni ko'rsatadi.

9.e. Misol.

Faraz qilaylik, barcha identifikatorlar uchun ularning foydalanilishidan qat`i nazar id belgini qaytaruvchi leksik tahlilchi bor bo'lsin. Faraz qilaylik, yana bizning tilimiz proseduralarni qavslarga olingan parametrlari bilan birga ismlari bo'yicha chaqiradi; xuddi shu sintaksisdan to'plamlar bilan ishlash uchun ham foydalaniladi. To'plam indekslari va proseduralarning parametrlari bir-biridan katta farq qilganliklari sababli biz parametrlar va indekslarni ro'yxatini yaratish uchun turli mahsulotlardan foydalanamiz. SHunday qilib, bizning grammatika quyidagi turdagi mahsulotga ega bo'lishi mumkin:

(1) stmt	id (parameter_list)
(2) stmt	expr :=expr
(3) parameter_list	parameter_list, parameter
(4) parameter_list	parameter
(5) parameter	id

(6) expr	id(expr_list)
(7) expr	id
(8) expr_list	expr_list, expr
(9) expr_list	expr

A(I,J) dan boshlanuvchi ma`lumot sintaksis tahlilchiga id(id.id) belgilar oqimi kabi uzatiladi. Birinchi uchta belgini stekka o'tkazilgandan so'ng PS tahlilchi quyidagi konfigurasiyaga ega bo'ladi:

Stek	Kirish
... id (id	,id...

Ko'rinib turibdiki, stekning cho'qqisida id belgi aylantirilishi kerak, lekin qaysi mahsulot bilan? To'g'ri tanlov agar A prosedura bo'lsa (5) mahsulot, agar A to'plam bo'lsa (7). Stekning qiymati nima ekanligini biz belgilar jadvalidan foydalanib bilishimiz mumkin, u erga esa A ni e`lon kilinayotganda kiritilgan bo'ladi.

Echimlardan biri id belgini (1) mahsulotdagi procid bilan almashtirish va intellektualrok leksik tahlilchidan foydalanishdir.

Ushbu leksik tahlilchi prosedura ismini ifodalovchi identifikatorni anglashda procid ni qaytaradi. Bunday usul leksik tahlilchidan belgini qaytarishdan avval belgilar jadvaliga murojaat etishni talab qiladi.

Agar biz ushbu o'zgarishlarni kiritsak, A(I,J) ni qayta ishlashda sintaksis tahlilchi yoki yuqorida keltirilgan konfigurasiyada, yoki quyidagi konfigurasiyada bo'lishi mumkin:

Stek	Kirish
... procid (id	,id...

Birinchi holda (7) mahsulot bo'yicha aylantirishni tanlaymiz; oxirgi holda (5) mahsulot bo'yicha. E`tibor bering, tanlov stekda cho'qqidan uchinchi turgan, xatto aylantirishda ishtirok etmayotgan, belgi orkali aniqlanayapti.

Tahlilni boshqarish uchun PS tahlilchi stekning «ichidagi» ma`lumotlardan foydalanishi mumkin.

9.9.Belgilar jadvali bilan ishlash

Sintaksis tahlilchi ko'pincha kontekst-ozod grammatikadan foydalanganligi sababli, tilning kontekst-ozod bo'laklarini aniklash usulini topishimiz kerak. Masalan, ko`pgina tillarda identifikatorlardan ularni ifodalamaguncha foydalanib bo'lmaydi, xuddi shunday yana dasturda turli toifadagi qiymatlardan foydalanishga ham chegaralar mavjud. Ifodalangan identifikatorlarni va ularning turlarini saqlab turish uchun ko'pgina kompilyatorlarda belgilar jadvalidan foydalaniladi.

Identifikator ifodalanayotganda, masalan int a; a ni aniqlovchi amalga oshirilishi deb ataladi. Lekin, a boshqa kontekstda ham uchrashi mumkin: a=4 yoki a+v yoki read(a) bu erda a ning amaliy amalga oshirilishi nazarga tutilayapti.

Identifikatorni aniqlovchi amalga oshirilishida kompilyator ob`ektni belgilar jadvaliga joylaydi. Amalga oshiriluvchi holda esa belgilar jadvalida mos aniqlovchi ob`ektni elementini, kompilyasiya vaqtida talab qilingan uni turini va boshqa belgilarini bilish uchun, qidiriladi.

Ko'pgina dasturlash tillarida bitta identifikatordan dasturning turli bo'laklarida turli ob`ektlarni ifodalash uchun foydalaniladi.

Bunday hollarda dasturning strukturasi ushbu ob`ektlarni farqlashga yordam beradi, masalan

```
{
int a;
.....
}
.....
{
char a;
.....
}
```

Ushbu holda a ikkita turli blokda ikkita turli ob`ektni ifodalaydi. Belgilar jadvali bitta identifikatordan ikki marta foydalanishda identifikatorlarni farqlash uchun xuddi dasturdagi blokli strukturaga ega bo'lishi kerak.

Ko'pgina zamonaviy yuqori daraja tillari quyidagi xususiyatlarga ega bo'lishi kerak:

Identifikatorni aniqlovchi amalga oshirilishi amalga oshirilishidan ko'ra matnda vaqtlirok paydo bo'lishi kerak.

Identifikatorni amaliy ishlatilishida mos keluvchi aniqlovchi amalga oshirilish ushbu identifikatorni ifodalash saqlangan blokda joylashadi.

Bitta blokda identifikator bir martadan ortiq ifodalanishi mumkin emas.

Kompilyatorning belgilar jadvalida identifikator haqidagi, kompilyasiya jarayonida kerak bo'ladigan, boshqa ma`lumotlar ham saqlanishi mumkin. Masalan, identifikatorning manzili, yoki kostantaning qiymati.

Dastur ichida belgilar jadvalining amalga oshirilishi to'plam yoki struktura zanjiri ko'rinishida bo'lishi mumkin. Bu erda har bir element o'zidan keyingi, balki o'zidan oldingi elementga ham ko'rsatadi.

To'plam ko'rinishidagi amalga oshirilishi.

Bunday tashkil etishning yutuqlari:

Belgilar jadvaliga tezlikda xotira ajratilishi (bir marta to'plamni e`lon etilishi paytida amalga oshiriladi).

To'plamning boshka elementlarini joylashtirish xaqidagi ma`lumotni saqlash zaruriyati yo'qligi sababli, joyni iqtisod qilishga erishiladi.

Tezkor qidiruv usullarini amalga oshirishning soddaligiga erishiladi (stekni imitasiyasi va x.k.).

Kamchiliklari:

Kichkina dasturlar va kam identifikatorlar ishtirok etgan holdagi translyasiya jarayonida xotiraning foydasiz ishlatilishi, chunki to'plamning kattagina qismi ishlatilmay qolib ketadi.

Katta dasturlarni translyasiyasida vaqtida shunday xolat yuzaga kelishi mumkinki, o'zgaruvchilar haqidagi ma`lumotni joylashtirish uchun fizik xotira bor, lekin to'plam esa to'lib bo'lgan. Bu holda mos ravishda to'lib ketish sababli translyasiyani amalga oshirib bo'lmaydi.

Zanjirli struktura ko'rinishidagi amalga oshirilishi (bog'langan ro'yxat).

Bunday tashkil etishning **yutuqlaridan** biri bu xotira resurslaridan to'liq foydalanishdir.

Kamchiliklari:

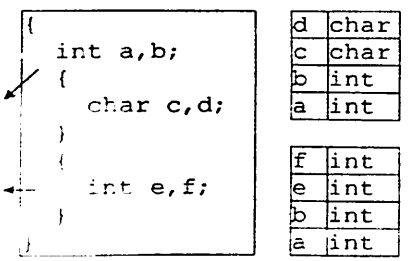
1.Xotira ajratilishi nisbatan sekin amalga oshiriladi, chunki har bir element uchun alohida ajratilidi.

2.Xotiraning qo'shimcha xarajatlanishi, ya`ni keyingi va oldingi elementga ko'rsatkichlar saqlanadi.

Amaliyotda yana ushbu ko'rinishlarning kombinasiyasi ham uchrashi mumkin.

Ifodalanish uchrashi bilan belgilar jadvalining mos elementi stekning yuqori qismiga joylashadi, blokdan chiqishda esa, ushbu blokdagi ifodalashga mos keluvchi, belgilar jadvalining barcha elementlari stekdan ketadilar. Stekning ko'rsatkichi blokka kirish paytidagi holatigacha pasayadi. Natijada tahlilning ixtiyoriy vaqtida belgilar jadvalining joriy identifikatorlarga mos keluvchi elementlari stekda joylashadi, ular bilan bog'liq identifikatorlarni amaliy va aniqlovchi realizasiyalari stekda yuqoridan pastga qidiruvni talab qiladi. Zanjirli struktura o'rniga stekdan foydalanish zanjir strukturasida ko'rsatkichlar band etgan joyni iqtisod qilish imkonini beradi.

Qarab chiqilgan usul quyidagi 53-rasmda keltirilgan.



53-rasm. Stek bilan amalga oshirilish varianti

9.10.Sintaksis xatoliklar vaqtida tiklash

Dastur matnini tahlili jarayonida translyator xatoliklar haqidagi mos diagnostikani berishi va grammatik tahlil jarayonini, keyingi mumkin bo'lgan xatoliklarni topish uchun, davom ettirishi kerak.

Ishni davom ettirish uchun :

1. noto'g'ri dastur egasi xaqiqatda nimalarni nazarda tutganligi haqida fikr yuritish kerak.
2. kiruvchi ketma-ketlikni qandaydir qismini o'tkazib yuborish kerak.
3. matnni tiklashga urinish va mumkin bo'lmagan hollarda ketma-ketlikni ba`zi bo'laklarini o'tkazib yuborish kerak.

Dasturchining dasturda nimalarni nazarda tutganligi haqida to'g'ri xulosa chiqarish anchagina qiyin masala, shuning uchun ushbu usul kam qo'llaniladi.

Ikkinchi yo'nalish bo'yicha quyidagicha ko'rsatmalar berish mumkin.

Oddiy strukturali til holida tiklash jarayoni anchagina engillashadi. Agar xatolik yuz bergan holda kiruvchi ketma-ketlikning ba`zi qismini o'tkazib yuborilsa, u holda til albatta noto'g'ri ishlatilish imkoniyati kam bo'lgan va grammatik tahlilda tiklash uchun foydalanish mumkin bo'lgan kalit so'zlar (xizmatchi so'zlar)dan iborat bo'lishi kerak.

Grammatik tahlil dasturini qurish bo'yicha:

Noto'g'ri konstruksiya uchragan vaqtda prosedura kiruvchi matnni qachonki qoida bo'yicha tilning konstruksiyasidan keyin keladigan belgi uchragunicha o'tkazib yuborishi kerak. Bu har bir grammatik tahlil proseduraslga uni aktivlashtirish vaqtida tashqi belgilar to'plami (qaralayotgan konstruksiyadan keyin kelauvchi) aniq bo'lishi kerakligini anglatadi. Har bir prosedura oxirida aniq quyidagi tekshiriv qo'yiladi: kiruvchi matnning keyingi belgisi ushbu tashqi belgilar orasida mavjudmi? Xatolikni topganidan so'ng prosedura mustaqil ravishda xatolik haqida ma`lumot berib va ishni to'xtatmasdan matnni qaerdan boshlab tahlilni tiklash kerak bo'lgan joyigacha qarab, chiqishi kerak.

SHunday qilib, grammatik tahlilning proseduraslidan to'g'ri tugatilishdan boshqa xalokatli chiqish yo'llari bo'lmaganligi maqsadga muvofiqdir.

Dastur matnidagi katta bo'laklarni keyingi tashqi belgilargacha tashlab ketish yaxshi natijalarga olib kelmasligi mumkin. SHuning uchun belgilar to'plamiga konstruksiyani boshlanishini belgilovchi va tashlab ketish mumkin bo'lmagan xizmatchi so'zlarni ko'shib qo'yiladi. SHu yo'l bilan tahlilning proseduraslga parametrlar sifatida oddiy tashqi belgilar emas, balki tiklash belgilari uzatiladi. Ko'pincha tiklash belgilari to'plami boshidan boshlab alohida xizmatchi so'zlardan tashkil topadi va grammatik tahlilning maqsadchalari darajasi kelishi davomida maqsadchalarning tashqi belgilari bilan asta-sekin to'ldirib boriladi. YUqoridagi tekshiruvni bajarish uchun umumiy funksiyani kiritamiz va uni test deb ataymiz. Ushbu funksiya uchta parametrdan iborat:

Mumkin bo'lgan keyingi belgilar to'plami s1; agar joriy belgi unga tegishli bo'lmasa, u holda xatolik yuz beradi.

Qo'shimcha tiklash belgilari to'plami s2; ularning paydo bo'lishi bu xatoliklarni anglatadi, lekin ularning ba`zilarini o'tkazib yuborib bo'lmaydi.

n rakami xatolikka beriladi, agar funksiya uni topsa.

```
void test(char[ ] s1, char [ ] s2, int n)
{
  If(!belongsTo(sym,s1)
  {
   error(n);
   s1= unite (s1,s2);
     while(!belongsTo(sym<s1))
     sym = fgetc(input);
     }
  }
```

Biron bir chizma barcha noto'g'ri konstruksiyalarning barchasi uchun qo'llanilishi birdek foydali bo'lmasligi mumkin. Ixtiyoriy tiklash chizmasi ham ba`zi bir xato konstruksiyalarni qayta ishlay olmasligi mumkin. Lekin yaxshi translyator quyidagi xususiyatlarga ega bo'lishi shart:

- hech qanday kiruvchi ketma-ketlik xalokatli holatlarga olib kelmasligi kerak.

- til qoidalari bo'yicha qonuniy bo'lmagan barcha konstruksiyalar topilishi va belgilanishi shart.

- dasturchining xatolari to'g'ri diagnostika qilishishi kerak, va translyatorning ishini hech qanday orqaga surmasligi shart.

Birinchi ikkita talab so'zsiz bajarilishi shart, oxirgi xususiyat esa bu istakdir, chunki amaliyotda har qanday holat bo'lishi va ba`zi xatoliklardan to'lik qochishning iloji bo'lmay qolishi ham mumkin.

Sinov savollari

1.Sintaksis tahlilchining bajaradigan asosiy funksiyalari haqida ma`lumot bering.

2.Tahlil daraxti va amallar daraxti orasida qanday farq mavjud?

3.Jadvalli –boshqaruvchi va dasturli boshqaruvchi sintaksis tahlilchilarning yutuq va kamchiliklarini ayting.

4.Tahlilning qanday usulini maqsadga yo'naltirilgan deb ham atash mumkin?

5.YUqorilovchi tahlil chaptomonlimi yoki o'ngtomonlimi?

6.O'ngsentensial ko'rinishning asosi tushunchasi haqida ma`lumot bering.

7.«Asoslarni qirqish» jarayoni nima ?

8.PS-tahlilchi qanday to'rt xarakatni bajaradi?

9.Aktiv prefiks nima?

10.PS tahlilchini ishidagi qarama-qarshilikka misol keltiring.

11.Sintaksis tahlilda belgilar jadvalidan foydalanish zaruriyati nimada?

12.Aniqlovchi realizasiyani amaliy realizasiyadan farqi nimada?

13.Belgilar jadvalining to'plam va bog'langan ro'yxat ko'rinishidagi realizasiyasining yutuq va kamchiliklarini sanab bering.

14.Sintaksis tahlil jarayonida yuz bergan xatoliklarni tiklashda qo'llanilishi mumkin bo'lgan yo'nalishlar haqida ma`lumot bering. Ularning qaysi biri foydalirokdir?

15.Qanday belgilar tiklash belgilari hisoblanadilar?

10.BO'LIM
Semantik tahlil bosqichlari

Semantik tahlil va kodni generasiyalashga tayyorgarlik. Semantik tahlilchining belgilanishi. Semantik tahlil bosqichlari. Dasturlash tillari leksik birliklarini identifikasiyalash. Xotirani taqsimlash

10.1.Semantik tahlil va kodni generasiyalashga tayyorgarlik

Biz bir necha marta eslatib o'tganimizdek, dasturlash tillarining deyarli barchasi, qat`iyroq aytganda, KO-tillar bo'lib hisoblanmaydilar. SHu sababli kiruvchi tilning belgilar zanjirini to'liq tahlilini kompilyator KO-tillar ramkasida KO-grammatika va MX –avtomatlar yordamida bajara olmaydi. Ko'pgina dasturlash tillarining to'liq anglovchisini KB-tillar ramkasida qurish mumkin, chunki barcha real dasturlash tillari kontekst-bog'liqdirlar.

SHunday qilib, dasturlash tili uchun to'liq anglovchini KB-til asosida qurish mumkin. Ammo ma`lumki, bunday anglovchi, hisoblash tizimi zanjirlarning bajarilishi uchun talab etiladigan, kiruvchi zanjirning uzunligidan eksponensial bog'liqlikka ega. Bunday anglovchi asosida qurilgan kompilyator, yoki ish tezligi nuqtai nazaridan, yoki zarur xotira hajmi nuqtai nazaridan foydasizroqdir. SHu sababli bunday kompilyatorlardan deyarli amaliyotda foydalanilmaydi, barcha mavjud kompilyatorlar esa kiruvchi zanjirni tahlili bosqichida kiruvchi tilning faqat sintaksis konstruksiyalarini, uning semantikasini hisobga olmagan holda, tekshiradilar.

Kompilyatorlarning foydalilik xususiyatini oshirish maqsadida kiruvchi tilning zanjirlarini tahlili ikki bosqichda bajariladi: birinchi bosqich — KO-tillar sinfining mashhur anglovchilaridan biri asosida, sintaksis tahlil; ikkinchi bosqich — kiruvchi zanjirning semantik tahlili. Kiruvchi dasturning semantik to'g'riligini tekshirish uchun tilning topilgan leksik birliklari haqidagi barcha ma`lumotlarni bilish zarur. Bu ma`lumotlar, sintaksis anglovchi tomonidan topilgan, konstruksiyalar asosida leksemalar jadvalida joylashadilar. Bunday konstruksiyalarning misoli bo'lib konstantalar va identifikatorlarni ifodalash bloklari hisoblanadilar (agar ular til semantikada ko'zda tutilgan yoki operatorlar, u yoki bu identifikator birinchi marta uchrasa (agar ifodalash birinchi marta foydalanish sababli yuz bersa). SHuning uchun kiruvchi dasturning to'liq sintaksis tahlili faqat uning sintaksis tahlilining tugatilishidan so'ng amalga oshirilishi mumkin.

SHunday qilib, semantik tahlil uchun kiruvchi ma`lumotlar bo'lib quyidagilar xizmat qiladilar:

- Identifikatorlar jadvali;
- Kiruvchi til sintaksis konstruksiyalarining tahlili natijalari.

Sintaksis tahlil bajarilishi natijalari kompilyatorda dasturlarni ichki tasvirlashning ko'rinishlaridan biri orqali ifodalanishi mumkin. Qoida bo'yicha semantik tahlil bosqichida sintaksis tahlil daraxtlarning turli variantlaridan foydalaniladi, chunki semantik tahlilchini avvalombor kiruvchi dasturning strukturasi qiziqtiradi.

Semantik tahlil ko'pincha kompilyasiya jarayonining ikki bosqichida bajariladi: sintaksis ajratish va kodni generasiyalashga tayyorgarlik bosqichining boshida. Birinchi holatda har doim tilning aniq bir sintaksis konstruksiyasini anglashdan keyin uning, identifikatorlar jadvali asosida, semantik tekshirishi bajariladi (bunday konstruksiyalar bo'lib, proseduralar, funksiyalar va kiruvchi til operatorlari bloklari hisoblanadilar. Ikkinchi holatda, sintaksis ajratishning barcha fazasini tugatgandan so'ng, identifikatorlar jadvali asosida, dasturning to'liq semantik tahlili bajariladi (bu erga, masalan, ifodalanmagan identifikatorlarni qidirish). Gohida semantik tahlilni kompilyasiyaning alohida bosqichi sifatida ajratiladi.

Har bir kompilyatorda ko'pincha semantik tahlilchini ikki varianti ham mavjud bo'ladi. Ularning aniq amalga oshirilishi kompilyatorning versiyasi va kiruvchi til semantikasidan bog'liq.

10.2.Semantik tahlil bosqichlari

Semantik tahlilchi quyidagi asosiy xarakatlarni amalga oshiradi:

- kiruvchi dasturda kiruvchi til semantikasi kelishuvlariga rioya qilishni tekshirish. Ushbu tekshiruv dasturning kiruvchi zanjirlarini kiruvchi dasturlash tili semantikasini talablariga mos qo'yishdan iborat.

a:=v+s ;

- kompilyatorda dasturlarni ichki ifodalashni, kiruvchi til semantikasida aniq ko'zda tutilmagan operatorlar va xarakatlar bilan to'ldirish;
- dasturlash tillarining, kiruvchi til bilan to'g'ridan-to'g'ri bog'lanmagan, elementar semantik (ma`noviy) normalarini tekshirish. Ushbu tekshiruv ko'pgina zamonaviy dasturlash tillari ko'rsatadigan servis funksiya hisoblanadi. Ushbuni bajarilishi majburiy shart emas.
- tilni leksik o'lchovlarini identifikasiyasi. O'zgaruvchilarni, toifalarni, proseduralarni, funksiyalarni va boshqa dasturlashning leksik tillarni identifikasiyasi– bu berilgan ob`ektlar va ularning kiruvchi dastur matnidagi ismlari o'rtasidagi bir qiymatlilikni o'rnatishdir.

Kiruvchi dasturdagi semantik kelishuvlarning rioya qilinishini tekshirish dastur kiruvchi zanjirlarini kiruvchi dasturlash tili talablariga mos qo'yish orqali bajariladi. Har bir dasturlash tili, sintaksis ajratish bosqichida tekshirib bo'lmaydigan, aniq berilgan va maxsus semantik kelishuvlarga ega. Semantik tahlilchi xuddi ana shularni birinchi navbatda tekshiradi. Bunday kelishuvlarning misollari bo'lib quyidagi talablar hisoblanadilar:

- yo'naltirishga ega har bir belgi dasturda bir marta qatnashishi shart;
- xar bir identifikator bir marta ifodalanishi shart, va hech bir identifikator bir martadan ortiq marta ifodalanishi mumkin emas (ifodalashlarning blokli strukturasini hisobga olgan holda);
- ifodalardagi barcha amallar va operandlar ushbu ifoda yoki amal uchun mumkin bo'lgan toifaga ega bo'lishi shart;
- ifodalardagi o'zgaruvchilar toifalari o'zaro kelishilgan bo'lishlari shart;

- proseduralar va funksiyalarni chaqirishda faktik parametrlar soni va toifalari formal parametrlar soni va toifalari bilan moslashishi shart.

Bu faqat shu toifadagi talablarning tahminiy tarkibidir. Semantik tahlilchi tekshirishi kerak bo'lgan talablarning aniq tarkibi kiruvchi til semantikasi bilan qattiq bog'langan (masalan, ba`zi bir tillar ba`zi aniq bir toifadagi identifikatorlarni ifodalamaslikga ruxsat etadilar).

Masalan, agar biz quyidagi ko'rinishga ega Pascal tili operatorini olsak:

```
a := b + s;
```

u holda sintaksis tahlil nuqtai nazaridan bu absolyut to'g'ri operator bo'ladi. Lekin biz bu operatorni kiruvchi til (Pascal) nuqtai nazaridan to'g'ri bo'ladimi yoki yo'qmi, unga kiruvchi barcha leksik elementlari uchun semantik talablarni tekshirmasdan turib, ayta olmaymiz. Bunday elementlar bo'lib bu erda a, b va s identifikatorlar hisoblanadilar. Ularning nimalarni ifodalashlarini bilmasdan avval, biz faqat, yuqoridagi keltirilgan operatorni to'g'riligini tasdiqlashgina emas, balki uning ma`nosini ham tushuna olmaymiz. Ushbu identifikatorlarni ifodalanishini bilishimiz zarur. Ulardan bittasi ifodalanmasdan qolgan holda ham xatolik mavjud bo'ladi. Agar ular sonli o'zgaruvchilar va konstantalar bo'lsa, u holda biz qo'shish operatori bilan ishlashimiz kerak bo'ladi, agar bular qatorli o'zgaruvchilar va konstantalar bo'lsa — u holda qatorlarni konkatenasiyasi operatori bilan ishlash kerak bo'ladi. Bundan tashqari, a identifikator, masalan, hech qachon konstanta bo'la olmaydi — aks holda o'zlashtirish operatorining semantikasi buzilgan bo'ladi. Xuddi shuninglek, identifikatorlardan birontasi son ham bo'lishi mumkin emas, qolganlari qatorlar yoki, to'plam yoki strukturalarning identifikatorlari bo'lishi ham mumkin emas — ya`ni argumentlarning bunday tarkibi qo'shish amali uchun mumkin bo'lmagan holatdir. Bu faqat kompilyator kiruvchi til semantikasi nuqtai nazaridan tekshirishi kerak bo'lgan kelishuvlarning bir bo'lagidir (ushbu misolda - Pascal).

YAna shuni ta`kidlash kerakki, semantik kelishuvlardan faqatgina operatorning to'g'riligi emas, balki uning ma`nosi ham bog'liqdir. Haqiqatan, algebraik qo'shish amallari va qatorlarni konkatenasiyasi amallari, ko'rib o'tilgan misolda bitta «+» amal belgisi bilan belgilanishlariga qaramay, turli ma`noga egadir. Xuddi shuningdek, semantik tahlilchidan dasturning natijaviy kodi ham bog'liq. Agar kiruvchi tilning qandaydir semantik talablari bajarilmasa, u holda kompilyator xatolik yuz berganligi haqida ma`lumot chiqaradi va kompilyasiya jarayoni shu erda tugatiladi.

Kompilyatorda dasturlarni ichki ifodalashni, kiruvchi til semantikasida aniq ko'zda tutilmagan operatorlar va xarakatlar bilan to'ldirish ifodalardagi operandlarning toifalarini aylantirish va funksiyalar proseduralariga parametrlarni uzatish bilan qattiq bog'liqdir.

Agar yuqoridagi Pascal tilining elementar operatoriga qaytsak,

```
a := b + s;
```

u holda shuni aytish mumkinki, bu erda ikki amal bajarilmokda: bitta amal qo'shish amali (yoki konkatenasiya, operandlar toifasiga bog'liq holda) va yana bitta amal natijani o'zlashtirish amali. Mos ravishda natijaviy dastur kodi ham tug'ilishi mumkin.

Ammo bularning barchasi ham sodda emas. Faraz qilaylik, qaerdadir ko'rilayotgan operatordan avval, biz uning operandlarini quyidagicha ifodalanishiga ega bo'laylik:

```
var
   a: real;
   b: integer;
   s: double;
```

bu ifodalanishlardan ko'rinib turibdiki, a — Pascal tilining haqiqiy o'zgaruvchisi, b butunsonli o'zgaruvchi, s — ikkilik aniqlikdagi haqiqiy o'zgaruvchidir. U holda qaralayotgan operatorning ma`nosi kiruvchi til nuqtai nazaridan sezilarli ravishda o'zgaradi, chunki Pascal tilida turli toifadagi operandlar ustida to'g'ridan-to'g'ri amallar bajarish mumkin emas. Ushbu til uchun qabul qilingan, toifalarni birini-biriga aylantirish qoidalari mavjud. Bu aylantirishlarni kim bajaradi?

Bu ishlarni dasturni ishlab chiqaruvchisi bajaradi — lekin u holda bu aylantirishlar kiruvchi dastur matnida aniq ko'rinishda keltirilishi kerak edi (biz ko'rgan misolda esa bunday emas). Boshqa holatda bu ishni kompilyator tomonidan keltirib chiqarilgan kod bajaradi, bu holda kiruvchi dastur matnida aniq ko'rinishdagi aylantirishlar mavjud bo'lmaydi, lekin kiruvchi tilning semantikasida aniq ko'zda tutilmagan semantik kelishuvlar bo'ladi.

Toifalarni aylantirish –bu semantik kelishuvlar asosida dastur kodiga, kompilyator tomonidan noaniq tarzda qo'shiladigan, amallarning faqat bittasidir. Bunday amallarning boshqa misoli bo'lib, ma`lumotlarni murakkab strukturasi elementlariga murojaat qilishdagi, manzillarni hisoblash amallari xizmat qilishi mumkin. Bunday amallarning yana boshqa variantlari ham mavjud (toifalarni aylantirish –bu faqat eng keng tarqalgan misolidir).

SHunday qilib, sintaksis tahlilchi tomonidan bajarilgan xarakatlar kompilyator tomonidan yaratiladigan natijaviy dastur kodiga sezilarli ta`sir ko'rsatadilar.

Dasturlash tillarining, kiruvchi til bilan to'g'ridan –to'g'ri bog'lanmagan, elementar ma`noviy normalarini tekshirish, bu servis funksiya bo'lib, ular ko'pgina zamonaviy kompilyatorlarda mavjuddirlar. Bu funksiya kompilyator tomonidan, ko'pgina zamonaviy dasturlash tillariga qo'llaniladigan, bajarilishi barcha dastur matnining to'liqligicha ma`noviy bog'liqligi kabi, uning alohida bo'laklari bilan ham bog'liq bo'lgan, ba`zi bir kelishuvlarni tekshirishni ta`minlaydi.

Bunday kelishuvlarning misoli sifatida quyidagi talablarni keltirish mumkin:

- har bir o'zgaruvchi yoki konstanta dasturda juda bo'lmaganda bir marta ishtirok etishi shart;
- har bir o'zgaruvchi, dastur bajarilishining ixtiyoriy qadamida, o'zining birinchi marta foydalanishigacha aniqlangan bo'lishi shart (o'zgaruvchining birinchi ishlatilishida o'zgaruvchiga har doim qandaydir qiymatni o'zlashtirilishi birga kelishi shart);
- funksiyaning natijasi uni bajarilishining ixtiyoriy qadamida aniqlangan bo'lishi shart;
- kiruvchi dasturdagi har bir operator, juda bo'lmaganda bir marta bajarilish

imkoniyatiga ega bo'lishi shart;

- shart va tanlov operatorlari dasturni o'zlarining har bir shohi bo'yicha bajarilishi qadamlariga ega bo'lishni ko'zda tutishlari kerak;
- takrorlash operatorlari takrorlashni tugatish imkoniyatini ko'zda tutgan bo'lishlari shart.

Albatta, bu asosiy kelishuvlarning tahminiy sanab o'tilishidir. Tekshirilayotgan kelishuvlarning aniq tarkibi til semantikasidan bog'liqdir. Ammo, tilning, semantik tahlilchi tomonidan qattiq tekshiriladigan, semantik talablaridan farqli ravishda, ushbu keltirilgan kelishuvlarning bajarilishi majburiy hisoblanmaydi. SHu sababli, qanday aniq kelishuvlar tekshiriladi, va ular qanday qayta ishlanadi, bu ishlab chiqaruvchi tomonidan kiritilgan, kompilyatorning sifati va funksiyalaridan bog'liqdir. Oddiy kompilyator semantik tahlilning ushbu bosqichini bajara olmasligi, bitta ham bunday kelishuvni tekshira olmasligi, mumkin, xattoki kiruvchi til semantikasi nuqtai nazaridan bu mumkin bo'lsa ham. Bunday toifa kelishuvlarning majburiy emasligi, ularning semantik tahlilchi tomonidan qayta ishlanishiga doir, yana bir xususiyatni yuklaydi. Ularga amal qilinmasligi xato sifatida ko'rilmasligi kerak. Xatto agar kompilyator o'zining haqligiga to'liq ishonsa ham, qandaydir kelishuvga amal qilinmayotganlik fakti, kiruvchi dasturni kompilyasiyasini tugatishga olib kelmasligi kerak. Ko'pincha bunday toifadagi kelishuvlarga amal qilmaslik faktining yuz berishi kompilyator tomonidan «ogohlantirish» (warning) sifatida qaraladi. Kompilyator foydalanuvchiga talablardan birining bajarilmaganligi haqida, kompilyasiya jarayonini tugatmasdan, ya`ni u foydalanuvchining e`tiborini kiruvchi dasturdagi o'sha joyga qaratish orqali, ma`lumot beradi. Ushbu «ogohlantirish» qanday qarash (boshlang'ich kodga o'zgartirish kiritish yoki ushbu faktni inkor etish), bu endi dasturni ishlab chiqaruvchisining javobgarligidagi masalalardir. Ko'rsatilgan kelishuvlarning majburiy emasligi, shu bilan tushuntiriladiki, bironta ham kompilyator boshlang'ich dastur matnini to'liq tushunish va baholashga qodir emas, Dasturning ma`nosini tushunishga faqatgina inson qodir bo'lganligi sababli (yomon yozilgan dastur uchun — uni ishlab chiqaruvchisi, yoki boshka hollarda qandaydir guruhlar), u ana shu semantik kelishuvlarga javobgar bo'lishi shart.

C tilidagi ushbu funksiyani ifodalovchi dastur bo'lagini ko'ramiz:

```
int f_test(int a)
{  int b,c;
   b = 0;
   s = 0;
   if (b=1) { return a; }
   s = a + b; }
```

Ixtiyoriy zamonaviy kompilyator bu tildan kiruvchi dasturning berilgan joyida ko'pgina «noaniqliklar»ni topadi. Masalan, s o'zgaruvchi ifodalangan, unga qiymat o'zlashtirilgan, lekin undan hech qaerda foydalanilmaydi; b o'zgaruvchini qiymati b = 0; ko'rinishdagi operatorda keltirilgan, undan ham hech qaerda foydalanilmaydi, shartli operator ham ma`noga ega emas, chunki, har doim faqat, bitta shoh bo'yicha

qadamni bajarilishini ko'zda tutadi (va demak, operator s=a+ b; hech qachon bajarilmaydi). Bu holda, kompilyator yana bitta, C tili uchun xarakterli bo'lgan, ogohlantirishni beradi, if(b=1) operatorda o'zlashtirish shartda turibdi (bu til intaksisida ham, semantikasida ham ta`qiqlanmagan, lekin C tilidagi eng ko'p tarqalgan ma`noviy xatolik hisoblanadi). Bu bo'lakning ma`nosi (to'g'rirogi, ma`nosizligi) kompilyator tomonidan to'g'ri tushuniladi va qayta ishlanadi.

Umuman olganda qo'shimcha semantik kelishuvlarning tekshiruvi kompilyatorlarning foydali funksiyasi hisoblanadi, lekin boshlang'ich dastur matnining ma`nosi bo'yicha javobgarlik, har doim avvalgidek, ishlab chiqaruvchida qoladi.

10.3.Dasturlash tillarining leksik birliklarini identifikasiyalash

Dasturlash tillarining o'zgaruvchilarni, toifalarni, proseduralarni, funksiyalarni va boshqa leksik birliklarini identifikasiyalash - bu berilgan ob`ektlar, va ularning boshlang'ich dastur matnidagi ismlari o'rtasidagi bir qiymatli moslikni o'rnatishdan iboratdir. Tilning leksik birliklarini identifikasiyalash ko'pincha semantik tahlil bosqichida bajariladi.

Qoida bo'yicha, ko'pgina dasturlash tillari boshlang'ich dasturda leksik birliklar ismlari, o'zaro ham va tilning sintaksis konstruksiyalari kalit so'zlari bilan ham, mos tushmasliklarini talab qiladilar. Ammo, ko'pincha buning o'zi leksik birliklar va ularning ismlari o'rtasidagi bir qiymatli moslikni o'rnatish uchun etarli emasdir, chunki ushbu ismlarni qo'llanilishiga doir, til tomonidan yuklatilgan, qo'shimcha ma`noviy (semantik) chegaralar mavjud.

Masalan, ko'pgina dasturlash tillarining lokal o'zgaruvchilari «ko'rinish sohasi» deb ataladigan o'zgaruvchini qo'llanilishini chegaralaydigan, ushbu o'zgaruvchi ifodalangan, kirish dasturining blokiga ega bo'ladilar. Bu esa o'sha o'zgaruvchidan, bir tomondan, o'z ko'rinish sohasidan tashqarida foydalanish mumkin emasligini anglatadi, boshqa tomondan, o'zgaruvchi ismi unikal, ya`ni yagona bo'lmay, bu ismdan ikkita turli ko'rinish sohasida ikkita turli bir xil ismli o'zgaruvchining mavjud bo'lishi mumkinligini bildiradi. Bunday toifadagi yana boshqa, unikallikni chegaralaydigan, misol sifatida, C dasturlash tilidagi ikkita turli funksiyalar yoki proseduralar turli argumentlarga ega bo'lsalar, bir xil ismga ega bo'lishlari mumkinligini keltirish mumkin. Bunday chegaralarning to'liq to'plami aniq dasturlash tilining semantikasidan bog'liq bo'ladi. Ularning barchasi tilni ifodalanishida aniq beriladi, lekin leksik tahlil bosqichida to'liq aniqlanishlari mumkin emas, shu sababli kompilyatordan, sintaksis va semantik tahlil bosqichlarida, qo'shimcha xarakatlarni talab qiladilar. Ushbu xarakatlarning umumiy yo'nalishi tilning har bir leksik birliklariga, barcha kiruvchi dastur chegarasida, unikal ism berish va so'ngra ushbu ismdan natijaviy dasturni sintezida foydalanishdan iboratdir. Kompilyatorning tilning o'zgaruvchilarini, konstantalarini, funksiyalarini, proseduralarini va boshqa leksik birliklarini identifikasiyalash uchun tahminiy xarakatlari to'plamini quyidagicha keltirish mumkin:

lokal o'zgaruvchilarning ismlari, ular ifodalangan bloklar (funksiyalar, proseduralar) ismlarini qo'shish yo'li bilan to'ldiriladi;

kiruvchi dasturning ichki o'zgaruvchilar va funksiya modullari ismlari o'zlarining ismlari bilan to'ldiriladi, bu esa faqat ichki ismlarga tegishli bo'lib, agar o'zgaruvchi yoki funksiyaga tashqaridan murojat etish mumkin bo'lsa bu holat mumkin emas;

ob`ektlarga yo'naltirilgan dasturlash tillarida ob`ektlarga (sinflarga) tegishli proseduralar va funksiyalarning ismlari, ular tegishli bo'lgan ob`ekt (sinf) ismlari bilan to'ldiriladi;

proseduralar va funksiyalarning ismlari ularning formal argumentlarining toifalariga bog'liq ravishda modifikasiyalanadilar.

Albatta, bu kompilyatorning xarakatlarini to'liq to'plami emas, har bir kompilyator o'zining xarakatlari to'plamiga ega bo'lishi mumkin. Ulardan qaysilaridan foydalanish va amaliyotda qo'llash kiruvchi dastur tilidan va ishlab chiqaruvchisidan bog'liq. Tilning leksik birliklariga o'zlashtiriladigan unikal ismlar, faqat, kiruvchi dasturning ichki ifodalanishlarida, kompilyator tomonidan foydalaniladi va dasturni tuzgan inson ular bilan to'qnashmaydi. Lekin ular foydalanuvchiga ba`zi bir holatlarda kerak bo'lishlari mumkin — masalan, dasturni qayta ishlashda, assembler tilidagi natijaviy dastur matnini hosil etishda yoki bir dasturlash tili uchun kompilyator tomonidan tashkil etilgan kutubxonadan boshqa tilda foydalanish uchun (yoki kompilyatorning boshqa versiyasida foydalanish). U holda, foydalanuvchi kompilyator qanday qoidalar asosida, kiruvchi dasturning leksik birliklari uchun, unikal ismlarni keltirib chiqarayotganini bilishi shart bo'ladi.

Ko'pgina zamonaviy kompilyatorlarda (va ular tomonidan qayta ishlanuvchi kiruvchi tillarda) tilning leksik birliklari uchun kompilyator tomonidan unikal ismlarni keltirib chiqarish jarayonini tugatish imkonini beradigan, maxsus qurilmalar va kalit so'zlar ko'zda tutilgan. Bu so'zlar yana tilning maxsus sintaksis konstruksiyalarida ham hisobga olingan (bu konstruksiyalar, export yoki external so'zlaridan tashkil etuvchilar). Agar foydalanuvchi ushbu qurilmalardan foydalansa, u holda kompilyator ko'rsatilgan leksik birliklar uchun unikal ismlarni keltirib chiqarish mexanizmini qo'llamaydi. Bu holda dasturni ishlab chiqaruvchisi, barcha kiruvchi dastur chegarasida, yoki xatto barcha loyixa chegarasida, ushbu leksik birlikning ismini unikalligi uchun javobgardir (agar bir nechta turli kiruvchi moduldan foydalanilayotgan bo'lsa). Agar unikallik talabi bajarilmasa, kompilyasiya jarayonida sintaksis yoki semantik xatoliklar yoki dasturiy ta`minotning keyingi bosqichlarida boshqa xatoliklar yuzaga kelishi mumkin. Turli dasturlash tillarida, eng keng qo'llaniladigan, leksik birliklar proseduralarning va funksiyalarning ismlari bo'lganligi sababli, ushbu masala ularga tegishlidir.

10.4.Xotirani taqsimlash

Xotirani taqsimlanishi. Xotirani taqsimlash tamoyillari

Xotirani taqsimlash-bu boshlang'ich dasturning leksik birliklariga kerak bo'ladigan manzilni, o'lchamni va xotira sohasi atributlarini mos qo'yish jarayonidir.

Xotira sohasi —bu qandaydir yo'l bilan mantiqiy birlashtirilgan ma`lumotlar uchun ajratilgan xotira yacheykasining blokidir. Bunday birlashmalarning ma`nosi kiruvchi til semantikasi bilan beriladi.

Xotirani taqsimlash tilning leksik birliklari –o'zgaruvchilar, konstantalar, funksiyalar va x.k. bilan ishlaydi va yana bu birliklar haqidagi leksik va sintaksis bosqichlarda olingan ma`lumotlar bilan ishlaydi. Kompilyatorda xotirani taqsimlash jarayonida boshlang'ich ma`lumotlar bo'lib, leksik tahlilchi tomonidan tashkil etilgan identifikatorlar jadvali va sintaksis tahlil natijasida olingan e`lon qilish bo'lagi xizmat qiladi (ifodalash sohasi). Dasturning e`lon qilish bo'lagi barcha dasturlash tillarida aniq ko'rinishda mavjud bo'lmasligi mumkin, ba`zi bir tillar konstanta va o'zgaruvchilarning ifodalanishi uchun qo'shimcha semantik kelishuvlarni ko'zda tutadi, bundan tashqari, xotirani taqsimlashdan avval, tilning leksik birliklarini identifikasiyasini bajarish kerak bo'ladi. SHu sababli xotirani taqsimlash boshlang'ich dastur matnining semantik tahlilidan so'ng bajariladi.

Zamonaviy kompilyatorlarda xotirani taqsimlash jarayoni xotirani absolyut manzillari emas, balki nisbiy manzillari yacheykalari bilan ishlaydi. Xotirani taqsimlash natijaviy dastur kodini generasiyalashdan avval bajariladi, chunki uning natijalaridan kodni generasiyalash jarayonida foydalanilishi kerak.

Dasturlash tillarining barchasida «ma`lumotlarning baza toifasi» degan tushuncha mavjud (asosiy yoki «skalyar» toifalar). Baza toifasidagi leksik birliklar uchun zarur bo'ladigan xotira sohasi o'lchami avvaldan ma`lum bo'ladi. U tilning semantikasi va kompilyator tomonidan tashkil etilgan natijaviy dastur bajarilishi kerak bo'lgan, hisoblash tizimining arxitekturasidan bog'liq ravishda aniqlanadi. Baza toifalarining leksik birliklari uchun ajratilgan xotira o'lchami kompilyatorning versiyasidan bog'liq emas, bu esa boshlang'ich ma`lumotlarni moslashish va ko'chirish imkoniyatini ta`minlaydi. Bu qoidaga kompilyatorlarni ishlab chiqaruvchilar qattiq amal qiladilar.

Dasturlarni ishlab chiqaruvchilar uchun ideal variant bo'lib, baza toifalari uchun xotira o'lchami faqatgina til semantikasidan bog'liq bo'lgan kompilyator to'g'ri kelar edi. Lekin natijaviy dasturni hisoblash tizimining arxitekturasidan bog'liqlik holatini to'liq yo'qotishni, har doim ham, amalga oshirib bo'lmaydi. U holda kompilyatorlar va dasturlash tillarini yaratuvchilar ushbu bog'liqlikni minimumga keltiruvchi mexanizmlarni ishlab chiqadilar.

Masalan, C dasturlash tilida baza toifalari bo'lib char, int, long int va x.k. toifalar hisoblanadilar (haqiqatda bu toifalar uchtadan ko'proq), Pascal dasturlash tilida esa ular — byte, char, word, integer va x.k. C tilida int baza toifasini o'lchami 12-razryadli prosessorlar bazasidagi kompyuter arxitekturasi uchun 2 baytni tashkil etadi, 32-razryadli prosessorlar uchun esa — 4 baytdir.

Ushbu tilda boshlang'ich dasturning ishlab chiqaruvchilari bu ma`lumotni bilishlari mumkin (u kompilyatorning har bir versiyasi uchun ma`lum), lekin uni dasturda to'g'ridan-to'g'ri ishlatilsa, u holda kompьyuterning aniq arxitekturasiga qattiq bog'liq bo'ladi. Bunday bog'liqlikni yo'qotish uchun dasturlash tillarining

ma`lumotlarning toifalari uchun xotira o'lchamini aniqlash mexanizmidan foydalanish foydaliroq bo'ladi, — C tilida, masalan, bu sizeof funksiyasidir.

Ma`lumotlarning murakkab strukturalari uchun ushbu strukturalarni semantikasi (ma`nosi) bilan aniqlanadigan xotirani taqsimlash qoidalaridan foydalaniladi. Bu qoidalar etarli darajada oddiy bo'lib barcha dasturlash tillarida bir xildir.

Ma`lumotlarning asosiy struktura ko'rinishlariga xotirani taqsimlash qoidalari quyidagichadir:

- to'plamlar uchun — to'plamdagi elementlar sonini bitta element uchun ajratilgan xotira o'lchamiga ko'paytirish (ushbu qoidani qatorlar uchun ham qo'llash mumkin, lekin ko'pgina tillarda qatorlar yana qo'shimcha fiksirlangan hajm haqidagi ma`lumotlarni ham o'z ichiga oladi);
- strukturalar uchun (ismli maydonli yozuvlar) — strukturaning barcha maydonlari bo'yicha xotira o'lchamlari yig'indisi;
- birlashmalar uchun (umumiy sohalar, variantli yozuvlar) — birlashmadagi maksimal maydon o'lchami;
- ob`ektlarni (sinflarni) amalga oshirish uchun — xuddi shunday ismli maydonlar strukturalar uchun xotira o'lchamiga ob`ektlarga yo'naltirilgan til ning xizmatchi ma`lumotlari uchun xotirani qo'shilgani (fiksirlangan hajm).

Kiruvchi tilning murakkab strukturali ma`lumotlari uchun ajratilgan xotira hajmi rekursiv ravishda hisoblanadi. Masalan, agar strukturalar to'plami mavjud bo'lsa, u holda ushbu to'plam uchun ajratilgan xotira hajmini hisoblashda, to'plamning bitta elementi uchun zarur bo'lgan xotirani hisoblashning struktura xotirasini hisoblash prosedurasi chaqiriladi.

Agar tilning ifodalashlar bo'lagi toifalarning daraxti ko'rinishida tasvirlangan bo'lsa, xotira hajmini bunday aniqlash yo'nalishi juda qulaydir. U holda daraxtning har bir cho'qqisi toifasi egallagan xotira hajmini hisoblash uchun, ushbu cho'qqini barcha avlodlari uchun xotira hajmini hisoblash kerak, so'ngra cho'qqining o'zi bilan bevosita bog'liq formulani qo'llash mumkin (bu mexanizm, kodni generasiyalashda qo'llaniladigan, sintaksis-boshqaruvchi tarjimaga o'xshash). Tilning ifodalash bo'lagi uchun bunday toifadagi daraxt ko'rinishidagi konstruksiyalarni sintaksis tahlil quradi.

Tilning hamma leksik birliklari ham o'zi uchun xotira ajratishni talab qilmaydi. Tilning qanday elementlari uchun xotira yacheykasi ajratish kerak yoki yo'qligini kompilyatorning amalga oshirilishi va foydalanilayotgan hisoblash tizimining arxitekturasidan bog'liq ravishda aniqlanadi. Butunsonli konstantalarni statik xotirada joylashtirish mumkin, yoki bevosita natijaviy dastur matnida ham joylashtirish mumkin (bu barcha zamonaviy hisoblash tizimlari imkoniyatidir), xuddi shu tushunchalar suzuvchi nuqtali konstantalarga ham tegishlidir, lekin ularning dastur matnida joylashtirilishi har doim ham mumkin bo'lavermaydi. Bundan tashqari, natijaviy dastur egallaydigan, xotirani iqtisod qilish maqsadida, tilning turli elementlariga kompilyator xotirani bitta yacheykasini ajratishi mumkin. Masalan, xotiraning bitta sohasida bir xil qatorli konstantalar joylashtirilishi mumkin, yoki

hech qachon bir vaqtda ishlatilmaydigan, turli lokal o'zgaruvchilar joylashtirilishi mumkin.

Tilning turli leksik birliklari va ma`lumotlar strukturalari egallaydigan xotira hajmi haqida gapirganda, xotira sohasining turli leksik birliklar uchun ajratilgan chegarasining tenglashtirilishi bilan bog'liq bir holatni eslatish kerak bo'ladi. Ko'pgina zamonaviy hisoblash tizimlarining arxitekturasi, ma`lumotlarning qayta ishlanishini foydaliroq bo'lishini ta`minlash uchun, ma`lumotlar tanlanadigan manzil qandaydir aniqlangan baytlar soniga martalik bo'lishini, ko'zda tutadi (bu 2, 4, 8 yoki 16 bayt). Zamonaviy kompilyatorlar hisoblash tizimining natijaviy dastur yo'naltirilgan xususiyatlarini hisobga oladilar. Ma`lumotlarni taqsimlashda leksik birliklarga xotira sohasini joylashtirishda optimalroq yo'l tutishlari mumkin. CHunki, leksik birlikga ajratilgan xotira o'lchami, har doim ham, ko'rsatilgan baytlar soniga martalik bo'la olmaydi, u holda natijaviy dastur uchun ajratilgan umumiy xotirada foydalanilmaydigan sohalar paydo bo'lishi mumkin.

Masalan, agar biz C tili o'zgaruvchilarining quyidagicha ifodalanishiga ega bo'lsak:

```
static char c1, s2, sZ;
```

u holda, char toifasidagi bitta o'zgaruvchiga 1 bayt xotira ajratilishini bilgan holda, yuqoridagi ifodalangan o'zgaruvchilar uchun 3 bayt xotira talab etilishini kutishimiz mumkin. Ammo xotiraga murojaat etishning manzil martaliklari 4 bayt etib o'rnatilgan bo'lsa, u holda ushbu o'zgaruvchilar uchun 12 bayt xotira ajratiladi va ulardan 9 bayti ishlatilmay qoladi.

Boshlang'ich dasturning ishlab chiqaruvchisi uchun kompilyatorning xotira sohasini qanday taqsimlanishining bilishning ahamiyati yo'q. Ko'pincha kompilyator optimal usulni, ajratilgan sohalarni chegaralarini o'zgartirgan holda, o'zi tanlaydi. Agar bir dasturlash tilida yozilgan dastur ma`lumotlaridan boshqa dasturlash tilida yozilgan dastur foydalanayotgan bo'lsa bu savol o'rtaga chiqishi mumkin (ko'pincha, assembler tilidagi), kamroq hollarda bunday muammolar bir xil kiruvchi tildan foydalanayotgan ikkita turli kompilyatordan foydalanishda paydo bo'ladi. Ko'pgina kompilyatorlar foydalanuvchiga bu holatda ushbu chegaralardan foydalanish kerak yoki yo'qligini ko'rsatish imkoniyatini beradilar.

Xotirani lokal va global xotiraga, dinamik va statik xotiraga ajratish mumkin.

Global va lokal xotira

Xotiraning global sohasi — bu natijaviy dasturni inisializasiyasi vaqtida bir marta ajratiladigan va dasturni bajarilish jarayonida har doim amal qiladigan xotira sohasidir. Qoida bo'yicha, xotiraning global sohasiga boshlang'ich dasturning ixtiyoriy bo'lagidan murojaat qilish mumkin bo'ladi, ammo ko'pgina dasturlash tillari hatto xotiraning global sohasiga murojaat qilishga sintaksis va semantik cheklovlarni qo'yish imkoniyatiga ega bo'ladilar. Bu holda xotira sohalari va ular bilan bog'liq leksik birliklar global holda qoladilar, cheklovlar esa faqat ulardan kiruvchi tildagi

dastur matnida foydalanilishiga qo'yiladi (va bu cheklovlarni aylanib o'tish mumkin).

Xotiraning lokal sohasi — bu qandaydir natijaviy dastur bo'lagining boshlanishi oldidan ajratiladigan (blok, funksiyalar, proseduralar yoki operatorlar) xotira bo'lib u ushbu bo'lakni bajarilishini tugashi bilan ozod qilinishi mumkin. Xotiraning lokal sohasiga ushbu dastur bo'lagining tashqarisidan, xotira ajratilgan sohadan boshqa, murojaat taqiqlangan.

Bu tilning ham sintaksis, ham semantik qoidalari bilan aniqlangan va shuningdek natijaviy dastur kodi bilan aniqlangan. Kiruvchi til tomonidan qo'yilgan cheklovlarni aylanib o'tilgan hollarda ham, bunday xotira sohalaridan, ya`ni ularning «ko'rinish sohasi»dan foydalanish natijaviy dastur uchun katastrofik holga olib kelishi mumkin.

Xotirani lokal va global sohalarga taqsimlanishi kiruvchi dastur tili semantikasi bilan to'liq aniqlanadi. Kiruvchi tilning sintaksis konstruksiyalari ma`nosini bilgan holda, ulardan qaysi biri global sohaga tegishli, qaysi biri lokal sohaga tegishli ekanligini aniq aytish mumkin bo'ladi. Gohida kiruvchi tilda ba`zi bir konstruksiyalar uchun aniq ajratishlar mavjud emas, u holda ularni u yoki bu xotira sohasiga tegishli ekanligini aniqlash masalasini echish kompilyatorni ishlab chiqaruvchilari zimmasiga yuklatiladi va foydalanilayotgan kompilyatorning versiyasidan bog'liq bo'lishi mumkin.

Tilning semantik xususiyatlarini, xotirani taqsimlash modulini ishlab chiqarayotganlarida, kompilyatorni tashkil etuvchilari hisobga olishlari shart.

Statik va dinamik xotira

Xotiraning statik sohasi — bu o'lchami dasturning kompilyasiya jarayonida aniq bo'lgan xotira sohasidir. Statik xotira sohasi uchun yana uning o'lchami ham ma`lum bo'lganligi sababli, kompilyator har doim ushbu xotira sohasini ajratishi mumkin va ularni dasturning mos elementlari bilan bog'laydi. SHu sababli xotiraning statik sohasi uchun kompilyator qandaydir manzilni keltirib chiqaradi (qoida bo'yicha, bu dasturdagi nisbiy manzil). Statik xotira sohasiga kiruvchi tilning ko'pgina o'zgaruvchilari va konstantalari kelib tushadilar.

Xotiraning statik sohasi kompilyator tomonidan eng oddiy qayta ishlanadi, chunki ular o'z manzillari bilan to'g'ridan to'g'ri bog'langanlar.

Xotiraning dinamik sohasi — bu o'lchami dasturning kompilyasiya jarayonida aniq bo'lmagan xotira sohasidir. Dinamik xotira sohasi o'lchami faqat natijaviy dasturning bajarilishi jarayonida aniqlanadi. SHu sababli xotiraning dinamik sohasi uchun kompilyator bevosita manzil ajrata olmaydi — dinamik xotira sohasi uchun kompilyator xotirani taqsimlash uchun javob beradigan kod bo'lagini keltirib chiqaradi (uning ajratilishi va ozod etilishi). Xotiraning dinamik sohasi bilan, ob`ektlarga mo'ljallangan dasturlash tillaridagi, ko'rsatkichlar va ob`ekt (sinflar) nusxalari ustidagi ko'pgina amallar bog'liq.

Xotiraning dinamik sohasini. o`z navbatida, foydalanuvchi tomonidan ajratilgan, xotiraning dinamik sohasi, va kompilyator tomonidan ajratilgan, xotiraning dinamik sohasiga ajratish mumkin.

Foydalanuvchi tomonidan ajratilgan xotiraning dinamik sohasi, kiruvchi dasturni ishlab chiqaruvchisi, dastur matnida xotirani taqsimlash bilan bog'liq funksiyalardan aniq foydalangan holida paydo bo'ladi (bunday funksiyalarning misoli bo'lib, New va Dispose Pascal tilida, mallos va free S tilida, new va del C++ tilida va x.k.). Xotirani taqsimlash funksiyalari yoki OT zahiralaridan to'g'ridan-to'g'ri, yoki kiruvchi til vositalaridan foydalanishi mumkin. Bu holda xotirani o'z vaqtida ajratilishi va ozod etilishiga, kompilyator emas, ishlab chiqaruvchining o'zi javob beradi.

Xotirani dinamik sohasi kompilyator tomonidan ajratiladigan hol. Bu holat foydalanuvchi kiruvchi dastur matnida ma`lumotlar toifalarini aniq ko'rinishda bermagan, va ushbu ma`lumotlar ustidagi amallar xotirani qayta taqsimlashni ko'zda tutadigan hollarda paydo bo'ladi. Bunday toifalarning misoli bo'lib, ba`zi bir dasturlash tillaridagi dinamik to'plamlar, ob`ektlarga mo'ljallangan dasturlashda ob`ektlar (sinflar) ustidagi ko'pgina amallarni keltirish mumkin (masalan—Object Pascal tilining Borland Delphi versiyasidagi st ma`lumotlar toifasi). Bu holda kompilyatorning o'zi, dasturning barcha elementlari uchun o'z vaqtida xotirani ajratilishiga va foydalanishdan so'ng o'z vaqtida ozod etilishiga, kodni tug'ilishi uchun javob beradi. Ob`ektlarga mo'ljallangan dasturlash tillarining ko'pgina kompilyatorlari ushbu maqsadlar uchun, barcha holatlarda, ya`ni dinamik xotiradan aniq va aniq bo'lmagan holda foydalanilganda, maxsus xotira menedjeridan foydalanadilar. Xotira menedjeri kodi natijaviy dastur matniga qo'shiladi yoki alohida kutubxona ko'rinishida keltiriladi.

Statik xotira sohasi va dinamik xotira sohasi o'z o'zidan global va lokal bo'lishlari mumkin.

Proseduralarning displey xotirasi (funksiyalar). Displey xotiraning stekli tashkil etilishi

Proseduralarning displey xotirasi (funksiyalar) — bu ushbu prosedurada (funksiyalarda) qayta ishlanishi mumkin bo'lgan ma`lumotlar sohasidir.

Proseduralarning displey xotirasi (funksiyalar) quyidagi tarkiblarni o'z ichiga oladi:

- dasturning barcha global ma`lumotlari (o'zgaruvchilar va konstantalar);
- proseduralarning formal argumentlari;
- berilgan proseduraning lokal ma`lumotlari (o'zgaruvchilar va konstantalar).

Global ma`lumotlar bilan hech qanday muammo tug'ilmaydi — chunki ularning manzillari aniq va ularning barchasi xotirani taqsimlash bosqichida o'rnatiladi. Xatto bu ma`lumotlarga xotira dinamik ravishda ajratilsa ham, kompilyatorga ushbu xotiraga qanday murojaat qilish va barcha dastur uchun qanday murojaat qilish birqiymatli

aniq. SHu sababli, u ushbu ma`lumotlarga, qaerdan murojaat etilayotganligiga bog'liq bo'lmagan holda, murojaat uchun kodni tashkil etishi mumkin.

Proseduraning lokal parametrlari va o'zgaruvchilari holida bu murakkabroqdir. Ular proseduraning lokal xotirasiga tegishli va shu sababli u bilan qandaydir bog'liqdirlar. Lokal parametrlar va o'zgaruvchilarni mos prosedura yoki funksiyaning kodi bilan bog'lashning bir qancha variantlari mavjud. Ularga proseduralarni displey xotirasini tashkil etishning quyidagi variantlari mos qo'yiladi:

- proseduralarni displey xotirasini statik tashkil etish;
- proseduralarni displey xotirasini dinamik tashkil etish;
- proseduralarni displey xotirasini stekli tashkil etish.

Proseduralarni (funksiyalarni) displey xotirasini statik tashkil etish boshlang'ich dastur matnining har bir prosedurasi (funksiyasi) bilan xotirani taqsimlash vaqtida kompilyator proseduraning lokal ma`lumotlarini (o'zgaruvchilar va parametrlar) joylashtirish uchun fiksirlangan xotirani bog'lashini ko'zda tutadi. Ushbu xotira sohasini manzili xotirani taqsimlash bosqichida fiksirlanadi. Proseduraning (funksiyaning) har bir formal parametriga va har bir lokal o'zgaruvchiga ushbu xotira sohasida o'z yacheykalar guruhi mos keladi. Alohida yacheykalar guruhi qaytish manzillarini, ya`ni proseduraning bajarilishidan so'ng boshqaruv uzatilishi kerak bo'lgan manzil, saqlash uchun belgilanadi.

U holda kompilyator proseduraning (funksiyaning) har bir chaqirilish nuqtasida, proseduraning chaqirishidagi faktik parametrlarini joylaydigan prosedura xotira sohasi yacheykalaridagi, uning formal parametrlariga mos, kodni vujudga keltiradi. Proseduraning chaqirilishi oldidan, bevosita chaqirish joyidan keyingi, proseduraning o'ziga boshqaruvni uzatish uchun, kodning bo'lagiga ko'rsatuvchi, qaytish manzili saqlanadi. Prosedura kodida lokal ma`lumotlarga murojaat (parametrlarga va o'zgaruvchilarga), ya`ni har bir prosedura uchun xotirani taqsimlash bosqichidan so'ng aniqlanadigan, fiksirlangan manzillar bo'yicha amalga oshiriladi. Proseduralarni (fuknsiyalarni) bajarilishini tugaganidan so'ng xotiraning fiksirlangan yacheykalaridan qaytish kodi chiqarib olinadi va u bo'yicha boshqaruv uzatiladi.

Proseduralarni displey xotirasini dinamik tashkil etish statik tashkil etish kabi tamoyillarga asoslangan, ammo proseduraning lokal ma`lumotlari uchun xotira unda har safar chaqirishdan avval dinamik ravishda ajratiladi, prosedura tugashi bilan ozod etiladi. Har bir prosedura bilan bog'liq xotira sohasi manzili ushbu holda fiksirlangan bo`la olmaydi — uni qandaydir yo'l bilan proseduraga (funksiyaga) uzatish kerak. Ushbu maqsad uchun prosessorning manzillash registri deb atalgan registridan foydalaniladi.

U holda prosedurani chaqirishda quyidagi xarakatlarni bajarish zarur:

- proseduraning lokal ma`lumotlarini saqlash uchun zarur bo'lgan xotira sohasini ajratish va uning manzilini eslab qolish;
- proseduraning parametrlari qiymatini to'ldirish va manzil qaytarish;
- chaqirish nuqtasida manzillash registrini xolatini saqlash;
- proseduraning xotira sohasini manzillash registriga joylashtirish va

proseduraga boshqaruvni uzatish.

Ushbu xarakatlarni bajargandan so'ng chaqirilgan prosedura kodi bajarilishi mumkin. Unda barcha lokal ma`lumotlar va parametrlarga murojaat manzillash registri orqali amalga oshiriladi.

Proseduradan qaytishda quyidagi xarakatlarni amalga oshirish zarur:

- proseduraning xotira sohasidan qaytish manzilini tanlash (chaqirish nuqtasidan) va unga boshqaruvni uzatish;
- manzillash registri ko'rsatayotgan xotira sohasini ozod qilish;
- qaytish nuqtasiga nisbatan surilish bo'yicha saqlangan qiymatni tanlash va uni manzillash registriga joylashtirish.

SHundan so'ng prosedurani chaqirishdan keyingi natijaviy dastur kodini ishini bajarilishini davom ettirish mumkin bo'ladi.

Ushbu chizma rekursiv chaqirishlarga yo'l qo'ysa ham, keng tarqalgan. Bu bir qator sabablarga asoslangan. Birinchidan, u xotirani doimiy dinamik qayta taqsimlashni talab qiladi — bu esa, chaqiriqlarni bajarilishini sekinlashtiradigan, etarli darajada murakkab amaldir. Ikkinchidan, bunday chizma lokal ma`lumotlarning etarli darajada murakkab manzillash chizmasini ko'zda tutadi. Va nihoyat, u oraliq ma`lumotlarni bevosita dasturning bajariluvchi kodida saqlashni talab qiladi, bu esa ma`lum murakkabliklarni keltirib chiqaradi.

Hozirgi vaqtda ushbu chizmadan foydalanilmayapti, chunki barcha ko'rsatkichlari bo'yicha displey xotirani stekli tashkil etish unga nisbatan ustundir.

Prosedurani (funksiyani) displey xotirasini stekli tashkil etish shunga asoslanganki, chaqirish vaqtida proseduraning barcha parametrlari va qaytish manzili, barcha natijaviy dastur uchun yagona bo'lgan parametrlar stekiga joylanadi, dasturni bajarilishi tugaganidan so'ng esa ular stekdan olib (chiqarib) tashlanadi. Proseduraning kodi stekning yuqorisiga nisbatan siljishi bo'yicha, lokal ma`lumotlarga manzillanadi.

Ushbu chizma zamonaviy kompilyatorlarda keng tarqalgan va ushbu mavzuni quyida to'liqroq ko'rib o'tamiz.

Proseduralar (funksiyalar) displey xotirasini stekli tashkil etish

Ushbu chizmada kompilyator barcha dastur uchun yagona bo'lgan parametrlar stekini tashkil etadi. Stekning yuqorisi prosessorning registrlaridan biri bilan manzillanadi, ya`ni stek registri bilan. Ma`lumotlarga murojaat uchun baza registri deb ataladigan, yana bitta prosessor registridan foydalanish qulay. Natijaviy dasturni bajarilishining boshida stek bo'sh bo'ladi.

U holda prosedurani chaqirishda quyidagi xarakatlarni bajarish zarur:

- proseduraning barcha parametrlarini stekka joylashtirish;
- stekda qaytish manzilini saqlash va boshqaruvni chaqirilayotgan proseduraga uzatish;
- stekda baza registrini qiymatini saqlash;

- stek registrini holatini baza registrida saqlash;
- chaqirilayotgan proseduraning bajarilishining boshida, stekda unga zarur bo'lgan barcha lokal ma`lumotlarni joylashtirish.

Ushbu xarakatlarni bajarishdan so'ng chaqirilgan proseduraning kodi bajarilishi mumkin. Undagi barcha lokal ma`lumotlar va parametrlarga murojaat baza registri orqali bajarilishi mumkin (parametrlar stekda baza registrida ko'rsatilgan joydan pastda yotadi, lokal o'zgaruvchilar va konstantalar — baza registrida ko'rsatilgan joydan yuqorida, ammo stek registrida ko'rsatilganidan pastda, yotadilar).

Proseduradan qaytishda quyidagi xarakatlarni bajarish zarur:

- stekdan proseduraning barcha lokal ma`lumotlarini tanlash;
- stekdan baza registrining qiymatini tanlash;
- stekdan qaytish manzilini tanlash;
- boshqaruvni qaytish manzili bo'yicha uzatish va stekdan proseduraning barcha parametrlarni tanlash.

SHundan so'ng, prosedurani chaqirishdan keyingi, natijaviy dastur kodini bajarilishini davom ettirish mumkin.

Hozirda xotiraning stekli tashkil etilishi zamonaviy kompilyatorlarda keng tarqalgan. Barcha mavjud yuqori daraja dasturlash tillari xotirani ushbu tashkil etilishi chizmasini ko'zda tutadilar, undan yana assembler tilidagi dasturlarda ham foydalaniladi. Prosedura va funksiyalar displey xotirasini stekli tashkil etilishini keng tarqalganligi sababi yana bu bilan bog'liqki, bu chizmada proseduralarni chaqirishdagi bajariladigan ko'pgina amallar, zamonaviy hisoblash tizimlarida, mos mashina komandalari ko'rinishlarida amalga oshirilgan. Bu parametrlar stekining barcha amallari, shuningdek prosedurani chaqirish komandalari (stekda qaytish manzilini avtomatik ravishda saqlovchi) va chaqirilgan proseduradan qaytish (stekdan qaytish manzilini tanlash). Bundan uslub prosedura displey xotirasini stekli tashkil etishda chaqiriqlarni bajarilishini sezilarli ravishda soddalashtiradi va tezlashtiradi.

Prosedura displey xotirasini stekli tashkil chaqiriqlarni rekursiv tashkil etishni imkoniyatini yaratadi (statik chizmadan farqli ravishda) va dinamik chizmadan ko'ra ustunliklarga ega. Ammo ushbu chizmada bitta sezilarli kamchilik mavjud — parametrlar stekiga ajratilgan xotira foydasiz ishlatiladi.

Ko'rinib turibdiki, parametrlar steki barcha parametrlarni, lokal ma`lumotlarni va natijaviy dasturdagi proseduraning chaqirishning juda chuqur ichki chaqirishlari manzillarini, saqlash uchun etarli bo'lgan o'lchamga ega bo'lishi kerak. Biz bilamizki, kompilyator ham, uni ishlab chiqaruvchisi ham, dasturda qanday maksimal chaqiriqlar bo'lishi mumkinligini aniq ayta olmaydi. Demak, mos ravishda, natijaviy dastur uchun bajarilish davrida, qanday steklar chuqurligi talab etilishi ham noma`lum. Stek o'lchami dastur ishlab chiqaruvchisi tomonidan «zahirali» tanlanadi, Boshqa tomondan, natijaviy dastur o'zining bajarilish vaqtida stekning ko'p qismidan foydalanmaydi, demak, kompyuter operativ xotirasi ham foydalanilmay qoladi, chunki parametrlar stekidan boshqa maqsadlarda foydalanish mumkin emas.

Prosedura va funksiyalar displey xotirasini stekli tashkil etish deyarli barcha zamonaviy kompilyatorlarda mavjud. Bundan tashqari bir yuqori daraja kiruvchi dasturlash tilida yordamida qurilgan prosedura va funksiyalardan, har doim ham boshqa dasturlash tilida yozilgan dasturlarda foydalanish mumkin bo'lmaydi. Gap shundaki, stekli tashkil etish prosedura displey xotirasini tashkil etishning asosiy tamoyillarini aniqlaydi, lekin ushbu chizmani amalga oshirish mexanizmini aniqlamaydi. Turli kiruvchi tillarda ushbu chizmani amalga oshirishning ba`zi bo'laklari prosedura parametrlarini stekka joylash va undan olish tartibi bilan farqlanadilar. Ushbu savollar turli kiruvchi dasturlash tillarining parametrlarni chaqirish va uzatish haqidagi qabul qilingan kelishuvlarida aniqlanadilar.
Parametrlar stekka to'g'ri tartibda joylashtirilishi mumkin (proseduraning ifodalanishida keltirilgan tartibda) va teskari tartibda. Stekdan ular yoki proseduradan qaytish vaqtida, yoki, bevosita qaytishdan keyin olinadilar. Bundan tashqari, dasturlarni foydalililik xususiyatini oshirish va prosedura va funksiyalarni chaqirish tezligini ko'paytirish uchun kompilyatorlar parametrlarni ba`zi bo'laklarini steklar orqali emas, balki prosessorning ozod registrlari orqali ham uzatishni ko'zda tutishlari mumkin. Ishlab chiqaruvchiga, kiruvchi dasturlash tilidagi, parametrlarni qanday uzatish qabul qilinganligi to'g'risidagi kelishuvlarni bilish zaruriyati yo'q. Boshlang'ich dastur matni ichidagi chaqiriqlarni qayta ishlashda kompilyatorning o'zi prosedura va funksiyalarni bajarish uchun kodni tashkillashtiradi. Rivojlanganroq kompilyatorlar chaqiruv kodini prosedura (funksiya) parametrlarini soni va toifasidan bog'liq ravishda hal qiladilar. Ammo, agar ishlab chiqaruvchiga bir kiruvchi til asosida tashkil etilgan prosedura va funksiyalardan, boshqa kiruvchi tildagi dasturda, foydalanish zaruriyati tug'ilsa, u holda kompilyator ushbu prosedura va funksiyalar uchun qabul qilingan parametrlarni uzatish haqidagi kelishuvlarni avvaldan bilmasligi mumkin. Bunday hollarda, ishlab chiqaruvchining o'zi parametrlarni uzatish kelishuvlari ikkala tilda ham bir xil bo'lishi haqida qayg'urishi kerak bo'ladi, chunki parametrlarni uzatishdagi mos kelmaslik natijaviy dasturni to'liq ish faoliyatini tugatishga olib kelishi mumkin. SHuning uchun, ko'pgina dasturlash tillarida, ishlab chiqaruvchiga parametrlarni qanday uzatish kelishuvidan foydalanishni aniq ko'rsatuvchi, maxsus kalit so'zlar mavjud.

Ma`lumotlar toifalari uchun xotira (RTTI-ma`lumot)

Ob`ektlarga yo'naltirilgan dasturlash tillarining zamonaviy kompilyatorlari natijaviy dastur faqatgina o'zgaruvchilar, konstantalar va boshqa ma`lumotlar tuzilmalarinigina qayta ishlabgina qolmay, balki boshlang'ich dasturda ifodalangan ma`lumotlar toifalari haqidagi ma`lumotlarni ham qayta ishlashini ko'zda tutadilar. Ushbu ma`lumotlar RTTI —Run Time Type Information — «bajarilish vaqtidagi toifalar haqidagi ma`lumotlar» deb ataladi.

RTTI- ma`lumotlar tarkibi kiruvchi til semantikasi va kompilyatorning amalga oshirilishidan tashkil topadi. Qoida bo'yicha, ob`ektlarga yo'naltirilgan dasturlash tillarining ixtiyoriy ma`lumotlar toifalari uchun, ularni toifalarini taqqoslash uchun foydalaniladigan, ma`lumotlar toifasining unikal identifikatori yaratiladi.

Ushbu identifikator uchun. natijaviy dastur kodida kompilyator foydalanadigan, toifa ismi va yana boshqa hizmatchi ma`lumotlar saqlanadi. Ushbu barcha ma`lumotlarga u yoki bu ma`noda foydalanuvchi murojaat etishi mumkin bo'ladi. RTTI-ma`lumotlarni saqlashdan yana bir maqsad ma`lumotlar — bu barcha ob`ektlarga yo'naltirilgan dasturlash tillarida ko'zda tutilgan, virtual prosedura va funksiyalarni chaqirishni korrekt mexanizmini («keyin bog'lanish» deb atalgan), ta`minlashdan iboratdir [15,16].

RTTI-ma`lumotlar RTTI-jadval ko'rinishida natijaviy dasturda saqlanadi. RTTI-jadval o'zida, natijaviy dasturni bajarilishi boshida yaratiladigan va to'ldiriladigan, global statik ma`lumotlar tuzilmasini ifodalaydi. Ob`ektlarga yo'naltirilgan kiruvchi tilning kompilyatori, natijaviy dasturda kodni, RTTI-jadvalni to'ldirilishi uchun javobgar, natijaviy dastur kodini tug'ilishi uchun javob beradi.

RTTI-jadvalda har bir ma`lumot toifasiga o'z ma`lumot sohasi mos keladi va unda ushbu ma`lumot toifasiga doir barcha zarur ma`lumotlar saqlanadi, ya`ni uning dasturning umumiy ma`lumotlar toifalari darajalanishidagi boshqa ma`lumotlar toifalari bilan o'zaro aloqalarini, shuningdek ushbu toifa bilan bog'liq barcha virtual prosedur va funksiyalar kodlariga barcha ko'rsatkichlarni saqlanadi. Ushbu barcha ma`lumotlar va ko'rsatkichlarni har bir ma`lumot toifasi uchun RTTI-jadvalda kompilyator tomonidan avtomatik ravishda tug'iladigan kod joylashtiradi.

Umuman olganda, ushbu barcha ma`lumotlarni bevosita xotirada, u yoki bu toifadagi har bir ob`ekt nusxasi (sinf) uchun statik yoki dinamik ajratilgan xotirada xotirada joylashtirish mumkin edi. Ammo, ushbu ma`lumot bir xil toifadagi i ob`ektlar uchun mos tushganligi sababli, bu holat operativ xotiradan norasional foydalanishga olib kelar edi. SHuning uchun kompilyator ma`lumotlar toifalari bo'yicha barcha ma`lumotlarni bir joyga joylashtiradi, har bir ob`ekt nusxasiga esa (sinf) xotira ajratish vaqtida unga mos ma`lumotlar toifasi bilash ushbu ob`ektni bog'lovchi juda katta bo'lmagan hizmatchi ma`lumotlarni qo'shadi (ushbu hizmatchi ma`lumotlar RTTI-jadvalda kerakli ma`lumotlar sohasiga ko'rsatkich hisoblanadilar). Ob`ekt nusxalariga (sinflar) statik xotira ajratish vaqtida kompilyator zarur hizmatchi ma`lumotlarni o'zi joylashtiradi, dinamik xotira ajratish vaqtida esa, ushbu ma`lumotlarni dasturni bajarilishi vaqtida to'ldiradigan (chunki RTTI-jadval ma`lumotlarning statik sohasi hisoblanadi, uning manzili aniq va fiksirlangan) — kodni tug'diradi.

Kompilyator har doim avtomatik ravishda natijaviy dasturda RTTI-jadvalni tashkil etish va turli toifadagi ob`ektlar bilan o'zaro aloqalarga javob beruvchi kodni quradi. RTTI-jadvalga kompilyator tomonidan joylashtirilgan xizmatchi ma`lumotlar haqida ishlab chiqaruvchining g'amho'rlik qilishi shart emas va bunday xarakatlarni amalga oshira olmaydi ham. Faqat qandaydir dastur dinamik ravishda yuklangan kutubxona bilan o'zaro xarakatlarni amalga oshirgan holdagina muammolar yuzaga kelishi mumkin. Bunday holda, dastur va kutubxona ma`lumotlari toifalarining mos kelmasliklari bilan kelib chiqqan holatlar, hatto ularning ikkalasi ham bitta kiruvchi tilda bitta kompilyator yordamida qurilgan bo'lsalar ham, yuzaga kelishi mumkin. Bu erda kimpilyator tomonidan tug'ilgan dastur kodiing inisializasiyasi o'zining RTTI-jadvalining tashkil etilishi va to'ldirilishiga javobgar, o'sha kompilyatorda

tug'ilgan kutubxonani kodi inisializasiyasi esa o`z RTTI-jadvaliga javobgardir. Agar dastur va dinamik yuklangan kutubxona bir turli toifadagi ma`lumotlar bilan ishlasalar, u holda, ushbu RTTI-jadvallar to`liq mos tushadilar, ammo boshqa boshqa jadvallar hisoblanadilar. Ushbu holatni, dasturni ishlab chiqaruvchisi ma`lumotlar toifalarini mosligini tekshirish jarayonida e`tiborga olishi kerak. YAnada katta murakkabliklar, agar ob`ektlarga mo'ljallangan dasturlash tili asosida qurilgan dastur, boshqa ob`ektlarga mo'ljallangan dasturlash tili asosida qurilgan dastur yoki kutubxona bilan o'zaro xarakatlarni amalga oshirgan bo'lsa, kelib chiqishi mumkin. CHunki RTTI-jadvallar formati va ularda saqlanuvchi xizmatchi ma`lumotlar tarkibi barcha dasturlash tillari uchun maxsuslashtirilmagan shu sababli ular qanday kompilyator versiyasidan foydalanilayotganligiga to'liq bog'liqdirlar. Mos ravishda, kompilyator tomonidan tug'iladigan natijaviy dastur kodi ham farqlanadi.

Dasturlash tilining ixtiyoriy o'zgaruvchisi kabi qandaydir sinfning ob`ektiga kompilyasiya bosqichida xotira statik ravishda ajratiladi yoki dasturning bajarilishi bosqichida dinamik ravishda ajratiladi. Boshqa dasturlash tillaridan farqli ravishda, hatto C dan C++ o'zgaruvchini asosiy dasturni ifodalanishigacha emas, balki asosiy dasturning ixtiyoriy joyida aniqlash imkonini yaratadi.

Ob`ekt sinfiga xotiraning dinamik taqsimlanishda dasturni bajarilish bosqichida ozod (dinamik) xotira hisobidan ajratiladi.

C++ tilining eng muhim oldinga surilgan juda kamchilik tushunadigan imkoniyatlaridan biri – bu Run-time turidagi ma`lumotlar xisoblanadi. Run-time turidagi ma`lumotlar –bu shunday tizimni tashkil etadiki, ular sizning dasturingizga ob`ektlarni bajarilish vaqtida anglash imkonini beradilar. Run-time C++ tilining maxsus original bo'lagi emasdir. Lekin, uning til tarkibiga qo'shilishi juda ko'p vaqtlardan beri sabrsizlik bilan kutilgan edi. CHunki shunday katta bo'lmagan, lekin muhim muammolar sinfi mavjudki, ularning echimi bevosita Run-time turidagi ma`lumotlarni kiritilishi bilan osonlashadi. Bugungi kunga kelib, C++ ning barcha asosiy kompilyatorlari Run-time turidagi ma`lumotlarga egadirlar va bu spesifikasiya dasturlar to'plamini amalga oshirishda juda keng qo'llanilmoqda.

C++ dasturlash tilida bajarilish vaqti turi haqidagi ma`lumot (RTTI) quyidagi ikkita kalit so'zlar orqali qo'llaniladi: typeid va dunamic_cast. RTTI katta bo'lagi C++ da polimorf sinflarga nisbatan qo'llaniladi.

RTTI ma`lumotlar nima uchun kerak?

Balkim bajarilish vaqtlari turlari haqidagi ma`lumot siz uchun yangilikdir, chunki C, Pascal, BASIC yoki FORTRAN kabi nopimorf dasturlash tillarida bunday ma`lumot yo'q. Nopolimorf tillarda u talab etilmaydi, chunki har bir ob`ektning turi kompilyasiya bosqichida aniqlanadi (ya`ni dasturni yozilishi jarayonida). Lekin, C++ga o'xshash polimorf tillarda shunday holatlar yuzaga kelishi mumkinki, ob`ektning turi kompilyasiya bosqichida noaniq bo`ladi, ob`ektning aniq tabiati dasturni bajarilish bosqichida aniqlanishi mumkin. C++ dasturlash tili baza sinfiga ko'rsatkichlar, virtual funksiyalar va sinflar ierarxiyasidan foydalangan holda polimorfizmni amalga oshiradi. Baza sinfiga

ko'rsatkichlar bunday yo'nalishda ushbu baza sinfiga tegishli ob`ektlarni yoki ushbu baza sinfidan keltirilib chiqariladigan ixtiyoriy ob`ektga ko'rsatish uchun foydalanilishi mumkin. SHunday qilib, har bir aniq vaqt ichida bunday ko`rsatkich qanday ob`ekt turiga ko'rsatishini avvaldan aniqlash mumkin emas. Ob`ektning turini aniqlash dasturni bajarilish vaqtida amalga oshirilishi kerak. Ko'pchilik hollarda buni amalga oshirish osongina RTTIsiz amalga oshadi, lekin RTTIga zaruriyat tug'iladigan holatlar ham ko'plab uchrab turadi.

Masalani to'liqroq yoritish uchun quyidagicha faraz qilamiz. SHunday **V** polimorf baza sinfi va undan keltirib chiqarilgan sinflar mavjud bo'lsin. So'ngra faraz qilaylik, shunday **F** funksiyani yozish talab qilinsinki, bu funksiya V turidagi ob`ekt va shu tur ob`ektlar keltirib chiqargan ob`ektlar bilan ishlasin. Bu imkoniyat mumkin bo'lishi uchun **F** funksiya bitta * **V** parametrni aniqlaydi, ya`ni baza sinfiga ko'rsatkichni aniqlaydi, Bu esa **F** funksiya **V&** turdagi ko'rsatkichlar va **V** ob`ekt keltirib chiqargan ob`ekt turlariga ko'rsatkichlarni qabul qilishini anglatadi. Lekin, agar Siz yana bitta **V** ob`ektdan keltirib chiqarilgan va **V** sinfdagi ob`ektlar ega bo'lmagan xususiyatlarga ega **D1** ob`ektni aniqlasangiz, **F** funksiya bu ob`ektga e`tibor berishi kerak bo'lsa, u holda nima yuz beradi? **F** funksiya ko'rsatkich unga uzatayotgan ob`ektlardan qaysilari **D1** turga tegishli, qaysilari tegishli emasligini qanday aniqlaydi. Boshqacha qilib aytganda, funksiya uning parametri ko'rsatayotgan qanday ob`ektlar ustida amallar bajara olishini qanday biladi? **RTTI** tizimi xuddi shunday muammolarni hal qilish uchun ishlab chiqilgan.

Muammoni yaxshiroq tushunish uchun aniq misol qarab chiqamiz. Misolimizda X,Y koordinata juftligini saqlash uchun oddiy daraja sinfidan foydalanamiz. Ushbu **coord** deb nomlangan baza sinfi quyida keltirilgan.

```
//Bazali polimorf sinf
 class coord {protected:
int x,y; //koordinatalar qiymati
public: coord()
{x=0; y=0;}
coord(int i, int j)
 {x=i; y= j;}
void get_xy (int &i, int &j)
{ i=x; j=y:}void set_xy (int i, int j)
{x=i; y= j:}virtual void show() {cout<< x<< ","<<y;}};
```

Ushbu sinf asosida ikkita **translate_coord** va **abs_coord** nomli sinflar keltirib chiqarilgan. **translate_coord** sinfi **coord** baza sinfini **deltaX** va **deltaY** qiymatlari bilan ko`rsatuvchi koordinata aylantirishlarini hal qiladi.**abs_coord** sinfi esa koordinatalarni absolyut qiymatlarini saqlashni tashkil etadi.

Sinflarning bunday shajarasini tashkil etib, endi **ptr** ko'rsatkich ko'rsatayotgan nuqtaning qaerda joylashganligini kvadrant qiymatini ma`lum qiluvchi funksiyani qaraymiz.

```
void quadrant(coord *ptr)
{int x,y;ptr-> get_xy(x,y);
if(x>0 && y>0) cout<<"Nuqta birinchi kvadrantda joylashgan \n";
else if(x<0 && y>0)cout<<"Nuqta ikkinchi kvadrantda joylashgan \n";
else if(x<0 && y<0)cout<<"Nuqta uchinchi kvadrantda joylashgan\n";
else if(x>0&&y<0)cout<<"Nuqta to'rtinchi kvadrantda joylashgan \n";
else cout <<"Nuqta koordinata boshida joylashgan \n";}
```

Ushbu funksiya ko'rsatilgan nuqtaning koordinatalarini oladi va nuqta joylashgan kvadrantni topadigandek tuyuladi, ammo ushbu funksiyaning parametri **coord*** ko'rsatkich bo'lgani uchun uni **coord** dan keltirib chiqarilgan ixtiyoriy sinf ob`ektlariga ko'rsatkich sifatida ko'rsatib chaqirish mumkin. Barcha muammo ana shu erda yashiringan. Masalan, biz ushbu funksiyani **translate_coord** turidagi ob`ektga ko'rsatkich bilan chaqirsak nima bo'ladi? Kamida funksiya nuqta joylashgan kvadrantni noto'g'ri aniqlaydi, chunki u **deltaX** va **deltaY** aylantirishlar qiymatlarini e`tiborga olmaydi. Ushbu muammoni hal qilishni quyida keltirilgan misolda keltiramiz.

```
quadrant(): // Kvadrant raqamini bildiruvchi funksiya.
void quadrant(coord *p);
{translate_coord *p;
 int x,y;
 int dx,dy;
p=(translate_coord *)
 ptr;p->get_xy(x,y);
p->get_trans(dx,dy);
x+=dx;  // normallashtirish
y+=dy;
if(x>0&&y>0) cout<<"Nuqta birinchi kvadrantda joylashgan \n";
else if(x<0&&y>0) cout<<"Nuqta ikkinchi kvadrantda joylashgan \n";
 else if(x<0&&y<0) cout<<"Nuqta uchinchi kvadrantda joylashgan \n";
else if(x>0&&y<0) cout<<"Nuqta to'rtinchi kvadrantda joylashgan \n";
 else cout<<"Nuqta koordinata boshida joylashgan \n";
}
```

Ushbu versiya ptr ko'rsatkichni translate_coord sinfi ob`ektining ko'rsatkichiga aylantiradi, get_trans funksiyasini chaqirib koordinat aylantirishlar qiymatini oladi, shundan so'ng koordinatalarni normallashtiradi. Lekin endi

boshqa muammoga duch kelamiz. quadrant funksiyasi coord yoki abs_coord turidagi ob`ektga ko'rsatkich bilan chaqirilgan bo'lsachi? Bu sinflarning hech biri get_trans a`zo funksiyasini aniqlamaydi. Ushbu dastur bilan ko'pgina g'alati holatlar yuz berishi mumkin, hattoki kompyuterni buzilishiga ham olib kelishi mumkin. Xuddi shu muammoni RTTI tizimi yordamida hal qilish mumkin.

Echilishi uchun RTTI tizimi talab etiladigan masalalar sinflarning kutubxona sinflaridan keltirib chiqariladigan murakkab ierarxiyasi bilan bog'liqdirlar. Anchagina sodda masalalarning echilishida RTTI tizimi talab etilmaydi, chunki ularni C++ dasturlash tilining boshqa mexanizmlari yordamida hal qilsa bo'ladi.

Sinov savollari

1.Cemantik tahlil uchun kiruvchi ma`lumotlar bo'lib qanday ma`lumotlar xizmat qiladilar?

2.Semantik tahlil kompilyasiya jarayonining qaysi bosqichida bajariladi?

3.Semantik tahlilchi amalga oshiradigan asosiy xarakatlar haqida ma`lumot bering.

4.Kiruvchi til semantikasi kelishuvlariga rioya qilish deganda nimalarni nazarda tutilayapti?

5.Tilni leksik o'lchovlarini identifikasiyasi nima uchun zarur?

6.Toifalarni aylantirish haqida ma`lumot bering.

7.Nima uchun semantik tahlil bosqichida boshlang'ich dastur matnining ma`nosi bo'yicha javobgarlik ishlab chiqaruvchida qoladi?

8. Xotirani taqsimlash jarayoni haqida ma`lumot bering.

9. Xotira sohasi haqida ma`lumot bering.

10.Zamonaviy kompilyatorlarda xotirani taqsimlash jarayoni qanday amalga oshiriladi ?

11.Dasturlash tillarining «ma`lumotlarning baza toifasi» degan tushunchasi nimani bildiradi?

12.Xotirani ko'rinishlari haqida ma`lumot bering.

13. Xotiraning global sohasi haqida ma`lumot bering.

14. Xotiraning lokal sohasi haqida ma`lumot bering.

15. Xotiraning statik sohasi haqida ma`lumot bering.

16. Xotiraning dinamik sohasi haqida ma`lumot bering.

17. Proseduralarning displey xotirasi haqida ma`lumot bering.

18. Proseduralarning displey xotirasining (funksiyalar) tarkiblari haqida ma`lumot bering.

11.BO'LIM
Kodni generasiyalash

Kodni generasiyalash usullari. Kodni generasiyalashning asosiy tamoyillari. Sintaksis boshqaruvchi tarjima. Dasturlarni ichki tasvirlash usullari

11.1.Kodni generasiyalash. Kodni generasiyalash usullari

Kodni generasiyalashning asosiy tamoyillari

Ob`ekt kodni generasiyalash deb boshlang'ich dastur matnini ichki ifodalanishini kompilyator tomonidan chiquvchi tilning belgilar zanjiriga aylantirilishiga aytiladi. Ob`ekt kodni generasiyalash assembler tilidagi yoki bevosita mashina tilidagi natijaviy ob`ekt dasturni keltirib chiqaradi (mashina kodlaridagi). Dasturni ichki tasvirlanishi kompilyatorning amalga oshirilishidan bog'liq holda ixtiyoriy strukturaga ega bo'lishi mumkin, natijaviy dastur esa har doim komandalarning chiziqli ketma-ketligidan iborat bo'ladi. SHu sababli ob`ekt kodini generasiyasi (ob`ekt dasturni) ixtiyoriy holatda murakkab sintaksis strukturalarni chiziqli zanjirlarga aylantirish xarakatlarini bajarishi shart.

Kodni generasiyalashni sintaksis daraxtda va identifikatorlar jadvalida saqlanuvchi ma`lumotdan aniqlangan funksiya deb hisoblash mumkin. SHu sababli, ob`ekt kodni generasiyasi, dasturning sintaksis tahlili bajarilgandan va kodni generasiyalashga tayyorgarlik bo'yicha barcha zarur xarakatlar amalga oshirilgandan so'ng, ya`ni funksiyalar va o'zgaruvchilarga manzillar fazosi tarqatilgan, boshlang'ich dasturning sintaksis konstruksiyalarida o'zgaruvchilar, konstantalar va funksiyalarning ismlari va toifalarini mosliklari o'rnatilgandan so'ng bajariladi. Kiruvchi dasturni generasiya bajaradigan komandalar ketma-ketligiga akslantirish xarakteri kiruvchi tildan, natijaviy dastur yo'naltirilgan hisoblash tizimining arxitekturasidan va albatta ob`ekt kodining istalgan sifatidan bog'liqdir.

Ideal holatda kompilyator barcha kiruvchi dastur matnini sintaksis tahlilini bajarishi, so'ngra uning semantik tahlilini amalga oshirishi, so'ngra kodni generasiyalashga tayyorgarlik ko'rishi va bevosita kodni generasiyalashi kerak. Ammo kompilyatorning bunday ish chizmasi amaliyotda hech qachon qo'llanilmaydi. Gap shundaki, umumiy holatda bironta ham semantik tahlilchi va hech bir kompilyator barcha kiruvchi dastur matnini to'liqligicha tahlil qilish va baholashga qodir emaslar. Semantik tahlilning formal usullari kiruvchi dasturlarning juda kam bo'laklari uchun qo'llaniladi. SHu sababli kompilyatorda barcha kiruvchi dastur asosida chiquvchi ekvivalent dasturni keltirib chiqarishning amaliy imkoniyati yo'q.

Qoida bo'yicha, kompilyator natijaviy kodni generasiyasini bosqichma-bosqich kiruvchi dasturning tugallangan sintaksis konstruksiyalari asosida bajaradi. Kompilyator kiruvchi dastur matnidan tugallangan sintaksis konstruksiyani ajratadi, u uchun natijaviy kodni bo'lagini tuzadi va uni chiquvchi dastur matniga joylaydi. So'ngra u keyingi sintaksis konstruksiyaga o'tadi. SHunday qilib jarayon barcha kiruvchi dastur qarab chiqilmagunicha davom etadi. Tahlil qilinuvchi tugallangan sintaksis konstruksiyalar sifatida operatorlar, operatorlar bloki, proseduralar va

funksiyalarning ifodalanishi kelishi mumkin. Anik tarkib kiruvchi til va kompilyatorning amalga oshirilishidan bog'liq bo'ladi.

Kiruvchi tilning har bir sintaksis konstruksiyasining ma`nosini (semantikasini) uning toifasidan aniqlash mumkin. Toifa esa kiruvchi tilning grammatikasi asosida sintaksis tahlilchi tomonidan aniqlanadi. Sintaksis konstruksiyalarning toifalarining misoli bo'lib takrorlash operatorlari, shart operatorlari, tanlov operatorlari va x.k. xizmat qilishi mumkin. Turli dasturlash tillari uchun sintaksis konstruksiyalarning bir xil toifalari xarakterlidir, bu holatda ular sintakis nuqtai nazaridan farqlanadilar (til grammatikasida berilgan), ammo o'xshash ma`noga ega bo'ladilar (semantika bilan aniqlangan). Sintaksis konstruksiyaning toifasidan bog'liq ravishda, ushbu sintaksis konstruksiyaga mos, natijaviy dastur kodining generasiyasi bajariladi. Turli kiruvchi dasturlash tillarining semantika nuqtai nazaridan o'xshash konstruksiyalari uchun yangi natijaviy kod tug'ilishi mumkin.

11.2.Sintaksis boshqaruvchi tarjima

Kiruvchi til sintaksis konstruksiyalari uchun kompilyator natijaviy dastur kodini qura olishi uchun, ko'pincha sintaksis- boshqariluvchi- tarjima (SB-tarjima) deb atalgan usuldan foydalaniladi. Sintaksis- boshqariluvchi- tarjima – bu sintaksis tahlil natijalari asosida dasturning natijaviy kodini keltirib chiqarish usulidir. Usulni tushunishni qulaylashtirish uchun, sintaksis tahlil natijasi sintaksis daraxt (yoki amallar daraxti) ko'rinishida ifodalangan deb hisoblashimiz mumkin, ammo aniq kompilyatorlarda bu har doim ham shunday emas.

Sintaksis-boshqaruvchi-tarjima bu natijaviy dasturni kodi tug'ilishining sintaksis tahlil natijalari asosidagi asosiy usulidir.

Sintaksis boshqaruvchi tarjimaning maqsadi quyidagichadir: N sintaksis tahlil daraxtining har bir cho'qqisi bilan C(N) qandaydir oraliq kodni zanjiri bog'langan. N cho'qqi uchun kod aniq bir fiksirlangan tartibdagi ketma-ketlik C(N) kodiga, N ga to'g'ridan-to'g'ri avlod hisoblangan barcha cho'qqilar bilan bog'langan ketma-ketlikni zanjirlash (konkatenasiya) yo'li bilan quriladi.

O'z navbatida N cho'qqini to'g'ridan-to'g'ri avlodlarini kodlari ketma-ketligini qurish uchun, uning avlodlari uchun kodlar ketma-ketligini topish talab etiladi- N cho'qqini ikkinchi darajali avlodlarini topish –va x.k.

O'girish pastdan yuqoriga daraxt strukturasida aniqlangan, qat`iy o'rnatilgan tartibda ketadi.

Berilgan sintaksis tahlilning daraxti bo'yicha sintaksis –boshqariluvchi – tarjimani qurish uchun daraxt ildizi uchun kodlar ketma-ketligini topish zarur. SHu sababli daraxtning, zanjirini keltirib chiqaruvchi, har bir cho'qqisi uchun kodni shunday tanlash kerakki, ushbu daraxt ildiziga yozilgan kod daraxt ko'rinishida ifodalangan barcha operator uchun boshlang'ich bo'lib hisoblansin. Umumiy holda C(N) kodni, barcha, N cho'qqi qatnashgan, holatlarda uchraydigan, birko'rinishli interpretasiyasiga ega bo'lish zarur. Bu masala haqiqatda echish mumkin bo'lmagan masala, chunki daraxtning har bir cho'qqisini ma`nosini (semantikasini) baholashni talab etadi. Sintaksis –boshqariluvchi-tarjimani qo'llagan holda kodni, daraxtning har

bir cho`qqisi uchun, interpretasiyasi masalasi faqat kompilyatorni ishlab chiqaruvchisi tomonidan echiladi.

Kompilyatorning ba`zi bir modellarida kiruvchi dastur sintaksis tahlili va natijaviy dastur kodini generasiyasi bitta fazaga birlashtirilgan variantlari ham bo'lishi mumkin. Bunday modelni kompilyatorda kodni generasiyalash amallari va sintaksis tahlilni amallari birgalikda bajariladigan ko'rinishda tasavvur qilish mumkin.

Agar kompilyatorda sintaksis tahlil fazalari va kodni generasiyalash birga birlashtirilgan bo'lsa, u holda bunday kompilyatorlar uchun sintaksis boshqaruvchi –kompilyasiya termini ishlatiladi. Sintaksis boshqaruvchi -kompilyatorni barcha kiruvchi tillar uchun qo'llab bo'lmaydi, lekin sintaksis boshqaruvchi tarjimani barchasiga qo'llash mumkin.

Ammo, sintaksis –boshqaruvchi –kompilyasiya asosida ko'pgina keng tarqalgan KO-tillar sinflari uchun, xususan LR- va LL-tillar uchun tarjimani qurish mumkinligi ma`lum.

Sintaksis –boshqaruvchi-tarjima va sintaksis-boshqaruvchi-kompilyasiya jarayonlarida faqat chiquvchi tilning matni keltirib chiqarilib qolmay, balki kompilyator tomonidan bajariladigan ba`zi bir qo'shimcha xarakatlar ham amalga oshiriladi. Umumiy holda sintaksis –boshqaruvchi-tarjima chizmalari quyidagi xarakatlarni bajarishni ko'zda tutadilar:

- chiquvchi ma`lumotlar oqimiga, kompilyator ishining natijasini (chiqishini) ifodalovchi, mashina kodlarini yoki assembler komandalarini joylashtirish;
- foydalanuvchiga topilgan xatoliklar va ogohlantirishlar haqida ma`lumotlar berish (natijaviy dastur uchun foydalaniladigan oqimdan farqli bo'lgan chiquvchi oqimga joylashtiriladigan ma`lumotlar);
- ba`zi bir komandalarning kompilyator tomonidan bajarilishi kerakligini ko'rsatuvchi komandalarni keltirib chiqarish va bajarish (masalan, identifikatorlar jadvalida joylashgan ma`lumotlar ustida bajariladigan amallar).

Quyida dasturlarni kompilyatorda ichki tasvirlash usullarini qarab chiqiladi.

11.3.Dasturlarni ichki tasvirlash usullari

Kompilyatorda boshlang'ich dasturni sintaksis konstruksiyalarni ichki tasvirlashning turli usullari mavjud. Sintaksis tahlil bosqichida ko'pincha chiqish daraxti deb atalgan ko'rinishdan foydalaniladi (uning qurish usullari yuqorida ko'rilgan edi). Lekin sintaksis tahlil bosqichlarida foydalaniladigan ko'rinishlar ob`ekt kodini generasiyalash va optimallash ishlarida noqulaylik tug'diradilar. SHuning uchun optimallashdan va bevosita generasiyalashdan avval dasturlarning ichki ko'rinishlari mos yozuvlardan biriga aylantirilishi mumkin.

Dasturlarni ichki tasvirlashlarning barcha ko'rinishlari o'zida ikkita tamoyil jihatidan turli bo'lgan operatorlar va operandlardan tashkil topadilar. Ichki tasvirlash ko'rinishlarining orasidagi farqlari bu operatorlar va operandlarning o'zaro bog'lanishlari bilan bog'liqdir. Xuddi shuningdek, operatorlar va operandlar bir-

birlaridan, agar ular ixtiyoriy tartibda uchrasalar, farq qilishlari shart. Operator va operandlarning farqlanishlari uchun avval aytib o'tganimizdek, kiruvchi til semantikasini bo'yicha ishlovchi kompilyatorni ishlab chiqaruvchisi javob beradi. Dasturlarni ichki tasvirlashning quyidagi ko'rinishlari ma`lum:

- sintaksis daraxtni ifodalovchi bog'langan ro'yxatli strukturalar;
- ko'pmanzilli aniq nomlangan natijali kod (tetradalar);
- ko'pmanzilli aniq ismga ega bo'lmagan natijali kod (triadalar);
- amallarning teskari (postfiks) polyak yozuvlari;
- assembler kodi va mashina komandalari.

Har bir aniq kompilyatorda ushbu, ishlab chiqaruvchi tomonidan tanlangan, ko'rinishlardan biri foydalanilishi mumkin. Lekin ko'pincha kompilyator dasturlarning faqat bitta ichki tasvirlash ko'rinishidan foydalanish bilan chegaralanib qolmaydi. Kompilyasiyaning turli fazalarida, o'tishlarni bajarilishi jarayonida biri biriga aylantiriluvchi, turli ko'rinishli formalardan foydalanilishi mumkin. YUqorida sanab o'tilgan ko'rinishlardan barchasidan ham zamonaviy kompilyatorlarda keng foydalanilmaydi. Ba`zi bir, natijaviy kodni sezilmas darajada optimallashtiruvchi, kompilyatorlar, boshlang'ich dastur matni tahlili bo'yicha ob`ekt kodini generasiyalaydilar. Bu holatda, kompilyatorning sintaksis tahlili, semantik tahlili, ob`ekt kodini generasiyalashga tayyorgarlik va generasiyalash bir o'tishda bajarilishi birlashtirilganda sintaksis-boshqaruvchi-kompilyasiya chizmasi qo'llaniladi. U holda, dasturlarni ichki tasvirlash faqat tahlil algoritmining ketma-ketlik qadamlari ko'rinishida shartli mavjud bo'ladi. Ixtiyoriy holatda, kompilyator har doim dasturlarni tasvirlashning mashina komandalari ko'rinishi bilan ishlaydi- aks holda u natijaviy dasturni qura olmaydi.

Quyida, barcha sanab o'tilgan ko'rinishlar alohida to'liq ko'rib chiqiladi.

Sintaksis daraxtlar

Sintaksis daraxtlar avval ko'rib o'tilgan edi. Sintaksis daraxt – bu struktura sintaksis tahlilchini ishi natijasini ifodalaydi. U kiruvchi til konstruksiyasi sintaksisini aks ettiradi va amallarning to'liq o'zaroaloqasini o'zida jamlaydi. Ko'rinib turibdiki, sintaksis daraxtlar – bu dasturlarni ichki ifodalashning mashinaga bog'liq bo'lmagan ko'rinishidir.

Sintaksis daraxtlarning kamchiligi shundaki, murakkab bog'langan strukturaga ega bo'lganliklari sababli, ular trivial ko'rinishda natijaviy dasturning chiziqli ketma-ketligiga aylantirila olmaydilar.

SHunga qaramay, ular dasturning ichki ifodalashlari bilan, natijaviy dasturning komandalariga bevosita murojat etish zaruriyati bo'lmagan bosqichlarda, ishlashda qulaydirlar.

Sintaksis daraxtlar dasturlarni ichki ifodalashning boshqa, o'zida chiziqli ro'yxatlarni, kiruvchi til semantikasini hisobga olgan holda, ifodalovchi, ko'rinishlariga aylantirilishlari mumkin. Bunday toifadagi algoritmlar quyida ko'rib chiqiladi. Bu aylantirishlar sintaksis-boshqariluvchi-tarjima tamoyillari asosida bajariladi.

Ochiq nomlangan natijali ko'pmanzilli kod (tetradalar)

Tetradalar to'rtta tarkibiy qismdan iborat amallar yozuvini ifodalaydilar: amal, ikkita operand va amal natijasi. Masalan, tetradalar quyidagicha ifodalanishi mumkin:

<amal>(<1operand>,<2operand>,<natija>).

Tetradalar chiziqli komandalar ketma-ketligi bo'lib, biri biridan keyin hisoblanadilar. Har bir tetrada ketma-ketlikda quyidagicha hisoblanadi: tetrada bilan berilgan amal operandlar ustida bajariladi va natija tetrada bilan berilgan o'zgaruvchiga joylanadi. Agar operandlardan qaysidir biri (yoki ikkalasi ham) yo'q bo'lsa (masalan, agar tetrada unar amalni ifodalasa), u holda u yoki tushirib qoldirilishi yoki bo'sh operand bilan o'zgartirilishi mumkin (qabul qilingan yozuv ko'rinishidan bog'liq holda). Tetradaning hisoblash natijasi hech qachon tushirib qoldirilishi mumkin emas, aks holda tetrada ma`nosini yo'qotadi.

Tetradalarning hisoblash tartibini, faqat bu tartibni maqsadli o'zgartiruvchi tetradalar mavjud bo'lsagina, o'zgartirish mumkin (masalan, qandaydir shartlarda bir necha qadam oldinga yoki orqaga o'tishni chaqiruvchi tetradalar).

Tetradalar chiziqli bog'liq bo'lganliklari sababli, ular uchun tetradalar ketma-ketligini komandalar ketma-ketligiga aylantiradigan trivial algoritmni yozish etarlidir. Ularning sintaksis daraxtlardan ustunligi ham ana shundadir. Assembler komandalaridan farqli ravishda tetradalar, natijaviy dastur yo'naltirilgan, hisoblash tizimining arxitekturasiga bog'liq emas. SHu sababli ular dasturlarni ichki ifodalashning mashinaga bog'liq bo'lmagan ko'rinishini ifodalaydilar.

Tetradalar triadalarga nisbatan, o'zlarini ifodalash uchun, ko'proq xotira talab etadilar, ular yana amallar orasidagi o'zaroaloqani aniq ifodalamaydilar.

Bulardan tashqari yana tetradalarni mashina kodlariga aylantirishda bir qancha qiyinchiliklar mavjud, chunki ular assembler komandalari va mashina kodlariga yomon aks etadilar, yana ko'pgina zamonaviy kompyuterlarning komandalar to'plamida trek operandli amallar kam uchraydi.

Masalan, tetrada ko'rinishida yozilgan A:=B*C+D-B*10 ifoda quyidagi ko'rinishga ega bo'ladi:

1. * (B, C, T1)
2. + (T1,D, T2)
3. * (B, 10,TZ)
4. - (T2,TZ,T4)
5. := (T4,0, A)

Bu erda barcha amallar mos belgilar bilan belgilangan (bu erda o'zlashtirish ham amal hisoblanadi). T1,T2,T3,T4 identifikatorlar, tetradalarning hisoblash natijalarini saqlovchi, vaqtinchalik o'zgaruvchilarni ifodalaydilar. SHuni e`tiborga olish kerakki, oxirgi, faqat bitta operandni talab qiluvchi, tetradada (o'zlashtirish), ikkinchi operand sifatida qiymatga ega bo'lmagan nol kelmoqda.

Ko'pgina kompilyatorlar dasturning ob`ekt kodini qurish uchun optimallash uchun qulay bo'lgan ichki formalardan foydalanadilar. Masalan: X:= Y+Z gap quyidagi to'rtlik orqali ifodalanadi.

(PLUS_OP, Sy, Sz,Sx), bu ifoda Sy belgilar jadvali bilan yacheykada aniqlangan o'zgaruvchini Sz yacheykada aniqlangan o'zgaruvchi bilan (PLUS_OP) qo'shib va natijani Sx yacheykada saqlashni anglatadi. Endi boshlang'ich gap mustaqil birlik sifatida ifodalanadi, uni kod generatori joylashish manzilidan qatьi nazar qayta ishlay oladi. SHunday qilib, optimallovchi amallar ketma-ketligini kodni generasiyalash jarayonini murakkablashtirmasdan o'zgartirishi mumkin.

Unar operatorlar uchun to'rtlikning ikkinchi operandini maydonini e`tiborga olmaslik mumkin, ikkitadan ortiq operandlarni talab qiladigan amallarni esa bir necha to'rtliklardan tashkil topgan ketma-ketliklar ko'rinishida ifodalash mumkin.

Masalan, quyidagi operatorni X:= F(A,B,C,D) uchta to'rtlik ko'rinishidagi guruh sifatida yozish mumkin.

(F1,A,B, T1)
(F2, T1,C, T2)
(F, T2,D,X)

F1 va F2 funksiyalar oraliq hisoblashlarni amalga oshiradilar, T1 va T2 yacheykalar esa ushbu xarakatlarning natijalarini saqlash uchun mo'ljallangan. Dasturni faktik qurish vaqtida kod generatori ob`ekt kodida F1 (bu amal uchun F2 va F amallar orqali) amalni to'g'ri ifodalashi mumkin.

Ochiq nomlanmagan natijali ko'pmanzilli kod (triadalar)

<amal>(<1operand><2operand>)

Triadalar 3 holatli amallar yozuvlarini ifodalaydilar.

Triadalar komandalarning chiziqli ketma –ketligini ifodalaydilar. Triada ko'rinishida yozilgan ifodalarni hisoblashda, ular biri biridan keyin ketma-ket bajariladilar. Ketma-ketlikdagi har bir triada quyidagicha hisoblanadi: triada bilan berilgan amal operandlar ustida bajariladi, agar operandlarning biri sifatida (yoki ikkala operand) boshqa triadaga ko'rsatkich berilsa, u holda o'sha triadaning hisoblash natijasi olinadi. Triadalarning xususiyatlari shuki, bir yoki ikkala operand boshqa triadaga ko'rsatkich bo'lishi mumkin. Tetradalardan farqli ravishda bu erda xotirani tarqatilishiga javob beruvchi algoritmni bo'lishi talab etiladi (hisoblashlarni oraliq natijalarini saqlovchi). Triadalar o'zini tasvirlash uchun kamroq xotira talab etadilar, ular o'zlari o'rtasidagi amallarning o'zaro aloqalarini ochiq ifodalaydilar.

Triadaning hisoblash natijalarini vaqtinchalik xotirada saqlash kerak, chunki bu natija keyingi triada tomonidan talab etilishi mumkin. Agar operandlardan qaysidir biri yo'q bo'lsa (masalan, agar triada unar amalni ifodalasa), u holda u

yoki tushirib qoldirilishi yoki bo'sh operand bilan o'zgartirilishi mumkin (qabul qilingan yozuv ko'rinishidan bog'liq holda).

Triadalarning hisoblash tartibini, xuddi tetradalar kabi, faqat bu tartibni maqsadli o'zgartiruvchi triadalar mavjud bo'lsagina, o'zgartirish mumkin (masalan, qandaydir shartlarda bir necha qadam oldinga yoki orqaga o'tishni chaqiruvchi triadalar).

Triadalar komandalarning chiziqli ketma-ketligini tashkil etib, ular uchun triadalarni ketma-ketligini natijaviy dastur komandalari ketma-ketligiga aylantirish algoritmini yozish etarlidir. Bu esa ularning sintaksis daraxtlardan ustunligini anglatadi. Bu erda maxsus xotirani tarqatishga javobgar va hisoblashlarni oraliq natijalarini saqlovchi algoritmdan foydalanish zarur, chunki vaqtinchalik o'zgaruvchilardan bu maqsadlarda foydalanilmaydi. Xuddi shu tetrada va triadalarning farqidir.

Tetradalar kabi triadalar ham, natijaviy dasturlar yo'naltirilgan, hisoblash tizimining arxitekturasidan bog'liq emaslar. SHu sababli, ular dasturlarni ichki tasvirlashning mashinaga- bog'liq bo'lmagan ko'rinishini ifodalaydilar.

Triadalar o'zini ifodalash uchun, tetradalarga qaraganda, kam xotira talab etadilar, amallarni o'zaro aloqasini aniq ifodalaydilar va bu ularning qo'llanilishini qulaylashtiradi.

Maxsus xotirani tarqatishga javobgar va hisoblashlarni oraliq natijalarini saqlovchi algoritmdan foydalanish zaruriyati kamchilik hisoblanmaydi, chunki natijalarni faqatgina mavjud vaqtinchalik xotira yacheykalariga joylashtirish emas, balki prosessor registrlariga ham joylashtirish qulaydir.

Bu esa ba`zi bir ustunliklarni yaratadi. Triadalar, tetradalarga qaraganda, ikkimanzilli mashina komandalariga yaqinroq bo'lib, xuddi shu komandalar zamonaviy kompyuterlar komandalari to'plamida keng tarqalgan.

Masalan, triada ko'rinishida yozilgan ushbu ifoda A:=B*C+D-B*10 quyidagi ko'rinishga ega bo'ladi:

1. * (B, C)
2. + (^1,D)
3. * (B, 10)
4. - (^2,^3)
5. := (A,^4)

Bu erda amallar mos belgi bilan belgilangan (bu erda o'zlashtirish ham amal hisoblanadi), ^ belgi esa bir triada operandini boshqa triada natijasiga ko'rsatishini bildiradi.

Amallarning teskari polyak yozuvi

Teskari (postfiks) polyak yozuvi — ifodalarni hisoblash uchun amallar va operandlarini yozishning juda qulay usulidir. Bu ko'rinish amal ishoralari belgilari operandlardan keyin yozilishini ko'zda tutadi. Teskari polyak yozuvi to'liq keyinchalik ko'rib chiqiladi.

Assembler kodi va mashina komandalari.

Mashina komandalari shu bilan qulayki, foydalanishda dasturning ichki tasvirlanishi to'liq ob`ekt kodiga mos keladi va murakkab aylantirishlar talab etilmaydi.

Assembler komandalari bu mashina komandalarining yozilish ko'rinishidir, shuning uchun dasturlarni ichki tasvirlash usullari ulardan ko'p farq qilmaydilar. Lekin, dasturlarni ichki tasvirlash uchun, assembler komandalaridan yoki mashina komandalaridan foydalanish, amallarni o'zaroxarakatlarini aks ettirish uchun, qo'shimcha strukturalarni talab etadi. Ko'rinib turibdiki, bu holatda dasturlarni ichki tasvirlash natijaviy kod yo'naltirilgan hisoblash tizimining arxitekturasidan bog'liq bo'ladi. Demak, kompilyatorni boshqa natijaviy kodga mo'ljallantirishda dasturlarni ichki tasvirlash ko'rinishini ham, uning qayta ishlash usullarini ham qayta qurish talab etiladi (triada va tetradalardan foydalanishda bu talab etilmaydi).

Mashina komandalari — bu natijaviy dastur yozilishi kerak bo'lgan tildir. SHuning uchun kompilyator ular bilan ishlashi shart. Bundan tashqari, faqat qayta ishlanuvchi mashina komandalari (yoki ularning assembler komandalari ko'rinishidagi ifodasi), yordamida natijaviy dasturni foydaliroq ishlashiga erishish mumkin. Bundan kelib chiqadiki, ixtiyoriy kompilyator natijaviy dasturning mashina komandalari ko'rinishidagi ifodasi bilan ishlaydi, lekin ularning qayta ishlanishi kodni generasiyalash fazasining tugatish bosqichlarida amalga oshiriladi.

11.4.Postfiks yozuv

Teskari polyak yozuvi — bu amallarning postfiks yozilishidir. U polyak matematik olimi YA. Lukashevich tomonidan ixtiro qilingan va shu sababli uning nomi bilan ataladi.

Kundalik amaliyotda uchraydigan qavslarni o'z ichiga olgan oddiy arifmetik ifodalarni infeks ifodalar deb ataladi, chunki amal ishorasi operandlar orasida joylashadi. Xarakatlarning bajarilish tartibi bunday ifodalarda amallarning kattalik darajasi va qavslar bilan belgilanadi. Bunday ifodalarni hisoblash va kompilyasiyalash ularni avvaldan amallarni bajarilish tartibini aniqlash maqsadida tahlil qilishni anglatadi. Arifmetik ifodalarni qavslarsiz yozish usullari mavjudki, ularda xarakatlar tartibi ifodadagi amal belgilari tartibi bilan beriladi. YOzuvlarning bunday ko'rinishini polyak yozuvlari yoki qavssiz yozuvlar deb ataladi. Bu amallarning postfiks yozilishidir. Amallarning infeks yozuviga nisbatan, polyak yozuvi operandlari o'z tartibi bo'yicha, amal ishoralari esa qat`iy bajarilish tartibida keladi. Hisoblash uchun stekdan foydalaniladi. Kamchiligi: Biz stekning yuqorisi bilangina ishlay olishimiz sababli, ifodalarni optimallashning iloji yo'q. Ifodalar chapdan o'ngga qarab chiqiladi va elementlar quyidagi qoida bo'yicha qayta ishlanadi:

Algoritm:

1. Agar operand belgisi uchrasa, u holda u stekka joylanadi (stekning yuqorisi bo'lib hisoblanadi)

2. Agar unar amal belgisi uchrasa, u holda bitta operand stekdan tortib chiqariladi, amal bajariladi va natija stekning yuqorisiga joylashadi.

3. Agar binar amal belgisi uchrasa, u holda ikkita operand stekdan tortib chiqariladi, amal bajariladi va natija stekning yuqorisiga joylashadi.

Polyak yozuvi prefiks bo'lishi mumkin, bu holda amal ishorasi operandlardan avval keladi, va postfiks bo'lishi mumkin, bu holda amal ishorasi operandlardan keyin keladi. Bunday qavssiz ifodalarni hisoblash va kompilyasiyalash qavsli ifodalarga qaraganda soddaroqdir, chunki amallar ifodalanish tartibida bajariladi va avvaldan tahlil talab etilmaydi.

Prefiks polyak yozuvi (PrPZ) quyidagicha aniqlanadi.

Agar infiks ifoda E bitta a operanddan iborat bo'lsa, u holda PrPZ ifoda E bu a.

Agar infiks ifoda E1*E2, bu erda * amal belgisi, E1 va E2 operandlar uchun infiks ifoda, u holda ushbu PrPz *E1'E2' , bu erda E1',E2' – E1 va E2 ifodalarning PrPZ idir.

Agar (E) infiks ifoda bo'lsa, ushbu ifodaning PrPZi PrPz E dir.

Sanab o'tilgan qoidalar berilgan infiks ifodaning PrPZni qurish tartibini aniqlaydilar. Masalan, (a+b)*(s-d) ifoda uchun PrPZ ni quyidagicha qurish mumkin.

Birinchi bajariluvchi amalning operandlarini aniqlaymiz.

E1= (a+b) va E2 = (s-d).

Aniqlashlarga ko'ra prefiks yozuv E1*E2 bu *E1'E2' bu erda E1',E2' – E1 va E2 ifodalarning PrPZ idir. Bu ifodalar uchun prefiks yozuvlarni qurishni bajaramiz.

E1' = +ab, E2'= -sd va natijada quyidagi ko'rinishni olamiz:

*+ab-sd

Postfiks yozuvda esa amal ishoralari operandlardan keyin qo'yilishi bilan farqlanadi. Masalan, infiks yozuv (A+V) ga postfiks yozuv AV+ mos keladi.

Postfiks polyak yozuvi (PoPZ) quyidagicha aniqlanadi.

Agar infiks yozuv E bitta a operanddan iborat bo'lsa, u holda E ni PoPZ a bo'ladi.

Agar infiks ifoda E1*E2, bu erda * amal belgisi, E1 va E2 operandlar uchun infiks ifoda, u holda ushbu PoPz E1'E2'* , bu erda E1',E2' – E1 va E2 ifodalarning PoPZ idir.

Agar (E) infiks ifoda bo'lsa, ushbu ifodaning PoPZi PoPz E dir.

Xuddi yuqoridagidek misolga PoPZni ko'ramiz.

(a+b)*(s-d)

Tashqi amallarning operandlarini belgilaymiz.

E1= (a+b) va E2 = (s-d).
Bularning postfiks yozuvlarini topamiz:
E1' = ab+, E2'= sd-
Ifodaga qo'yib E1'E2'* va natijada ab+sd-* ga ega bo'lamiz.

Ifodalarning postfiks yozuvi ikkita kuchli xususiyatga ega bo'lib, ularni shuning uchun ham kompilyatorlarda keng qo'llaniladi.

Ixtiyoriy ifodani yozish uchun qavslar shart emas, chunki amalda ishtirok etuvchi operator operanddan keyin keladi, va operandlarni ko'rsatishdagi noaniqlik bo'lmaydi. Masalan, (A+V)+S postfiks yozuvda AV+S+, A+(V+S) esa postfiks yozuvda AVS+ ko'rinishda yoziladi.

Navbatdagi operatorni o'qish vaqtida mos operandlar o'qib bo'lingan bo'ladilar va operator boshqa qo'shimcha berilganlarni o'qishsiz bajarilishi mumkin. YUqorida aytilganlar binar amallarga ham tegishli bo'lib ularni unar amallarga ham qo'llash qiyin emas. Unar operator (teskari ~) ni argumentdan keyin qo'yish etarlidir. Masalan infiks yozuv ~A A~ ko'rinishda bo'ladi, ~(A+V) ifoda esa AV+~ ko'rinishga ega bo'ladi.

YUqoridagi xususiyatlarga asosan postfiks ko'rinishdagi ifoda quyidagi oddiy algoritm orqali hisoblanishi mumkin.

```
while(lex!=NULL)      // ifodada leksemalar qolmagunicha
{
  lex= getNextLex( );   // keyingi leksemani o'qish
  if(isOperand(lex))    // leksema, agar operand bo'lsa
    push(lex);          // leksemani stekka yozish
  if(isOperator(lex))   // leksema, agar operator bo'lsa
push( performOperation(lex, pop( ), pop( ) ));

/* oxirgi stekka yozilgan leksemalar ko'rsatayotgan amallarni bajarish va
ushbu elementlarni amal natijasi bilan almashtirish; */

}
```

Hisoblashlar natijasida ifodaning qiymati stekning yagona elementi bo'lib qoladi. Bunday stek bilan amallar bajaruvchi algoritm ixtiyoriy kompilyatorning asosini tashkil etadi. Ko'pincha sintaksis tahlil bloki dasturni postfiks ko'rinishga aylantiradi, undan keyin kod generatori ob'ekt dasturni yuqoridagi usul bilan ifodalarni qarab chiqqan holda ko'radi. Ko'rinib turibdiki, postfiks yozuvlarni hisoblash qiyinchilik tug'dirmaydi, lekin infiks yozuvlarni postfiksga aylantirish murakkabroq ish. Faraz qilaylik til mahsuloti quyidagi ko'rinishga ega bo'lsin.

$A \rightarrow V \; i \; S$

Bu erda A,V va S noterminal belgilardir, i esa terminal belgidir. Amal ishoralari terminal belgilar deb hisoblaymiz. U holda bu mahsulot noterminal A infiks yozuv ekanligini va unda V va S «i» operator bilan birgalikda qatnashayotganligini bildiradi. Xuddi shunday ifodaning postfiks ko'rinishi ham V va S operandlardan tashkil etiladi va ulardan so'ng «i» operatori keladi.

SHunday qilib, bunday mahsulot uchun postfiks yozuv quyidagi ko'rinishga ega bo'ladi.

V uchun postfiks yozuv
S uchun postfiks yozuv
va

Agar barcha terminal belgilar chiquvchi faylga shu tartibda chiqarilsalar, u holda barcha tilning frazalari postfiks ko'rinishida ifodalanadilar.

11.5.Infiks ko'rinishdagi ifodani postfiks ko'rinishdagi ifodaga aylantirish

Qavslarga olingan ifodaning qismlari ushbu ifodaning eng ichki qavslari darajasida, avval postfiks ko'rinishiga, ularni bitta operand sifatida qarash uchun, aylantirilishlari kerak. Bunday yo'nalishda aylantirishda ifodaning barchasidan qavslarni yo'qotilishiga kelinadi. Guruhdagi oxirgi ochilgan qavslar juftligi ushbu guruhdagi aylantirilgan ifodani o'z ichiga oladi. «Oxirgi ochilgan ifoda birinchi hisoblanadi» tamoyili stekdan foydalanishni ko'zda tutadi.

Infiks ko'rinishdagi quyidagi ikkita ifodani qaraymiz:

A+V*S va (A+V)*S va ularga mos postfiks ko'rinishlari quyidagicha bo'ladi: AVS*+ va AV+S*. Ushbu ko'rinishlardagi xar bir holatda operandlarning kelish tartibi boshlang'ich ifodadagi operandlarning kelish tartibiga mos tushadi. Birinchi ifodani qaraganimizda (A+V)*S) birinchi operand A tezda postfiks ifodaga joylashtirilishi mumkin. Ko'rinib turibdiki, «+» belgi ushbu ifodaga ikkinchi operand joylashmasdan turib joylasha olmaydi. Demak, u (ya`ni «+» belgi) saqlab qo'yilishi, va keyinchalik olinib kerakli joyga joylashtirilishi kerak. V operandni qaraganimizdan so'ng ushbu operand A operanddan so'ng joylashtiriladi. Ushbu vaqtga kelib qarab chiqilganlar ikkitani tashkil etadi. «+» belgini olish va postfiks yozuvga joylashtirishga hali vaqtli, chunki undan keyin yuqoriroq ustunlikka ega bo'lgan «*» belgi keladi. Ikkinchi ifodada qavslarni borligi «+» belgini birinchi bo'lib bajarilishiga sabab bo'ladi.

SHunday qilib, infiks ko'rinishni postfiks ko'rinishga aylantirishda ustunlik qoidalari asosiy rol o'ynaydi, ularni hisobga olish uchun quyidagi funksiyani kiritamiz:

rrecedence (oper1,oper2), bu erda oper1 va oper2 amallarni anglatuvchi belgilar. Ushbu funksiya agar oper1 oper2 ga nisbatan yuqori yoki teng ustunlikka ega bo'lsa TPUE qiymat qaytaradi. Aks holda rrecedence (oper1,oper2) FALSE qaytaradi. Masalan, rrecedence («*»,»+») va rrecedence («+»,»+») -xaqiqat, rrecedence («+»,»*»)- yolg'on. Teng ustunlikning tagiga chizilgan, chunki postfiks yozuvning noto`g`ri tashkil etilishi teng ustunlikka ega amallar uchun tipik xatoliklarning biri hisoblanadi.

Endi infiks ko'rinishda ifodalangan va qavslari bo'lmagan qatorni postfiks qatorga aylantirish algoritmini maketini qaraymiz. Kiruvchi qatorda qavslar yo'q

deb hisoblaganimiz uchun, amallarning bajarilish tartibini belgilash yagona ularning ustunlik darajalariga bog'liqdir.

```
1. (Postfiks qatorga « » ni o'rnatamiz);
2. (opstk ismli stekni tozalash);
3. while ( kirishda hali belgilar mavjud) {
4.       read(symb);
5.       if(isOperand(symb)= =TRUE) { // belgi bu -operand
6.          ( postfiks qatorga belgini qo'shish);
7.        } else  {  // belgi bu - amal
8.  while(empty(stack0= =FALSE)&&
   (procedence(stacktop(opstk), symb) = = TRUE)) {
                smbtp = pop(opstk);
```

/* smbtp symb ga nisbatan katta ustunlikka ega, shuning uchun u postfiks qatorga joylanishi mumkin. */

```
                (smbtp ni postfiks qatorga qo'shish);
9.  } // end while
```

/* ushbu nuqtada yoki opstk bo'sh, yoki symb stacktop(opstk) dan katta ustunlikka ega. symb ni postfiks qatorga keyingi amal (balki ustunroq bo'lgan) o'qilmagunicha qo'shib bo'lmaydi. SHunday qilib, symb ni saqlashimiz kerak*/

```
10.push(opstk,symb);
11.} // end if
12.} // end while
```

/*ushbu vaqtda qator butunlay qarab bo'lingan bo'ladi. Stekda qolgan boshqa amallarni postfiks qatorga joylashtirish kerak bo'ladi*/

```
13. while ( empty(opstk) = = FALSE) {
smbtp = pop(opstk);
14.(smbtp ni postfiks qatorga qo'shish);
15.} // end while
```

Qavslar bilan ishlash imkoniyatini ta`minlash uchun ochilgan qavs o'qilgandan so'ng stekka joylanadi. Bu quyidagi precedence(op,"(") =FALSE qoidani amalning ixtiyoriy yopilgan qavsdan tashqari belgisi uchun o'rnatish orqali bajariladi. Xuddi shunday, precedence("(", or) =FALSE chap qavsdan keyin keluvchi amal belgisi stekka yozilishi uchun qoidani o'rnatish kerak.

YOpilgan qavsni o'qilgandan keyin barcha amallar stekdan o'qilishi va postfiks qatorga joylashtirilishi kerak. Bu esa precedence(op,"(") =TRUE qoidani barcha chap qavsdan tashqari op amallar uchun o'rnatiladi. Ushbu amallarni stekdan o'qish va ochilgan qavsni yopish uchun quyidagi amalni bajarish kerak bo'ladi. Ochilgan qavs stekdan olib tashlanadi va yopilgan qavs bilan birgalikda tashlab yuboriladi. Ikkala qavs so'ngra , stekka xam postfiks qatorga ham joylashtirilmaydi. precedence("(", ")") =FALSE funksiyani o'rnatamiz. Bu esa

bizga yopilgan qavs kelgan vaqtdagi 8 qatordan boshlanuvchi takrorlash tashlab ketilishi, ochilgan qavs esa postfiks qatorga joylanmasligini kafolatlaydi.

Bajarilish esa 12 qatordan davom etadi. Lekin, yopilgan qavs stekka joylanmasligi uchun 12 qator quyidagi operator bilan almashtiriladi:

```
if((empty(opstk)= = TRUE || (symb < > ")")
push(opstk, symb);
else smbtp(opstk);
```

YUqorida kelishilganidek, precedence funksiyasi uchun va 12 qatordagi o'zgartirishlar uchun qarab chiqilgan algoritm ixtiyoriy qatorni infiks ko'rinishdan postfiks ko'rinishga aylantirish uchun foydalanilishi mumkin.

Qavslarni ustunliklari qoidalarini umumlashtiramiz:
precedence("(", or) = =FALSE ixtiyoriy or amal uchun
precedence(or,")") = =FALSE ")" qavsdan tashqari barcha amal uchun
precedence(op,")") = =FALSE "(" qavsdan tashqari barcha amal uchun
precedence(")", or) ==”noanik” ixtiyoriy or amal uchun (ikkita ko'rsatilgan amalni bajarishga urinish xatolik hisoblanadi).

Misol 1. A+V*S Postfiks qatorning symb qiymati va opstk har bir belgini qarab chiqilgandan so'ng keltiriladi, opstk cho'qqisi o'ngda (14-jadval) joylashadi.

14-jadval

Qator	Symb	Postfiks qator	opstk
1	A	A	
2	+	A	+
3	V	AV	+
4	*	AV	+*
5	S	AVS	+*
6		AVS*	+
7		AVS*+	

1,3,5 qatorlar operand belgisini (symb) postfiks qatorga tezda joylanishiga mos keladi.

2 qatorda esa amal keladi, stek esa bo'sh, amal stekka joylanadi. 4 qatorda * belgisining ustunligi stek cho'qqisida joylashgan + ga qaraganda yuqoriroq, shuning uchun yangi belgi stekka joylanadi. 6 va 7 qadamlarda kiruvchi qator bo'sh, shuning uchun stekdan elementlar o'qiladi va postfiks qatorga joylanadi.

Misol 2.
(A+V)*S

15-jadval

Qator	Symb	Postfiks qator	opstk
1	(		(
2	A	A	)
3	+	A	(+
4	V	AV	(+
5	)	AV+	
6	*	AV+	*
8		AV+S*	

Ushbu misolda (15-jadval) o'ng qavs uchrashi bilan stekdan elementlar chap qavs uchragunicha olina boshlaydi, shundan so'ng ikkala qavslar tashlab yuboriladi. Amallarni bajarilish ustunliklarini o'zgartirish uchun qavslardan foydalanish ularning postfiks ko'rinishdagi joylashishlarining, 1 misoldagidan farqli ravishda, ketma-ketligiga keltiradi

Stekdan bu erda avval qaralgan amallarni saqlab turish uchun foydalaniladi. Agar joriy qaralayotgan amal stekning cho'qqisidagi amalga nisbatan yuqori ustunlikka ega bo'lsa, u holda bu yangi amal stekka yoziladi. Bu esa stekdan elementlarni oxirgi postfiks qatorga yozish uchun tanlovda ushbu yangi amal undan avval oldinda turgan amaldan oldinga joylashtiriladi (bu to'g'ri hisoblanadi, chunki u yuqoriroq ustunlikka ega). Boshqa tomondan, agar yangi amalning ustunligi stekdagi amaldan kichik yoki teng bo'lsa, u holda stek cho'qqisida joylashgan amal birinchi bajarilishi kerak. Demak, u stekdan olinadi va chiquvchi qatorga joylashtiriladi, joriy qaralayotgan belgi esa keyingi stek cho'qqisini egallagan element bilan taqqoslanadi va x.k. Kiruvchi qatorga qavslarni kiritib hisoblashlar ketma-ketligini o'zgartirish mumkin. CHap qavs uchraganda u stekka yoziladi. Unga mos o'ng qavs uchraganda barcha ushbu qavslar orasidagi amallar chiquvchi qatorga joylashtiriladi va ular ushbu qavslardan keyin keladigan barcha amallardan avval bajariladi.

11.6.Operatorlarning postfiks yozuvi (IF va boshqalar)

Ifodalar uchun postfiks yozuv anchagina sodda tashkil etiladi. Operatorlar uchun esa bu jarayon anchagina murakkabdir. Ushbu jarayoni IF operatori misolida ko'rib chiqamiz. Uning sintaksisi quyidagicha berilgan:

<IF_STMT> :: + IF <IFODA> THEN
<GAPLAR RO'YXATI >
{ELSE <GAPLAR RO'YXATI > }
END

Bu erda figurali qavslar metabelgilar bo'lib, bu ularning shart emasligini bildiradi. Ushbu gapni bajarilishi uchun ifodani baholash kerak, agar uning qiymati yolg'on bo'lsa, boshqaruvni ELSE (agar u bor bo'lsa) so'zidan keyin keluvchi operatorlar guruhiga, yoki gapni END so'zi bilan tugallanadigan oxiriga uzatish kerak. Lekin ELSE va END kalit so'zlari tahlilning ushbu bosqichida hali o'qilgani yo'q, shuning uchun biz boshqaruvni aniq qaerga uzatish kerakligini bilmaymiz.

Bu murakkablikni boshqaruvni uzatish nuqtasini aniqlash yo'li bilan hal qilish mumkin. Ushbu nuqtani biz tizimli belgi deb ataymiz va tahlilning joriy bosqichida ushbu belgini ismini ifodadan so'ng postfiks yozuvga joylashtiramiz. Qachonki dastur matnida ELSE kalit so'zi (yoki END) uchraganda xuddi shunday tizimli belgini mos postfiks yozuvga joylashtiramiz. SHunday qilib tizimli belgi tahlil qilinayotgan dasturning ikki joyida uchraydi: ifodada boshqaruvni uzatiladigan nuqtani ko'rsatish bajarilayotgan nuqtada, boshqaruvni uzatilayotgan nuqtada. Ushbu berilganlar yordamida kod generatori mos manzil aloqasini o'rnatadi.

THEN kalit so'zidan keyingi operatorlarni ro'yxatini tashlab avvalga o'tadigan buyruqni ko'rish mumkin, lekin END kalit so'zini qayta ishlash davomida ba`zi bir muammolar tug`ilishi mumkin. Agar IF gapida ELSE so'zi tushirib qoldirilgan bo'lsa, u holda boshqaruv END so'ziga mos nuqtaga uzatiladi. Lekin ELSE so'zi IF gapi tarkibiga kirgan bo'lsa, u holda, boshqaruv ushbu so'zga mos nuqtaga uzatiladi va THEN so'zidan keyin keluvchi operatorlardan END so'ziga mos nuqtaga shartsiz o'tish bajarilishi kerak. Ushbu hol THEN so'zidan keyin keluvchi operatorlarni bajarilishidan so'ng, ELSE so'zidan keyin kelgan operatorlarni tashlab o'tish imkonini beradi. Ikkala holatda ham, END so'zi gapda bitta buyruq orqali oldinga uzatishni nuqtasini ifodalaydi.

Ushbu holat tufayli, IF gapini amalga oshiruvchi ob`ekt kodini bo'laklarini bir biri bilan bog'lash imkoniyati tug'iladi.

YUqoridagilarni hisobga olgan holda, IF gapini postfiks yozuvini quyidagicha ifodalash mumkin:

<IFODA >
IF so'zi uchun oldinga o'tish belgisi
IF
<GAPLAR RO'YXATI >
{ Aylanib o'tish belgisi /* ELSE operatorlar ro'yxati uchun aylanib o'tishlar belgisi */
Oldinga o'tish belgisi
/* THEN operatorining ro'yxatini aylanib o'tishda boshqaruv uzatiladigan nuqta */
ELSE
<GAPLAR RO'YXATI >
ELSE operatorining aylanib o'tishlar ro'yxatini yoki boshqaruvni oldinga uzatish nuqtasini belgilaydigan tizimli belgi
END_IF

Bu erda IF binar operator hisoblanadi, u ifodaning haqiqiyligini tekshiradi va agar uning qiymati nulga (YOLGON) teng bo'lsa, THEN operatorlari ro'yxatini aylanib o'tadi, ELSE esa –binar operator bo'lib ELSE operatorlari ro'yxatiga o'tishlar buyruqlarini ko'radi va oldinga o'tish belgisini aniqlaydi. Xuddi shunday xarakatlarni END_IF yozuvi ham bajaradi.

11.7.Ifodalarni teskari polyak yozuviga o'girish uchun sintaksis – boshqaruvchi-kompilyasiya chizmasi

Ifodalarni infiks yozuv ko'rinishidan teskari polyak yozuvi ko'rinishiga aylantirishning ko'pgina algoritmlari mavjud. Quyida sintaksis –boshqaruvchi-kompilyasiya chizmasi asosida arifmetik ifodalarni +, -, * va / amallariga ega tili uchun algoritmi qaraladi. Algoritmning asosi sifatida arifmetik ifodalarning, ko'p marta misol sifatida qaralgan, grammatikasi tanlanadi, Bu grammatika quyida keltirilgan.

G({+,-,/,*,a,b}, {S,T,E}, P, S);

R

S -» S+t | S-t | t
T -» T*E | T/E | E E -> (S) | a | b

Sintaksis -boshqaruvchi-kompilyasiya chizmasini R chiquvchi belgilar zanjiri mavjud va ko'rsatkichning ushbu zanjirdagi holati r ma'lum deb faraz qilgan holda quramiz. Anglovchi, grammatika qoidalari bo'yicha alsternativalarni tanlab yoki hosil qilishni bajargan holda, chiquvchi zanjirga belgilarni yozishi mumkin va undagi ko'rsatkichning joriy holatini o'zgartirishi mumkin.

Bunday usul bilan qurilgan sintaksis -boshqaruvchi-kompilyasiya chizmasi arifmetik ifodalarni teskari polyak yozuvi ko'rinishiga aylantirish uchun juda soddadir. U quyida keltirilgan. Bu erda har bir grammatika qoidasi bilan, har bir qoidaning o'ng qismidan keyin keladigan, qandaydir ; (nuqta vergul) orqali yozilgan xarakatlar bog'liq. Agar hech qanday xarakatlarni amalga oshirish kerak bo'lmasa yozuvda bo'sh belgi *(X)* keladi.

S -> S+T; R(p) = "+", r=r+1
S -> S-T; R(p) = "-", r=r+1
S -> T; X
T -> T*E; R(p) = "*", r=r+1
T -> T/E; R(p) = V", r=r+1
T -> E; X
E -> (S); *X*
E -> a; R(p) - "a", r=r+1
E -> b; R(p) - "b", p=p+1

Ushbu sintaksis –boshqaruvchi-kompilyasiya chizmasini ixtiyoriy, berilgan grammatika asosida kiruvchi zanjirni tahlilini amalga oshiruvchi, qaytishsiz anglovchi uchun foydalanish mumkin. U holda sintaksis –boshqaruvchi-tarjima tamoyillariga mos ravishda, har safar qandaydir qoida asosida kodni hosil qilishni bajara turib, anglovchi ushbu qoida bilan bog'langan xarakatlarni ham bajaradi. Uning ishi natijasida boshlang'ich ifodani teskari polyak yozuvi ko'rinishidagi ifodasini o'zida saqlovchi R zanjir quriladi (boshqacha aytganda bu holatda avtomat faqat anglovchini emas, balki zanjirlarni aylantiruvchini ham ifodalaydi.

Keltirilgan sintaksis –boshqaruvchi-kompilyasiya chizmasi, bir tomondan ifodalarni teskari polyak yozuviga aylantirish qanchalik oson ish ekanligini ko'rsatsa, boshqa tomondan sintaksis –boshqaruvchi-kompilyasiya qanday ishlashini namoyish qiladi.

11.8.Sintaksis –boshqaruvchi tarjima chizmalari

YUqorida, natijaviy dasturni chiziqli komandalar ketma-ketligini olish yoki kompilyatorda, sintaksis tahlil natijalari asosida, dasturni ichki ifodalanishini imkonini beradigan, sintaksis-boshqaruvchi tarjima tamoyili ifodalangan edi. Endi, kirishida amallar daraxtini oladigan va u asosida natijaviy dasturning chiziqli bo'laklari uchun ob`ekt kodini yaratadigan, kodni generasiyalash algoritmini variantini ko'ramiz. Sintaksis-boshqaruvchi tarjima chizmasini arifmetik ifodalar uchun misolini qaraymiz. Bu chizmalar etarli darajada oddiy va ular asosida kompilyatorda kodni

generasiyalash vaqtida sintaksis-boshqaruvchi tarjima qanday bajarilishini ko'rsatish mumkin bo'ladi.

Sintaksis tahlil natijalarini ifodalash ko'rinishi sifatida amallar daraxtini olamiz (amallar daraxtini qurish usuli yuqorida ko'rib o'tilgan). Ob`ekt kodni ichki ifodalanishini qurish uchun (bundan buyon -kodni) amallar daraxti bo'yicha daraxtga murojaatni oddiygina rekursiv prosedurasidan foydalanish mumkin. Bu erda boshqa usullardan ham foydalanish mumkin bo'ladi, faqat daraxtda har doim pastda yotuvchi amallar yuqoridagi amallarga nisbatan oldin bajarilishi tamoyili buzilmasligi muhim (bir daraja amallarining tartibi muhim emas, bu holat natijaga ta`sir ko'rsatmaydi va daraxt cho'qqilarini tartibidan bog'liq bo'ladi).

Amallar daraxti bo'yicha kodni generasiyalash prosedurasi avvalombor, daraxt bog'lamini toifasini aniqlashi kerak. U, daraxt bog'lami hosil qilingan belgining, amal toifasiga mos keladi. Bog'lam toifasini aniqlagandan so'ng prosedura, daraxt bog'lami uchun, amal toifasiga mos, kodni quradi. Agar navbatdagi *r* darajaning joriy bog'lamining barcha bog'lamlari daraxt barglari bo'lsa, u holda kodga ushbu barglarga mos operandlar qo'shiladi, va tashkil etilgan kod proseduraning bajarilishi natijasi hisoblanadi. Aks holda prosedura rekursiv ravishda, daraxtning pastdagi bog'lamlari uchun kodni generasiyalash uchun, o'zini o'zi chaqirishi kerak va bajarilish natijasini tug'ilgan kodga qo'shishi kerak. Faktik ravishda, kodni generasiyalash prosedurasi daraxtning har bir bog'lami uchun, joriy bog'lam va pastdagi bog'lamlar bilan bog'langan, komandalar zanjirini konkatenasiyasini bajarishi shart. Zanjirlarni konkatenasiyasi pastdagi bog'lamlar bilan bog'langan amallar, joriy bog'lam bilan bog'liq amal bajarilgunicha bajarilishini nazarda tutadi.

SHu sababli ob`ekt kodini daraxt bo'yicha ichki ifodalanishini qurish uchun, birinchi navbatda, ob`ekt kodini ifodalashni, chiqish daraxtini joriy bog'lami ko'rinishiga mos to'rtta holat uchun ko'rinishini ishlab chiqarish zarur.

Sinov savollari

1.Kodni generasiyalash jarayoni haqida ma`lumot bering.

2. Kodni generasiyalashni sintaksis daraxtda va identifikatorlar jadvalida saqlanuvchi ma`lumotdan aniqlangan funksiya deb hisoblanishining sabablarini ayting.

3. Sintaksis-boshqaruvchi-tarjima nima?

4.Sintaksis boshqaruvchi -kompilyator nima?

5.Tetradalar nima?

6.Generasiya qilinayotgan kodni ichki ifodalanishida tetrada qanday bo'laklardan tashkil topadi?

7.Kodni generasiyalashda tetradani qo'llashning yutuqlari nimalarda ko'rinadi?

8.Triadalardan foydalanishning yutuq va kamchiliklari haqida ma`lumot bering.

9.Triadalardan foydalanishning yutuq va kamchiliklari haqida ma`lumot bering.

10. Assembler kodi va mashina komandalaridan nima uchun foydalaniladi?

11. Assembler kodi va mashina komandalaridan nima uchun foydalaniladi?

12.Ifodalarning infiks, postfiks va prefiks yozuvlarini farqlarini aytib bering.

13.Kompilyatorlarda foydalanilishi nuqtai nazaridan postfiks yozuvning ustunligi nimada?

14.Infiks yozuvlarni postfiks yozuvlarga aylantirishning formal usullari mavjudmi?

15.Postfiks yozuvida ifodalangan quyidagi ifodani a=2; v=4; qiymatlarda hisoblang (barcha o'zgaruvchilar va sonli konstantalar 1 uzunlikka ega).

1a+1a3--*v+

16.Ifodaning infiks yozuvini postfiks yozuviga aylantiring.

(a+(a-v))*(v+2-a)

12.BO'LIM
Kodni optimallash

Kodni optimallash. Kodni optimallashning asosiy usullari. Kodni optimallashning umumiy tamoyillari. Dasturning chiziqli bo'laklarini optimallash. Optimallashning mashinaga-bog'liqlik usullari

12.1.Kodni optimallash. Kodni optimallashning asosiy usullari Kodni optimallashning asosiy tamoyillari

YUqorida aytilganidek, ko'pgina hollarda kodni generasiyalash kompilyator tomonidan kiruvchi dasturning barchasi uchun to'liq holda bajarilmaydi, balki uning alohida konstruksiyalari uchun ketma-ketlikda bajariladi. Kiruvchi dasturning har bir sintaksis konstruksiyasi uchun olingan natijaviy kodni bo'laklari ham ketma-ket ravishda umumiy natijaviy kodni dasturiga birlashtiriladi. Bu holatda natijaviy kodning turli bo'laklar orasidagi aloqalari to'liq ravishda e`tiborga olinmaydi.

O'z navbatida, kiruvchi tilning turli sintaksis konstruksiyalari uchun natijaviy kodni qurishda sintaksis –boshqaruvchi-tarjima usulidan foydalaniladi. U yana daraxt strukturasi bo'yicha qurilgan kod zanjirlarini, ularning o'zaro aloqalarini hisobga olmagan holda, birlashtiradilar. Bunday yo'l bilan qurilgan natijaviy dasturning kodi ortiqcha komandalar va ma`lumotlarga ega bo'lishi mumkin. Bu esa natijaviy dastur bajarilish foydaliligini kamaytiradi. Haqiqatda kompilyator shu bilan kodni generasiyalashni tugatishi mumkin, chunki natijaviy dastur qurilgan va u kiruvchi til ma`nosi (semantikasi) bo'yicha yozilgan dasturga ma`no jihatidan ekvivalent hisoblanadi. Ammo natijaviy dasturni foydaliligi uni ishlab chiqaruvchisi uchun juda muhim, shu sababli ko'pgina zamonaviy kompilyatorlar kompilyasiyaning yana bir bosqichini bajaradilar, ya`ni natijaviy dasturni optimallashni, dasturni foydaliligini mumkin qadar oshirish maqsadida. Quyidagi ikki muhim holatni ta`kidlash kerak: birinchidan, kodni generasiyalash alohida bosqichida optimallashni ajratish — bu majburiy qadam. Kompilyator qurilgan kodni optimallashni bajarishga majbur, chunki u kiruvchi dastur matnining barchasini to'liq semantik tahlilini bir vaqtda amalga oshirish va uni ma`nosini baholash va bulardan kelib chiqqan holda natijaviy dasturni qurishni bajara olmaydi. Optimallash, natijaviy dastur birdaniga qurilmay, bosqichlar bo'yicha qurilishi sababli ham, zarur. Ikkinchidan, optimallash —bu kompilyasiyaning majburiy bo`lmagan bo'lagidir. Kompilyator umuman optimallashni bajarmasligi va natijaviy dastur, bu holda ham, to'g'ri bo'lishi, kompilyatorning o'zi esa funksiyalarini to'liq bajarishi mumkin. Lekin amaliyotda barcha kompilyatorlar u yoki bu holda optimallashni bajaradilar, chunki ularni ishlab chiqaruvchilari dasturiy ta`minotni ishlab chiqarish vositalari bozorida yaxshi o'rinni egallashga urinadilar. Bu erda esa optimallash, natijaviy dasturning foydalilik xususiyatiga sezilarli ta`sir etgan holda, muhim bo'lgan faktor hisoblanadi.

Endi «optimallash» tushunchasiga ta`rif beramiz.

Dasturlarni optimallash bu foydaliroq natijali ob`ekt dastur olish maqsadida kompьyuter dasturidagi amallarning o'zgartirish va tartibga solish bilan bog'liq qayta ishlashdir. Optimallash, kod generasiyasini tayyorlash fazasi bo'yicha va

kodni generasiyalash fazasi bo'yicha, bir necha marta bajarilishi mumkin, Natijaviy dasturnning foydalilik ko'rsatkichi bo'lib quyidagi ikki kriteriylardan foydalaniladi:

1)natijaviy dasturning bajarilishi uchun zarur bo'lgan xotira hajmi (uning barcha ma`lumotlari va kodini saqlash uchun) va

2) dasturning bajarilish tezligi.

Har doim ham ushbu ikki kriteriylarni qanoatlantiruvchi optimallashni bajarish mumkin bo'lmaydi.

Ko'pincha dasturdagi ma`lumotlar hajmini zaruriy kamaytirilishi, dasturning bajarilish tezligini kamayishiga olib keladi yoki aksincha. SHu sababli optimallash uchun ko'pincha yuqorida aytilgan kriteriylardan biri tanlanadi. Optimallash kriteriysini tanlashni ko'pincha foydalanuvchi bevosita kompilyatorni sozlash jarayonida bajaradi.

Ammo, optimallash kriteriysini tanlangan holda ham, umumiy holda, eng qisqa va eng tez ishlaydigan kiruvchi dasturga mos, natijaviy dastur kodini qurish mumkin bo'lmaydi. Gap shundaki, berilgan kiruvchi dastur uchun ekvivalent bo'lgan, eng qisqa va eng tez, natijaviy dastur kodini qurish uchun algoritmik usulni topish mumkin emas. Bu masala amalda echimga ega bo'lmagan masaladir. Kiruvchi ma`lumotlarning juda katta mumkin bo'lgan soni uchun, juda ko'p marta tezlashtirish mumkin bo'lgan algoritmlar mavjud, lekin bu algoritmlar boshqa kiruvchi ma`lumotlar to'plami uchun ular optimal bo'lmasliklari mumkin. Bundan tashqari kompilyator barcha kiruvchi dasturning semantik tahlili uchun kerak bo'ladigan chegaralangan vositalarga ega. Optimallash bosqichida bajarilishi mumkin bo'lgan ish - bu berilgan dastur ustida ushbu dasturni foydalilik xususiyatini oshirish maqsadida aylantirishlar ketma-ketligini bajarishdan iboratdir.

U yoki bu kompilyator yordamida olingan natijaviy dasturni foydaliligini baholash uchun, ko'pincha uni unga ekvivalent dasturga, ya`ni boshlang'ich dasturdan olingan va assembler tilida yozilgan, dasturga taqqoslash amalga oshiriladi (usha algoritmni amalga oshiruvchi dastur). Eng yaxshi optimallash kompilyatorlari, yuqori daraja dasturlash tilida yozilgan, sifati jihatidan assembler tilida yozilgan dasturlardan qolishmaydigan, murakkab boshlang'ich dasturlardan natijaviy ob`ekt dasturlarni olishlari mumkin. Ko'pincha yuqori daraja dasturlash tillarida yozilgan dasturlarni foydalilik xususiyatlarini, assembler yordamida tuzilgan dasturlarga moslik darajasi 1,1-1,3 ga teng. YA`ni yuqori daraja dasturlash tili kompilyatorida qurilgan ob`ekt dastur, assembler tili yordamida qurilgan unga ekvivalent bo'lgan ob`ekt dasturga nisbatan ko'pincha 10-30 % ko'p, komandalarga ega bo'ladi va 10-30 % sekinrok ishlaydi.

Bu esa, assembler tilida va yuqori daraja dasturlash tilida dasturlarni ishlab chiqarish uchun sarflanadigan xarajatlarni taqqoslaganda, juda xam yomon natijalar emas.

Optimallashni kodni generasiyalashning, sintaksis tahlilni tugatishidan boshlab va oxirgi natijaviy dastur kodini tug'ilish jarayonigacha, ixtiyoriy bosqichida bajarish

mumkin. Agar kompilyator dasturlarni tasvirlashning bir nechta turli ko`rinishlaridan foydalanayotgan bo'lsa, u holda ularning har biri optimallashtirilishi mumkin, bu erda ichki tasvirlashning har bir ko'rinishi turli optimallash usullariga mo'ljallangan. SHunday qilib, optimallash kompilyatorda kodni generasiyalash bosqichida bir necha marta bajarilishi mumkin.

Aylantirishlarni optimallashni ikki asosiy ko'rinishini farqlaydilar:

1) kiruvchi dastur matnini natijaviy ob`ekt tiliga bog'liq bo'lmagan holda uning ichki tasvirlanishini ko'rinishida aylantirish. Ushbu berilgan aylantirishlar maqsadli hisoblash tizimining arxitekturasidan bog'liq emas. Ular avvaldan yaxshi tanish bo'lgan matematik va mantiqiy aylantirishlarga asoslangan.

2)natijaviy ob`ekt dasturni aylantirish. Ushbu guruh aylantirishlari maqsadli hisoblash tizimining arxitekturasidan bog'liq.

Masalan, optimallashda xesh-xotira hajmi va markaziy prosessorni konveyer amallarini tashkil etish usullarini e`tiborga olinishi mumkin. Ko'pgina hollarda bu aylantirishlar kompilyatorning amalga oshirilishidan qattiq bog'liq bo'ladi va kompilyatorni ishlab chiqaruvchilari uchun «nou-xau» hisoblanadi. Xuddi ana shu toifa optimallashtirish aylantirishlari natijaviy kodni foydaliligini sezilarli oshirish imkonini beradilar. Foydalanilayotgan optimallash usullari hech qachon, hech qanday sharoitda ham boshlang'ich dastur ma`nosiga o'zgartirish kiritmasliklari shart (ya`ni dasturni optimallashdan so'ng dasturni natijasi o'zgarmasligi kerak).

Birinchi ko'rinish aylantirishlari uchun ko'pincha muammolar tug'ilmaydi. Ikkinchi ko'rinish aylantirishlari uchun ba`zi murakkabliklar kelib chiqishi mumkin, chunki kompilyatorni ishlab chiqaruvchilari foydalanadigan barcha optimallash usullarining hammasi ham, nazariy nuqtai nazardan asoslangan va barcha mumkin bo'lgan boshlang'ich dasturlar ko'rinishlari uchun isbotlangan bo'ladilar. Xuddi ana aylantirishlar boshlang'ich dastur ma`nosiga ta`sir ko'rsatishlari mumkin. SHu sababli, ko'pgina kompilyatorlar optimallash usullarini ba`zilarini o'chirish imkoniyatini ko'zda tutadilar. Ba`zi hollarda optimallash, dastur ma`nosini dasturni ishlab chiqaruvchisi ko'zda tutgan ko'rinishda emas boshqa ko'rinishga olib keladi, lekin bu kompilyatorning optimallash qismidagi xatoliklar tufayli emas, balki foydalanuvchining optimallash bilan bog'liq ba`zi bir aspektlarni e`tiborga olmaganligi sababli yuz beradi. Masalan, kompilyator dasturdan avvaldan ma`lum natijaga ega qandaydir funksiyani chaqirishni olib tashlashi mumkin, lekin agar bu funksiya «nojuya samaraga» ega bo'lsa, ya`ni global xotirada qandaydir qiymatlarni o'zgartirsa, dasturning ma`nosi o'zgarishi mumkin. Bu boshlang'ich dastur matnining yomon dasturlanganligidan darak beradi. Bunday xatoliklarni topish anchagina qiyinchiliklar tug'diradi, ularni topish uchun dasturning, dastur ishlab chiqaruvchisiga, semantik tahlilchi tomonidan berilgan, ogohlantirishlariga e`tibor berish, yoki optimallashni o'chirish kerak bo'ladi. Optimallashni boshlang'ich dastur matnini qayta ishlash jarayonida qo'llash ham maqsadga muvofiq emas.

Zamonaviy kompilyatorlarda faqat umumiy optimallash kriteriylarini tanlash imkoniyati emas, balki optimallashdan foydalanishda qo'llaniladigan ba`zi bir alohida usullarni ham tanlash imkoniyati mavjud.

Dasturlarni aylantirish usullari kiruvchi til sintaksis konstruksiyalarining toifalaridan bog'liq bo'ladi. Amaliyotda dasturlash tillarining ko'pgina konstruksiyalari uchun optimallash usullari ishlab chiqilgan.

Optimallash quyidagi sintaksis konstruksiyalar uchun bajarilishi mumkin:

1.dasturning chiziqli bo'laklari;

2.mantiqiy ifodalar;

3.takrorlashlar;

4.prosedura va fuksiyalarning chaqiriqlari;

5.kiruvchi tilning boshqa konstruksiyalari.

Barcha hollarda optimallashning mashinaga –bog'liq usullar kabi, mashinaga-bog'liq bo'lmagan usullardan ham foydalanilishi mumkin.

Keyinchalik optimallashning faqat ba`zi bir mashinaga- bog'liq usullari, birinchi navbatda, dasturlarning chiziqli bo'laklariga tegishli usullar, qaraladi. Bu erda ushbu usullarning faqat umumiy ko'rinishi qaraladi.

12.2.Dasturlarning chiziqli bo'laklarini optimallash. CHiziqli bo'laklarni optimallash tamoyillari

Dasturning chiziqli bo'lagi — bu amallarning ketma-ketlik tartibida bajariluvchi amallar bo'lib, ular bitta kirish va bitta chiqishga ega. Ko'pincha chiziqli bo'lak arifmetik amallar va o'zgaruvchilarga qiymat o'zlashtiruvchi operatorlardan tashkil topgan hisoblashlar ketma-ketligini o'z ichiga oladi.

Ixtiyoriy dastur hisoblashlarni bajarish va qiymatlarni o'zlashtirishni ko'zda tutadi, shu sababli, chiziqli bo'laklar ixtiyoriy dasturda uchraydilar. Aniq dasturlarda ular dastur kodining sezilarli bo'lagini tashkil etadilar. SHu sababli chiziqli bo'laklar uchun kodni optimallash usullarini keng spektri ishlab chiqarilgan.

Bundan tashqari, ixtiyoriy chiziqli bo'lakning xarakterli xususiyati bo'lib uning tarkibiga kiruvchi amallarning bajarilish ketma-ketligi tartibi hisoblanadi. Dasturning chiziqli bo'lagining tarkibidagi hech bir amal tushirib qoldirilishi mumkin emas, biron bir amal o'ziga qo'shni amaldan ko'p marta bajarilishi mumkin emas (aks holda bu dasturning bo'lagi chiziqli bo'lak bo'lmay qoladi). Bu esa dasturning chiziqli bo'laklarini optimallash masalasini sezilarli ravishda soddalashtiradi. CHiziqli bo'lak amallarining barchasi ketma-ketlikda bajarilishi sababli, ularni bajarilish tartibida raqamlash mumkin.

Dasturning chiziqli bo'laklarini tashkil etuvchi amallar uchun quyidagi ko'rinishdagi optimallash aylantirishlari qo'llanilishi mumkin:

- foydasiz o'zlashtirishlarni yo'qotish;
- ortiqcha hisoblashlarni olib tashlash (ortiqcha amallarni);
- ob`ekt kodini amallarini bajarish;
- amallarni o'rin almashtirish;
- arifmetik aylantirishlar.

Foydasiz o'zlashtirishlarni yo'qotish – bu agar dasturning chiziqli bo'lagi tarkibida qandaydir ixtiyoriy i raqamli A o'zgaruvchiga qiymat o'zlashtirish amali mavjud bo'lsa, xuddi shuningdek j raqamli o'sha A o'zgaruvchiga qiymat o'zlashtirish amali mavjud bo'lsa, i<j bo'lib, va i va j o'rtasidagi biron bir amalda A o'zgaruvchini qiymatidan foydalanilmasa, u holda i raqamli o'zlashtirish amali foydasiz hisoblanadi. Foydasiz qiymat o'zlashtirish amali o'zgaruvchiga qiymat beradi, lekin bu qiymatdan hech qaerda foydalanilmaydi. Bunday amal dastur ma'nosiga zarar etmagan holda olib tashlanishi mumkin. Umumiy holda, foydasiz bo'lib faqat o'zlashtirish amali emas, balki chiziqli bo'lakning ixtiyoriy boshqa, natijalari hech qaerda ishlatilmaydigan, amallari ham bo'lishi mumkin.

Masalan, quyidagi dastur bo'lagidagi

A :=B*S;

D:= B+C;

A:=D*C;

A: =B*C; o'zlashtirish amali foydasiz bo'lib, uni dastur matnidan olib tashlash mumkin. Bu erda o'zlashtirish amali bilan birgalikda, natijada foydasiz bo'lgan, ko'paytirish amali ham olib tashlanishi mumkin.

Foydasiz o'zlashtirish amalini topish har doim ham yuqorida ko'rsatilgan misoldagi kabi oson kechmaydi. Agar chiziqli bo'lakdagi ikkita o'zlashtirish amalining ko'rsatkichlari (xotira manzili) ustida xarakatlar bajarishda yoki «nojuya samara»ga ega funksiyalarni chaqirishda yuzaga kelishi mumkin.

Masalan, quyidagi dastur bo'lagida

P:=@A;

A:= B*C;

D:= P^+C;

A:= D*C;

A:=B*C; o'zlashtirish amali endi foydasiz emas, hatto bu erda u aniq ko'rinib turmasa ham. Bu holatda, agar 1 raqamli qiymat o'zlashtirilgan o'zgaruvchi i va j amallari o'rtasida bir marta ham uchramasligi haqidagi oddiy tamoyilga rioya qilmaslik to'g'ri bo'ladi, bu holda i raqamli o'zlashtirish amali foydasiz hisoblanadi.

Har doim ham, o'zgaruvchiga o'zlashtirilgan qiymatdan foydalanilayaptimi yoki yo'qmi aniqlash mumkin bo'lmaydi. U holda ortiqcha o'zlashtirishlarni yo'qotish etarli darajada murakkab, amallarning xotira manzillari va ko'rsatkichlarini hisobga oluvchi, masala bo'lib qoladi.

Ortiqcha hisoblashlarni yo'qotish (ortiqcha amallarni) ob'ekt koddan bir xil operandlarni qayta ishlovchi amallarni topish va yo'qotishni o'z ichiga oladi. i tartib raqamli chiziqli bo'lak amali ortiqcha hisoblanadi, agar unga o'xshash j raqamli amal mavjud bo'lsa, j<i va ushbu amal qayta ishlovchi hech qanday operand, hech qanday 1 va j tartib raqamli amallar o'rtasidagi amal bilan, o'zgartirilmagan bo'lsa.

Ob'ekt kodni yaratish (svertka) —bu boshlang'ich dasturni kompilsiya jarayoni vaqtida, operandlarining qiymatlari ma'lum bo'lgan holda, amallarni bajarishidir. Ushbu holni trivial misoli bo'lib, barcha operandlari konstantalar bo'lgan ifodalarni

hisoblash hisoblanadi. Bundan murakkabrok variantlarda o'zgaruvchilar va funksiyalarning ma`lum qiymatlarini e`tiborga olish kerak bo'ladi.

Kompilyatorga har doim ham, xatto ifoda uni bajarilishiga imkon bersa ham, ob`ekt kodini yaratish mumkin bo'lmasligi mumkin. Masalan, A:=2*B*C*3; ifoda quyidagi ko'rinishga olib kelinishi mumkin A:=6*B*C; , lekin hisoblash tartibida A:=2*(B*(C*3)); buni ko'rish juda xam oson emas. Ob`ekt kodini yaratishning foydaliroq bajarilishiga erishish uchun, uni bajarilishini boshqa usulning bajarilishi bilan o'rin almashtirish imkoni mavjud –ya`ni amallarni o'rin almashtirilishi.

Dasturlashning yaxshi uslubi bo'lib, kompilyator ob`ekt kodini yaratishini osonlashtirish uchun, konstantalar ustida bajariladigan amallarni birlashtirish hisoblanadi. Masalan, agar, mos matematik konstantani ifodalovchi, P1=3.14 konstanta mavjud bo'lsa, u holda b=sin(2P1a); ifodani B:=sin(2*A*PI); ko'rinishda yozishdan ko'ra, B:=sin(2*PI*A); ko'rinishda yoki xatto B:=sin((2*PI)*A); ko'rinishda yozish ma`qul.

Amallarni o'rin almashtirish, dasturni foydaliligini oshirish maqsadida, oxirgi hisoblashlar natijaga ta`sir ko'rsatmaydigan, amallarni tartibini o'zgartirishdagn iboratdir.

Masalan, quyidagi ifodada ko'paytirish amalini A:=2*B*3*C; oxirgi natijani o'zgartirmagan holda o'rnini almashtirish mumkin va quyidagicha tartibda bajarish mumkin A:=(2*3)*(B*C);. U holda amallar sonini kamaytirish va ob`ekt kodini yaratish imkoni paydo bo'ladi.

Boshqa A:=(B+C)+(D+E); ifodada juda bo'lmaganda, oraliq natijani saqlab turish uchun, bitta xotira yacheykasini talab qilish kerak bo'ladi (yoki prosessorni registrini). Lekin uni quyidagi tartibda bajarishda A:=B+(C+(D+E)); bitta registr bilan ishlash mumkin bo'ladi, natija esa o'zgarmaydi.

Amallardagi o'rin almashtirishlar ko'pincha, ularning bajarilish tartibi maqsadli hisoblash tizimining arxitektura xususiyatlaridan bog'liq holda, dasturning foydaliligiga ta`sir ko'rsatadigan holatlarda, ahamiyat kasb etadi (prosessor registrlaridan foydalanish, ma`lumotlarni qayta ishlash konveyerlari va x.k.). Bu xususiyatlar keyinchalik optimallashning mashinaga bog'liq usullariga bag'ishlangan bo'limda to'liqroq qaraladi.

Arifmetik aylantirishlar mavjud algebraik va mantiqiy qonunlar asosida amallarning bajarilish tartibini va xarakter o'zgarishlarini o'zlarida ifodalaydilar.

Masalan, A:=B*C+B*D; ifoda quyidagi ifodaga almashtirilishi mumkin A:=B*(C+D);. Bu holda oxirgi natija o'zgarmay qoladi, lekin ob`ekt kodi bitta ko'paytirish amaliga kam amallarni o'zida jamlaydi.

Arifmetik aylantirishlarga yana, darajaga ko'tarish amalini ko'paytirish amaliga o'zgartirish; 2 ga butun bo'linuvchi butunsonli konstantalarni – surilish amallarini bajarish amaliga o'zgartirish kabi xarakatlarni misol sifatida keltirish mumkin. Ikkala holatda ham dasturni tezligini, murakkab amallarni soddaroq amallarga almashtirish orqali, oshirish mumkin.

Quyida ikki usul: ob`ekt kodini yaratish va ortiqcha amallarni olib tashlash usuli to'liq qaraladi.

12.3.Ob`ekt kodini hosil qilish

Ob`ekt kodini hosil qilish — bu boshlang'ich dasturni kompilsiya jarayoni vaqtida, operandlarining qiymatlari ma`lum bo'lgan holda, amallarni bajarishidir. Ushbu amallarni natijaviy dasturda ko'pmarta bajarishning zaruriyati yo'q — ularni bir marta, dasturni kompilyasiyasi jarayonida, bajarish etarlidir.

Ushbu holning oddiy misoli bo'lib, barcha operandlari konstantalar bo'lgan ifodalarni kompilyatorda bajarish hisoblanadi.

Qiymatlari boshqa amallarning bajarilishi natijasida ma`lum bo'ladigan amallarni aniqlash jarayoni murakkabroqdir. Bu maqsadlar uchun, chiziqli bo'laklarni optimallashtirishda, ob`ekt kodini hosil qilishning maxsus algoritmidan foydalaniladi. Ushbu algoritm maxsus, barcha o'zgaruvchilar uchun, qiymatlari ma`lum bo'lgan, (<o'zgaruvchi>,<konstanta>) juftligini o'zida saqlovchi T jadval bilan ishlaydi. Bundan tashqari, algoritm ob`ekt kodini hosil qilishi natijasida kodni generasiyalanishini talab qilmaydigan dasturni ichki tasvirlangan amallarini belgilaydi. CHunki ushbu algoritmni bajarilishida amallarning o'zaro aloqalari hisobga olinadi, uning uchun qulay tasvirlash ko'rinishi triadalar hisoblanadi, amallarning boshqa tasvirlash ko'rinishlarida (tetradalar yoki assembler komandalari kabi) esa, bir amallarning natijalarini boshqa amallarning operandlari bilan aloqalarini aks ettirish uchun, qo'shimcha strukturalar talab etiladi. Triadalar uchun ob`ekt kodini hosil qilish algoritmini bajarilishini qaraymiz. Ob`ekt kodini keltirib chiqarishni talab qilmaydigan amallarni belgilash uchun, maxsus ko'rinishdagi quyidagi triadadan foydalanamiz S(K,O).

Triadalarni ob`ekt kodini hosil qilish algoritmi triadaning chiziqli bo'laklarini ketma-ket qarab chiqadi va har bir triada uchun quyidagilarni bajaradi:

1. Agar operand, T jadvaldan olingan o'zgaruvchi bo'lsa, u holda operand konstantaning mos qiymatiga almashtiriladi.

2. Agar operand C(K,0) toifadagi triadaga ko'rsatkich bo'lsa, u holda operand K konstantaning qiymatiga almashtiriladi.

3. Agar triadaning barcha operandlari konstantalardan iborat bo'lsa, u holda triada yaratilishi mumkin. U holda ushbu triada bajariladi va uning o'rniga maxsus C(K,0) ko'rinishdagi triada joylashadi, bu erda K — yaratilgan triadaning bajarilish natijasi bo'lgan konstantadir. (Maxsus triada uchun kodni generasiyalashda ob`ekt kod keltirib chiqarilmaydi, shu sababli uni olib tashlash mumkin.)

4. Agar triada A:=B, ko'rinishdagi o'zlashtirish hisoblansa u holda:

- agar B — konstanta, u holda A konstantaning qiymati bilan T jadvalga kiritiladi (agar u erda A uchun eski qiymat mavjud bo'lsa, u holda eski qiymat yo'qotiladi);

- agar B — konstanta bo'lmasa, u holda A umuman T jadvaldan olib tashlanadi, agar u bor bo'lsa.

Algoritmni bajarilishiga misol qaraymiz.

Faraz qilaylik boshlang'ich dastur matni bo'lagi quyidagi ko'rinishga ega bo'lsin (Pascal toifasidagi tilda yozilgan):

I:=I+1;
I:=3:
J:=6*I+I;

Uning triada ko'rinishidagi ichki tasvirlanishi quyidagi ko'rinishga ega bo'ladi:

1. + (I,I)
2. := (I,^1)
3. := (I,3)
4. * (6, I)
5. + (^4, I)
6. := (J, ^5)

Ob`ekt kodini hosil qilish algoritmining bajarilish jarayoni 16–jadvalda ko'rsatilgan.

16- jadval. Algoritmning ishlash misoli

Triada	1-qadam	2-qadam	Z-qadam	4-qadam	5-qadam	6-qadam
1	C (2, 0)	C (2, 0)	C (2, 0)	C (2, 0)	C (2, 0)	C (2, 0)
2	:=(I,^1)	:= (I, 2)	:= (I, 2)	:= (I, 2)	:= (I, 2)	:= (I, 2)
3	:= (I, 3)	:= (I, 3)	:= (I, 3)	:= (I, 3)	:= (I, 3)	:= (I, 3)
4	* (6,I)	* (6,I)	* (6,I)	C (18, 0)	C (18, 0)	C (18, 0)
5	+ (^4, I)	+ (^4,I)	+ (^4, I)	+ (^4, I)	C(21,0) 1	C (21, 0)
6	:=(J, ^5)	:=(J, ^5)	:=(J, ^5)	:=(J,^5)	:=(J, ^5)	:=(I,21)
T	(,)	(I,2)	(I,3)	(I,3)	(I,3)	(I,3) (J,21)

Agar maxsus toifadagi C(K,O) ko'rinishli triadalarni olib tashlasak (ob`ekt kodini keltirib chiqarmaydigan), u holda hosil qilish natijasida triadalarning quyidagi ketma-ketligiga ega bo'lamiz:

1. := (I, 2)
2. := (I, 3)
3. := (J, 21)

Ob`ekt kodini hosil qilish algoritmi, dasturning chiziqli bo'laklaridan, kompilyasiya jarayonida natijalari ma`lum bo'lgan amallarni olib tashlash imkoniyatini beradi. Bular hisobiga esa dasturning bajarilish vaqti va natijaviy dastur kodi hajmini qisqartirish imkoniyatini yuzaga keladi.

Ob`ekt kodini hosil qilish amalda faqat dasturning chiziqli bo'laklari uchun emas, balki boshqa holatlarda ham bajarilishi mumkin. Operandlar sifatida konstantlar kelsalar, dasturning bajarilish mantig'i ahamiyatga ega emas, ob`ekt kodini hosil qilish ixtiyoriy holda bajarilishi mumkin. Agar o'zgaruvchilarni ma`lum qiymatlarini hisobga olish zarur bo'lsa, u holda natijaviy dastur bajarililshi mantig'ini ham e`tiborga olish kerak bo'ladi.

SHu sababli dasturning chiziqli bo'lmagan bo'laklari uchun (tarmoqlanish va takrorlashlar) algoritm, triadalarning chiziqli ro'yxatini qarab chiqishdan ko'ra, murakkabroq bo'ladi.

12.4.Ortiqcha amallarni olib tashlash

Ortiqcha amallarni olib tashlash algoritmi amallarni ularning kelib tushishi tartibida qaraydi. YAratish algoritmi kabi, ortiqcha amallarni olib tashlash algoritmiga ham triadalar bilan ishlash osonroqdir, chunki ular amallarning o'zaro aloqalarini to'liq aks ettira oladilar. Triadalar uchun ortiqcha amallarni olib tashlash algoritmini qaraymiz.

Triadalar va o'zgaruvchilarning ichki bog'liqligini tekshirib turish uchun, algoritm ularga ba`zi bir, bog'liqlik sonlari deb atalgan, qiymatlarni, quyidagi qoidalar bo'yicha, o'zlashtiradi:

- avvalda har bir o'zgaruvchi uchun uning bog'liqlik soni 0 ga teng, chunki dastur ishining boshlanishida o'zgaruvchining qiymati hech qanday triadadan bog'liq bo'lmaydi;
- i –chi, A o'zgaruvchiga qandaydir qiymat o'zlashtirilgan, triadani qayta ishlashdan so'ng, A (dep(A)) bog'liqlik soni i qiymatni oladi, chunki endi A ning qiymati berilgan i –chi triadadan bog'liq bo'ladi;
- i-chi triadani qayta ishlashda uning bog'liqlik soni (dep(i)) 1+(operandlar bog'liqligining maksimal soni) qiymatga teng deb qabul qilinadi.

SHunday qilib, triadalar va o'zgaruvchilar uchun bog'liqlik sonidan foydalanishda aytish mumkinki, agar i-chi triada j-chi triadaga (j<i) o'xshash bo'lsa, u holda i-chi triada, faqat dep(i)=dep(j) bo'lgan holda, ortiqcha hisoblanadi.

Ortiqcha amallarni olib tashlash algoritmi o'z ishida maxsus ko'rinishdagi SAME(j,0) triadasidan foydalanadi. Agar bunday triada i raqamli joyda uchrasa, u holda bu, kiruvchi triadalar ketma-ketligida qandaydir i triada j triadaga o'xshashligini bildiradi.

Ortiqcha amallarni olib tashlash algoritmi triadaning chiziqli bo'laklarini ketma-ket qarab chiqadi. U quyidagi. har bir triada uchun bajariladigan, qadamlardan tashkil topadi:

1. Agar triadaning qandaydir operandi maxsus ko'rinishdagi SAME (j, 0) triadaga ko'rsatsa, u holda, u j (^j) raqamli triadaga ko'rsatkichga almashtiriladi.

2. i raqamli joriy triadaning bog'liqlik soni, uning operandlari bog'liqlik sonidan kelib chiqib, hisoblanadi.

3. Agar triadalar ro'yxatining qarab chiqilgan bo'lagida o'xshash j-chi triada mavjud bo'lsa, bu erda j<i va dep(i)=dep(j) bo'lsa, u holda joriy i triada maxsus ko'rinishdagi SAME(j,0) triadaga almashtiriladi.

4. Agar joriy triada o'zlashtirish bo'lsa, u holda mos o'zgaruvchini bog'liqlik soni hisoblanadi.

Aniq misolda algoritm ishini qarab chiqamiz:

```
D:=D+C*B;
A:=D+C*B;
C:=D+C*B;
```

Dasturning ushbu bo'lagiga quyidagi triadalar ketak-ketligi mos keladi:

1. * (C, B)
2. + (D, ^1)
3. := (D, ^2)
4. * (C, B)
5. + (D, ^4)
6. := (A, ^5)
7. * (C, B)
8. + (D, ^7)
9. := (CS, ^8)

Ko'rinib turibdiki, ushbu misolda ba'zi bir amallar bir xil operandlar ustida ikki marta hisoblanmokda, demak, ular ortiqcha va ularni olib tashlash mumkin. Ortiqcha amallarni olib tashlash algoritmining ishi quyidagi jadvalda aks etadi.

17-jadval.Ortiqcha amallarni olib tashlash algoritmi ishi

i qayta ishlanuvchi triada	O'zgaruvchilarni bog'liqlik soni				Triadalarni bog'liqlik soni dep(i)	Algoritmni bajarilishi natijasida olingan triadalar
	A	B	C	D		
1) * (C, B)	0	0	0	0	1	1) * (C, B)
2) + (D, ^1)	0	0	0	0	2	2) + (D, ^1)
3) := (D, ^2)	0	0	0	3	3	3) := (D, ^2)
4) * (C, B)	0	0	0	3	1	4) SAME (1, 0)
5) + (D, ^4)	0	0	0	3	4	5) + (D, ^1)
6) := (A, ^5)	6	0	0	3	5	6) := (A, ^5)
7) * (C, B)	6	0	0	3	1	7) SAME (1, 0)
8) + (D, ^7)	6	0	0	3	4	8) SAME (5, 0)
9) := (C, ^8)	6	0	9	3	5	9) := (C, ^5)

Endi, agar SAME(j,0) ko'rinishdagi maxsus triadani olib tashlasak, u holda algoritmni bajarilishi natijasida quyidagi triadalar ketma-ketligini olamiz:

1. * (C, B)
2. + (D, 1)
3. := (D, ^2)
4. + (D, ^1)
5. := (A, ^4)
6. := (C, ^4)

Ahamiyat berib qarasak, natijaviy ketma-ketlikda triadalarning raqamlanishi va bir triadadagi boshqa triadaga yo'naltirish raqamlanishi o'zgardi. Agar kompilyatorda

yo'naltirish sifatida triadalarning raqamini emas, balki bevosita ularga ko'rsatkichlardan foydalansak, u holda ushbu variantdagi yo'naltirishlarga talab qolmaydi.

Ortiqcha amallarni olib tashlash algoritmi bir xil operandlar ustida bir xil amallarni takroriy ravishda bajarishdan qochish imkonini beradi.

Ushbu usul bo'yicha optimallash natijasida natijaviy dastur kodini hajmi va bajarish vaqti qisqaradi.

12.5.Dasturlarni optimallashning boshqa usullari. Mantiqiy ifodalarni hisoblashlarni optimallash

Mantiqiy ifodalarni optimallash xususiyatlari shundaki, har doim ham natijani bilish uchun ifodaning hisoblashlarini to'liq bajarish zarur emas. Gohida, birinchi amalning natijasi bo'yicha yoki birinchi operandning qiymati bo'yicha, barcha ifodaning hisoblash natijasini avvaldan aniqlash mumkin bo'ladi.

Amal operandning qandaydir qiymati uchun avvaldan aniqlangan deb ataladi, agar uning natijasi faqatgina shu operanddan bog'liq bo'lsa va boshqa operandlarning qiymatiga nisbatan (invariant) o'zgarmay qolsa.

(og) mantiqiy qo'shish amali quyidagi mantiqiy qiymat uchun «haqiqat» (true) avvaldan aniqlangan hisoblanadi, mantiqiy ko'paytirish amali esa — «yolg'on» (false) mantiqiy qiymat uchun avvaldan aniqlangandir.

Bu faktlar mantiqiy ifodalarni optimallash uchun foydalanilishi mumkin. Haqiqatan, mantiqiy qo'shish ketma-ketligida «haqiqat» qiymatini olib yoki mantiqiy ko'paytirish ketma-ketligida «yolg'on» qiymatini olib, keyingi hisoblashlarni amalga oshirish zaruriyati qolmaydi — chunki natija aniqlangan.

Masalan, quyidagi ifodani A og B og C or D hisoblashdan ma`no yo'q, agar A o'zgaruvchining qiymati True («haqiqat») ekanligi ma`lum bo'lsa.

Kompilyatorlar mantiqiy ifodalarni hisoblashning ob`ekt kodini shunday ko'radilarki, ifodalarni hisoblash, uni qiymati avvaldan aniqlanishi bilanoq, tugatiladi. Bu esa natijaviy dasturni bajarishdagi hisoblashlarni tezlashtirish imkonini yaratadi. Mantiqiy ifodalarni buleva algebrasi qonunlari asosidagi aylantirishlar va amallarni o'rin almashtirishlari orqali ushbu usulning foydaliligi anchagina ko'paytirilishi mumkin.

Bunday aylantirishlar har doim ham dastur ma`nosiga invariant emaslar. Masalan, quyidagi ifodani hisoblashda A or F(B) or G(C), agar A o'zgaruvchining qiymati true hisoblansa, F va G funksiyalar chaqirilmaydi va bajarilmaydi. Agar ushbu funksiyalarning natijalari faqat ular qaytaradigan qiymatlar bo'lsa va ular «nojuya samara»ga ega bo'lsalar, u holda dasturning semantikasi o'zgarishi mumkin.

Ko'pgina hollarda zamonaviy kompilyatorlar bunday toifadagi optimallashni olib tashlash imkoniyatiga ega bo'ladilar va u holda barcha mantiqiy ifodalar, natijaning avvaldan aniqlanishini hisobga olmay, oxirigacha hisoblanadilar.

Mantiqiy ifodalarni hisoblashning ushbu xususiyatini e`tiborga olish yaxshi uslub hisoblanadi. U holda mantiqiy ifodalardagi operandlardan birinchi navbatda ifodaning

barcha qiymatlarini aniqlaydiganlarini hisoblaydilar. Bundan tashqari, funksiyalarning qiymatini, ortiqcha murojaatlardan qochish uchun mantiqiy ifodaning boshida emas, oxirida hisoblash yaxshiroq. SHunday qilib, yuqorida ko'rilgan ifodani quyidagi ko'rinishda yozish va hisoblash A or F(B) or G(C) yaxshiroq, F(V) or G(C) or A dan ko'ra.

Avvaldan aniqlangan natijaga faqatgina mantiqiy ifodalargina ega bo'lmaydilar. Ba`zi bir matematik amallar va funksiyalar ham bu xususiyatga ega bo'lishlari mumkin. Masalan, 0 ga ko'paytirishni bajarishdan ma`no yo'q.

Bunday aylantirishlar toifasining boshqa misoli – bu invariant operandlar uchun hisoblashalrni olib tashlashdir.

Operandning ba`zi bir qiymatiga nisbatan amal invariant deb ataladi, agar uning natijasi ushbu operanddan bog'liq bo'lmasa va boshqa operandlar orqali aniqlansa.

Masalan, (or) mantiqiy qo'shish «yolg'on» (False) qiymatiga nisbatan invariant, mantiqiy ko'paytirish (and)— «haqiqat» qiymati uchun invariant; algebraik qo'shish 0 ga nisbatan invariant, algebraik ko'paytirish 1 ga nisbatan invariant.

Bunday amallarning bajarilishini, agar ularga invariant operand ma`lum bo'lsa, dastur matnidan olib tashlash mumkin.

12.6.Proseduralar va funksiyalarga parametrlarni uzatishni optimallash

YUqorida proseduralar va funksiyalarga parametrlarni uzatishni steklar orqali amalga oshirilishi qaralgan edi. Ushbu usul ko'pgina dasturlash tillariga qo'llaniladi va deyarli barcha maqsadli hisoblash tizimlarning turli arxitekturali zamonaviy kompilyatorlarida foydalaniladi (natijaviy dastur bajariladigan).

Ushbu usul amalda qo'llanish usun juda qulay va ko'pgina arxitekturalarda yaxshi apparat qo'llanmasiga ega. Ammo u, agar prosedura va funksiyalar kam parametrlar ustida murakkab bo'lmagan amallarni bajargan taqdirida, foydasizdir. U holda har doim prosedura va funksiyalarni chaqirishda kompilyator stekda uning faktik parametrlarini joylashtirish uchun ob`ekt kodni tashkil etadi, undan chiqishda esa, parametrlar bilan band, yacheykalarni bo'shatish uchun kodni tashkil etadi. Natijaviy dasturni bajarilishida bu kod harakatlanadi. Agar prosedura yoki funksiya katta bo'lmagan hajmdagi hisoblashlarni bajarsalar, stekda parametrlarni joylashtirish uchun kod va ularga murojaat, ushbu prosedura va funksiyalar uchun tashkil etilgan kodning sezilarli qismini tashkil etishlari mumkin. Agar bu funksiya ko'p marta chaqirilsa, u holda bu kod uning bajarilishini foydaliligini sezilarli ravishda kamaytiradi. Natijaviy dasturni foydaliligini, kod xarajatlarini kamaytirish va parametrlarni prosedura va funksiyalarga uzatishni bajarilish vaqtini qisqartirish hisobiga, oshiradigan usullar mavjud:

- Parametrlarni prosessor registrlari orqali uzatish;
- CHaqiruvchi ob`ekt kodiga funksiyalar kodini qo'yish.

Parametrlarni prosessor registrlari orqali uzatish usuli prosedura va funksiyalarga uzatilayotgan barcha parametrlarni yoki ularning bir qismini, stekda emas, bevosita

prosessor registrlarida joylashtirish imkoniyatini beradi. Bu esa funksiya paraemtrlarini qayta ishlashni tezlashtirish imkonini yaratadi, chunki prosessor registrlari bilan ishlash har doim ham, stek joylashgan operativ xotira yacheykalariga nisbatan, tezrok amalga oshiriladi. Agar barcha parametrlarni registrlarga joylashtirishga erishilsa, u holda kodning hajmi ham kamayadi, chunki stekga parametrlarni joylashtirishdagi amallar ham kerak bo'lmaydi.

Biz bilamizki, prosessorning, parametrlarni prosedura va funksiyalarga uzatishda foydalaniladigan, registrlari, ushbu prosedura va funksiyaning ichida, hisoblashlar uchun, to'g'ridan-to'g'ri xarakatlana olmaydilar. CHunki prosessorning dasturiy murojaat etilishi mumkin bo'lgan registrlari chegaralangan sonlidir, shu sababli, murakkab hisoblashlarni bajarishda taqsimlash muammolari paydo bo'lishi mumkin. U holda kompilyator tanlashi kerak bo'ladi: yoki registrlardan parametrlarni uzatish uchun foydalanish va hisoblash foydaliligini kamaytirish (oraliq natijalarni bir qismini, operativ xotirada yoki stekda joylashtirish kerak bo'ladi), yoki ozod registrlardan hisoblashlar uchun foydalanib va parametrlarni uzatish foydaliligini kamaytirish. SHu sababli ushbu usulni qo'llanilishi prosessorning maqsadli hisoblash tizimidagi dasturiy murojaat mumkin bo'lgan registrlari sonidan va kompilyatordagi registrlarni taqsimlash algoritmidan bog'liq bo'ladi. Zamonaviy prosessorlarda dasturiy murojaat mumkin bo'lgan registrlari soni har doim, yangi va yangi ishlab chiqarish variantlari hisobiga ko'payib bormoqda, shuning uchun ushbu usulning qiymati ham oshib bormoqda.

Bu usul bir qancha kamchiliklarga ega. Birinchidan, ko'rinib turibdiki, u maqsadli hisoblash tizimining arxitekturasidan bog'liq bo'ladi. Ikkinchidan, ushbu usul bilan optimallashtirilgan prosedura va funksiyalardan hech qanday ko'rinishda kutubxona proseduralari va funksiyalari sifatida foydalanib bo'lmaydi. Bu esa prosessor registrlari orqali parametrlarni uzatish usulining standartlashmaganligi bilan bog'liqdir (parametrlarni steklar orqali uzatishdan farqli ravishda) va kompilyatorning amalga oshirilishidan bog'liqdirlar. Va nihoyat, ushbu usuldan, agar prosedura yoki funksiyaning biron erida parametrlarga manzillar (ko'rsatkichlar) bilan amallar bajarish kerak bo'lib qolsa ham foydalanib bo'lmaydi.

Ko'pgina zamonaviy kompilyatorlarda prosedura va funksiyalarga parametrlarni uzatish usuli kompilyator tomonidan avtomatik ravishda, har bir prosedura va funksiya uchun, kodni generasiyalash jarayonida tanlanadi. Ammo ishlab chiqaruvchi prosessor registrlari orqali parametrlarni uzatishni, yoki aniq bir funksiyalar uchun (buning uchun kiruvchi tilning maxsus kalit so`zlaridan foydalaniladi), yoki dasturdagi barcha funksiyalar uchun (buning uchun kompilyatorni o'rnatish komandalaridan foydalaniladi), taqiqlash imkoniyatiga ega.

Ba`zi bir dasturlash tillari (masalan. C va C++ kabi) kiruvchi dasturni ishlab chiqaruvchisiga qanday parametrlar yoki proseduraning lokal parametrlarini prosessor registrlariga aniq joylashtirish imkoniyatini beradilar. Bu holda kompilyator ozod registrlarni birinchi navbatda o'sha parametrlar va o'zgaruvchilar uchun taqsimlashga harakat qiladi.

CHaqiruvchi ob`ekt kodiga funksiya kodini qo'yish usuli (inline-qo'yish deb ataladi) funksiyaning ob`ekt kodi bevosita chaqiriluvchi ob`ekt kodiga har safar chaqirish joyida kiritilishiga asoslangan.

Kiruvchi dasturning ishlab chiqaruvchisi uchun bu funksiya (inline-funksiya) boshqa ixtiyoriy funksiyalardan farq qilmaydi, lekin uni, natijaviy dasturda chasirish uchun butunlay boshqa mexanizmdan foydalaniladi. Amalda, natijaviy ob`ekt kodida funksiyani chaqirilishi bajarilmaydi, balki funksiya tomonidan bajariladigan barcha hisoblashlar bevosita chaqirilayotgan kodning o'zida bajariladi.

Ko'rinib turibdiki, umumiy holda optimallashning ushbu usuli dasturni bajarilish tezligini oshirish bilangina cheklanmay, yana ob`ekt kod hajmini ham oshiradi. Tezlik funksiyani chaqirish bilan bog'liq, erda faqat chaqirish funksiyasi emas, balki barcha parametrlarni uzatish harakatlari ham birgalikda olingan, amallar hisobiga oshadi. Hisoblashlar bu holda bevosita funksiyaning faktik parametrlari bilan olib boriladi. Kod hajmi oshib boradi, chunki har safar funksiya kodini uni chaqirish joyiga qo'shish kerak bo'ladi. Agar funksiya juda sodda bo'lsa va o'zida bir nechta mashina komandalarini jamlagan bo'lsa, u holda natijaviy dastur kodini qisqartirish ham mumkin bo'ladi, chunki kodga uning chaqirish joyiga funksiyani o'zini qo'shishda uning parametrlarini uzatish bilan bog'liq amallar olib tashlanadi.

Ushbu usul juda sodda funksiya va proseduralarga qo'llanilishi mumkin, aks holda natijaviy kod hajmi sezilarli ravishda juda oshib ketishi mumkin. Bundan tashqari, u faqat manzillar bo'yicha, bilvosita RTTI jadvallari orqali manzillashda emas, chaqirilgan funksiyalarga qo'llanilishi mumkin. Ba`zi bir kompilyatorlar ularni faqat ketma-ket, takrorlashlarsiz, hisoblashlarni ko'zda tutadigan funksiyalarga qo'llashga ruxsat beradilar.

Bir qator dasturlash tillari (masalan, C++) ishlab chiqaruvchiga u qanday funksiyalar uchun inline-qo'yishni aniq ko'rsatish imkoniyatini beradilar. C++, dasturlash tilida masalan, bu maqsad uchun kiruvchi tilning inline kalit so'zi xizmat qiladi.

12.7.Takrorlashlarni optimallash

Dasturda takrorlash deb takroran bajarilishi mumkin bo'lgan dasturning ixtiyoriy bo'lagining ketma-ketligiga aytiladi.

Takrorlashlar ko'pgina dasturlarga xosdir. Ko'pgina dasturlarda ko'p marta bajariladigan takrorlashlar bor. Ko'pgina dasturlash tillari takrorlashlarni tashkil etishga mo'ljallangan maxsus sintaksis konstruksiyalarga ega. Ko'rinib turibdiki, bunday takrorlashlar sintaksis tahlil bosqichida oson topiladi.

Ammo ob`ekt dastur nuqtai nazaridan yuqorida aniqlangan takrorlash tushunchasi umumiy hisoblanadi. U o'z ichiga faqat til sintaksisida aniq ifodalangan takrorlashlarnigina olmaydi. Takrorlashlar boshlang'ich dasturda, boshqaruvni uzatishni amalga oshiruvchi, ixtiyoriy konstruksiya yordamida tashkil etilishi mumkin (avvalombor, shart operatorlari va shartsiz o'tish operatorlari yordamida). Natijaviy ob`ekt dastur uchun boshlang'ich dasturda takrorlash qanday konstruksiya yordamida tashkil etilishi muhim emas.

Boshlang'ich dasturda barcha takrorlashlarni topish uchun dasturni boshqarish grafini qurishga asoslangan usullardan foydalaniladi. Takrorlashlar ko'pincha o'z ichiga, hisoblashlarni bajarishga mo'ljallangan, bitta yoki bir qancha chiziqli bo'laklarni oladi. SHu sababli, chiziqli bo'laklarni optimallash usullari takrorlashlarni ham foydaliligini oshirish imkoniyatini yaratadilar. Ammo dasturlarni optimallashning takrorlashlarni optimallashga mo'ljallangan maxsus usullari ham mavjud.

Takrorlashlarni optimallash uchun quyidagi usullardan foydalaniladi:

- takrorlashlardan invariant hisoblashlarni chiqarib tashlash;
- amallarni induktiv o'zgaruvchilar bilan almashtirish;
- takrorlashlarni qo'shish va tarmoqlash.

Takrorlashlardan invariant hisoblashlarni chiqarib tashlash - bu operandlari takrorlashlarning bajarilishi jarayonida o'zgarmay qoladigan amallarni takrorlash chegaralaridan chiqarib tashlashdir. Ko'rinib turibdiki, bunday amallar takrorlash boshlangunicha bir marta bajarilishi mumkin va olingan natijalar so'ngra takrorlashda foydalanilishi mumkin. Masalan, quyidagi takrorlash

```
for i:=1 to 10 do
begin
A[i]:= V *S*A[i];
```

end; agar V va S o'zgaruvchilarning qiymati takrorlash tanasining hech erida o'zgarmasa, quyidagi amallarning ketma-ketligiga almashtirilishi mumkin

```
D:=V*S;
for i:=1 to 10 do
begin
A[i]:= D*A[i];
```

end;

Bu erda ko'paytirish amali V*S faqat bir marta bajariladi, birinchi variantda esa u, 10 marta bir xil natija bilan bajarilar edi.

Operandlarning takrorlash tanasidagi o'zgarmasligi har doim ham ko'rinib turmasligi mumkin. O'zgaruvchilarga qiymat o'zlashtirishning mavjud emasligi ularning o'zgarmasligining ishonchli belgisi bo'lib hisoblanmasligi mumkin. Umumiy holda kompilyatorga o'zgaruvchilarni invariantligiga u yoki bu holda ta`sir ko'rsatuvchi barcha amallarni e`tiborga olishiga to'g'ri keladi. Bunday amallar bo'lib ko'rsatkichlar ustidagi amallar, funksiyalarni chaqirish (hattoki agar o'zgaruvchilarning o'zlari funksiyaga parametrlar sifatida uzatilmasalar, u ularning qiymatlarini «nojuya samara») orqali o'zgartirishi mumkin. Kompilyasiya jarayonidagi barcha ko'rsatkichlarni (manzillarni) o'zgarishlari va barcha «nojuya samara»larni ko'rib turish mumkin bo'lmaganligi sababli, kompilyator har safar, u yoki bu o'zgaruvchini invariantligini shubha ostiga qo'yadigan amallar takrorlash tanasida paydo bo'lganida, takrorlashdan invariant o'zgaruvchilarni chiqarib tashlashni rad etishga majbur bo'ladi.

Amallarni induktiv o'zgaruvchilar bilan almashtirish – bu dastur tanasidagi murakkab amallarni induktiv o'zgaruvchilar bilan oddiyroq amallarga o'zgartirishdan iboratdir. Qoida bo'yicha, ko'paytirish qo'shishga almashtiriladi. O'zgaruvchi takrorlash jarayonida induktiv deyiladi, agar uning qiymati takrorlashni bajarish jarayonida arifmetik progressiyani tashkil etsa. Takrorlash jarayonida bunday o'zgaruvchilar bir nechta bo'lishi mumkin, u holda takrorlash tanasida ularni bitta yagona o'zgaruvchiga o'zgartirish mumkin, har bir o'zgaruvchi uchun aniq qiymatlar esa mos munosabat koeffisientlari yordamida hisoblanadilar (takrorlash chegarasidan tashqarida barcha o'zgaruvchilarga qiymatlar o'zlashtirilgan bo'lishlari kerak, agar, albatta ulardan foydalanilayotgan bo'lsa).

Eng sodda induktiv o'zgaruvchi bo'lib, takrorlashni sanoqchi-o'zgaruvchi hisoblanadi (for toifasidagi takrorlash, ko'pgina zamonaviy dasturlash tillarining sintaksisida uchraydi). Induktiv o'zgaruvchilarning murakkab holatlari takrorlash tanasida maxsus tahlilni talab etadilar. Induktiv o'zgaruvchilarni topilganidan so'ng, takrorlash tanasidagi ular foydalanilayotgan amallarni tahlil qilish zarur. Bunday amallarning bir qanchasi soddalashtirilishi mumkin. Qoida bo'yicha, bu erda gap ko'paytirishni qo'shishga almashtirish haqida ketmoqda.

Masalan, quyidagi takrorlash

```
S := 10;
for i:=1 to N do A[i]:= i*S;
```

quyidagi amallar ketma-ketligiga almashtirilishi mumkin

```
S := 10;
T := S; i := 1;
  while i <= 10 do
 begin
   A[i]:= T;
   T:= T + 10;
   i := i + 1;
 end:
```

Bu erda Pascal tilining sintaksisidan foydalanilgan, T esa bu qandaydir yangi vaqtinchalik o'zgaruvchi (ushbu maqsadda mavjud S o'zgaruvchidan foydalanish to'g'ri bo'lmaydi, chunki uning qiymati ushbu takrorlash tugagandan keyin ham ishlatilishi mumkin). Natijada N ta ko'paytirish amalini bajarishni, N ta qo'shish amalini bajarishga almashtirish bajariladi (ular esa tezrok bajariladilar). Takrorlashning birinchi variantida induktiv o'zgaruvchi bo'lib i, ikkinchi variantida esa induktiv o'zgaruvchilar i va T hisoblanadi. Boshqa misolda

```
S := 10;
for i:=1 to N do R:=R + F(S);
S := S + 10;
```

ikkita induktiv o'zgaruvchilar — i va S. Agar ularni bittaga almashtirilsa, u holda i o'zgaruvchining umuman ma'nosi qolmaydi, u holda ushbu takrorlashni quyidagi amallarni ketma-ketligiga almashtirish mumkin

```
S := 10; M := 10 + N*10;
while S <= M do begin R := R + F(S); S := S + 10; end;
```

Bu erda I o'zgaruvchi uchun N qo'shish amalini M yangi vaqtinchalik o'garuvchi kiritish orqali almashtirish mumkin (avvalgi misol kabi bu erda xam Pascal tili sintaksisidan foydalanildi).

Zamonaviy aniq kompilyatorlarda bunday almashtirishlardan juda kam foydalaniladi, chunki ular etarli darajada murakkab tahlilni talab etadilar, erishilgan yutuq esa juda ham katta emas — qo'shish va ko'paytirish amallarini bajarilish tezligining farqi, boshqa ko'pgina amallar kabi, zamonaviy hisoblash tizimlarida juda ham sezilarli darajada emas. Bundan tashqari takrorlashlarning shunday variantlari mavjudki, ularda ko'rsatilgan aylantirish usullarining foydasi munozarali masala hisoblanadi.

Takrorlashlarni qo'shish va tarmoqlash tashkil etishning ikkita turli variantini ko'zda tutadi: ikkita bir- birini ichiga kirgan takrorlashlarni birga qo'shish va takrorlashni amallarning chiziqli ketma-ketligiga almashtirish. Ikkita bir- birini ichiga kirgan takrorlashlarni qo'shishni quyidagi misolda ko'rsatish mumkin

```
for i:=1 to N do
for j:=1 to M do A[i,j] := 0;
```

Bu erda ikki o'lchamli to'plamga boshlang'ich qiymat berish jarayoni amalga oshirilmokda. Lekin ob'ekt kodda ikki o'lchamli to'plam — bu N*M bo'lgan xotira o'lchamining sohasidir, shuning uchun (ob'ekt kodi nuqtai nazaridan, kiruvchi til emas) bu amalni quyidagicha ifodalash mumkin:

```
K := N*M;
for i:=1 to K do A[i] := 0;
```

Takrorlashlarni tarmoqlashni bajarilish soni kompilyasiya jarayonida avvaldan ma'lum takrorlashlar uchun bajarish mumkin. U holda N martalik takrorlashni N ta amallarning, takrorlash tanasidan tuzilgan, chiziqli ketma-ketligiga almashtirish mumkin.

Masalan, quyidagi takrorlashni

```
for i:=1 to 3 do A[i] := i;
```

quyidagi amallar bilan almashtirish mumkin

```
A[1] := 1;
A[2] := 2;
A[3] := 3;
```

Bu erda, induktiv o'zgaruvchili amallarni chiqarib tashlash hisobiga tezlikdagi ko'p bo'lmagan yutuqga erishiladi. ammo dastur hajmi sezilarli ravishda oshishi mumkin.

12.8.Optimallashning mashinaga-bog'liq usullari

Optimallashning mashinaga-bog'liq usullari, natijaviy dastur bajariladigan, aniq maqsadli hisoblash tizimining arxitekturasiga mo'ljallangan. Qoida bo'yicha, har bir kompilyator bitta aniq maqsadli hisoblash tizimining arxitekturasiga mo'ljallangan.

Gohida kompilyatorni qo'yilishida mumkin bo'lgan maqsadli arxitekturalardan birini aniq ko'rsatish mumkin bo'ladi. Ixtiyoriy holatda ham natijaviy dastur aniq berilgan maqsadli arxitektura uchun tuziladi.

Hisoblash tizimining arxitekturasi bu tizimning apparat va dasturiy tashkil etuvchilarini va tizimni yagonaligi nuqtai nazaridan ular orasidagi o'zaro aloqani ifodalanishidir. «Arxitektura» tushunchasi o'z ichiga maqsadli hisoblash tizimining apparat va dasturiy vositalarining xususiyatlarini oladi. Optimallashning mashinaga-bog'liq usullarini bajarishda kompilyator uning u yoki bu tashkil etuvchilarini e'tiborga olishi mumkin. Arxitekturaning qanday aniq xususiyatlarini e'tiborga olinishi, kompilyatorni amalga oshirilishidan bog'liq va uni ishlab chiqaruvchilari tomonidan aniqlanadi.

Hozirgi kunda hisoblash tizimlarining mavjud arxitekturalari soni juda ko'p. SHu sababli, ularga mo'ljallangan barcha optimallash usullarini qarab chiqishning imkoni yo'q. Ushbu masala qiziqtirganlar esa maxsus adabiyotlarga murojaat etishlari mumkin. Quyida esa fakat mashinaga-bog'liq optimallashning asosiy ikki aspektini varaymiz: prosessor registrlarini taqsimlash va parallel hisoblashlar uchun kodni generasiyalash.

Prosessor registrlarini taqsimlash

Zamonaviy hisoblash tizimlari uning bazasida qurilgan, prosessorlar, bir qancha dasturiy –mumkin bo'lgan registrlarga ega bo'ladilar. Ularning ba'zi bo'laklari qandaydir aniq maqsadlarni bajarishga mo'ljallangan bo'lishlari mumkin (masalan, registr — stekni ko'rsatkichi yoki registr — komanda-sanoqchisi), boshqalari esa turli amallarni bajarishda ixtiyoriy ravishda foydalanilishi mumkin («umumiy belgilangan registrlar»). Operandlarni qiymatlarini va hisoblash natijalarini saqlash uchun umumiy belgilangan registrlardan foydalanish dasturning tezligini oshirishga erishish imkonini beradi, chunki prosessor registrlari ustidagi xarakatlar har doim xotira yacheykalariga nisbatan tezrok bajariladilar. Bundan tashqari, bir qancha prosessorlarda barcha amallar xotira yacheykalarida bajarilmasligi mumkin, shu sababli, ko'pincha registrga operandni avvaldan yuklash talab etiladi. Amalni bajarilish natijasi ham ko'pincha registrda bo'ladi, va agar zarur bo'lsa uni xotira yacheykasiga yozish kerak bo'ladi.

Prosessorning mumkin bo'lgan dasturiy registrlarni cheklangan sonlidir. SHu sababli, ularni hisoblashlarni bajarish jarayonida taqsimlash masalasi qo'yiladi. Ushbu masala bilan har bir zamonaviy kompilyatorning natijaviy dastur kodini generasiyalash bo'lagida mavjud bo'lgan registrlarni taqsimlash algoritmi shug'ullanadi.

Registrlarni, hisoblashlarni oraliq va oxirgi natijalarini saqlash uchun, taqsimlashda, shunday holatlar yuzaga kelishi mumkinki, u yoki bu o'zgaruvchini qiymatini (u bilan bog'liq xotira yacheykasi) registrga, keyingi hisoblashlarni amalga oshirish uchun, yuklash kerak bo'ladi, va bu vaqtda barcha mumkin bo'lgan registrlar band bo'ladilar. U holda kompilyatorga o'zgaruvchini yuklash bo'yicha kodni tashkil etishdan avval, xotira yacheykasidan bir qiymatni yozish uchun kodni generasiyalash kerak bo'ladi (registrni ozod qilish uchun). SHuni ham esda tutish kerakki, yozilgan qiymatni keyinchalik yana registrga yuklash kerak bo'lishi mumkin. Qiymatini

xotiraga yozish kerak registrni tanlash masalasi kelib chiqadi. Prosessor registrini almashtirish taktikasini ishlab chiqish zaruriyati paydo bo'ladi. U kompьyuter xotirasidagi jarayonlar va varaqlarni almashtirish taktikasi bilan o'xshashdir. Ammo operasion tizimdagi xotira dispetcheridan farqi shundaki, kompilyator kodni tahlil qiladi va yozilgan qiymatlardan qaysi birlari keyingi keladigan hisoblashlar uchun kerak bo'lishini aniqlaydi.

Registrlarni umumiy taqsimlashdan tashqari maxsus xarakterdagi registrlarni taqsimlash algoritmlaridan foydalanish mumkin. Masalan, ko'pgina prosessorlarda registr-akkumulyator mavjud bo'lib, u turli arifmetik amallarni bajarishga mo'ljallangan (u bilan amallar yoki tezroq bajariladilar, yoki qisqarok ko'rinishga ega bo'ladilar). SHu sababli unga ko'pincha foydalaniladigan operandlarni yuklashga harakat qilinadi; undan, koida bo'yicha, alohida operatorlar va funksiyalarning natijalarini uzatish uchun foydalaniladi. YAna takrorlashlar sonini sanoqchi registrlar, baza ko'rsatkichlari uchun mo'ljallangan registlar mavjud bo'ladilar. Kompilyator ularni o'zlari mo'ljallangan maqsadlari uchun taqsimlashga harakat qilishi kerak bo'ladi.

Hisoblashlarni parallel bajarishni ta`minlaydigan prosessorlar uchun kodni optimallash

Ko'pgina zamonaviy prosessorlar bir nechta amallarni parallel bajarish imkoniyatini beradilar. Bu erda gap arifmetik amallar ustida ketmokda.

Kompilyator bunday yo'l bilan ob`ekt kodni yaratishida barcha operandlari bir biridan bog'liq bo'lmagan maksimal mumkin bo'lgan qo'shni amallarni bo'lishini nazarda tutadi. Bu masalani global ma`noda barcha dastur uchun to'liq echilishi mumkin bo'lmagan masaladir, lekin, aniq operator uchun, bu uning amallarini bajarilish tartibidir. Bu holatda optimal variantni topish masalasi amallarni o'rin almashtirishni bajarilishiga keltiriladi (agar, albatta bu mumkin bo'lsa). Bu erda, optimal variant amallarning xarakteridan, hamda, parallel hisoblashlarni amalga oshirish uchun prosessorda mavjud bo'lgan konveyerlar sonidan bog'liq bo'ladi. Masalan, A+B+C+D+E+F amalni prosessorda bir oqimli ma`lumotlarni qayta ishlashda quyidagi tartibda bajarish yaxshiroq. ((((A+B)+C)+D)+E)+F. U holda oraliq natijalarni saqlash uchun kam yacheykalar talab etiladi, bajarilish tezligi esa bu holda amallarni tartibidan bog'liq bo'lmaydi.

Ushbu amal ikki oqimli ma`lumotlarni qayta ishlash prosessorida bajarilish tezligini oshirish maqsadida quyidagi tartibda amalga oshirilishi mumkin ((A+V)+S)+ +((D+E)+F). U holda, quyidagi amallar A+V va D+E, va ularning natijalarini qo'shish parallel rejimda amalga oshirilishi mumkin. Komandalarni aniq tartibi, oraliq natijalarni saqlash uchun registrlarni taqsimlash prosessorning toifasidan bog'liq bo'ladi.

Uch oqimli ma`lumotlarni qayta ishlash prosessorida ushbu amalni uch ko'rinishga quyidagicha bo'lish mumkin (A+B)+(C+D)+(E+F). Endi uch amal A+V, C+D va E+F parallel bajarilishi mumkin. To'g'ri, bu erda ularning natijalari ketma-ket qayta ishlanadilar.

Sinov savollari

1. Dasturlarni optimallash jarayoni nima?

2. Kodni optimallashning asosiy usullari haqida ma`lumot bering.

3. Kodni optimallashning asosiy tamoyillari haqida ma`lumot bering.

4.Dasturning chiziqli bo'laklari uchun optimallash haqida ma`lumot bering.

5.Mantiqiy ifodalar uchun optimallash haqida ma`lumot bering.

6.Takrorlashlar uchun optimallash haqida ma`lumot bering.

7.Prosedura va funksiyalarni chaqiriqlari uchun optimallash haqida ma`lumot bering.

8.Natijaviy dasturnning foydalilik ko'rsatkichi kriteriylari haqida ma`lumot bering.

9. Prosessor registrlarini umumiy taqsimlashning optimallash jarayonidagi o'rni nimada?

10.Hisoblashlarni parallel bajarishni ta`minlaydigan prosessorlar uchun kodni optimallash haqida ma`lumot bering.

Foydalanilgan adabiyotlar

1. Гордеев А.З., Молчанов А.Д. Системное программное обеспечение. – Спб-Петер, 2002. -734с.
2. Кревский И.Г. , Селиверстов М.И., Григорьева К.В. Формальные языки, грамматики и основы построения трансляторов: Учебное пособие: (под ред. д.т.н. проф. А.М.Бершадского): изд. Пенз.гос.ун-та 2002 г. -124 с.
3. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов. –СПб: Питер, 2003.-396 с.
4. Афанасьев А.Н. Формальные языки и грамматики: Учебная школа: УлГТУ, 1997. – 84 с.
5. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции -: Мир, 1979.-487с.
6. Компаниец Р.И. Системное программирование. Основы построения трансляторов. СПб.:Корна принт., 2000. -256 с.
7. Дьяконов В.Ю. Системное программирование. Высш.шк.. 1990. - 221 с.
8. Ананченко И. Программное обеспечение для создания электронных книг.Радиомир. Ваш компьютер.-2005.
9. Шаньгин В.Ф. Программное обеспечение микро ЭВМ. Выс.шк.1988.-126 с.
10. Брябрин В.М. Программное обеспечение персональных ЭВМ. Москва.: Наука. 1990.-272 с.

11. Браун С. Операционная система UNIX – М.: Мир, 1986.-463 с.
12. Вендров А.М. Case – технология. Современные методы и средства проектирования информационных систем. –М: Финансы и статистика, 1998 -361с.
13. Вирт Н. Алгоритмы и структуры данных – М.; МИР, 1989.– 360 с.
14. Дворогин А.И. Основы трансляции. Учебное пособие.– Волгоград. ВолГТУ,1999.80г.
15. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование- М.; Издательство МГТУ имени Н.Э.Баумана, 2003.-367 с.

Қўшимча адабиётлар.

1. Карпов Б.И. Паралов Т.Х. С++: -СПб: Питер, 2002-210 с.
2. Киселев С.Ю. Технология разработки программного обеспечения. Информационная система.-СПб: СПбГУАП, 1998.-107 с.
3. Майерс Дж. Надежность программного обеспечения. – М: Мир, 1997.360 с.
4. Мельников Б.Ф. Подклассы класса контекстно – свободных языков – М.: МГУ, 1995. – 174 с.
5. Компаниец Р.И. Маньков Е.В. Филиппов Н.Е. Системное программирование, основы построения трансляторов. Учебное пособие для средних и высших учебных заведений. СПб: Карона, 2000-256 с.
6. А.Левин. Самоучитель полезных программ. Питер. Санкт-Петербург, 2002.
7. Карпов Б.И. Delphi: Специальный справочник. – СПб: Питер, 2001-648 с.
8. Карпов Б.И. Visual Basic Специальный справочник. – СПб: Питер, 2000-415с.
9. Карпов С.Ю. Теория автоматов. Учебные пособия для вузов. –СПб: Питер, 2003.-201с.
10. Бейбир Р.Л. Программное обеспечение без ошибок. Радио и связь.1996. -176 с.
11. Бонч-Буревич М.А. Программное обеспечение персональных ЭВМ. СПб.:СПбГУТ, 1994. -104 с.
12. WWW.codecrojekt.ru
13. WWW. master.ru
14. WWW.bdn_borland.com
15. http://microsofft.com