MIRZO ULUG'BEK NOMIDAGI
O'ZBEKISTON MILLIY UNIVERSITETI

GAYNAZAROV S.M.

DASTURLASH TIZIMLARI

O‘ZBEKISTON RESPUBLIKASI
OLIY TA’LIM, FAN VA INNOVATSIYA VAZIRLIGI

MIRZO ULUG‘BEK NOMIDAGI
O‘ZBEKISTON MILLIY UNIVERSITETI

GAYNAZAROV S.M.

DASTURLASH TIZIMLARI

DARSLIK

O‘zbekiston Respublikasi **Oliy ta’lim, fan va innovatsiya vazirligi** tomonidan **5330200 – Informatika va axborot texnologiyalari (tarmoqlar bo‘yicha)**, yo‘nalishi talabalari uchun darslik sifatida tavsiya etilgan

Toshkent
“Ma’rifat”
2023

UO'K: 004.42(075)
KBK: 32.973-018.2ya7
G 15

Gaynazarov S.M. Dasturlash tizimlari. Darslik.
–T.: "Ma'rifat", 2023. 432 bet.

Darslikda dasturning hayot davri, dasturlash tizimini tuzilishi va ishlash tamoyili, translyatorlar ishlash tamoyili, translyatorlar tuzilishi, identifikator jadvalini tuzish usullari, formal tillar va grammatikalarning sinflari, belgili zanjirlar bilan ishlovlar, muntazam grammatikalar, chekli avtomatlar, kontekstdan erkin grammatika, magazin xotirali avtomatlar, tushuvchi va ko'tariluvchi tahlil, leksik va sintaktik tahlilchilarni yaratish va ishlash, sintaktik boshqaruvli tarjima, semantik tahlil, dasturni ichki tasvirga o'tkazish, kod interpretatsiyasi, dastur kodini optimallashtirish, kod generatsiyasi va xotira taqsimoti kabi asosiy tushunchalari yoritilgan.

Oliy o'quv yurtlarining 5130200 – "Amaliy matematika va informatika" yo'nalishining bakalavriat hamda 5A130202 – "Amaliy matematika va axborot texnologiyalari" mutaxassisligining magistrantlari uchun, Kompyuter texnologiyalari va informatika sohasidagi barcha yo'nalish va mutaxassislik talabalari ham foydalanishlari mumkin.

Ilmiy muxarrir:
M.X.Xakimov – Mirzo Ulug'bek nomidagi O'zbekiston Milliy Universiteti
"Algoritmlar va dasturlash texnologiyalari" kafedrasining professori, t.f.n.

Taqrizchilar:
M.M.Aripov – Mirzo Ulug'bek nomidagi O'zbekiston Milliy Universiteti
"Amaliy matematika va informatika" kafedrasining professori, f.-m.f.d.

R.D.Aloyev – Mirzo Ulug'bek nomidagi O'zbekiston Milliy Universiteti
"Matematik modellashtirish va kriptoanaliz"
kafedrasining professori, f.-m.f.d.

Darslik O'zbekiston Respublikasi Oliy va o'rta maxsus ta'lim Vazirligini 2019-yil 4-oktyabrdagi 892-sonli buyrug'iga asosan nashrga tavsiya etilgan (ro'yxatga olish raqami 654-034).

ISBN: 978-9910-9887-9-0

KIRISH

Translyatorlarni yaratish- bu juda qiziq soha. Bu soha asosida formal tillar nazariyasi turibdi. Ushbu darslikning asosiy maqsadi – shu nazariya asoslarini yoritishdan iborat. Bu nazariyaning asosiy qismlari klassika bo'lib qolgan va uning asosida talaba biror bir notrivial til uchun o'z translyatorini yaratishi mumkin.

Informatikani o'rganuvchi har qanday odamda savol tug'ilishi mumkin: translyatorlarni qurish nazariyasi va amaliyoti bo'yicha bilimlar kerakmi? Chunki, ishlatiladigan tillar uncha ko'p emas va ular uchun kompilyatorlar yaratib bo'lingan. Bu savolni javobi shunday - amalda keng tarqalgan algoritmik dasturlash tillardan tashqari, doimo maxsus tillarga talab o'smoqda. Bunday tillar aniq sohalarda ma'lumotlarni uzatish va qayta ishlashda qo'llanadi. Aynan shu maxsus tillar uchun kompilyatorlar yaratish kerak bo'ladi.

Formal maxsus tillar barcha sohalarda kerak bo'ladi, ularni tuzimini ta'riflash, shu tilda berilgan ma'lumotni to'g'ri qabul qilish va mashina tiliga o'tkazish vazifasini maxsus kompilyatorlar bajaradi.

Bunday mukammal kompilyatorni faqat nazariya asosida sifatli va ishonchli yaratish mumkin.

Bu darslik O'zbekiston Milliy Universitet "Matematika" fakultetida o'qilayotgan mavzular asosida yozilgan. Undan tashqari bu fan tematikasi hozirda dolzarb bo'lishiga qaramay, adabiyotlar juda kam va chop etilganiga ancha vaqt bo'lgan. O'zbek tilida deyarli yo'q desak ham adashmaymiz.

Muallif aminki, bu darslik oz bo'lsa ham kerakli adabiyotlarni o'rnini qoplaydi.

Muallif ushbu darslikni ilk bor o'zbek tilida yaratilishi, uning ahamiyati va mas'uliyatini chuqur anglagan holda, hamda ustida bajariladigan ijodiy ish jarayonidagi vazifalarni aniq kelishib olgan holda olib bordi va buni e'tirof etishni o'z vazifasi deb maqsad qo'ygan.

Ushbu darslik to'rtta bo'limdan iborat bo'lib, har bir bo'limga maqsadli aniq funksional vazifa yuklatilgan.

I- bo‘limda boshlang‘ich asosiy tushunchalar, dasturlash tizimlarni asosi, dasturni hayot davri, uning asosiy fazalari keltirilgan. Dasturlash tizimlarni komponentalari, shu jumladan matn tahrirchi, kompilyator, jamlovchi, yuklovchi to‘g‘risida ma’lumotlar berilgan. Xotirani taqsimlash usullari ko‘rsatilgan, qism-dasturlar kutubxonalari ochib ko‘rsatilgan. Kompilyatorni asosiy vazifasi va bosqichlari keltirilgan. Shu jumladan leksik tahlil, sintaktik tahlil, dasturni ichki tasviri, identifikatorlar jadvalini tashkil qilish to‘g‘risida ma’lumotlar keltirilgan. Obyekt kodni generatsiyasi, dasturni optimallashtirish usullari batafsil keltirilgan.

II-bo‘limda zamonaviy dasturlash tizimlarni to‘g‘risida qisqacha ma’lumot keltirilgan. Ushbu bo‘limda komponent va vizual dasturlashni vazifalari aniqlangan. Borland kompaniyasini dasturlash tizimlari to‘g‘risida ma’lumotlar berilgan. Mikrosoft kompaniyasini dasturlash tizimlari to‘g‘risida aytib o‘tilgan. Unix, Lexus, GNU tizimlarni dasturiy ta’minoti to‘g‘risida ham ma’lumot berilgan.

III- bo‘limda kompilyator yaratish asosida turgan formal tillar nazariyasi keltirilgan. Ushbu bo‘limda formal grammatika ta’rifi, zanjirlarni keltirib chiqarish, grammatikani sinflash, muntazam grammatika, chekli avtomatlar, kontekstdan erkin grammatika, grammatika aniqlovchilar, sintaktik daraxtlar va sintaktik tahlil usullari to‘g‘risida ma’lumot keltirilgan.

IV- bo‘lim translyatsiya nazariyasining elementlarini amalga oshirishi ko‘rsatilgan. Bu bo‘limda dasturlash tili modelida kompilyatorni barcha bosqichlarining algoritmlari va dasturlari keltirib o‘tilgan. Dasturlar C++ tilida qayd etilgan. Kompilyatorlarni yaratishda leksik tahlilchini yaratish, sintaktik tahlilchini yaratish dasturlari to‘g‘risida ma’lumot berilgan va bu dasturlarni ishlatish usuli ko‘rsatilgan.

Har bir nazariy boblardan so‘ng, nazorat uchun savollar, amaliy boblardan so‘ng esa amaliyot uchun topshiriqlar keltirilgan.

Muallif, darslikni to‘liq ko‘rib chiqib, undagi kamchiliklarni to‘ldirishida ko‘maklashgan ilmiy muharrir dotsent M.X. Xakimovga

o‘zining katta minnatdorchiligini bildiradi.

Darslikdan 5330100 - Axborot tizimlarining matematik va dasturiy ta’minoti, 5330200 – Informatika va axborot texnologiyalari (tarmoqlar bo‘yicha), 5330203 - Amaliy matematika va informatika, 5A330100 - Ahborot tizimlari matematik va dasturiy ta’minoti (tarmoqlar bo‘yicha), 5A330203 - Amaliy matematika va informatika, 5A330204 - Axborot tizimlari (tarmoqlar bo‘yicha), 5110700 - Informatikani o‘qitish metodikasi, 5111000 - Kasb ta’limi (yo‘nalishlar bo‘yicha), 5A220202 - Axborot xizmatlari texnologiyalari (faoliyat turlari bo‘yicha) talabalari, tadqiqotchi-izlanuvchilar ham foydalanishlari mumkin.

Muallif

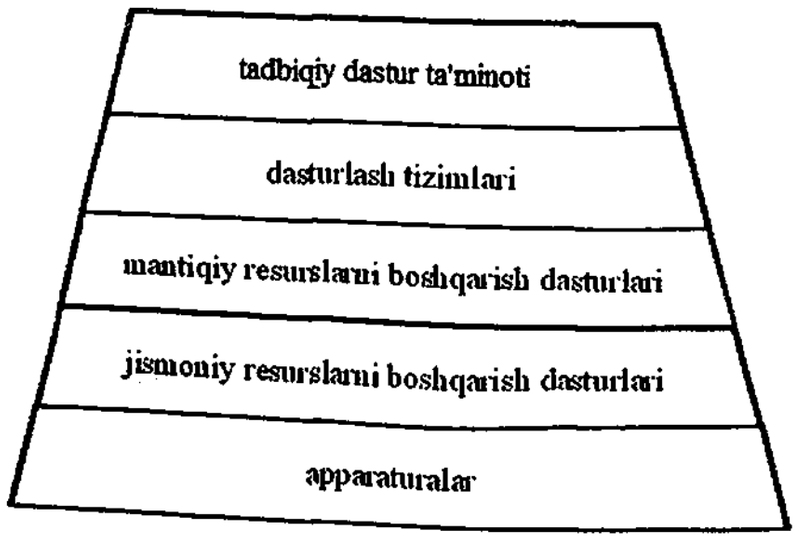
Sentyabr 2023 yil

I.BO'LIM. DASTURLASH TIZIMLARI

1- BOB. DASTUR MAHSULOTINING HAYOT DAVRI

Tayanch iboralar: *dastur, dastur mahsuloti, dasturlash tizimlari, integranlangan dastur mahsuloti, dastur yaratish bosqichlari, yaratish va kuzatish bosqichlari, talablarni aniqlash va tahlil qilish, loyihalash, dastur matnni yozish ("kodlash"), dastur majmuasini jamlash va integratsiyalash, verifikatsiyalash, testlash va sozlash, "qora" quti, "oq" quti, hujjatlar yaratish, joriy qilish(qo'llash), nusxalash, kuzatish, kaskad qaytish chizmasi, spiral chizmasi.*

Dastur – apparat ta'minotida dasturlash tizimlari mantiqiy resurslarni boshqarish dasturlari va tadbiqiy dasturlar ta`minoti orasida joy egallaydi (1.1-rasm).



1.1-rasm. Dastur-apparat ta'minoti.

Ta'rif: ***dastur mahsulotini butun hayoti davrida qo'llab quvvatlab turadigan dastur vositalar majmuasi, dasturlash tizimlari deb nomlanadi.***

Dasturchilar mehnat natijasini belgilashlari uchun quyidagi atamalar ishlatiladi: dastur, dastur mahsuloti va tizimli(yoki integrallangan) dastur muhiti.

Dastur - biror bir alohida masalani yechish uchun, yagona muallif tomonidan ma'lum operatsion muhitida yaratiladi. Dastur muallifdan ajralmasdir, odatda uning hajmi katta emas.

Dastur mahsuloti, bir nechta operatsion muhitlarda muallifsiz ishlashi mumkin. Dastur mahsulotini tekshirish, o'zgartirish va bajarishda muallifning ishtiroki kerak bo'lmaydi (u muallifdan ajratilgan). Dastur mahsulotining sifati oddiy dasturga nisbatan ancha yuqori turadi. Dastur mahsulotidan foydalanish va unga o'zgartirishlar kiritish uchun bir nechta hujjatlar yaratiladi. Dastur mahsulotining sozlovchisi bo'lishi kerak (setup.exe) va uni sozlashda, biror bir sozlash parametrlar qiymati berilishi shart.

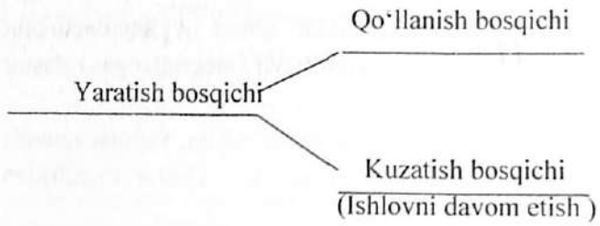
Tizimli (integranlangan) dastur muhiti bu dastur vositalar majmuasidir (paket). Integrallangan dastur muhit sifatida Microsoft Office paketini keltirish mumkin. Bu paket kelishilgan interfeysli, o'nga yaqin dastur mahsulotlardan iborat. Paketning har bir komponentasi o'ziga xos bo'lgan maxsus vositasiga qaramasdan, parametrlarni berish, ishlash tartibotlari va foydalanuvchi amallari barcha komponentalarda bir xil va o'xshashdir. Komponentalar orasida berilganlarning uzatishini tashkil qilish mumkin. Masalan, Yexcelda tuzilgan murakkab jadvalni PowerPointda qulay ko'rinishda taqdimot qilish mumkin.

Dasturning hayot davrining bosqichlari

Dastur ustida ish olib borish butun hayoti davrida davom etadi.

Dastur hayoti uch bosqichdan iboratdir: yaratish bosqichi, qo'llanish bosqichi va kuzatish bosqichi.

Dasturni yaratishda, uning ustida ishlovchi jamoa ish olib borishi mumkin. Kuzatishni esa boshqa bir jamoa bajarishi mumkin. Dastur mahsulotini ko'pincha umuman boshqa odamlar ishlatishi mumkin. Yaratish bosqichi oldin bajariladi, qolgan ikki faza baravariga bajarilishi mumkin (1.2-rasm).



1.2-rasm. Dasturni yaratish bosqichlari

Hajmi katta va uzoq ishlatiladigan dasturlarda kuzatish bosqichi ishlovni davom etish deb ataladi. Kuzatish bosqichning kerakligining ikki sababi bor. Birinchidan, katta dasturda albatta test jarayonida aniqlanmagan bir nechta xatolar mavjud bo'lishi mumkin. Ikkinchidan, dastur rivojlanishi kerak: yangi zaruriyatlar paydo bo'lishi, hisoblash mashinalarga yangi qurilmalar ulanishi va ular bilan dasturning ishlashi uchun qo'shimcha kodlar kiritish kerak bo'lishi mumkin bo'ladi.

Yaratish bosqichdagi jarayonni noto'g'ri tashkil qilish, yaratish jarayonida to'g'ri kelmagan dasturlash tizimlar ishlatish, dastur hayot davrida ishlov bilan kuzatish orasida uzilishlarga olib kelishi mumkin. Ishlov namunalariga va qoidalariga rioya qilmagan dasturlar bunday holatlarga olib kelishi mumkin. Agar bunday uzilish mavjud bo'lsa, ishlov bilan qo'llanish orasida ham uzilish bo'ladi (1.3-rasm.)

1.3-rasm. Dasturni yaratish bosqichdagi uzilish

Shuni aytib o'tish kerakki, ko'pincha kuzatish bosqichini yaratuvchi jamoa vakillari davom ettiradi.

Yaratish va kuzatish bosqichlardagi vazifa va ishlar

Yaratish va kuzatish bosqichlarida quyidagi ishlarni bajaririlishi zarur:

- *Talablarni aniqlash va tahlil qilish*
- *Loyihalash*
- *Dastur matnini yozish ("kodlash")*
- *Dastur majmuosini jamlash va integratsiyalash*
- *Verifikatsiyalash, testlash va sozlash*
- *Hujjatlar yaratish*
- *Joriy qilish (qo'llash)*
- *Nusxalash*
- *Kuzatish (barcha yuqorida ko'rsatilgan ishlarni qaytarish)*

Talablarni aniqlash va tahlil qilish

Masalani qo'yish, aniqlash va tahlil qilish ishlari umuman olganda rasmiylashtirilmagan. Lekin, bunda ishlov bosqichiga va natijaviy dastur mahsulotini sifatiga katta ta'sir qiladi. Bu vazifada foydalanuvchi talablarini aniqlashi lozim. Buning uchun ko'pincha iste'molchi bilan birgalikda umumiy atamalar va iboralar lug'atini tuzish kerak bo'ladi va ular yordamida foydalanuvchilar bilan muloqat qilinadi. Tanlangan atamalar va iboralar orqali avtomatizatsiyalash qilish kerak bo'lgan obyektlar ta'riflanadi.

Birinchi qadamda turli materiallar yaratiladi: oddiy matndan boshlab, qisman rasmiylashtirilgan tasniflargacha. Qisman rasmiylashtirish uchun maxsus dastur vositalari yaratilgan, shu jumladan talablarni tavsiflash tillari.

Bunday tillar har xil bo'lishi mumkin:

- Yechim jadvallari. Bu jadvallar kiruvchi va chiquvchi berilganlar orasidagi bog'liqlarni aks ettiradi. Jadvalda berilganlarga shartlar qo'yiladi, berilganlar dastur ishlashiga va chiquvchi berilganlarga ta'siri ko'rsatiladi.
- Funksional diagrammalar. Diagrammalar graf ko'rinishda bo'lib, uning tugunlarida kiruvchi berilganlar, ularga qo'yilgan shart va cheklamalar turadi. Undan tashqari, diagrammalarda berilganlarni qayta ishlash yo'llari ko'rsatiladi.

• Tasniflarni berish tillari. Bu tillar talablarni ta'riflab berish uchun ishlatiladi. CLU, MSC, SDL va hokazo tillar aynan shuning uchun ishlatiladi.

Birinchi vazifa natijasida ta'riflab berilgan talablar, ya'ni tashqi tasnif, foydalanuvchi nuqtayi nazaridan tavsiflangan tizim hosil bo'ladi.

Loyihalash

Butun tizimga tegishli bo'lgan loyihalash vazifasi - murakkab jarayondir. Bu vazifa ko'pincha ikki bo'limdan iborat: tizimni tuzilishini loyihalash ("katta" loyihalash), va o'zaro bog'langan tizim qismlar majmuasini loyihalash ("kichik" loyihalash). Loyihalashda bajarilayotgan jarayon, murakkablikni boshqarish deb nomlanadi.

Har qanday murakkab tizimlar o'z shajarasiga ega. Dastur shajarasini har bir pog'onasi bu o'zaro bog'langan qism-dasturlardir. Bu qismlar o'z navbatda pog'onaostilardan tuzilib, shajara tuzilishiga ega bo'lishi mumkin. Loyihalash bosqichida "Bo'l va boshqar" usuli keng qo'llanadi. Murakkab dastur majmualarni loyihalashda ularni avvalo, kichik tizim qismlarga bo'lib tashlanadi va har bir qism ustidan alohida ishlovlar olib boriladi ("avtonom"). Bu loyihalash jarayonida eng murakkabi – umumiy tizimni qismlarga bo'lish asosini aniqlashdir. Butun tizimni qismlarga bo'lish ***dekompozitsiya*** deb nomlanadi. Bitta dastur tizimini har xil usul bilan dekompozitsiyalash mumkin.

Algoritmik dekompozitsiya – dastur tizimni ishlatiladigan algoritmlar bo'yicha bo'ladi. Bunday usul tuzimli dasturlash va "ustidan ostiga" loyihalash deb nomlanadi. Bo'lish natijasida olingan har bir qism-dasturi (moduli) o'ziga xos ishni bajaradi. Bu yondashuvda berilganlar passiv rol o'ynaydi.

Obyektga yo'naltirilgan dekompozitsiya -- tizimda alohida obyektlarni ajratish bilan bog'liqdir. Obyektlar xabarlarni qabul qilib, ularga biror bir amal orqali javob beradi. Bunday modelda algoritmlar o'z mustaqilligini yo'qotadi va ular ajratilgan obyektlar ustidan amallarga aylanadi. Bu usulda berilganlar faol rol o'ynaydi, chunki ular obyektdir. Bu obyektlar nafaqat ma'lumotga, balki faollikka ham egadir.

Algoritmik dekompozitsiya yuz bergan hodisalar ketma–ketligiga katta ahamiyat bersa, obyekt dekompozitsiyasi bevosita obyektga e'tibor

beradi va obyektlararo bo'lgan hodisalarga, amal sifatida javob beradi. Dasturdagi jarayon obyektdan boshqa obyektlarga uzatilgan xabarlardan tashkil topadi. Hozirgi vaqtda obyektga yo'naltirilgan usul ustuvorlik qiladi.

Ikkinchi vazifa natijasida quyidagilar hosil bo'ladi:

1. Tizim qismlardan tuzilgan shajaraviy chizmasi.
2. Har bir qismning funksionalligi va interfeysi, ya'ni tashqi ko'rinishi.
3. Har bir modul yoki obyektdagi berilganlarni ichki tuzilmasi.
4. Har bir modulda yoki obyektda berilganlarning qayta ishlash algoritmlari.

Bunda 1 va 2 faoliyatlar "katta", 3- va 4-si "kichik" loyihalashga kiradi.

Dastur matnini yozish ("kodlash")

Matnini yozish dasturlash yoki kodlash deb nomlanadi. Bu vazifa natijasida biror bir dasturlash tilida dastur matni hosil bo'ladi. Dastur matnini yozish uchun, qo'yilgan masalaga, talablarga va dekompozitsiya usullariga ko'ra, til tanlanadi. Masalan, obyektga yo'naltirilgan dekompozitsiyani, albatta obyektga yo'naltirilgan dasturlash tilida yozish kerak (C++ tili).

Verifikatsiyalash, testlash va sozlash

Dasturni to'g'riligini tekshirish jarayoni ***verifikatsiya*** deb nomlanadi.

Dasturni tekshirish uchun maxsus nazariy usullardan foydalaniladi. Bulardan biri ***validatsiya***, ya'ni dasturni to'g'riligini mantiqiy usullar yordamida isbotlash. Lekin, amalda test asosida bo'lgan, verifikatsiyani evristik usullari keng tarqalgan. Tuzilgan dasturlarda xatoliklarni aniqlovchi jarayon ***dasturni testlash*** deb nomlanadi.

Testlash jarayonida dasturni boshlang'ich talablar va tasniflarga to'g'ri kelmasligi aniqlanadi. Asosan ikkita testlash usullari bor, bu "qora" va "oq" qutilar. ***"Qora" quti*** usulida, dastur kodi berilmaydi, faqat kiruvchi qiymatlar va kutilgan natijalar beriladi. ***"Oq" quti*** usulida, aksincha dastur kodi ochiq, dasturning nazorat nuqtalarida oraliq natijalar beriladi va shunga qarab test o'tkaziladi. Testlash jarayonida faqat xatolar bor yoki

yo‘qligi aniqlansa, aksincha ***dasturni sozlash*** jarayonida xatolarni topish va ularni to‘g‘rilash amallari bajariladi. Testlash va sozlash natijasida berilgan testlar bo‘yicha xatolar topilib, tuzatilganligi isbotlanadi.

Dastur majmuasini jamlash va integratsiyalash

Jamlash vazifasida dasturni alohida qismlari bir-biriga ulanadi va bitta katta tizimli dastur ta’minotiga keltiriladi. Ko‘pincha dastur yig‘ilgandan keyin, yana bir bor majmua testdan o‘tadi, bu testlashda qism-dasturlari birgalikda ishlashi tekshiriladi.

Hujjatlarni yaratish

Bu vazifa jarayonida bir necha hujjatlar yaratiladi. Tizim dasturchiga qo‘llanma, foydalanuvchiga qo‘llanma, yordamchi va hokazolar. Dasturni joriy etishda, nusxalash va kuzatishda bunday hujjatlar bo‘lishi katta ahamiyatga ega bo‘ladi.

Joriy qilish (qo‘llash)

Yaratilgan dastur mahsulotini qo‘llashda, buyurtmachining ishtiroki, ***joriy qilish*** deb nomlanadi. Bu vazifada ko‘p tashkiliy muommolar yechiladi, shu jumladan, foydalanuvchilarni o‘qitish, aniq tashkilotda va aniq berilganlar bilan dasturni ishlashi va barqarorligini tekshirish ishlari olib boriladi. Agar tashkilot tarmoqda ko‘p ishchi joylarga ega bo‘lsa, dastur mahsuloti ***nusxalanadi.***

Kuzatish vazifasida ishlovchi va foydalanuvchilar orasidagi aloqalar davom etadi. Bu vazifa o‘z ichiga oldingi barcha bosqichlarni qamrab oladi.

Ishlov bajarish usullari

Ideal holatda ishlov jarayoni, kam bog‘langan holda, barcha vazifalarni ketma–ket bajarilishidan iborat. Bunday ishlov chizmasi pasayuvchi usul deb nomlanadi va u kamdan-kam ishlatiladi. Ko‘pincha pog‘ona chizmasi ishlatiladi va unda testlashni bitirgandan so‘ng yana talablarni aniqlash va tahlil qilish vazifasiga qaytiladi. Agar talablar aniq va to‘liq qo‘yilgan bo‘lsa, pog‘ona chizmasi qulaydir, lekin haqiqiy jarayon murakkab bo‘lgani uchun pog‘ona–qaytish chizmasi ishlatiladi (1.4-rasm).

Pog‘ona–qaytish chizmasi muammolarini yechish uchun spiral modeli ishlatiladi.

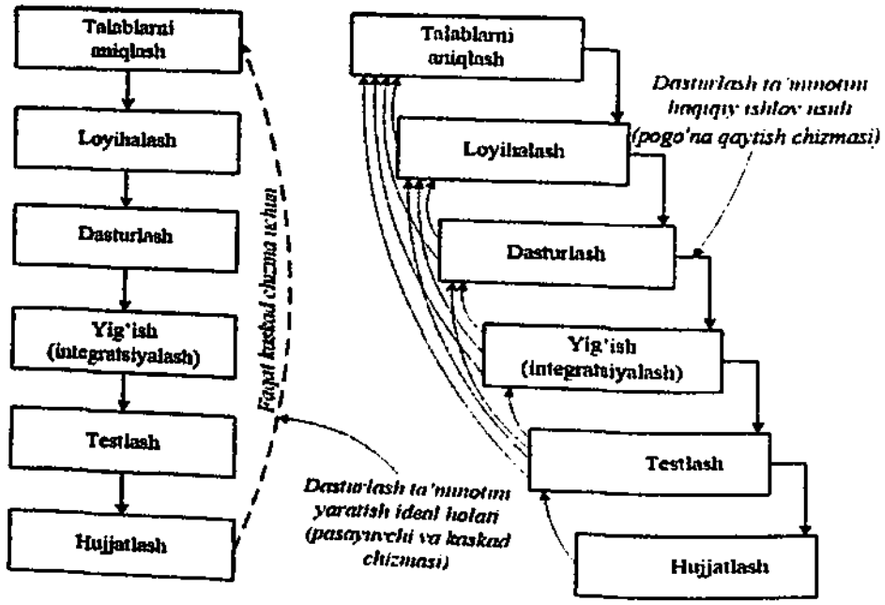
Bu modelda asosiy ahamiyat birinchi va ikkinchi vazifalarga qaratiladi, texnik yechimi na’munalar orqali tekshiriladi. Spiralning har bir aylanmasi dastur naqliga to‘g‘ri keladi, yangi naqlni tuzishni boshlaganda avvalgilarni kamchiliklari tahlil qilinadi, qo‘shimchalar kiritiladi va shu orqali dastur sifati oshiriladi (1.5-rasm).

Xulosa

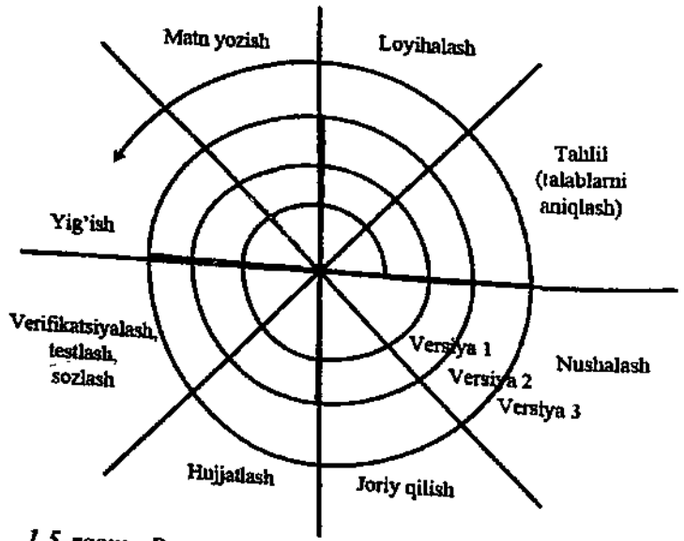
Dasturiy tizimlari apparat dasturiy ta’minot shajarasida tadbiqiy dasturlar ta’minotdan keyin turadi. Dasturlash tizimlari dastur mahsulotini butun hayoti davrida qo‘llab quvvatlashga mo‘ljallangan. Dastur mahsulotiga oddiy dastur, dastur mahsuloti va tizimli dastur muhiti kiradi. Dasturning hayot davri uchta bosqichdan iborat: yaratish bosqichi, qo‘llanish bosqichi va kuzatish bosqichi. Yaratish va kuzatish bosqichlarda quyidagi ishlar bajariladi: talablarni aniqlash va tahlil qilish; loyihalash; dastur matnini yozish (“kodlash”); dastur majmuosini jamlash va integratsiyalash; verifikatsiyalash, testlash va sozlash; hujjatlar yaratish; joriy qilish (qo‘llash), nusxalash; kuzatish (barcha yuqorida ko‘rsatilgan ishlarni qaytarish).

Nazorat uchun savollar

1. Dasturlash tizimlari nimadan iborat?
2. Dastur turlarini keltiring.
3. Dasturning hayot davri qanday bosqichlardan o‘tadi?
4. Dasturni yaratish va kuzatish bosqichlari qanday vazifalarni amalga oshiradi?
5. Dasturni yaratish usullarini keltiring.



1.4-rasm. Dastur ta'minotini yaratish chizmalari.



1.5-rasm. Dastur ta'minotini yaratishning spiral chizmasi.

2- BOB. DASTURLASH TIZIMINING TUZILISHI

Tayanch iboralar: *kompilyator vazifasi, matn tahrirchisi, jamlovchi, yuklovchi, tadbiqiy dastur kutubxonalari, sozlovchi, obyekt modullar, bajaruvchi dastur, integrannlangan dastur yaratish muhitlari, 4GL tili, RAD tizimi, resurs fayllar.*

Dasturlash tizimining yaratilish tarixi

Dasturlash tilida yozilgan har qanday dastur kodi kompilyator tomondan qayta ishlab mashina kodiga aylantiriladi. Shu sababli, kompilyatorlar dasturlash tizimlarining asosiy qismi deb hisoblanadi. Ularning asosiy vazifasi – yangi tadbiqiy va tizimli dasturlarni yuqori bosqichdagi tillar yordamida yaratishdir. Har qanday dasturning o'z hayot davri bor, loyihalashdan boshlab to ishlatilishigacha. Kompilyatorlar dastur ta'minotini yaratishda, ya'ni uni kodlash, tekshirish va sozlash jarayonlarida xizmat qiladi.

Lekin kompilyator, yangi dastur yaratish jarayoni bilan bog'langan barcha masalalarni to'liq yecha olmaydi. Shu sababli kompilyator boshqa dastur vositalar bilan birgalikda ishlatiladi.

Kompilyator bilan birgalikda quyidagi dasturiy vositalar qo'llanadi:

- matnni, ya'ni dastlabki dastur kodini terish uchun ishlatiladigan ***matn tahrirchilari***;
- kompilyator tomondan yaratilgan bir nechta obyekt modullar yaxlit ko'rinishda birlashtiradigan ***jamlovchilar***;
- ko'p holda ishlatiladigan obyekt modullar shaklidagi funksiyalar va qism-dasturlarni saqlaydigan ***tadbiqiy dastur kutubxonalar***;
- tayyor bo'lgan dasturni bajarilishini ta'minlaydigan ***yuklovchilar***;
- xatolarni qidirish va aniqlash uchun mahsus qadamma–qadam tartibotda ishlaydigan ***sozlovchilar***.

Boshlang'ich davrda kompilyatorlar alohida dastur vosita bo'lib faqat kiruvchi tilda yozilgan dastur kodni mashina tili kodiga tarjima qilardi. Kompilyator boshqa dastur vositalarga bog'lanmagan holda yaratilar edi. Dastur yaratuvchi o'zi dastur vositalarni bog'lardi va quyidagi qadamlarni bajarardi:

- biror matn tahrirchi yordamida yuqori bosqich tilida dastur kodini terib uni faylda saqlab qo'yardi;

- bu faylni kompilyatorga kiruvchi sifatida uzatardi;
- agar dasturda xatolar bo'lmasa, hosil bo'lgan mashina kodini obyekt fayllar ko'rinishda yozib qo'yardi;
- olingan obyekt fayllarni kutubxonadagi funksiyalar va qism-dasturlar bilan birgalikda jamlovchiga kirituvchi fayl qilib uzatardi;
- jamlovchidan bajarishga tayyor bo'lgan bajaruvchi dastur fayli olinardi;
- bajaruvchi faylni yuklovchiga uzatardi;
- dasturni bajarish va kerak bo'lsa sozlovchi yordamida uni to'g'ri bajarishini tekshirardi.

Yuqorida ko'rsatilgan barcha amallar ketma-ket buyruqlar bilan yozilar edi. Bunday buyruqlar satr ko'rinishda berilib, unda dastur vosita nomi va kerakli parametrlar (fayl nomi, tartibotlari va hokazo) yozilar, hamda unday satrlar maxsus fayl ko'rinishda saqlanardi (bat kengaytmasi bilan).

Keyinchalik barcha texnik dastur vositalari birlashtirib kompilyator bilan birga beriladigan bo'ldi. Undan tashqari obyekt modullar va kutubxonalar modullari bir xil formatda bo'ladigan bo'ldi.

Buyruq faylini yozish uchun maxsus Makefile tili yaratildi. Bu dastur vositasi yuqorida ko'rsatilgan jarayonni qulay shaklda ta'riflash imkoniyatini berardi lekin bu tilni o'zi ham murakkab bo'lib, ishlatuvchidan yuqori bilimni talab qilardi. Asosan bu usul katta mashinalar zamonida ishlatilar edi. Lekin ba'zi bir kompilyatorlarni haligacha shu usul bilan ishlatish mumkin (Assembler tili).

Keyingi qadamda integrallangan yaratish muhiti paydo bo'ldi va bu muhit o'z ichiga barcha dastur vositalarni qoplab oldi. Bunday muhitda, o'zining ichki tahrirchisida dastur kodini terib, undan tayyor bajaruvchi dasturni olish mumkin edi. Yendi yaratuvchi faqat tilni bilsa yetarli, qolgan barcha ishni muhitning o'zi bajarardi. Ana shunday muhitlarga misollar - Delphi, C++Builder, Visual C++ va hokazo.

Hozirgi zamonda operatsion tizimlar grafik tartibotida ishlagani sababli (Windows), ilovani grafik interfeysini yaratuvchi tillar mavjud bo'ldi va ular yordamida tadbiqiy dastur resurslari degan tushuncha kiritildi.

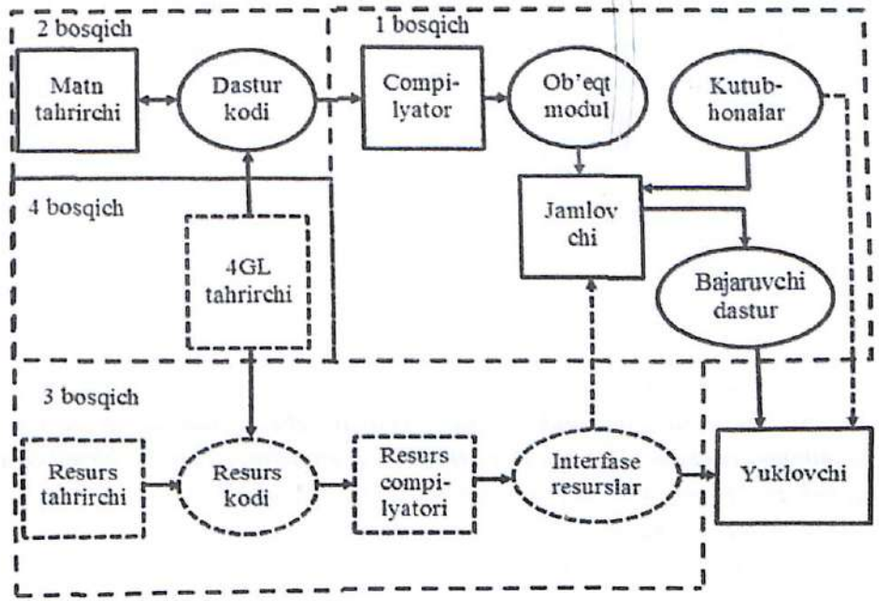
Dasturni algoritmiga bog'liq bo'lmagan holda, uni tashqi ko'rinishini ta'minlab beradigan ma'lumotlar to'plami tadbiqiy dastur resurslari deb nomlanadi. Resurslarga masalan dastur tomondan beriladigan xabarlar, grafik elementlarni joylanishi, ranglari hamda yozuvlari kiradi. Bularni yaratish uchun resurslar tahrirchisi va resurslar kompilyatori paydo bo'ldi.

Ko'rsatilgan barcha dastur – texnik vositalar majmuyisi hozirgi davrda yangi tushuncha, dasturlash tizimi tushunchasini tashkil qiladi.

Zamonaviy dasturlash tizimni tuzilishi

Dastur ta'minotini kodlash, testlash va sozlash uchun ishlatilgan barcha dastur vositalari majmuasi dasturlash tizimiga kiradi.

2.1-rasmda zamonaviy dasturlash tizimlarni umumiy tuzilishi keltirilgan. Rasmda barcha asosiy qismlar va ularni o'zaro bog'lanishlari ko'rsatilgan. Qayta ishlash muhitlarining rivojlanish bosqichlarini tashkil qilgan qismlar guruhi alohida ko'rsatilgan. Rasmdan ko'rinib turibdiki, zamonaviy dasturlash tizimlari bu dastur-texnik vositalarni murakkab majmuyisidir.



2.1-rasm. Dasturlash tizimlarni rivojlanish bosqichlari va umumiy tuzimi.

Hozirgi asosiy rivojlanish tizimlarning do'stona harakati va servis imkoniyatlarini oshirish yo'lida ketmoqda. Bugungi kunda, dastur yaratishga va qayta ishlashga sarflangan kuchlarni kamaytirish muhim ahamiyatga ega bo'lib qoldi.

Hozirgi zamonda dasturlash tizimlariga asosiy kiritilgan yangilikdan biri – to'rtinchi bosqich tili, ya'ni 4GL (four generation language) tilidir va dastur ta'minotini tezkor yaratish RAD (rapid application development) tizimlardir.

4GL tili loyihalash va dastur kodini yaratishda keng qo'llanilmoqda. Bu tilda grafik elementlardan foydalanib interfeys shaklini yaratadi. Shu jarayonda dastur hamda resurs kodiga kerakli o'zgarishlarni avtomatik ravishda generatsiya qilib qo'shadi. Buning uchun yuqori malakali dasturchi bo'lish shart emas. 4GL tillari, dasturlash tizimlarning rivojlanish jarayonida, to'rtinchi bosqich bo'ldi. 4GL tili asosida ta'riflangan dastur yuqori darajali tilga translyatsiya qilinadi va bu matn bilan professional dasturchi ishlaydi. Unga kerakli funksiyalarni qo'shadi.

Bunday usul ishni bo'lish imkoniyatini beradi, loyihachi o'z ishini bajaradi va umumiy konsepsiyani tuzadi. Dizayner interfeysni tashqi ko'rinishiga va dastur tuzuvchi dastur kodini yaratishga javob beradi.

RAD texnologiya dastur yaratishda tayyor komponentalardan foydalanishga imkoniyatlar beradi. Bunday komponentalar har xil sohalarga tegishli bo'lib, har bir foydalanuvchi o'ziga kerakligini ishlatishi mumkin bo'ladi. Ushbu usul kam kuch bilan tez vaqtda dastur yaratishga imkon beradi.

Xulosa

Dastur mahsulotini yaratishda, ya'ni uni qodlash, tekshirish va sozlash jarayonlarida muhim rol o'ynaydi. Har qanday dasturlash tizimning asosida kompilyator turadi. Kompilyator bilan birgalikda quyidagi dasturiy vositalar qo'llanadi: matn tahrirchilari; jamlovchilar; tadbiqiy dastur kutubxonalar; yuklovchilar; sozlovchilar. Oldin dastur yaratish qadamlari alohida buyruqlar orqali bajarilardi, hozirgi zamonda asosiy rivojlanish tizimlarning do'stona harakati va servis imkoniyatlarini oshirish yo'lida ketmoqda. Bugungi kunda, dastur yaratishga va qayta ishlashga sarflangan kuchlarni kamaytirish muhim ahamiyatga ega bo'lib qoldi, shuning uchun dastur yaratish jarayonida 4GL tillar, RAD texnologiyalar qo'llanmoqda.

Nazorat uchun savollar

1. Dasturlash tizimlarning asosiy vazifasini keltiring.
2. Kompilyator qanday ish bajaradi?
3. Kompilyator bilan ishlatiladigan qo'shimcha dastur vositalarini keltiring.
4. Zamonaviy dasturlash tizimining tuzilishini keltiring.

3- BOB. KLASSIK DASTURLASH TIZIMI

Tayanch iboralar: *matn tahrirchi, kompilyator, jamlovchi, yuklovchi, sozlovchi, qism-dasturlar kutubxonasi.*

Dasturlash tizimlaridagi matn tahrirchisining vazifasi

Matn tahrirchisi bu yuqori bosqichdagi tilda dastur kodini yaratish, o'zgartirish va qayta ishlash imkoniyatini beruvchi dasturdir.

Integranlangan yaratish muhitlari paydo bo'lishi bilan matn tahrirchisi shunday muhitlar ichiga kirib ketgan. Bunda dastur kodini qog'ozga yozish shart emas, uni to'g'ridan – to'g'ri terib ketish mumkin. Leksik tahlil dastur matnini terish jarayonida bajarilishi hamda kalit so'zlarni boshqa rangda chiqarish, ochiluvchi yordam menyulari orqali kerakli funksiyalar va boshqa elementlarni dasturga kiritish mumkin. Bunday imkoniyatlar matn tahrirchisini qo'shimcha funksiyalar bilan kengaytirish hisobiga yuzaga keldi.

Natijada matn tahrirchisi dasturlash tizimlarining asosiy qismlaridan biri bo'lib qoldi va u yordamida dasturni yaratish ancha yengillashdi.

Kompilyator vazifasi

Kompilyator bu har qanday dasturlash tizimida asosiy modul bo'lib, yuqori darajadagi tilda yozilgan dastur kodni mashina kodiga o'tkazadigan dastur vositasi hisoblanadi. Biz asosan shu vositani batafsil ko'rib chiqamiz. Kompilyatorsiz dasturlash tizimlarining ma'nosi qolmaydi. Barcha qolgan dasturiy vositalari kompilyatorning ish jarayonini ta'minlash uchun mo'ljallangan. Kompilyator ishlash natijasida obyektda modul hosil bo'ladi. Dasturning obyekt modulida mashina kodidan tashqari o'zgaruvchilar, funksiyalar va boshqa elementlardan biriktirilgan bo'ladi, bunday elementlar obyektlar deb nomlanadi. Bu modul alohida obyekt fayl bo'lib, tashqi xotiraga yozilib qo'yiladi.

Dasturlash tizimlarining birinchi bosqichidan boshlab integrallangan muhitigacha kompilyatorlar qatnashadi. Albatta, zamonaviy dasturlash tizimlarida foydalanuvchi asosan matn tahrirchisi bilan ishlaydi. Kompilyator avtomatik va oshkormas ravishda o'z ishini bajaradi va dasturchi faqat oxirgi natijani ko'radi.

Dasturlash tizimida asosiy kompilyatordan tashqari yana bir nechta kompilyatorlar bo'lishi mumkin. Masalan, assembler tili kompilyatori yoki resurs kodi kompilyatori. Ular bilan ham foydalanuvchi kam ishlaydi.

Kompilyatorni texnik xarakteristikalari natijaviy dasturni yaxshiligiga katta ta'sir qiladi.

Jamlovchi (Link)

Jamlovchi bu, kompilyator tomonidan hosil bo'lgan obyekt fayllarni va kutubxonadagi fayllarni bir-biriga bog'lab natijaviy bajaruvchi dasturni hosil qiluvchi dasturiy vositadir. Obyekt fayl yoki fayllar, toki ularda ishlatilgan modullar unga bog'lanmaguncha alohida ishlamaydi. Jamlovchi barcha modullarni ulab, yagona bajaruvchi fayl hosil qiladi.

Agarda biror qismda-dastur topilmasa jamlovchi xato beradi va bajaruvchi fayl hosil bo'lmaydi. Uning ishlash tamoili quyidagicha: birinchi modulni olib, unga boshlang'ich manzilni beradi va shu modulda chaqirilgan boshqa modullarni ketma-ket bog'lab chiqadi. Yig'ish jarayonida umumiy ismlar jadvali tuziladi.

Kompilyator bir qadamda faqat bitta modulni yaratadi. Keyinchalik obyekt modullar ustidan jamlovchi ishlaydi. Jamlovchining asosiy vazifasi – bir nechta obyekt modullarni bir–biriga bog'lashlan iborat. Bu modullarni bevosita kompilyatordan yoki kutubxonalardan olishi mumkin. Agar obyekt, dastur birligida ta'riflanmasdan va aniqlanmasdan ishlatilsa, u dastur birligiga nisbatan tashqi hisoblanadi. Bunday obyektlar tashqi havola deb hisoblanadi. Jamlovchi barcha tashqi obyektlarni aniqlaydi va ularni boshqa modullardan qidirib umumiy dasturga ulaydi va ularga manzil beradi.

Dastur ishlashdan oldin barcha tashqi havolalarni aniqlash va ular uchun xotira ajratish kerak. Aks holda jamlovchi hatolik to'g'risida habar beradi.

Undan tashqari, jamlovchi obyekt modulda ishlatilgan tashqi nomni va shu nom ta'riflangan obyekt moduldagi nomga mos kelishini nazorat qiladi. Agar obyektlar ismlari mos kelib, lekin semantikasi to'g'ri kelmasa, bunday hatolar qiyin aniqlanadi. Bunday hatolarni oldini olish faqat aniq ulash qoidalarni ishlatilishi kerak.

Jamlovchining vazifasi bu, xotira sohalarini tashkil qilish, ya'ni har bir modul uchun alohida joy ajratish bilan bog'liq. Har bir soha uchun nisbiy manzillar beriladi. Bu modullarni xaqiqiy manzili bajarish oldidan faqat yuklovchi tomondan aniqlanadi. Jamlovchi dasturni tezligini oshirmaydi, lekin u xotirani samarali ishlatishiga ta'sir qilishi mumkin. Masalan, dasturda biror bir kutubxonaga murojaat bo'lsa, u faqat kerakli qismlarni bajariluvchi dasturga qo'shadi.

Xullas, jamlovchining asosiy vazifalari quyidagicha:

- tashqi ismlar bo'yicha modullarni bir biriga bog'lash va yagona bajaruvchi dasturni yaratish;
- obyektlarning nisbiy manzillar jadvalini tashkil qilish;
- kerakli kutubxonalarni bajaruvchi dasturga statik qo'shish:
- dinamik kutubxonalarning funksiyalarini chaqiruvchi nuqtalar manzil jadvalini tashkil qilish.

Yuklovchi va sozlovchilar vazifalari

Obyekt modullar asosan nisbiy manzillar bilan ishlaydi. Jamlovchi ham bajaruvchi faylni nisbiy manzillarda tayyorlaydi. Bunday dasturni bajarishga qo'yganda nisbiy manzillar xaqiqiy manzillarga almashtiriladi. ***Bu jarayon manzillar translyatsiyasi deb nomlanadi va uni yuklovchi dastur vositasi bajaradi.***

Jamlovchining ishi, bajaruvchi dasturni jamlash bilan tugaydi. Natijaviy dasturda barcha manzillar boshlang'ich manzilga nisbatan aniqlangan bo'ladi. Bu dasturda faqat boshlang'ich manzil noma'lumdir. Buni na kompilyator va na jamlovchi bilmaydi, chunki bu ma'lumot faqat bajarish oldidan ma'lum bo'ladi. Jamlovchi dastur obyektlarini faqat nisbiy manzillar bo'yicha joylashtiradi.

Nisbiy manzillarni haqiqiy (absolyut) manzillarga qayta o'zgartirishni yuklovchi bajaradi. Yuklovchilar dasturlash tizimlari ichiga kiritilishi mumkin, lekin ko'pincha ular operatsion tizimga kiradi, chunki u bajaradigan funksiyalar nafaqat dastur ishlaydigan hisoblash tizimining arhitekturasiga bog'liq, balki bu tizimning aniq fizik konfiguratsiyaga (xotira o'lchoviga, protsessor razryadiga) ham bog'liqdir.

Agar dastur kompilyatsiyadan va jamlovchidan keyin bajarishga emas, interpretatorga uzatilsa, unda yuklovchi dasturlash tizimlarning tarkibida bo'lishi shart.

Yuklovchi o'z vazifasini bajarish uchun, jamlovchi tomonidan bajaruvchi fayl boshiga dastur obyektlarini nisbiy manzillar maxsus jadvalini joylashtirishi lozim.

Jadvaldagi ma'lumotlarni qayta ishlagandan so'ng, yuklovchi ba'zi bir buyruqlarga o'zgartirishlar kiritib natijaviy dasturni hosil qiladi.

Jadvalning tuzimi operatsion tizimga va hisoblash tizimning arxitekturasiga bog'liqdir.

Shu sababli, yuklovchi operatsion tizimning tarkibida bo'lgani ma'quldir. Undan tashqari, zamonaviy kompyuterlarda xotirani boshqarish murakkab bo'lishi va shunda manzillarni qayta o'zgartirish har xil xotira tizimiga bog'liq bo'lishi mumkin (segment, segment-sahifa va sahifa ko'rinishda xotirani tashkil qilish). Bu holatlarda faqat operatsion tizim orqali dasturni to'g'ri yuklash mumkin. Shu sababli yuklovchining funksiyalari operatsion tizimga berilgan.

Dastur bajarilishi bilan bog'lik yana bir vosita bor - bu sozlovchidir.

Sozlovchi-bu tadbiqiy dasturni ishlash jarayonini tekshiradigan va kuzatishini bajaradigan dasturiy vositadir.

Sozlovchi quyidagi funksiyalarni bajaradi:

- ketma-ket va qadamma-qadam natijaviy dasturni bajaradi;
- dasturni nazorat nuqtasigacha (to'xtash manzili) bajaradi va to'xtab turadi;
- dastur egallagan xotiradagi berilganlarni ko'radi va kerak bo'lsa o'zgartirishlar kiritadi.

Avvallari sozlovchilar alohida dasturiy vosita edi, ular faqat mashina kodidagi dasturlar bilan ishlardi.

Integranlangan muhitlarda sozlovchi dasturlar uning bir qismiga aylangan bo'lib, bevosita yuqori darajadagi tillar operatorlari bilan ishlaydi va har qadamda shu tildagi operatorlarni ko'rsatadi, va shu tilda yozilgan o'zgaruvchilarni ko'rsa bo'ladi.

Zamonaviy sozlovchilar rivojlangan interfeysga ega va ular matn tahriri bilan chuqur bog'langan.

Qism-dasturlar kutubxonasi

Qism-dasturlar kutubxonasi bu dasturlash tizimni bir qismi bo'lib, unda namunaviy qism-dasturlar, funksiyalar va komponentalar joylashadi.

Qism-dasturlar kutubxonasi ikki qismdan tashkil topgan. Birinchi qismida, obyekt faylar joylashgan bevosita qism-dasturni kodi, ikkinchisida, kutubxonaga tegishli funqtsiyalar prototipi, o'zgaruvchilarni va o'zgarmaslarni tavsiflari joylashadi.

Kutubxonadagi obyekt kodi jamlovchi orqali bajaruvchi dasturga qo'shiladi.

Kutubxonani sarloxa fayli kompilyatorga kutubxona turkimi to'g'risida ma'lumot beradi.

Dasturlash tizimida bir nechta kutubxona bo'lishi mumkin, ulardan biri asosiy bo'lib, unda tilga tegishli na'munaviy funksiyalar va boshqa muhim berilganlar saqlanadi. Bu kutubxona albatta kompilyator bilan birga ishlatiladi. Qolgan kutubxonalar dasturchi tomondan dasturga qo'shiladi.

Ba'zi bir tillar, faqat keng kutubxonasi mavjud bo'lgani uchun haligacha ishlatilib kelinmoqda. Masalan, Fortran tili juda katta matematik va fizik funksiyalar kutubxonasiga ega.

Zamonaviy operatsion tizimlar bajaruvchi dasturlarga dinamik kutubxonalarni ulashga imkon berdi.

Dinamik kutubxonalar modullari dasturga ishlash jarayonida qo'shiladi. Bu ancha qulayliklar yaratadi. Chunki asosiy dasturning kodi qiskaradi va bitta kutubxonani bir necha dasturlar ishlatish imkoniyati paydo bo'ladi.

Lekin, dinamik kutubxonalarni ishlatishda kamchiliklar ham bor. Masalan, kutubxona tuzuvchilar kodni o'zgartirib qo'yishi va natijada dastur noto'g'ri javob berishi mumkin. Bu kamchilikni oldini olish uchun, bir tomondan dastur tuzuvchilar funksiyani faqat ko'rsatilgan ko'rinishda ishlatishi kerak va ikkinchi tomondan kutubxonani yaratuvchi funksiyani mantig'ini o'zgartirmasligi kerak. Bu qoidani ikki taraf xam buzmaslikka harakat qilishi kerak.

Xulosa

Dastur avvol biror bir matn tahrirchisida terilishi kerak, zamonaviy dasturlash tizimlarda matn tahrirchisi bevosita tizim tarkibida bo'lib, uning funksiyasi ancha kengaytirilgan. Kompilyator dastlabki dastur kodini sintaktik tahlildan o'tkazib mashina kodini yaratadi. Dasturda hatoliklar mavjud bo'lsa, ular to'g'risida habar beradi. Jamlovchi

kompilyator tomonidan hosil bo'lgan obyekt fayllarni va kutubxonadagi fayllarni bir-biriga bog'lab natijaviy bajaruvchi dasturni hosil qiladi. Jamlovchi bajaruvchi faylni nisbiy manzillarda tayyorlaydi. Bunday dasturni bajarishga qo'yganda nisbiy manzillar xaqiqiy manzillarga almashtiriladi. Bu jarayon manzillar translyatsiyasi deb nomlanadi va uni yuklovchi dastur vositasi bajaradi. Sozlovchi tadbiqiy dasturni ishlash jarayonini tekshiradi va kuzatadi. Dastur yaratishda qism-dasturlar kutubhonalari keng ishlatiladi.

Nazorat uchun savollar

1. Matn tahrirchisini vazifasi nimadan iborat?
2. Kompilyatorning vazifasi nimadan iborat?
3. Jamlovchining vazifasi nimadan iborat?
4. Yuklovchining vazifasini keltiring.
5. Qism-dasturlar kutubxonasini vazifasi nimadan iborat?

4-BOB. TRANSLYATOR, KOMPILYATOR VA INTERPRETATOR

Tayanch iboralar: *translyator, kompilyator, interpretator*

Translyatorning formal ta'rifi

Translyator-bu kiruvchi tilda yozilgan dastlabki dasturni ekvivalent chiquvchi tildagi natijaviy dasturga o'tkazuvchi dasturdir.

Ko'rinib turibdiki, bu ta'rifda uch marta dastur so'zi ishlatilmoqda. Bu xato emas. Birinchidan, xaqiqatdan ham translyatorning o'zi mashina kodida yozilgan dasturdir. Translyator kompyuterdagi tizimli dasturiy ta'minotini bir qismi.

Ikkinchidan, translyatorga kiruvchi berilganlar sifatida dastlabki dasturning matni ishlatiladi. Ko'pincha bu belgilardan iborat bo'lgan matn faylidir. Bu matn kiruvchi tilni qoidalari bo'yicha yozilgan bo'ladi.

Uchinchidan, translyator ishlash natijasida hosil bo'lgan chiquvchi til qoidalarida yozilgan, chiquvchi fayl matni ham dasturdir. Translyator ta'rifida muhim shart bor, bu kiruvchi va chiquvchi dasturlarning ekvivalentligi, ya'ni dastlabki va natijaviy dasturlarni ma'nosi o'zgarmasligidir.

Translyatorni yaratish uchun avvalo kiruvchi va chiquvchi tillarni tanlash lozim. Translyatorni tarjimon deb hisoblasak ham bo'laveradi. Masalan, C++ tildagi dasturni Pascal tiliga tarjima qiluvchi dasturni translyator deb nomlash mumkin.

Agarda kiruvchi tildagi dastlabki dasturda xatoliklar bo'lmasa, translyator ishlash natijasida chiquvchi tilda natijaviy dasturni yaratadi. Aks holda translyator ishlash natijasida dastlabki dasturdagi xatoliklar to'g'risidagi xabarlarni chiqaradi (nima hato, qaysi satrda).

Kompilyator ta'rifi. Kompilyator va translyatorning farqi

Translyator tushunchasidan tashqari, unga ma'nosi yaqin bo'lgan kompilyator tushunchasi ham keng ishlatiladi.

Kompilyator bu kiruvchi tildagi dastlabki dasturni mashina tilidagi yoki assembler tilidagi ekvivalent obyekt dasturga tarjima qiluvchi translyator.

Kompilyatorni translyatordan farqi shundaki, kompilyatorda natijaviy dastur mashina tilida yoki assembler tilida bo'ladi. va

translyatorda umumiy holda natijaviy dastur har qanday tilda bo'lishi mumkin. Masalan, Pascaldan C++ga . Shuning uchun har qanday kompilyatorni, biz translyator deb atashimiz mumkin, lekin teskarisi o'rinli emas.

Kompilyatorni natijaviy dasturi obyekt dastur, obyekt kod yoki obyekt modul deyiladi. Natijaviy dastur yozilgan fayl obyekt fayldir. Obyekt dastur va bajaruvchi dastur orasida farq mavjud, birinchisi aniq manzillarga bog'lanmagan va bevosita kompyuterda bajarib bo'lmaydi.

Kompilyator bu eng keng tarqalgan translyatordir. Shu sababli biz asosan kompilyatorni ishlash tamoilini o'rganamiz.

Kompilyatorlar va translyatorlar odamlar tomonda yaratiladi. Umuman olganda, ularni mashina tilida yaratish mumkin, lekin bu juda uzoq va murakkab jarayon bo'ladi. Ko'pincha kompilyatorlar, boshqa birorta yuqori bosqich tilida yoziladi, yoki shu kompilyatorni oldingi naqlida yaratiladi.

Interpretator ta'rifi. Interpretator va translyatorning farqi

Bir-biriga o'xshagan kompilyator va translyator tushunchalaridan tashqari, ulardan tamoyil bo'yicha farq qiladigan interpretator tushunchasi ham mavjud.

Interpretator bu dastlabki tilda yozilgan kiruvchi dasturni o'zlashtirib, uni buyruqlarini muntazam bajaruvchi dasturdir.

Translyatordan tamoyil bo'yicha farqi shundaki, u natijaviy dastur yaratmaydi. Interpretator, dastlabki dastur matnini tahlil qiladi va kodning ma'nosiga qarab uni bajaradi. Agar joriy buyruqda xato bo'lsa, xabar berib ishni to'xtatadi.

Albatta u dastlabki dasturning har bir buyrug'ini mashina tiliga o'tkazadi va keyin bajaradi. Lekin, mashina kodini foydalanuvchi ko'rmaydi va uning kodga kirish imkoniyati yo'q. Bu mashina kodlari interpretator tomondan yaratiladi, bajariladi va o'chiriladi. Foydalanuvchi faqat natijani ko'rishi mumkin holos.

Translyator, kompilyator va interpretatorlarning vazifalari

Birinchi avlod EHM davrida, dasturlar bevosita mashina tilida yozilar edi. Bu juda qiyin va katta mexnat talab qiluvchi mashaqqatli ish edi. Lekin inson tilini bemalol tushunadigan kompyuter hanuzgacha

yaratilmadi va yaqin orada yaratilmasa ham kerak.

Shuning uchun dastur ta'minotining rivojlanishi yangi kompilyatorlar yaratish yo'lida ketmoqda.

Dastlabki kompilyatorlar Assembler tili uchun yaratilgan edi. Mashina tilini buyruqlari insonga tushunarli bo'lgan belgilar tarzda, ya'ni, mnemokod orqali Assembler tilida yoziladi. Assemblerda dasturni yaratish yengillashdi. Bunday dasturni mashina tiliga o'tkazadigan kompilyator ancha sodda bo'lib, xozirgacha ishlatilmoqda.

Keyingi bosqichda yuqori darajali tillar paydo bo'ldi. Bu tillarda tabiiy tillardagi so'zlar ishlatiladi, lekin bunday so'zlar soni ko'p emas. Tabiiy tildan farqi shundaki undagi gaplar aniq qoidalar bo'yicha tuziladi. Bu qoidalar formal grammatika nazariyasi bo'yicha ta'riflanadi. Yuqori bosqich tillari dasturlash jarayonini ancha soddalashtirdi va yengillashtirdi, ammo biroq dasturni hamma ham yoza olmasdi.

Oldin bunday kompilyatorlar bir yoki ikkita edi halos, keyinchalik o'nta, so'ngra esa 100dan ortiq bo'ldi.

Kompilyatorlar, interpretatorga nisbatan sodda va natijaviy dastur tezroq ishlaydi. Lekin kompilyatordan chiqgan dasturning kamchiligi ham bor. Bunday dastur aniq mashinani arxitekturasiga bog'langan. Hozir S, C++, Paskal tillar kompilyatorlari rivojlanmoqda. Interpretator faqat dastlabki til bilan bog'liq, va bitta dastlabki dastur har xil mashinalarda ishlashi mumkin. Interpretatorlar uncha rivojlanmagan edi,. Ma'lum interpretatorlardan biri Beysik edi. Bugungi kunda Internet rivojlanishi bilan interpretatorlar tez rivojlana boshladi. Zamonaviy interpretatorlar bu Java, JavaScript va C# tillar uchun yozilgan. Masalan Java tili interpretatori dastlabki tilda yozilgan dasturni oraliq sodda tilga o'tkazadi (Jet tili) va bu tildagi dastur internet orqali tarqatiladi. Har bir kompyuterda mavjud bo'lgan JVM dasturi Jetda yozilgan dasturni bagaradi.

Xulosa

Translyator-bu kiruvchi tilda yozilgan dastlabki dasturni ekvivalent chiquvchi tildagi natijaviy dasturga o'tkazuvchi dasturdir. Kompilyator bu kiruvchi tildagi dastlabki dasturni mashina tilidagi yoki assembler tilidagi ekvivalent obyekt dasturga tarjima qiluvchi translyator. Interpretator bu dastlabki tilda yozilgan kiruvchi dasturni o'zlashtirib, uni buyruqlarini

muntazam bajaruvchi dasturdir. Kompilyatorlar, interpretatorga nisbatan sodda va natijaviy dastur tezroq ishlaydi. Lekin kompilyatordan chiqgan aniq mashinani arxitekturasiga bog'langan. Interpretator faqat dastlabki til bilan bog'liq, va bitta dastlabki dastur har xil mashinalarda ishlashi mumkin.

Nazorat uchun savollar

1. Translyatorning ta'rifini keltiring.
2. Kompilyatorning ta'rifini keltiring.
3. Interpretarorning ta'rifini keltiring.
4. Translyator, kompilyator va interpretatorlarni farqi nimada?

5 - BOB. KOMPILYATORNING TUZILISHI

Tayanch iboralar: *kompilyator bosqichlari, kompilyatorning umumiy chizmasi, tahlil fazasi, sintez fazasi, leksik tahlil, sintaktik tahlil, semantik tahlil, dasturni ichki tasviri, identifikatorlar jadvali, dasturni optimallashtirish, xotirani taqsimlash, kod generatsiyasi, bir marotaba o'tishli kompilyator, ko'p marotaba o'tishli kompilyator.*

Kompilyator bosqichlari. Kompilyator ishining umumiy chizmasi

5.1-rasmda kompilyatorni ishlash umumiy chizmasi keltirilgan. Bunda to'liq chiziqlar ishlash tartibini, punktir chiziqlar ma'lumot oqimini ko'rsatadi. Undan ko'rinib turibdiki, kompilyatsiya jarayoni ikki asosiy bosqichdan iboratdir: tahlil va sintez, optimallashtirish fazasi shart emas.

Kompilyator formal tillar nuqtayi nazaridan ikkita asosiy funksiyani bajaradi:

Birinchidan, u kiruvchi tilni aniqlovchisi, ya'ni kiruvchi tilda yozilgan belgilar zanjirini qabul qilib, bu zanjir tilga mos kelishini aniqlaydi. Undan tashqari tilni qaysi qoidalar asosida bu zanjir tuzilganligini topib beradi. Zanjir foydalanuvchi tomondan yaratiladi. Bu qismni tahlil bosqichi bajaradi.

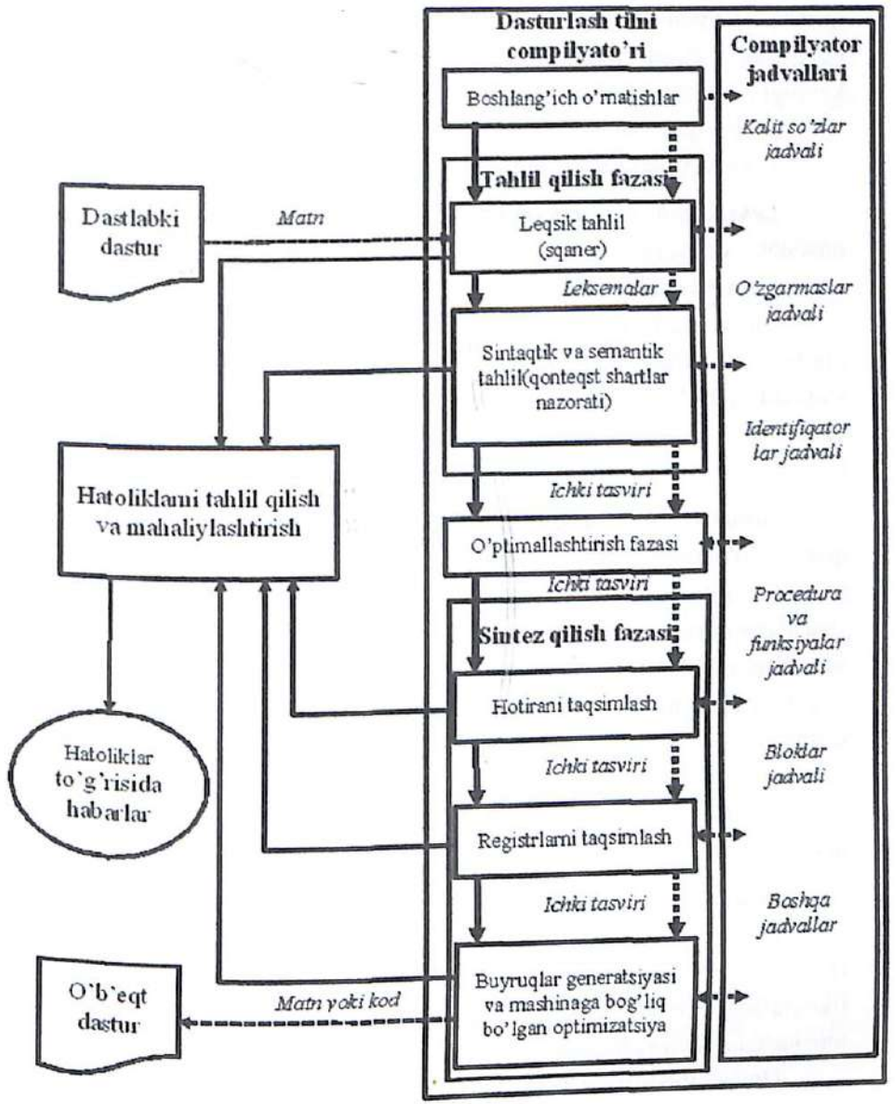
Ikkinchidan, kompilyator natijaviy dasturni yaratishi kerak. U mashina tilining ma'lum qoidalari bo'yicha chiquvchi, tilda chiquvchi zanjir tuzib beradi. Buni sintez bosqichi bajaradi.

Undan tashqari kompilyatorda xatolarni tahlil qilish va to'g'rilash qismi ham bor. Bu qism xato to'g'risida batafsil ma'lumot berishga harakat qiladi va iloji bo'lsa to'g'ri variantlarni taklif qiladi.

Har bir bosqich o'z navbatida bir necha mayda vazifalardan iboratdir va ular kompilyatsiya fazalari deb nomlanadi.

Yendi har bir kompilyatsiya fazalarni (qismlarni) va elementlarini qisqa ta'rifini berib o'tamiz.

Kompilyator jadvallari. Dasturni tahlil qilish jarayonida, undagi ta'riflardan, funksiyalardan, bloklarda hamda boshqa tuzilmalardan ma'lumotlar olinib jadvallarga tushiriladi va bu ma'lumotlar keyingi fazalarda ishlatiladi. Masalan, biror bir identifikator uchrasa, u to'g'risida ma'lumot identifikator jadvaliga tushadi va har bir faza ushbu identifikator uchun jadvalga qo'shimcha ma'lumot kiritadi.



5.1-rasm. Kompilyatorning ishlashidagi umumiy chizmasi.

Boshlang'ich o'rnatish. Dastur ustidan ishni boshlashdan oldin kompilyator jadvallarni boshlang'ich qiymatlar bilan to'ldiradi, ba'zi bir tartibotlarni o'rnatadi va shunga o'xshagan boshlang'ich ishlarni bajaradi.

Tahlil bosqichi. Tahlil bochqichida dastlabki dastur matni tahlil qilinadi hamda identifikatorlar jadvallari tuziladi va to'ldiriladi. Bu bosqichda ishlash natijasida kompilyator uchun tushunarli bo'lgan va keyingi bosqichda ishlatiladigan dasturning ichki tasviri hosil bo'ladi.

Bu bosqich uchta fazadan iborat:

Leksik, sintaktik va semantik tahlil.

Leksik tahlilchi (skaner). Leksik tahlil fazasini asosiy vazifasi dastlabki dasturni o'qib, uni leksemalarga ya'ni bo'linmas alohida so'zlarga ajratib berishdir.

Bu fazaga kiruvchi sifatida dastur matni bo'ladi. Chiquvchi ma'lumot kodlangan leksemalar ketma-ketligi bo'lib chiqadi va u sintaktik tahlilchiga uzatiladi. Dasturlash tillarida leksemaga - sonlar. identifikatorlar, hizmatchi so'zlar va ajratuvchilar kiradi. Undan tashqari kerak bo'lmagan belgilar (izohlar) olib tashlanadi.

Sintaktik va semantik tahlilchilar. Bu fazalar kompilyatorda asosiy qism hisoblanadi. Bu fazalarda dasturni to'g'ri yozilganligi aniqlanadi. ya'ni til sintaktik konstruksiyalarga mos kelishi tahlil qiladi. Masalan. shartli operator to'g'ri ketma – ketlikda yozilganligi.

Semantik tahlil, berilgan dastlabki dastur semantika nuqtayi nazardan to'g'ri yozilganligini aniqlaydi. Masalan, bir xil identifikatorlar e'lon qilinmaganligi, o'zgaruvchi ishlatilishidan oldin e'lon qilinganligi, xamda funksiyalarni parametrlari to'g'ri ishlatilganligi va hokazo. Bular kontekst shartlarga kiradi. Bu bosqich ishlashi natijasida jadvallarga qo'shimcha ma'lumotlar yoziladi va dasturni ichki tasviri hosil bo'ladi.

Dastlabki dasturning ichki tasviri.

Kompilyatorda dasturni ichki tasviri asosan dasturni keyingi qayta ishlash jarayoniga bog'liq. Ba'zi bir ichki tasvirlar dastur tuzimini fiksirlashga mos keladi, boshqalari optimallashtirish jarayoniga moslashgan, uchinchilari obyekt kodini yaratishga qulaydir.

Optimallashtirish fazasi. Optimallashtirish bu kompilyatorni muhim masalasidan biri. Yuqori bosqich tillarda samarali dasturlar tuzish uchun ularni albatta optimallashtirish lozim. Optimallashtirishni ikkita strategiyasi bor: dasturni tezligini oshirish va dasturni hajmini qisqartirish. Bu strategiyalarda ishlatilgan usullar ko'pincha qarama-qarshi, lekin o'hshash qismi ham yo'q emas.

Sintez bosqichi.

Kompilyatorni ikkinchi asosiy ishi bu natijaviy dasturni generatsiya qilishdir. Sintez bosqichida. ichki tasvir asosida va jadvallardagi ma'lumotlarga ko'ra, natijaviy dastur kodi yaratiladi. Ushbu natijada obyekt kod hosil bo'ladi, yoki assembler tilida matn tayyorlanadi.

Xotira va registrlarni taqsimlash fazasi.

Xotirani taqsimlash jarayoni kompilyatorda albatta bo'lishi shart. Bu jarayonda xotira bloklarga ajratiladi, registrlarga kerakli qiymatlar beriladi. Berilganlar xotira bloklariga joylashtiriladi va ularning chegaralari ma'lum birlikka keltiriladi.

Buyruqlarni generatsiya qilish va mashinaga bog'liq optimallashtirish fazalari.

Bu fazalarda kompilyator dasturning ichki tasvir ustidan yakuniy qayta o'zgartishlar bajaradi va dasturni mashina kodiga yoki assembler tiliga o'tkazadi.

Interpretatorda bu qism ichki tasvirdagi dasturni bevosita bajaradigan dasturga almashtiriladi. Lekin faqat kompilyatordan chiqgan kod to'g'ridan – to'g'ri ishlamasligi mumkin. Zamonaviy Si. Si++, Java tillar uchun kompilyator faqat dasturni bir qismini kompilyatsiya qiladi. Qolgan tayyor qismlar yoki tizimli dasturlashni boshqa komponentalar maxsus bog'lovchi dastur orqali yig'iladi.

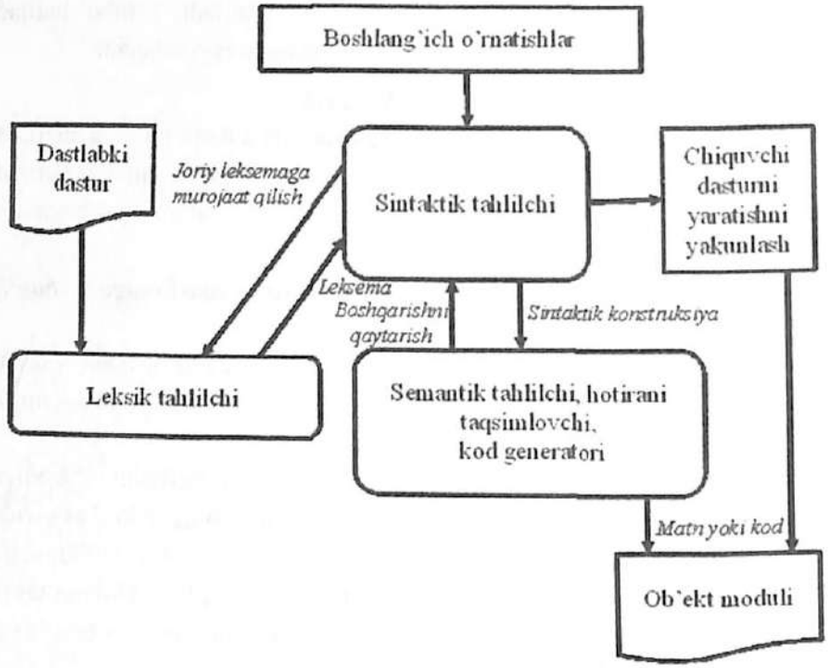
Kod generatsiya fazasida bevosita natijaviy kod yaratiladi. sintez bosqichida bu asosiy fazadir. Natijaviy dastur kodini yaratishdan tashqari, bu fazada mashinaga bog'liq optimallashtirish masalasi ham yechiladi. Bu jarayon dasturni tez va samarali ishlashiga olib keladi.

Bir marotaba va ko'p marotaba o'tishli kompilyator

Kompilyatsiya jarayoni bir nechta fazada bo'lishligini aytib o'tdik, aniq kompilyatorda bu fazalar farq qilishi mumkin: ba'zi birlari bo'linishi mumkin, ba'zi birlari qo'shilishi mumkin. Fazalarni bajarish ketma-ketligi ham har xil bo'lishi mumkin. Bunga bir necha narsa ta'sir qiladi. Shu jumladan xotira hajmi, translyator tezligi, optimallashtirish jarayoning mavjudligi. Asosiy bajarish usullari ikkita - bu bir marotaba yoki ko'p marotaba o'tish usuli.

Bir marotaba o'tish usulida (5.2-rasm), barcha fazalar birdaniga

ishlaydi. Leksik tahlilchi matndan joriy leksemani ajratib sintaktik tahlilchiga uzatadi. Sintaktik tahlilchi leksemani tahlil qilib, kerak bo'lsa, leksik tahlilchidan yana leksema so'raydi.



5.2 -rasm. Bir marotaba o'tishli kompilyator

Va natijada dasturni bir parchasi uchun ichki tasvirini taxlaydi va semantik tahlilchiga uzatadi.

Semantik tahlilchi parchaga ko'ra kerakli o'zgartirishlarni identifikator jadvallariga kiritadi va kod generatoriga boshqarishni uzatadi. Generator o'z navbatida parcha uchun obyekt kodini yaratadi va yana boshqarishni sintaktik tahlilchiga uzatadi. Shunday qilib, bir o'tishda parchaning obyekt kodi yaratiladi. Ba'zi bir kompilyatorlarda asosiy faza sintaktik tahlilchi bo'lib, ish undan boshlanadi.

Sintaktik tahlilchi, navbatdagi leksemani olish uchun leksik tahlilchiga (skanerga) murojaat qiladi va joriy konstruksiyani tashkil qilgandan so'ng, keyingi fazalarga uzatadi. Ular o'z qismini bajargandan keyin yana boshqarishni sintaktik tahlilchiga uzatadi.

Bir marotabali o'tish, asosan sodda tillar uchun qulaydir. Masalan

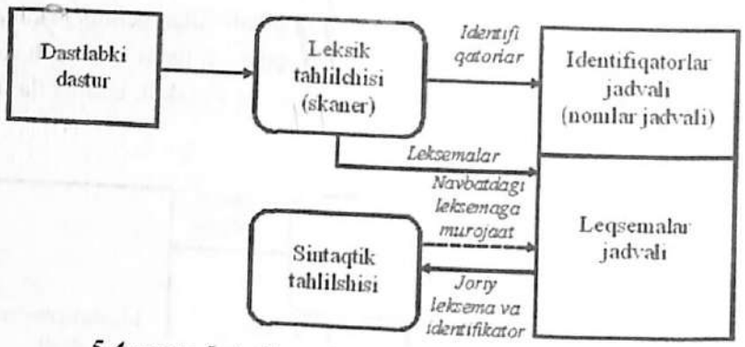
Beysik va Paskal tillari uchun. Lekin murakkab tillar uchun yaramaydi. Masalan, C++ kabi tillarda, ko'pmarotaba o'tishli kompilyatorlar ishlatiladi. Bir marotaba o'tish usulida leksik va sintaktik tahlilchilar (5.3-rasm) parallel ravishda ishlaydilar.

5.3-rasm. Sintaktik tahlilchi boshchiligida tahlil

Ko'p marotaba o'tishli kompilyatorlarda har bir faza alohida ishlab, o'z natijasini tashqi xotirada saqlab qoladi. Keyingi faza oldingi faza natijasini qayta ishlab, yana faylga yozib qo'yadi va hokazo. Obyekt faylini yaratguncha (5.1- rasm).

Ko'pincha uch marotaba o'tishli kompilyatorlar ishlatiladi. Unda birinchi o'tishda leksik tahlilchi, ikkinchi o'tishda sintaktik hamda semantik tahlilchilar, nihoyat uchinchi o'tishda kod generatsiya va optimizatsiya fazalari ishlaydi. Kiruvchi til qancha murakkab bo'lsa, shuncha o'tishlar soni ko'payadi. Ko'p marotabali o'tish usulida, leksik va sintaktik tahlilchilar ketma – ket ishlaydilar (5.4- rasm).

Kiruvchi tilni qayta o'zgartirish. Leksik tahlilchining asosiy vazifalaridan biri - bu leksemani dastur matndan ajratish va uni belgilash. Leksik tahlilchi asosida chekli avtomat turadi. Bu avtomat kiruvchi zanjirdan leksemani ajratib, turini aniqlaydi va uni kod sifatida qaytaradi. Lekin leksik tahlilchini vazifalari, avtomatga nisbatan keng. U ajratilgan leksemalarni mos jadvallarga yozishi kerak. Bu jarayonda joriy leksema jadvalda bor yoki yo'qligini aniqlashi. Undan tashqari matndan ortiqcha belgilarni olib tashlashi va nihoyat xatolik ro'y bersa, uni joyini ko'rsatishi kerak.



5.4-rasm. Leksik va sintaktik tahlilchilar munosabati.

Ko'p marotaba o'tishli kompilyatorlarda leksik tahlilchi dastur dastlabki kodini, qayta o'zgartirilgan shaklini, leksemalar jadvaliga o'tkazib tashqi xotiraga yozib qo'yishi kerak.

Xulosa

Komppilyator ikki bosqichdan iborat: bu tahlil va sintez bosqichlari. Tahlil bosqichi uchta fazadan iborat: leksik , semantik va sintaktik tahlillar. Bu bosqich ishlash natijasida obyetlar to'g'risida ma'lumotlar saqlaydigan jadvallar yaratiladi hamda dasturning ichki tasviri hosil bo'ladi. Hosil bo'lgan ichki tasvirdagi dastur optimallashtiriladi. Sintez bosqichida dastur uchun kerak bo'lgan hotira, registrlar taqsimlanadi. so'ng buyruqlar generatsiyasi bajarilib obyekt kodi hosil bo'ladi. Kompilyator ishlash jarayonida ko'p maratoba o'tishi yoki bir marotaba o'tish usullari qo'llanishi mumkin. Undan tashqari kompilyator ishni leksik tahlilchidan yoki sintaktik tahlilchidan boshlashi mumkin.

Nazorat uchun savollar

1. Analiz fazasiga qaysi bosqichlar kiradi?
2. Sintez fazasiga qaysi bosqichlar kiradi?
3. Leksik tahlil bosqichini vazifasi nimadan iborat?
4. Sintaktik tahlilchini vazifasini keltiring.
5. Kompilyator jadvali nima uchun kerak?
6. Dasturni ichki tasviri nimaga kerak?
7. Optimallashtirish jarayoni nima qiladi?
8. Xotirani taqsimlash jarayoni nimadan iborat?
9. Kod generatsiyasi nima vazifani bajaradi?

6 BOB. SEMANTIK TAHLIL ASOSLARI

Tayanch iboralar: *semantik kelishuvlar, kontekst shartlar, dasturlash qoidalari, nomlar sohasi.*

Semantik tahlil vazifasi

Semantik tahlilchi oldingi bosqichlarning barcha natijasidan to'liq foydalanadi. Unga leksik tahlilchi tomonidan tuzilgan identifikatorlar jadvallari, sintaktik tahlilchi tomonidan, til konstruksiyalarining ichki tasviri uzatiladi. Semantik tahlil o'z navbatida ikki bosqichga bo'linishi mumkin. Birinchi bosqichni sintaktik tahlilchi bilan parallel bajarishi mumkin. Ikkinchi bosqichni esa obyekt kodning generatsiyasidan oldida bajarishi mumkin. Birinchi bosqich, sintaktik tahlilchi navbatdagi til konstruksiyasini tahlil qilgandan so'ng bajariladi. Ikkinchi bosqich barcha dasturni to'liq tahlil qilib chiqadi.

Semantik tahlilchi asosiy vazifasi quyidagilardan iborat:

- kiruvchi dasturda, semantik kelishuvlar va kontekst shartlarni bajarilishini tekshirish;
- kiruvchi tilni semantikasiga tegishli bo'lgan, qo'shimcha operatorlarni ichki tasvirga qo'shish;
- kiruvchi tilga tegishli bo'lmagan, semantik me'yorlarni tekshirish.

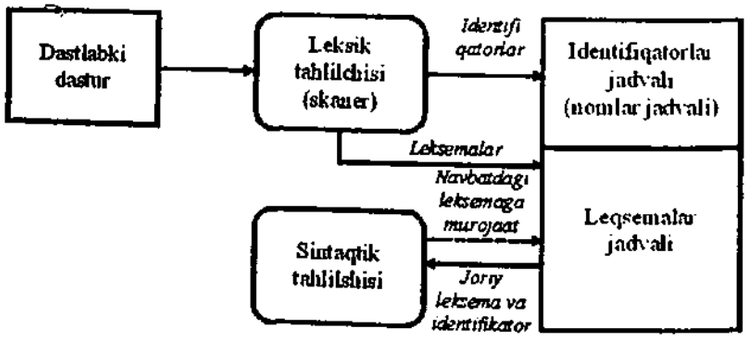
Semantik tahlilchi tomonidan bajarilgan tekshirishlar, **semantik tekshirishlar** deb nomlanadi.

Kontekst shartlarni tekshirish

Operatorlar yozilishidan, uni semantik to'g'ri yoki noto'g'riligini aytib bo'lmaydi. Semantik kelishuv va kontekst shartlarni tekshirishda, kiruvchi dastlabki dastur tilni semantik qoidalariga qanoatlanishini aniqlaydi. Bunda y qoidalar barcha tillarda mavjud va ularni sintaktik tahlil jarayonida tekshirib bo'lmaydi. Chunki sintaktik tahlilchi, dastlabki zanjirni faqat sintaktik nuqtayi nazardan to'g'ri yozilganligini tekshiradi.

Semantik tekshirishlarga quyidagilar kiradi:

1. Turlarni tekshirish. Agar biror bir amaldagi operandlar turi mos kelmasa, kompilyator xatolik to'g'risida xabar berishi lozim. Masalan, ko'rsatkichlarni qo'shishda.



5.4-rasm. Leksik va sintaktik tahlilchilar munosabati.

Ko'p marotaba o'tishli kompilyatorlarda leksik tahlilchi dastur dastlabki kodini, qayta o'zgartirilgan shaklini, leksemalar jadvaliga o'tkazib tashqi xotiraga yozib qo'yishi kerak.

Xulosa

Komppilyator ikki bosqichdan iborat: bu tahlil va sintez bosqichlari. Tahlil bosqichi uchta fazadan iborat: leksik , semantik va sintaktik tahlillar. Bu bosqich ishlash natijasida obyetlar to'g'risida ma'lumotlar saqlaydigan jadvallar yaratiladi hamda dasturning ichki tasviri hosil bo'ladi. Hosil bo'lgan ichki tasvirdagi dastur optimallashtiriladi. Sintez bosqichida dastur uchun kerak bo'lgan hotira, registrlar taqsimlanadi, so'ng buyruqlar generatsiyasi bajarilib obyekt kodi hosil bo'ladi. Kompilyator ishlash jarayonida ko'p maratoba o'tishi yoki bir marotaba o'tish usullari qo'llanishi mumkin. Undan tashqari kompilyator ishni leksik tahlilchidan yoki sintaktik tahlilchidan boshlashi mumkin.

Nazorat uchun savollar

1. Analiz fazasiga qaysi bosqichlar kiradi?
2. Sintez fazasiga qaysi bosqichlar kiradi?
3. Leksik tahlil bosqichini vazifasi nimadan iborat?
4. Sintaktik tahlilchini vazifasini keltiring.
5. Kompilyator jadvali nima uchun kerak?
6. Dasturni ichki tasviri nimaga kerak?
7. Optimallashtirish jarayoni nima qiladi?
8. Xotirani taqsimlash jarayoni nimadan iborat?
9. Kod generatsiyasi nima vazifani bajaradi?

6 BOB. SEMANTIK TAHLIL ASOSLARI

Tayanch iboralar: *semantik kelishuvlar, kontekst shartlar, dasturlash qoidalari, nomlar sohasi.*

Semantik tahlil vazifasi

Semantik tahlilchi oldingi bosqichlarning barcha natijasidan to'liq foydalanadi. Unga leksik tahlilchi tomonidan tuzilgan identifikatorlar jadvallari, sintaktik tahlilchi tomonidan, til konstruksiyalarining ichki tasviri uzatiladi. Semantik tahlil o'z navbatida ikki bosqichga bo'linishi mumkin. Birinchi bosqichni sintaktik tahlilchi bilan parallel bajarishi mumkin. Ikkinchi bosqichni esa obyekt kodning generatsiyasidan oldida bajarishi mumkin. Birinchi bosqich, sintaktik tahlilchi navbatdagi til konstruksiyasini tahlil qilgandan so'ng bajariladi. Ikkinchi bosqich barcha dasturni to'liq tahlil qilib chiqadi.

Semantik tahlilchi asosiy vazifasi quyidagilardan iborat:

- kiruvchi dasturda, semantik kelishuvlar va kontekst shartlarni bajarilishini tekshirish;
- kiruvchi tilni semantikasiga tegishli bo'lgan, qo'shimcha operatorlarni ichki tasvirga qo'shish;
- kiruvchi tilga tegishli bo'lmagan, semantik me'yorlarni tekshirish.

Semantik tahlilchi tomonidan bajarilgan tekshirishlar, **semantik tekshirishlar** deb nomlanadi.

Kontekst shartlarni tekshirish

Operatorlar yozilishidan, uni semantik to'g'ri yoki noto'g'riligini aytib bo'lmaydi. Semantik kelishuv va kontekst shartlarni tekshirishda, kiruvchi dastlabki dastur tilni semantik qoidalariga qanoatlanishini aniqlaydi. Bunda y qoidalar barcha tillarda mavjud va ularni sintaktik tahlil jarayonida tekshirib bo'lmaydi. Chunki sintaktik tahlilchi, dastlabki zanjirni faqat sintaktik nuqtayi nazardan to'g'ri yozilganligini tekshiradi.

Semantik tekshirishlarga quyidagilar kiradi:

1. Turlarni tekshirish. Agar biror bir amaldagi operandlar turi mos kelmasa, kompilyator xatolik to'g'risida xabar berishi lozim. Masalan, ko'rsatkichlarni qo'shishda.

2. Boshqarish amallarini tekshirish. Boshqarishni faqat ruhsat berilgan joyga o'tkazish mumkin. Masalan, C++ tilida break operatori faqat sikl yoki shartli tarmoq operatorlar tanasida uchrashi mumkin. Boshqa joyda ishlatilishi man etiladi.

3. Yagonaligini tekshirish. Ma'lum holatlarda obyekt bir marta uchrashi lozim. Masalan, case nishoni nomi noyob bo'lishi kerak, yoki enum sanagichdagi elementlar qaytarilmasligi kerak.

4. Nomlar bilan bog'langan tekshirishlar. Ba'zi bir hollarda bitta nom, ikki yoki undan ko'p marta ishlatilishi kerak. Shunda kompilyator bir xil nomlar ishlatilganligini tekshirish kerak. Masalan, Ada tilida protsedura, blok yoki sikl boshida va oxirida bir hil nomda bo'lishi kerak.

Ko'p tillarda kontekst shartlariga quyidagi semantik chegaralar kiradi:

- dasturda ishlatilgan har bir nom, ta'riflanishi lozim va bu faqat bir marta bo'lishi kerak;
- qiymat berish operatorning chap qismidagi o'zgaruvchini va o'ng qismidagi ifodaning turlari bir hil bo'lishi kerak (yoki bir sinfdagi turga kirishi kerak);
- shartli va takrorlash operatorlarining sharti sifatida, faqat mantiqiy ifoda bo'lishi mumkin;
- taqqoslash amallar operandlari butun bo'lishi kerak;
- ifoda operandlar turlari mos bo'lishi kerak;
- har bir nishon, faqat bitta joyda bo'lishi kerak;
- blok yoki funksiya ichida har bir identifikator bir marta ta'riflanishi kerak;
- funksiyani chaqirishda faktik parametrlar soni va turi formal parametrlar soniga hamda turiga mos kelishi kerak;
- har bir amal operandlari yoki sikl parametrlari va h.k. turlariga qo'yilgan shartlar bajarilishi kerak.

Aniq shartlar tilning semantikasiga bog'liq. Masalan, ba'zi bir tillarda identifikatorni ta'riflash shart emas. Ba'zi bir tillarda ifodaga kirgan, operandlar turi avtomatik ravishda qayta o'zgartiriladi. Funksiyani chaqirishda faktik parametrlar kam bo'lishi mumkin, qolgan parametrlar kelishuv bo'yicha qiymatlarga ega bo'ladi. Semantik tahlilchi quyidagi shartlarni dasturda buzilishi to'g'risida xabar berishi kerak.

Ba'zi bir tillarda, ifodalar ichidagi operandlar turlari har hil bo'lishi mumkin. Bu holatda semantik tahlilchi bunday joylarda operandlarni bir xil turga qayta o'zgartirish, amallarini dasturning ichki tasviriga qo'shish kerak bo'ladi. Umuman aytganda, semantik tahlilchi natijaviy obyekt kodga ancha o'zgartirishlar kiritishi mumkin.

Dasturlash qoidalarini tekshirish

Ko'p zamonaviy kompilyatorlar, nafaqat til semantikasiga ko'ra qo'yilgan chegaralar va talablarni, balki dasturlashni "to'g'ri" libosini ham tekshiradi. Yuqori bosqich tillarda, odobli dasturlash qoidalari yo'q emas:

- har bir o'zgaruvchi yoki ta'riflangan o'zgarmas dasturda hech bo'lmaganda bir marta qo'llanilishi kerak;
- har bir o'zgaruvchi o'zini birinchi marta qo'llanishida, qiymatga ega bo'lishi kerak (masalan qiymat berish operatorning o'ng qismida);
- har qanday vaziyatda funksiya qiymatga ega bo'lishi kerak;
- dasturda har bir operator hech bo'lmasa, bir marta bajarilish imkoniyatiga ega bo'lishi kerak;
- shartli operatorlarning har bir tarmog'i, bajarish imkoniyatiga ega bo'lishi kerak;
- sikl operatorlari yakunlanish imkoniyatiga ega bo'lishi kerak.

Identifikatorlarni nomlar fazolariga taqsimlash

Dasturdagi identifikatorlarni nomlar fazosiga taqsimlash, bu semantik tahlilchining muhim masalalaridan biridir. Nomlar fazosi bir tomondan dasturda ishlatilgan obyektlarni, ular e'lon qilingan bloklardan tashqari ishlatib bo'lmasligi, ikkinchi tomondan esa bir hil nomlarni har xil bloklarda ishlatish mumkinligini bildiradi.

Leksik tahlil bosqichida, bu muammoni yechishning mutlaqo ilojisi yo'q. Bu bochqichda faqat har hil yozilishi bo'yicha ajratadi. Keyingi bosqichda, bir hil ismlarga boshqa noyob nomlar berib, nomlar fazosiga ajratish kerak bo'ladi. Masalan:

- lokal obyekt nomlariga ular e'lon qilingan blok (funksiya, protsedura) nomlari qo'shiladi;
- modul ichida e'lon qilingan o'zgaruvchi va funksiyalarga modul nomi qo'shiladi;

- obyekt yo'naltirigan tillarda (Si++), sinf ichiga kirgan o'zgaruvchi va fuksiyalarga sinf nomi qo'shiladi;
- har bir qayta yuklanuvchi, funksiyaga qo'shimcha raqamlar qo'shiladi.

Tashqi (global) nomlarga, alohida noyob nomlar beriladi. Bu nomlar jamlovchi tomondan ham ishlatiladi.

Xulosa

Semantik tahlilchi asosiy vazifasi quyidagilardan iborat: kiruvchi dasturda, semantik kelishuvlar va kontekst shartlarni bajarilishini tekshirish; kiruvchi tilni semantikasiga tegishli bo'lgan, qo'shimcha operatorlarni ichki tasvirga qo'shish; kiruvchi tilga tegishli bo'lmagan, semantik me'yorlarni tekshirish. Semantik tekshirishlarga quyidagilar kiradi:turlarni tekshirish;boshqarish amallarini tekshirish;yagonaligini tekshiri;nomlar bilan bog'langan tekshirishlar. Ko'p tillarda kontekst shartlariga quyidagi semantik chegaralar kirishi mumkin: dasturda ishlatilgan har bir nom, ta'riflanishi lozim va bu faqat bir marta bo'lishi kerak; qiymat berish operatorning chap qismidagi o'zgaruvchini va o'ng qismidagi ifodaning turlari bir hil bo'lishi kerak va hokazo. Semantik tahlilchi identifikatorlarni nomlar fazosiga taqsimlaydi.

Nazorat uchun savollar

1. Semantik tahlil vazifasi nimadan iborat?
2. Semantik tekshirishlargan nimalar kiradi?
3. Kontekst shartlarga qo'yiladigan chegaralarni keltiring.
4. Umumdasturlash qoydalari nimadan iborat?
5. Identifikarolarni nomlar fazosiga taqsimlash to'g'risida ma'lumot bering.

7-BOB. DASTURNI ICHKI TASVIRI

Tayanch iboralar: *sintaktik daraxt, tetrada, triada, infiks yozuv, prefiks yozuv, postfiks yozuv, abstrakt mashinaning assembler tili.*

Dasturni ichki tasvirlash usullari

Sintaktik tahlilchini ishlash natijasida, dastlabki leksemalar ketma-ketligi biror bir ichki tasvirdagi matnga o'tkazilishi kerak. Ichki tasvirdagi matn, sintaktik tuzimni aks ettirishi kerak. Ichki tasvirdagi dastur, keyinchalik obyekt kodga aylantirilishi yoki interpretatsiya qilinishi mumkin.

Dasturni ichki tasvirini hususiyatlari quyidagicha:

- ichki tasvir tillari, dastlabki dasturni sintaktik tuzimini taqqoslaydi;
- ichki tasvir tilidagi matnini sintaktik tahlil jarayonida, avtomatik generatsiya qilish mumkin;
- ichki tasvirdagi til konstruksiyalari sodda bo'lgani sababli, u matnini obyekt kodga yengil translyatsiya qilish mumkin. Yoki samarali interpretatsiya qilish mumkin;

Kompilyator uchun eng soddasi, ichki tasvirni daraxt shaklida tashkil qilishdir. Lekin, tahlil daraxti juda ko'p ortiqcha ma'lumotlarga ega. Shu jumladan noterminal belgilarni ham o'z ichiga oladi.

Ichki tasvirda faqat operandlar va amallar qolishi kerak. Ularni ketma-ketligi sintaktik daraxtni aks ettirishi kerak.

Haqiqiy kompilyatorda dasturni ichki tasviri, quyidagi shakllar bilan berilishi mumkin:

- sintaktik daraxtni beruvchi ro'yhat tuzimli ichki tasviri;
- ko'p manzilli oshkor natijali kod (operator, operand, operand – natija ko'rinishdagi tetradalar);
- ko'p manzilli oshkormas natijali kod (operator, operand-natija, operand ko'rinishdagi triadalar);
- infiks yozuv (ifodalar oddiy yozuvga o'xshagan, amallar operandlar orasida yoziladi);
- prefiks yozuv (amallar operandlardan avval yoziladi);
- postfiks yozuv (amallar operandlardan so'ng yoziladi);
- abstrakt mashinani psevdokod tili.

Bitta kompilyatorda, bir necha usullar ishlatilishi mumkin. Agar kompilyator optimizatsiya masalasini yechmasa, unda bir o'tishda sintaktik, semantik va kod generatsiyasini bajarish chizmasi qo'llaniladi.

Bog'langan ro'yxat tuzimi (sintaktik daraxt)

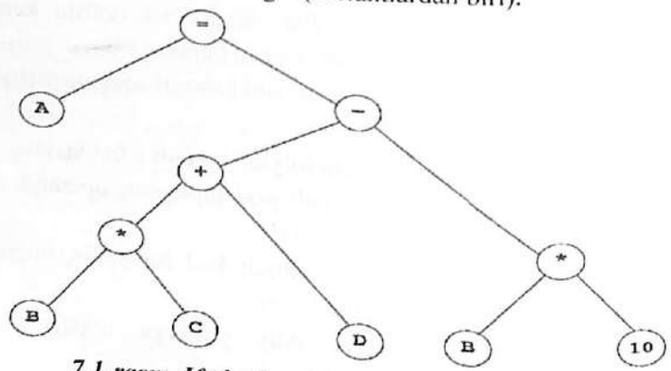
Bog'langan ro'yhat tuzimini, sintaktik tahlil jarayonida tuzish yengil va oson usuldir. Bunday usul sintaktik daraxtni to'g'ridan-to'g'ri tuzadi. Bunday daraxt tugunlarida amallar, barglarida operandlar joylashadi. Ko'pincha, operandlar identifikator va o'zgarmas jadvallari bilan bog'langan. Bunday tuzim dastlabki dastur, yozilgan dasturlash tilining sintaksisini aniq aks ettiradi.

Sintaktik daraxt tuzishdan oldin grammatikadan A→B kabi zanjir qoidalarni olib tashlash kerak.

Agar sintaktik daraxtda faqat amallar va operandlar mavjud bo'lsa, bunday daraxt amallar daraxti deb nomlanadi. Amallar daraxti bevosita va avtomatik ravishda sintaktik tahlilchi bilan quriladi. Bunday daraxtda noterminal belgilar va semantik ma'noga ega bo'lmagan belgilar (qavslar) olib tashlanadi.

Daraxtni qaysi bir tugunida amal va qaysinisida operand bo'lishi tilni grammatikasiga emas, balki semantikasiga bog'liq.

C++ tilida berilgan A = B * C + D – B * 10 arifmetik ifodani amallar daraxti 7.1-rasmda keltirilgan(variantlardan biri).



7.1-rasm. Ifodaning sintaktik amallar daraxti.

Amallar daraxti kompilyatorni kod generatsiyasidan oldin bo'lgan ichki bosqichlarida ishlatiladi. Bu daraxt amallar orasida bog'liklarni aks ettiradi. Daraxt orqali amallarni qayta o'zgartirish mumkin va

optimizatsiya ishlarini bajarish uchun, juda qulaydir. Bu usulni kamchiligi shundaki, uni obyekt dasturning chiziqli buyruqlariga o'tkazish murakkabdir.

Ko'p manzilli oshkor ravishda natijali kod (tetradalar)

Tetradalar bu amallarni to'rtta element orqali – bevosita amal, ikkita operand va amal natijasi shaklida yozishdir:

<amal> <operand1> <operand2> <natija>

Tetradalar chiziqli buyruqlar ketma-ketligi shaklida yoziladi. Masalan, A = B * C + D – B * 10 ifoda uchun tetradalar ketma-ketligi quyidagicha bo'ladi:

1. * B C T1
2. + T1 D T2
3. * B 10 T3
4. – T2 T3 T4
5. = T4 ? A

Tetrada shaklda berilgan ifoda, hech qanday ustuvorliksiz ketma-ket hisoblanadi. Agar biror bir operand mavjud bo'lmasa (unar amallarda), u operand yo'q ekanligini bildiruvchi belgi bilan yoziladi ("?"). Tetradada doimo natija yoziladi. Tetradalarni hisoblash ketma-ketligi faqat maxsus tetradalar bilan buzilishi mumkin. Bu tetradalar oldiga yoki orqaga o'tish amallarini bajaradi. Tetradalar ketma-ket yozilgani sababli, ularni osonlikcha obyekt kodga qayta o'zgartirish mumkin. Yoki assambler tiliga o'tkazish mumkin. Tetradalarni afzal tomoni shundaki, u kompyuter arxitekturasiga bog'lanmagan. Bu usulni asosiy kamchiligi shundaki, uch manzilli buyruqlardan iborat bo'lgan, hisoblash mashinalari kamdan-kam uchraydi.

Ko'p manzilli oshkormas natijali kod (triadalar)

Triadalar uchta element, amal va ikkita operand orqali beriladi:

<amal> <operand1> <operand2>

Agar triada amalida biror-bir operand sifatida oldingi natija ishlatilsa, unda operand o'rnida shu natija hosil qiladigan triadaga murojaat yoziladi. A=B*C+D–B*10 ifodani triada shaklida berilishi quyidagicha bo'ladi:

1. * B C
2. + ^1 D
3. * B 10
4. - ^2 ^3
5. = A ^4

Haqiqiy kompilyatorlarda murojaatlar, ko'rsatkich orqali amalga oshiriladi, shu sababli triadalar ro'yhat hususiyatiga ega va ularni qayta o'zgartirish mumkin.

Agar natijalarni biror bir vaqtincha xotirada saqlansa, triada tuzimi chiziqli bo'ladi. Triadalar, tetradalarga nisbatan, kam joy egallaydi. Undan tashqari triadalarni, zamonaviy ikki manzilli buyruqlarga o'tkazish juda qulaydir.

Infiks yozuv

Infiks yozuv faqat dastlabki dasturlarda ishlatiladi, bu usul inson uchun qulaydir. Lekin kompilyator uchun eng noqulay. Shu sababli bu usul ichki tasvirda ishlatilmaydi.

Prefiks yozuv

Prefiks yozuvni to'g'ri polyak yozuvi deb nomlashadi. Bu yozuvda amallar o'z operandlari oldida yoziladi. Bu yozuvni nokulayligi shundaki, amallar ketma-ketligi, bajarish ketma-ketlikdan farq qiladi. Masalan, yuqorida berilgan ifoda uchun prefiks yozuvi quyidagicha bo'ladi:

```
= A - + * B C D * B 10
```

Postfiks yozuv

Postfiks, ya'ni teskari polyak yozuvda (POLIZ) amallar operandlardan keyin yoziladi. Bizni misolimiz POLIZda quyidagicha yoziladi:

```
A B C * D + B 10 * - =
```

Bu usul kompilyator uchun juda qulay chunki:

- operandlar yozuvi, infiks yozuvdagi tartib ketma-ketligini o'zgartirmaydi;
- amallar yozuvi, bajarilish ketma-ketligida yoziladi (chapdan o'nga qarab);

- amallar bevosita o'z operandlaridan keyin yoziladi;
- qavslar ishlatilmaydi, amallar ustuvorligi hisobga olinmaydi.

POLIZ arifmetik ifodalarni translyatsiya qilish, juda ham ma'quldir.

Odatda, tilni boshqa konstruksiyalarini ham POLIZda yozish mumkin. Shu sababli dastlabki dasturni POLIZga o'tkazishni to'liq avtomatlashtirish mumkin.

POLIZni o'ziga yarasha kamchiliklari yo'q emas. POLIZda yozilgan dasturni optimallashtirish nuqtayi nazardan tahlil qilish juda qiyin.

Abstrakt mashina psevdokod tilidagi yozuv

Ko'pincha kompilyatorlar dasturni obyekt kodga yoki assembler tiliga o'tkazishdan oldin, uni psevdokod orqali yozadi. Psevdokod tilini assembler tilidan farqi shundaki, unda xotiraga chegara qo'yilmaydi. Kompyuter arxitekturasini hisobga olmaydi. Natijaviy bajaruvchi kodni yaratishdan oldin, kompilyatorlar mashinaga bog'liq optimizatsiya ishlarni bajaradi va bu ishlarni psevdokodda yozilgan ichki tasvirda ishlatadi.

Xulosa

Sintaktik tahlil natijasida leksemalar ketma-ketligi ichki tasvir matnga o'tkaziladi. Ichki tasvir shakli kuyidagicha bo'lishi mumkin: sintaktik daraxtni beruvchi ro'yhat tuzimli ichki tasviri; ko'p manzilli oshkor natijali kod (operator, operand, operand – natija ko'rinishdagi tetradalar); ko'p manzilli oshkormas natijali kod (operator, operand-natija, operand ko'rinishdagi triadalar); infiks yozuv (ifodalar oddiy yozuvga o'xshagan, amallar operandlar orasida yoziladi); prefiks yozuv (amallar operandlardan avval yoziladi); postfiks yozuv (amallar operandlardan so'ng yoziladi);abstrakt mashinani psevdokod tili. Har bir shaklni afzalligi va kamchiligi bor.

Nazorat uchun savollar

1. Dasturning ichki tasvir shakllarini keltiring.
2. Ichki tasvirning sintaktik daraxt shakli to'g'risida ma'lumot bering.
3. Tetrada tuzimi nimadan iborat?
4. Triada shakli nimadan iborat?
5. Infiks, postfiks va prefiks yozuvlarning bir-biridan farqi nimada?
6. Abstrakt mashinani assembler tili qanday shaklda bo'ladi?

8-BOB. DASTURNI OPTIMALLASHTIRISH

Tayanch iboralar: *mashinaga bog'lanmagan optimallashtirish, mashinaga bog'liq optimallashtirish,*

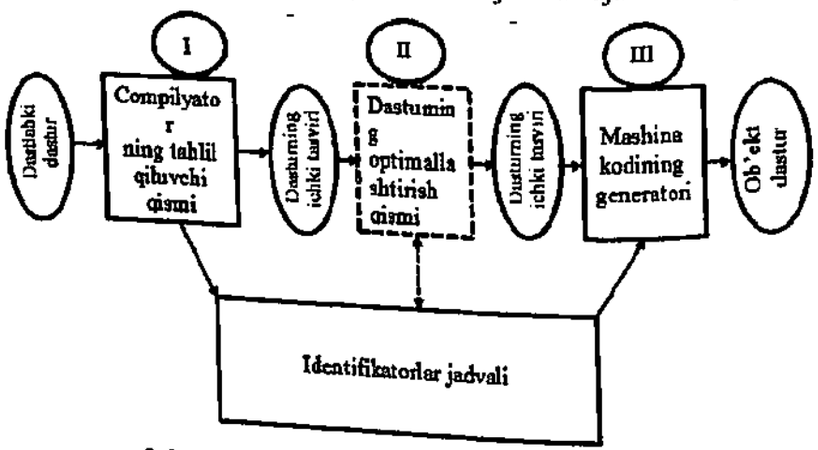
Dasturni optimallashtirish tushunchasi va usullari

Dasturni optimallashtirish deb, samarali obyekt dasturni hosil qilish uchun dasturni qayta tartiblash va amallarni o'zgartirish kabi qayta ishlovlarni tushunamiz. Dasturni optimallashtirish - noiloilik xarakatidir, chunki kompilyator dastlabki dasturni baholash.va ma'nosiga tushinish uchun to'liq semantik tahlil qila olmaydi.

Kompilyatorlarda dasturni optimallashtirishni, turli joylarda bajarishi mumkin. Boshlang'ich optimallashtirishni, foydalanuvchi o'zi bajarishi mumkin. Ba'zi bir tizimlarda profilirovuiklar mavjud. Ularning asosiy vazifasi, dasturni qaysi qismi ishlash jarayonida ko'p vaqt olishini aniqlashdan iborat.

Dastlabki dasturni optimallashtirish (8.1-rasmdagi I), texnik tafsiflarini yaxshilashda eng katta samara beradi. Bunday texnik tafsifga ikkita narsa kiradi:

Dasturni bajarish uchun ketgan xotira hajmi va bajarish tezligi.



8.1-rasm. Dasturni optimallashtirish jarayoni.

Ko'pincha berilganlarni kamaytirish, ish bajarilishi vaqtini oshirishga olib keladi va aksincha dasturni tezligini oshirish, xotirani hajmini

ko'payishiga olib keladi. Shu sababli optimallashtirish uchun qandaydir umumiy mezon olinadi. Bunday mezonlar quyidagicha bo'lishi mumkin:

- barcha qayta o'zgartirishlar ekvivalent bo'lishi kerak, ya'ni dastlabki dasturni semantikasi saqlanib qolinishi kerak;
- olingan yutuq va samaralar, qayta o'zgartirishga sarflangan harakatni qoplashi kerak. Yana bir muhim narsa bu optimallashtirish jarayonida qo'shimcha xatolar kirib, ularni tuzatishga yana vaqt sarflanishi ehtimoli yo'q emas;
- dasturni qayta o'zgartirish natijasida, uning tavsiflari o'rta hisobda yaxshilanishi kerak.

Dasturni ikkita usul bilan optimallashtirish mumkin:

- mashinaga bog'lanmaslik, ya'ni dastlabki dasturni (ichki tasvirli shaklda ham mumkin) qayta o'zgartirish, chiquvchi tilni yoki mashinani hisobga olmagan holda. Bu optimallarshtirish maxsus fazaga ajralib bajariladi (8.1- rasmdagi II) ;
- mashinaga bog'liq, ya'ni qayta o'zgartirishlar kompilyatorning chiquvchi tilidagi dasturda bajariladi. Bu optimallashtirish dasturni obyekt kodni generatsiya qilish jarayoni bilan birvarakayiga bajariladi. Yoki generatsiyadan keyin qo'shimcha fazada bajariladi (8.1- rasmdagi III).

Mashinaga bog'liq bo'lmagan optimallashtirish jarayonidagi qayta o'zgartirishlar, hisoblash tizimining arxitekturasiga bog'liq emas. Mashinaga bog'liq optimallashtirish jarayonida, apparat hisoblash tizimining hususiyatlari hisobga olinadi. Shu jumladan tashqi xotira mavjudligi, protsessorlarning o'zaro ishlashi, registrlar o'lchovi hamda soni.

Ko'pincha dasturni ichki tasviri qayta o'zgartiriladi. Chunki birinchidan, yuqori tilda yozilgan operatorlar ichki ko'rinishda aniq bo'lib qoladi. Masalan, s=s+a[i]*b[i] operatorda manzilni hisoblash qismi (sizeof(tur)*i) ko'rinmaydi, ya'ni a[i] va b[i] manzillari umumiy ifoda bilan hisoblanadi. Shu sababli uni oldin hisoblab qo'yish mumkin. Ikkinchidan, ichki ko'rinish mashinaga bog'lanmagan, shuning uchun boshqa mashinaga ko'chirganda xatoliklar ko'paymaydi.

Mashinaga bog'liq bo'lmagan optimallashtirish

Mashinaga bog'liq bo'lmagan optimallashtirishni asosiy qayta o'zgartirishlari alohida ifodalarni, dasturni chiziqli qismlarini, sikllar,

funksiya va protseduralarni chaqirish qismlari uchun bajariladi. Bir marotaba bajariladigan qism-dasturlari dastur tezligiga, umuman aytganda, ta'sir qilmaydi. Faqat dastur hajmiga ozgina ta`sir qilishi mumkin, shu sababli asosiy qayta o'zgartirishlar eng ichki sikllarda bajariladi. Optimallashtirishni bajarish uchun dastur chiziq qismlarga bo'linadi. Chiziqli qismlar bitta kirish va bitta chiqish nuqtasiga ega. Har bir dasturda bunday qismlar bor,. Ular ifodalarni va qiymat berish operatorlarini ketma-ket hisoblashdan iborat.

Chiziqli qismlar uchun quyidagi qayta o'zgartirishlar bajariladi:

- o'zgarmaslardan tuzilgan ifodalarni kompilyatsiya fazasida hisoblash;
- arifmetik qayta o'zgartirishlar;
- umumiy ifoda qismlarini olib tashlash;
- kerakmas qiymat berish operatorlari va boshqa operatorlarni olib tashlash hamda qiymatlarni nusxasini tarqatish;
- bog'liq bo'lmagan dasturni qo'shni qismlarini almashtirish;
- qatnamaydigan qism-dasturlarini olib tashlash;
- mantiq ifodalarni hisoblashni optimallashtirish.

Ma'lum o'zgarmaslarga ega bo'lgan operandlardan tuzilgan ifodalarni hisoblab qo'yish quyidagi holatlarda bajariladi:

- bevosita ifodada o'zgarmas ishlatilganda:

```
A = sin (2 * 3.14 * B);
```

- makro-kengaytmalardan keyin o'zgarmas-operandlar chiqib qolganda:

```
#define Pi 3.1415926 ;
A = sin (2 * Pi * B);
```

- til konstruksiyasini kompilyatsiya qilish natijasida, o'zgarmas-operandlar chiqib qolsa, masalan ko'p o'lchovli massivlarda:

```
int a [10][10][10], b [10][10][10], c [10][10][10];
a [3][4][i] = b [8][3][k] * c [3][2][j];
a [((3 * 10) + 4) * 10 + i] := b [((8 * 10) + 3) * 10 + k] *
c [((3 * 10) + 2) * 10 + j];
```

Kompilyator bu holatlarda hisoblashni bajarib, yangi o'zgarmaslarni o'zgarmaslar jadvaliga yozib qo'yishi kerak.

Arifmetik qayta o'zgartirishlar.

Arifmetik va mantiqiy tengliklar asosida kompilyator ifodani hisoblashni o'zgartirishi mumkin, masalan A=B*C+B*D ifodani A=B*(C+D) ga qayta o'zgartiradi.

Ba`zi bir amallarni, soddaroq amalga o'zgartirishi mumkin:

```
x := y ** 2 => x := y * y;
x := y * 2  => x := y + y;
```

Umumiy ifoda qismlarni olib tashlash (ortiqcha hisoblashlar) Chiziq qismida bir hil ifodalar qaytarilishi mumkin. Agar ularni operandlari shu qismda o'zgarilmasa, unda bunday ifoda bir marotaba yozilib, qolgan hollarda olib tashlanishi mumkin:

```
x=a+b; y=a+b; z=a+b; o'rniga
x=a+b; y=x; z=x; yoki
z=y=x=a+b
```

Kerak bo'lmagan qiymat berish operatori yoki boshqa operatorlarni olib tashlash.

Agar biror bir chiziqli qismda o'zgaruvchiga, ikkita qiymat berish operatorlari orasida, va birinchi qiymatni qo'llagan hech qanday operator uchramasa, bunday qiymatni berish befoyda. Shu sababli, u olib tashlanishi mumkin.

Bir o'zgaruvchiga boshqa o'zgaruvchining qiymatini berish holatlari uchrasa, ba`zi bir o'zgaruvchilarni olib tashlab, uning o'rniga nusxasini ishlatish mumkin. Buni "nusxalarni tarqatish" deb nomlanadi. Bunda ham xotiradan yutiladi hamda vaqt tejaladi. Masalan:

f:=g qiymat berish operatori mavjud bo'lsa, o'rniga g o'zgaruvchini ishlatish va qiymat berish operatorini olib tashlash mumkin. Umuman, f o'zgaruvchi ishlatilmaydi va u egallagan xotirani bo'shatish mumkin.

Bir biriga bog'lanmagan qo'shni dastur qismlarining joylarini almashtirish.

Ketma-ket turgan amallarda o'zgaruvchi yoki o'zgarmaslarni o'rnini almashtirish yo'li bilan, hotira va vaqtdan yutish mumkin. Masalan, A=2*B*3*C ifodani A=(B*C)*(2*3) ifodasiga qayta o'zgartirish va bunda o'zgarmaslardan iborat bo'lgan ifodani oldindan hisoblab qo'yish mumkin.

Quyidagi A=(B+C)+(D+E) ifodani hisoblashda qo'shimcha o'zgaruvchilar vaqtincha kerak bo'ladi. Agar bu ifodani A=((B+C)+D)+E ko'rinishga qayta o'zgartirilsa, bunday o'zgaruvchilar muhtoj yo'q.

Lekin qayta o'zgartirishlar aniqlikni yo'qotishga olib kelishi mumkin yoki umuman noto'g'ri qiymatni berishi mumkin. Masalan, butun sonli I/J*K ifodani I*K/Jga qayta o'zgartirsak, 11/8*3 qiymati 11*3/8 teng bo'lmaydi.

Dasturni erishib bo'lmaydigan bo'laklarni olib tashlash.

Buning uchun dasturni graf nazariyasi asosida, chuqur tahlil qilish kerak bo'ladi. Yerishib bo'lmaydigan qismlarni olib tashlashning oson yo'li, bu preprotsessor operatorlarini ishlatishdir.

Mantiq ifodalarni hisoblashni optimallashtirish.

Mantiq ifodalarni oxirigacha hisoblash shart emas. Ba'zi bir holatlarda birinchi amaldan keyin, natijasi ma'lum bo'lib qoladi. Masalan, diz'yunksiya amalida birinchi operand rost bo'lsa, qolgan amallarni bajarish shart emas. Ya'ni hisoblashni to'htatib qo'ysa bo'ladi, bu dasturni ichki tasvirini qisqartiradi hamda vaqtni tejaydi.

Funksiyalarni parametrlarini uzatishni va funksiya chaqirishini optimallashtirish.

Ma'lumki, funksiyani chaqirishda stek tashkil qilinib, unga parametrlar va natija yoziladi, bu jarayon qo'shimcha xotira hamda vaqt talab qiladi. Buning oldini olish uchun ikki usul mavjud: funksiya chaqirilgan joyga uning tanasini o'rnatish va parametrlarni global o'zgaruvchilar orqali uzatish. C++ tilda bu mexanizm (inline) funksiyalar orqali amalga oshirilgan. Undan tashqari, parametrlarni ma'lum registrlar yordamida uzatish mumkin. Umuman aytilganda, stekdan foydalanmaslik dasturni tezligini oshirishga olib keladi. Ba'zi bir tillarda (C++) o'zgaruvchilarni registrlarga joylashtirish imkoniyati bor va bu dasturni ancha optimallashtiradi.

Sikllarni optimallashtirish.

Qayta bajariladigan qism-dasturi sikl deb nomlanadi.

Dasturni siklik qismi nafaqat sikl operatorlar orqali, balki boshqa

operatorlar bilan ham tashkil etish mumkin. Ko'pincha sikl tanasi chiziqli qism-dasturlardan iborat bo'lib, unga chiziqli qismlarga tegishli barcha optimallashtirish usullarni qo'llash mumkin.

Sikllarni optimallashtirish uchun maxsus usullar ham yaratilgan:

- o'zgarmas xisoblarni sikldan tashqariga chiqarish;
- sikl o'zgaruvchilari bilan bajaradigan amallarni almashtirish;
- sikllarni birlashtirish,va bo'lib tashlab kengaytirish.

Siklni bajarish jarayonida, operandlari o'zgarmaydigan amallarni sikl tanasidan tashqariga chiqarib yuborish kerak.

Masalan,

```
for (i = 0; i < limit – 2; i ++) A [i] = B * C * A [i];
```

tsiklni quyidagi amallar ketma-ketligi bilan almashtirish mumkin:

```
D = B * C; k = limit – 2;
for (i = 0; i < k: i ++) A [i] = D * A [i];
```

Shunda B*C bir marta hisoblanadi.

Vektor arxitekturali EXMda sikllarni optimallashtirish mashinaga bog'liq bo'lib qoladi.

Sikl parametrini ko'paytirish amali uchragan operatorlarini, qo'shish amaliga o'zgartirish mumkin, masalan:

```
S = 10; for (i = 0; i<N; i ++) A [i] = i * S;
```

operatorlarni

```
S = 10; T = 0; for (i = 0; i< 10; i ++) {T = T + S, A [i] = T};
```

Ba'zi bir hollarda sikl parametridan voz kechish mumkin.

Masalan, quyidagi

```
S = 10; for (i = 0; i < N; i ++) {R = R + F (S), S = S + 10;}
```

operatorlar ketma-ketligini

```
S = 10; M = S + N * 10; while (S <= M){ R = R + F (S), S = S + 10;}
```

operatorlarga almashtirish mumkin. Bu misolda i o'zgaruvchini N marta qo'shish amali bajarilmaydi.

Bir nechta sikl operatorlarini birlashtirish mumkin:

Masalan:

```
for (i = 0; i<n; i ++) { S1; }
for (i = 0; i < n; i ++) { S2; }
```

operatorlar o'rniga

Quyidagi A=(B+C)+(D+E) ifodani hisoblashda qo'shimcha o'zgaruvchilar vaqtincha kerak bo'ladi. Agar bu ifodani A=((B+C)+D)+E ko'rinishga qayta o'zgartirilsa, bunday o'zgaruvchilar muhtoj yo'q.

Lekin qayta o'zgartirishlar aniqlikni yo'qotishga olib kelishi mumkin yoki umuman noto'g'ri qiymatni berishi mumkin. Masalan, butun sonli I/J*K ifodani I*K/Jga qayta o'zgartirsak, 11/8*3 qiymati 11*3/8 teng bo'lmaydi.

Dasturni erishib bo'lmaydigan bo'laklarni olib tashlash.

Buning uchun dasturni graf nazariyasi asosida, chuqur tahlil qilish kerak bo'ladi. Yerishib bo'lmaydigan qismlarni olib tashlashning oson yo'li, bu preprotsessor operatorlarini ishlatishdir.

Mantiq ifodalarni hisoblashni optimallashtirish.

Mantiq ifodalarni oxirigacha hisoblash shart emas. Ba'zi bir holatlarda birinchi amaldan keyin, natijasi ma'lum bo'lib qoladi. Masalan, diz'yunksiya amalida birinchi operand rost bo'lsa, qolgan amallarni bajarish shart emas. Ya'ni hisoblashni to'htatib qo'ysa bo'ladi, bu dasturni ichki tasvirini qisqartiradi hamda vaqtni tejaydi.

Funksiyalarni parametrlarini uzatishni va funksiya chaqirishini optimallashtirish.

Ma'lumki, funksiyani chaqirishda stek tashkil qilinib, unga parametrlar va natija yoziladi, bu jarayon qo'shimcha xotira hamda vaqt talab qiladi. Buning oldini olish uchun ikki usul mavjud: funksiya chaqirilgan joyga uning tanasini o'rnatish va parametrlarni global o'zgaruvchilar orqali uzatish. C++ tilda bu mexanizm (inline) funksiyalar orqali amalga oshirilgan. Undan tashqari, parametrlarni ma'lum registrlar yordamida uzatish mumkin. Umuman aytilganda, stekdan foydalanmaslik dasturni tezligini oshirishga olib keladi. Ba'zi bir tillarda (C++) o'zgaruvchilarni registrlarga joylashtirish imkoniyati bor va bu dasturni ancha optimallashtiradi.

Sikllarni optimallashtirish.

Qayta bajariladigan qism-dasturi sikl deb nomlanadi.

Dasturni siklik qismi nafaqat sikl operatorlar orqali, balki boshqa

operatorlar bilan ham tashkil etish mumkin. Ko'pincha sikl tanasi chiziqli qism-dasturlardan iborat bo'lib, unga chiziqli qismlarga tegishli barcha optimallashtirish usullarni qo'llash mumkin.

Sikllarni optimallashtirish uchun maxsus usullar ham yaratilgan:

- o'zgarmas xisoblarni sikldan tashqariga chiqarish;
- sikl o'zgaruvchilari bilan bajaradigan amallarni almashtirish;
- sikllarni birlashtirish,va bo'lib tashlab kengaytirish.

Siklni bajarish jarayonida, operandlari o'zgarmaydigan amallarni sikl tanasidan tashqariga chiqarib yuborish kerak.

Masalan,

```
for (i = 0; i < limit – 2; i ++) A [i] = B * C * A [i];
```

tsiklni quyidagi amallar ketma-ketligi bilan almashtirish mumkin:

```
D = B * C; k = limit – 2;
for (i = 0; i < k: i ++) A [i] = D * A [i];
```

Shunda B*C bir marta hisoblanadi.

Vektor arxitekturali EXMda sikllarni optimallashtirish mashinaga bog'liq bo'lib qoladi.

Sikl parametrini ko'paytirish amali uchragan operatorlarini, qo'shish amaliga o'zgartirish mumkin, masalan:

```
S = 10; for (i = 0; i<N; i ++) A [i] = i * S;
```

operatorlarni

```
S = 10; T = 0; for (i = 0; i< 10; i ++) {T = T + S, A [i] = T};
```

Ba'zi bir hollarda sikl parametridan voz kechish mumkin.

Masalan, quyidagi

```
S = 10; for (i = 0; i < N; i ++) {R = R + F (S), S = S + 10;}
```

operatorlar ketma-ketligini

```
S = 10; M = S + N * 10; while (S <= M){ R = R + F (S), S = S + 10;}
```

operatorlarga almashtirish mumkin. Bu misolda i o'zgaruvchini N marta qo'shish amali bajarilmaydi.

Bir nechta sikl operatorlarini birlashtirish mumkin:

Masalan:

```
for (i = 0; i<n; i ++) { S1; }
for (i = 0; i < n; i ++) { S2; }
```

operatorlar o'rniga

for (i = 0; i < n; i ++) { S1; S2; }

ishlatish mumkin.

Agar sikl operator tanasida shartli operator mavjud bo'lsa, sikl operatorni ikkiga ajratgani ma'kuldir:

```
for (i = 0; i < n; i ++)          if (x < y)
{ if (x < y) { S1; }              for (i = 0; i < n; i ++) { S1; }
  else { S2; } }                  else for (i = 0; i < n; i ++) { S2; }
```

Siklni kengaytirish natijasida, siklning qaytarish sonini kamaytirish mumkin va keyinchalik amallarni parallel bajarish mumkin:

```
for (i = 0; i < n; i ++)          for (i = 0; i < n; i += 2)
  { A [i] = B [i] * C [i]; }      { A [i] = B [i] * C [i];
                                    A [i+1] = B [i+1] * C [i+1]; }
```

Lekin shuni aytib o'tish kerakki, dasturlarni qayta o'zgartirish ba'zi bir hollarda xatoliklarga olib kelishi mumkin. Masalan:

for (i = 1; i<N; i ++) { ... A = i * B; ... }

operatorni

for (i = 1,A=0; i<N; i ++) { ... A = A + B; ... }

qayta o'zgartirish mumkin, lekin agarda A va B haqiqiy o'zgaruvchilar bo'lsa, iteratsiya oshgan sari yig'indi nuqsoni ham oshib ketadi,. Bu hato natijaga olib kelishi mumkin.

Mashinaga bog'lik bo'lgan optimallashtirish

Mashinaga bog'liq bo'lgan optimallashtirish usullarini hisoblash tizimining aniq arxitekturasiga, ya'ni uning apparat va dastur vositalariga mo'ljallangan.

Mashinaga bog'lik optimallashtirishda, quyidagi jihalar amalga oshiriladi:

- hisoblash mashinasining registrlar tuzilishini hisobga olish;
- keragidan ortiq buyruqlarni olib tashlash;
- boshqarish oqimini optimallashtirish va bajarilmaydigan qism-dasturlarni olib tashlash;
- dasturni murakkabligini kamaytirish;
- mashina imkoniyatlarini hisobga olish;

- texnik imkoniyatlarini hisobga olgan holda, siklik buyruqlarni birlashtirish yoki aksincha ajratish va ko'paytirish;
- EHMni vektor va konveyer arxitekturasini hisobga olish.

Muhim jihatlardan biri, bu registrlardan optimal foydalanish. Registrlarni bir guruhi dasturni boshqarish va nazorat qilish uchun qo'llanadi, boshqa guruhi arifmetik va mantiq amallarda keng ishlatiladi. Registrlarda joylashgan operandlar ustidan bajarilgan amallar, xotirada joylashgan operandlarga nisbatan ancha tez ishlaydi.

Undan tashqari, ba'zi bir arxitekturalarda umuman xotiradagi operandlarni faqat registrlarga ko'chirish mumkin. Amallar faqat registrlar bilan bajariladi. Bunday vaziyat, registrlarni taqsimlash va optimal ishlatish masalasini kompilyatorlarga qo'yadi.

Registrlarni taqsimlash usullardan biri, ularni faqat bitta kelishilgan obyektlarini saqlash. Masalan faqat protseduralar yoki funksiyalarning faktik parametrlarini saqlash uchun ishlatish. Bu albatta kompilyatorni yaratishni yengillashtiradi, lekin registrlarni samarali qo'llanishini chegaralaydi.

Ikkinchi usul, bu qism-dasturlarni bajarishini tahlil qilish. Dastur bloklarga bo'linadi va bir blokdan ikkinchi blokga o'tadigan o'zgaruvchilar registrlarda saqlanadi.

Uchinchi usul, registrlarning o'zaro munosabat grafini ranglashga asoslangan. Bu jarayon quyidagicha bajariladi:

- registrlar soni dasturdagi o'zgaruvchilar soniga teng deb hisoblanadi;
- agar ikki registr biror bir qiymatni birvarakayiga saqlashi kerak bo'lsa, ular (tugunlar) yoy bilan ulanadi;
- graf shunday rangga bo'yaladiki, hech qaysi qo'shni tugunlar bir hil rangda bo'lmasligi kerak,va ranglar soni haqiqiy registrlar soniga teng. Agar rang yetmasa, joriy tugun grafdan olib tashlanadi.

Hisobotlarni bajarishda registrlar soni yetarli bo'lmasligi mumkin. Bunday holatlarda, ba'zi bir registrlardagi qiymat xotiraga vaqtincha yuklatiladi va kerakli paytda qayta registrga yuklanadi. Kompilyator bunday registrlarni tanlashda qiymati uzoq ishlatimaydiganini oladi.

Yana shuni aytib o'tish kerakki, ba'zi dasturni ichki tasviridan

mashina kodiga o'tkazishda ortiqcha buyruqlar hosil bo'lib qoladi. Masalan registrdagi qiymat xotiraga yuklanadi va keyingi buyruq shu qiymatni registrga qayta yuklaydi. Shunga o'xshash ortiqcha buyruqlarni kompilyator olib tashlashga harakat qiladi.

Optimallashtirish jarayonida har bir buyruqga bajarish vaqti, apparat resurslarni ishlash nuqtayi nazaridan, ustuvorlik tavsiflamasi beriladi. Kompilyator "qimmat" buyruqlarni "arzon" buyruqlarga almashtiradi. Masalan, darajaga ko'tarish amalini, ba'zi hollarda chapga siljish amali bilan almashtirish mumkin. Agar bir guruh buyruqlarni, o'rniga boshqa buyruqlar ishlatilsa, bu undan ham yaxshi bo'ladi. Ba'zi bir mashinalarda ko'p uchraydigan hisoblashlar uchun, apparat buyruq mavjud bo'ladi. Shunda bunday hisoblashlarni, mos buyruqlar bilan almashtirish ma'quldir.

Hozirgi zamonda parallel va vektor hisoblash arxitekturalar keng rivojlanyapti. Bu imkoniyatlarni kompilyatorlar hisobga olishi mumkin. Masalan, ketma-ket bajaridigan bir oqimli arxitekturada A+B+C+D+E+F ifoda ((((A+B)+C)+D)+E)+F tartibda hisoblanadi. Agar mashina ikki oqimli arxitekturaga ega bo'lsa, bu ifoda quyidagicha bajarilish mumkin. ((A+B)+C)+((D+E)+F), bunda A+B va D+E amallar va natijani qo'shish amallarini bir vaqitning o'zida bajarish mumkin.

Shuni aytib o'tish kerakki, algoritm tuzish jarayonida dasturchi qatnashsa eng katta yutuq bo'ladi.

Masalan tartiblashda almashtirish algoritmi o'rniga tezkor tartiblash algoritmini ishlatish ancha vaqtni tejalishiga olib keladi.

100ta sonni tartiblashda real dastur 2,5 marta tez ishlaydi, 100000ta son uchun esa 1000 baravar tez!!!

Xulosa

Kompilyatorlarda dasturni optimallashtirishni, turli joylarda bajarishi mumkin. Ba'zi bir tizimlarda profilirovuiklar mavjud. Ularning asosiy vazifasi, dast Dasturni ikkita usul bilan optimallashtirish mumkin: mashinaga bog'lanmaslik, ya'ni dastlabki dasturni (ichki tasvirli shaklda ham mumkin) qayta o'zgartirish, chiquvchi tilni yoki mashinani hisobga olmagan holda hamda mashinaga bog'liq, ya'ni qayta o'zgartirishlar kompilyatorning chiquvchi tilidagi dasturda bajariladi. Mashinaga bog'liqmas optimallashtirishlarga kuyidagilar kirishi mumkin: o'zgarmaslardan tuzilgan ifodalarni kompilyatsiya fazasida hisoblash; arifmetik qayta o'zgartirishlar; umumiy ifoda qismlarini olib tashlash; mantiq ifodalarni hisoblashni optimallashtirish. Mashinaga bog'lik optimallashtirishda, quyidagi ishlar amalga oshiriladi:hisoblash mashinasining registrlar tuzilishini hisobga olish; keragidan ortiq buyruqlarni olib tashlash; mashina imkoniyatlarini hisobga olish; EHMni vektor va konveyer arxitekturasini hisobga olish.

Nazorat uchun savollar

1. Optimallashtirishni asosiy vazifasi nimadan iborat?
2. Dasturni optimallashtirish usullarini keltiring.
3. Mashinaga bog'liq bo'lmagan optimallashtirish nimadan iborat?
4. Mashinaga bog'liq optimallashtirish qanday bajariladi?

9-BOB. XOTIRANI TAQSIMLASH

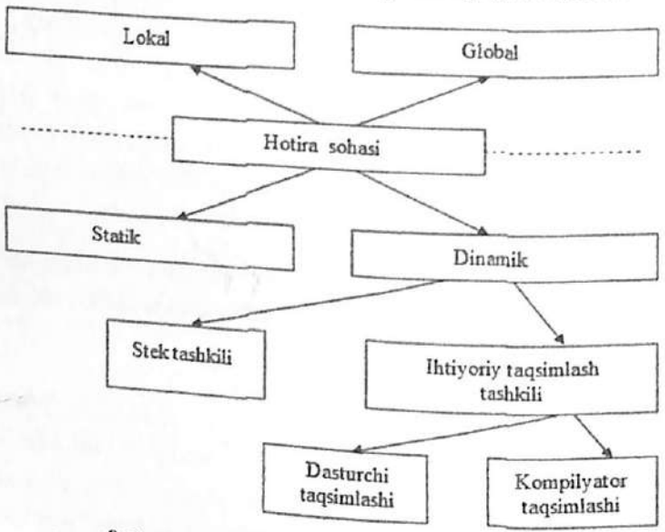
Tayanch iboralar: *lokal xotira, global hotira, statik taqsimlash, dinamik taqsimlash, stek usuli, "uyum" usuli, "ahlat jamlovchi" mexanizmi.*

Xotirani taqsimlashdagi asosiy usullar

Xotirani taqsimlash jarayonida kompilyator til birliklarga xotira manzilini beradi, ularni o'lchovini aniqlaydi va birlikka hususiyatlar (atributlar) beradi. Til birligi atamasini ishlatish sababi shundaki, xotira faqatgina berilganlarga emas, balki dastur bo'limlariga ham taqsimlanadi.

Barcha dasturlarni semantikasiga ko'ra, dastur bajarish vaqtida xotira sohasi quyidagi ma'lumotlarni saqlashi kerak bo'ladi:

- dastur kodini;
- dastur ishiga kerak bo'lgan berilganlarni;
- hisob davrida unga kerak bo'lgan tizimli dasturlar kodini;
- ishlash jarayonidagi joriy holat to'g'risidagi yozuvlarni.



9.1-rasm. Xotirani taqsimlash usullari.

Taqsimlanayotgan xotira sohasi ikkita xususiyatlarga ega – bu xotiradan dasturda foydalanish xususiyati va uni taqsimlash usuli xususiyati. Xotiradan foydalanish usuli bo'yicha xotira soxasi global va

lokal qismga bo'linadi hamda taqsimlash usuli bo'yicha statik va dinamik taqsimlashga bo'linadi (9.1-rasm).

Xotirani taqsimlashning eng sodda strategiyasi statik taqsimlashdir Unga ko'ra kompilyator xotirani – avtomatik ravishda statik sohalarga taqsimlaydi. Bu sohalarda taqsimlangan obyektlarni o'lchovi hech qachon o'zgarmaydi va ularga ko'rsatadigan manzillar ham o'zgarmaydi.

Faqat sohani boshlang'ich manzili o'zgarishi mumkin, bu ham bajarishdan oldin, ya'ni dasturni xotiraga yuklaganda bo'ladi.

Agar ob'ktlarning joyi va o'lchovi oldindan kompilyatsiya jarayonida ma'lum bo'lmasa, dinamik taqsimlash strategiyasi qo'llanadi. Dinamik taqsimlash strategiyasini bir nechta usul bilan tashkil qilish mumkin. Eng keng tarqalgan usullardan, stek ko'rinishda taqsimlash va ihtiyoriy ko'rinishda taqsimlash ("uyum"da taqsimlash)ni aytib o'tish mumkin.

Stek tuzilmasida xotiradan stek uchun ma'lum joy ajratiladi va xotira resursga ehtiyoj bo'lganda, stekdan joy ajratiladi. Bunda stek qoidasiga ko'ra ohirgi ajratilgan joy birinchi bo'shatiladi.

Ihtiyoriy taqsimlash tuzilmasida hech qanday qoida yo'q. Xotirani har qanday vaqtda egallash mumkin va kerak paytda bo'shatish mumkin. Bunday amallar oshkor ravishda dasturchi tomondan, yoki oshkormas ravishda, ya'ni avtomatik kompilyator tomonidan bajarilishi mumkin.

Strategiya va tuzilmani tanlash quyidagi mezonlarga asoslanadi:

- xotirani boshlang'ich taqsimlashdagi samaradorlik;
- "bo'sh xotira" toifasini tiklash samaradorligi;
- bo'sh xotira maydonlarini yig'ish samaradorligi.

Xotirani taqsimlashda dastlabki ma'lumot sifatida dasturni tahlil qilish jarayonida hosil bo'lgan kompilyator jadvallari kiradi. Bu ma'lumotlar barcha fazalardan yig'iladi, shu jumladan semantik tahlilda to'ldiriladi. Ajratilgan dastur obyektlariga nisbatan xotirani quyidagicha taqsimlash mumkin:

1. foydalanuvchi dastur kodi, tizimli dasturlar kodi, kiritish va chiqarish buferlari statik xotira sohasida joylashadi va bunda statik taqsimlash strategiyasi ishlatiladi.
2. turli hil berilganlar uchun:
   a) global, statik, o'zgarmaslar, translyatsiya jarayonida yaratilgan ichki tuzimlar. Masalan, polimorf sinflar uchun virtual

funksiyalar jadvali statik sohasida joylashadi va bunda statik taqsimlash strategiyasi ishlatiladi;

b) lokal o'zgaruvchilarga (masalan blok yoki funksiya ichida e'lon qilingan) dinamik taqsimlash strategiya qo'llanadi va stek tuzilmasi ishlatiladi,

c) oshkor ravishda xotira ajratilayotgan o'zgaruvchilarga (new va malloc funksiyalar orqali) va o'lchovi o'zgaradigan obyektlar uchun (STL turlar kutubxonasi) ihtiyoriy tuzilmali dinamik taqsimlash strategiyasi qo'llanadi, xotira "uyum"da ajratiladi;

3. dasturni bajarish jarayonining joriy holatini, blok, funksiya va protseduralarni faollashgani to'g'risidagi ma'lumotlarni saqlashda, norekursiv blok yoki protseduralar uchun statik taqsimlash strategiyasi qo'llanib, fiksirlangan statik sohada xotira ajratiladi va rekursiv obyektlar uchun dinamik taqsimlash strategiya qo'llanib, stek tuzilmasi ishlatiladi.

Protseduralar faollashganda, xotirada faktik parametrlar, lokal o'zgaruvchilar, vaqtincha ishlatilgan o'zgaruvchilar va qaytish manzili registrlar qiymati saqlanadi. Protsedura bajarishni tugatgach, bu ma'lumotlar tiklanadi yoki o'chiriladi. Ajratilgan xotira bo'shatiladi.

Global xotira sohasi dasturning bajarilishi boshlanganda bir marta ajratiladi va dastur ishini tamomlagunicha saqlanadi.

Lokal xotira sohasi, qism-dasturi (blok, funksiya, protsedura) ishni boshlaganda ajratiladi va tugatganda bo'shatiladi. Lokal xotirani qism-dasturidan tashqarida ishlatib bo'lmaydi.

Statik xotirani taqsimlash qiyinchilik tug'dirmaydi. Chunki statik xotiradagi obyektlarning o'lchovi va turi ma'lum bo'ladi.

Dinamik xotirani taqsimlash

Dinamik xotirani o'lchovi kompilyatsiya bosqichida noma'lum bo'lgani sababli, uni taqsimlash ancha qiyinchiliklar tug'diradi.

Bunday xotirani taqsimlash uchun kompilyator dastur ichiga, dinamik xotirani ajratuvchi va olib tashlovchi maxsus funksiyalarni qo'yadi.

Ko'pincha dinamik xotira sohasi bilan ko'rsatkich va sinf konstruktorlari bog'liq. Dinamik obyektlarni xotira maydonlariga dinamik bog'lash kerak bo'ladi. Dinamik xotira sohasida foydalanuvchi uchun xotira ajratiladi va kompilyatorga ham ajratiladi, lekin bu ajratishlar dastur

bajarilish jarayonida qilinadi.

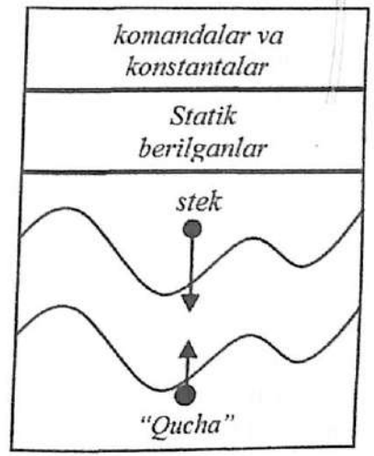
Foydalanuvchi, dinamik xotira sohasini ajratishi va bo'shatishi mumkin. Buning uchun u maxsus operatorlarni qo'llaydi (new va dispose Paskalda, malloc() va free() Sida, new va delete Si++da).

Bu operator va funksiyalar, o'z navbatida operatsion tizim imkoniyatlaridan foydalanishi mumkin yoki mustaqil ravishda katta statik ajratilgan xotira sohasida taqsimlashi mumkin. Qaysi usulni ishlatishdan qat'iy nazar, foydalanuvchi javobgarlikni to'liq o'z zimmasiga oladi.

Kompilyator tomondan ham dinamik xotira sohasi ajratilishi mumkin. Masalan, satrlarni ulash amali qo'shimcha xotira talab qiladi. Bu amalni bajarish uchun kompilyator kerakli uzunlikdagi dinamik xotirani ajratish va uni bo'shatish buyruqlarni dasturda tashkil qilishi kerak.

Alohida sohalarni xotirada joylashtirish umumiy tuzimi 9.2-rasmda keltirilgan.

Bu rasmdan ko'rinib turibdiki, noma'lum hajmdagi ikki dinamik sohalarga umumiy maydon ajratilib, ular qarama – qarshi yo'nalishda joylashtirilgan. Ikkita sohaning uzunligi noma'lum bo'lgani sababli, operatsion tizim doimo ularni kesishmasligini nazorat qilib turadi.

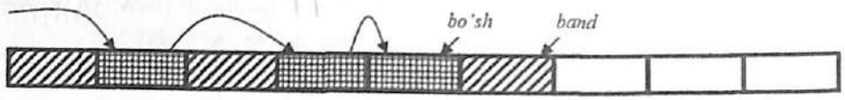


9.2-rasm. Dinamik xotiraning taqsimlanishi.

Dinamik xotirani taqsimlash texnologiyasi

XOTIRANI dinamik boshqaruv usullarini amalga oshirish foydalanuvchining talablariga bog'liqdir. Foydalanuvchining oshkor

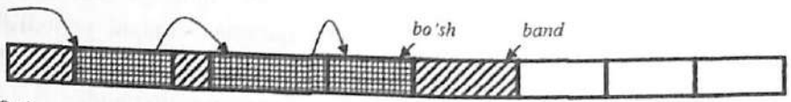
ravishda xotiraga bo'lgan talablari bo'yicha, xotira teng bloklar ko'rinishida ajratiladi (9.3-rasm).



9.3-rasm. Dinamik xotirani bloklarga taqsimlanishning birinchi usuli.

Bunday bloklarni bo'shagani sari, bo'sh bloklar ro'yhati tuziladi. Xotiraga bo'lgan so'nggi talablar bo'yicha, xotira shu ro'yhatdagi bo'sh bloklardan ajratiladi. Bu usuldagi yagona qiyinchilik – yakka elementar blokning o'lchovini aniqlashidir.

Ikkinchi usul bo'yicha bloklar o'lchovi fiksirlangan bo'lmasdan, har safar talablar parametrlariga asosan tanlanadi va blok ichida saqlanadi (9.4-rasm):



9.4-rasm. Dinamik hotirani bloklarga taqsimlanishning ikkinchi usuli.

Bu usulda xotira ajratishda muammo bo'lmaydi, lekin xotirani bo'shatish va qayta ajratish murakkablashadi. Bo'shatishda va qayta ajratishda qo'shni bloklarni birlashtirish va qolaversa zichlash kerak bo'ladi,. Ya'ni siljish kerak bo'lgan bloklar ichidagi obyektlarga bo'lgan ko'rsatkichlarni (manzillarni) o'zgartirish lozim bo'ladi.

Agar dasturda xotirani taqsimlash operatorlari ishlatilmasa, kompilyator tomondan avtomatik ravishda xotira taqsimlanadi va oshkormas ravishda bo'shatiladi. Bunday holatlarda "ahlat jamlovchi", protsedurasi ishlatiladi. Uning vazifasi xotirada bo'shatilgan sohalarni qidirib, ularni qayta ishlatishga tayyorlash. Agar oshkor ravishda taqsimlanadigan xotirani ajratishda muammo bo'lmasa, aksincha oshkormas ravishda bu katta muammodir.

Ajratiladigan xotira bloki foydalanuvchiga kerakli o'lchovda bo'lmasligi mumkin. Oshkormas ravishda ajratilgan bloklar uchun, unga bog'liq bo'lgan ko'rsatkichlar sonini saqlovchi qo'shimcha maxsus sanagichni kiritish yoki blokning bo'shligini bildiruvchi belgini tayinlash

mumkin Oshkormas ravishda ajratiladigan yoki bo'shatiladigan bloklar bilan ishlashda, uning bandligini aniqlash algoritmiga nisbatan turli hil bo'ladi.

Sanagich orqali blokni bandligini aniqlashda, doimo shu blokda joylashgan obyektlarga murojaat qilgan ko'rsatkichlar sonini nazorat qilishga to'g'ri keladi. Bunday usul barcha p = q, bunda p va q dinamik yaratilgan obyektlarga ko'rsatkichlar, qiymat berish operatorlarni nazorat qilish bilan bog'liq. Bu holda qiymat berish operatorini bajarishdan oldin p ko'rsatkich murojaat qilgan blokning sanagichi bittaga kamayadi va birdaniga q ko'rsatkich murojaat qilgan blokning sanagichi bittaga oshadi, so'ng operator bajariladi. Agar sanagichni qiymati nolga teng bo'lsa, blok bo'sh deb hisoblanadi. Bu usulda siklik murojaatlarni (halqa) qayta ishlashda muammo tug'iladi. Sababi bunday halqa ro'yhatga hech qaysi tashqi ko'rsatkich murojaat qilmasligi mumkin va bu holda xotirani bo'shatishda katta qiyinchilik tug'iladi.

Bloklarni bandligini aniqlovchi belgilar orqali yuqorida ko'rsatilgan muammoni yechsa bo'ladi. Agar biror bir obyektga xotira kerak bo'lsa, "ahlat jamlovchi" mexanizmi ishga tushadi va uning algoritmi quyidagicha:

- barcha oldin ajratilgan bloklar bo'sh deb belgilanadi;
- dasturdagi barcha ko'rsatkichlar murojaat qilgan bloklar, band deb belgilanadi;
- band blok ichidagi obyektlar maydoni sifatida ishlatilgan ko'rsatkichlar, murojaat qilgan bloklar ham band deb belgilanadi. Bu jarayon yangi band bo'lgan bloklar tugamaguncha iterativ qaytariladi;
- ko'rsatkichlar murojaat qilmagan barcha bloklar bo'sh qoladi va ular zichlanadi, zichlash jarayonida ko'rsatkichlarni qiymati o'zgartiriladi.

"Ahlat jamlovchi" jarayoni xotirani boshqarish tizimi tomonidan har qanday vaqtda bajarilishi mumkin. Bu mexanizmni samaradorligi, tizimli dasturlashni muhim tafsifnomasiga kiradi.

Xulosa

Dasturlarni semanikasiga ko'ra xotira sohasida quyidagi ma'lumotlarni saqlash kerak bo'ladi: dastur kodi; dastur berilganlari; kerak bo'lgan tizimli dastur kodi; joriy holat to'g'risidagi yozuvlar.

Hotiradan foydalanish usuli bo'yicha hotira sohasi global va lokal qismga bo'linadi.

Global qismdan dastur tuganlanishigacha ishlatiladigan va hamma joyda ko'rinadigan berilganlar saqlanadi. Lokal qismida faqat joriy funksiya yoki blokda qo'llanadigan berilganlar turadi va funksiya tamomlangandan so'ng bu berilganlar ko'rinmaydi. Hotirani taqsimlash bo'yicha statik va dinamik taqsimlashga bo'linadi. Statik ko'ra kompilyator xotirani obyektlarga avtomatik ravishda statik sohalarga taqsimlaydi Bu sohalarda taqsimlangan obyektlarni o'lchovi hech qachon o'zgarmaydi va ularga ko'rsatadigan manzillar ham o'zgarmaydi. Agar ob'ktlarning joyi va o'lchovi oldindan kompilyatsiya jarayonida ma'lum bo'lmasa, dinamik taqsimlash strategiyasi qo'llanadi. Dinamik taqsimlash strategiyasini bir nechta usul bilan tashkil qilish mumkin. Eng keng tarqalgan usullardan, stek ko'rinishda taqsimlash va ihtiyoriy ko'rinishda taqsimlash ("uyum"da taqsimlash)ni aytib o'tish mumkin.

Nazorat uchun savollar

1. Xotirani taqsimlash asosiy usullarni keltiring.
2. Dinamik xotirani taqsimlash nimadan iborat?
3. Dinamik xotirani taqsimlash texnologiyasi keltiring.

10 BOB. KOD GENERATSIYASI

Tayanch iboralar: *obyekt dastur, absolyut manzildagi kod, nisbiy manzildagi kod, assembler tilidagi kod, boshqa yuqori bosqich tilidagi kod, livermor sikllari.*

Kod generatsiyani asosiy vazifasi

Kompilyatorning ishlashidagi yakuniy jarayon - bu obyekt kodning generatsiyasidir. Kod generatoriga kiruvchi sifatida, boshlang'ich fazalarda hosil bo'lgan dastlabki dasturning ichki tasviri uzatiladi va chiquvchi sifatida ekvivalent chiquvchi obyekt dastur kodi yaratiladi.

Generatorning ish jarayoni optimizatsiya fazasiga bog'liq emas. Matematik nuqtayi nazaridan, samarali kodni generatsiya qilish muammosining yechimi yo'q. Lekin amalda kodni hosil qilishdan oldin optimallashtirish o'tkaziladi. Generatsiya qilishda barcha ma'lumotlar jadvallari ishlatiladi. Bu jadvallardagi ma'lumotlar dastur obyektlarining manzillarini aniqlash uchun ishlatiladi. Dasturni ichki tasvirida obyektlarga maxsus nomlar beriladi va bu nomlar dastlabki dasturdagi ismlardan farq qiladi.

Dasturni ichki tasviri turli bo'lishi mumkin, ularda xatoliklar bo'lmaydi.

Dasturning obyekt kodi shakllari

Aniq kompilyatorlar ishlash natijasida dasturning obyekt kodi hosil bo'ladi va uning shakli quyidagicha bo'lishi mumkin:

- absolyut manzillardagi mashina kodlari;
- nisbiy manzillardagi mashina kodlari;
- assembler tilidagi kod;
- boshqa yuqori bosqich tildagi kod.

Absolyut manzildagi kodni o'z afzalligi bor, bunday dastur fiksirlangan manzilga yuklanadi va shu zahoti bajariladi. Bunday tizimlar tilni o'rganish uchun ishlatiladi. Ba'zi bir tillarda (Algol-60, Paskal) aynan shu usul ishlatilgan.

Nisbiy manzildagi kodni afzalligi shundaki, har bir qism-dasturini (modulni) alohida kompilyatsiya qilib, maxsus jamlovchi dastur orqali ularni birlashtirib va natijaviy dastur hosil qilish mumkin. Bu tartibot

modullarni yig'ish uchun qo'shimcha xarajatlarni talab qiladi, lekin bog'liqmas kompilyatsiyani afzalligi bu xarajatlarni qoplaydi.

Agar tizimda tayyor assembler mavjud bo'lsa, unda assembler tilidagi kod ma'quldir. Undan tashqari, ko'p tillik dasturlash tizimlarida turli tillarda yozilgan komponentalarni bir-biri bilan bog'lash uchun shu shakl ishlatiladi.

Boshqa tildagi kod generatsiyasi, turli tilda yozilgan dasturlarni yagona mashinaga bog'liq bo'lmagan tilga o'tkazishda tez-tez ishlatiladi. Bu usul zamonaviy texnologiyalarda (Net Framework) qo'llanadi.

Har qanday kod generatori kiruvchining ichki ko'rinishi va natijaviy obyekt kodi shaklidan qat'iy nazar, ikkita asosiy masalani yechish kerak:

• berilganlar, obyektlar va dastur bo'laklari uchun xotirani taqsimlash;

• dasturni ichki tasvirida ishlatilgan tuzilmalar uchun ekvivalent semantik kodini qidirish va tanlash.

Kod generatsiyadagi muhim qo'shimcha funksiyalardan biri - bu standart operatorlar ketma-ketligini ajratib, ular o'rniga mashina apparaturasida mavjud bo'lgan buyruqlarga almashtirishdir. Bu jarayon dasturni optimallashtirishga olib keladi. Masalan, konveyer yoki vektor arxitekturali mashinalarda ***livermor sikllarini*** ajratish muhim masaladir. Livermor sikli - bu Livermor milliy laboratoriyasida yaratilgan 24ta maxsus ilmiy dasturlardir. Shu jumladan vektorlar, matritsalar ustidan matematik amallar, murakkab qidirish algoritmlari va boshqalar.
Masalan, 4-chi livermol sikli C tilida quyidagicha bo'ladi:

```
void Loop4 (int n, float *X, float *Y)
{ for (int k = 6; k < AR_1001; k += (AR_1001 - 7) / 2)
   { int lw = k - 6;
     for (int j = 4; j < n; j += 5) X [k - 1] -= X [lw ++] * Y [j];
     X [k - 1] *= Y [4];
   }
}
```

Dastur ichida maxsus andazalarni aniqlab, ularning o'rniga mashina arxitetkturasida mavjud bo'lgan, kerakli operatorlarni ishlatish orqali dastur tezligini 100 baravargacha oshirishi mumkin!!!

Xulosa

Kompilyator yakuni jarayonida obyekt kodini yaratadi. Bu jarayon kod generatsiyasi deb nomlanadi. Kod generatoriga kiruvchi sifatida ichki tasviri uzatiladi va chiquvchi sifatida ekvivalent chiquvchi obyekt dastur kodi yaratiladi Generatsiya qilishda barcha ma'lumotlar jadvallari ishlatiladi. Bu jadvallardagi ma'lumotlar dastur obyektlarining manzillarini aniqlash uchun ishlatiladi.

Aniq kompilyatorlar ishlash natijasida dasturning obyekt kodi quyidagicha shaklda bo'lishi mumkin: absolyut manzillardagi mashina kodlari; nisbiy manzillardagi mashina kodlari; assembler tilidagi kod; boshqa yuqori bosqich tildagi kod.

Har qanday kod generatori ikkita asosiy masalani yechish kerak: berilganlar, obyektlar va dastur bo'laklari uchun xotirani taqsimlash; dasturni ichki tasvirida ishlatilgan tuzilmalar uchun ekvivalent semantik kodini qidirish va tanlash.

Nazorat uchun savollar

1. Kod generatsiyasini asosiy vazifasi nimadan iborat?
2. Obyekt kodni shakllarini keltiring.

11-BOB. KUTUBXONALAR

Tayanch iboralar: *statik kutubhonalar, dinamik kutubhonalar, funksiyalar, protseduralar va makro ta'riflar kutubhonalari, sinflar kutubhonalari, komponentalar kutubhonalari.*

Kutubxona bilan ishlash texnikasi

Dasturlash tizimlarning farqlovchi xususiyatlardan biri bu qism-dasturlar kutubxonasining mavjudligidir. Texnik tarkibi bo'yicha kutubxonalar ikkita kategoriyaga bo'linadi: dastlabki tilning funksiyalar kutubxonasi va operatsion tizim funksiyalari. Kutubxonaning ma'lum qismi, masalan kutubxonaga kirgan komponentalarning tavsiflari kompilyatorga uzatilishi kerak. Kompilyator kutubxonadan chaqirilgan funksiyalarni parametrlar soni va turini dasturdagi chaqirish bilan solishtiradi. Faktik parametrlar dastlabki dasturda beriladi va formal parametrlar kutubxonada ta'riflanadi. Vaholanki kutubxona funksiyalari o'zi ham shu dasturlash tilida yozilishi mumkin. Kutubxonalar statik va dinamik bo'lishi mumkin.

Statik kutubxonalar

Statik kutubxonalar, bajaruvchi dastur ichiga qo'shiladigan protsedura va funksiyalardan iborat. Barcha statik kutubxonalar, bajaruvchi dasturga faqat bir marotaba yuklovchi tomondan qo'shiladi. Dasturga qo'shilgan komponentalar uning ajralmas qismi hisoblanib, kutubxonaga bog'liqmas bo'lib qoladi.

Agar kutubxonaga o'zgartirishlar kiritilsa dasturga o'zgartirish kiritish shart emas, chunki dastur kutubxona bilan aloqasi uzilgan bo'ladi.

Kutubxonada biror bir ishlatilgan komponentasida xatolik bo'lsa yoki algoritmi o'zgartirilsa, bajaruvchi dasturni to'g'ri ishlashi uchun uni qayta yig'ish lozim.

Kutubxona komponentlarni va obyekt modullarni bajaruvchi dasturga statik qo'shish, dasturni hajmini oshirishiga olib keladi. Hozirgi zamonda kutubxonalar komponentalar yuqori darajada rivojlandi va ularning miqdori bajaruvchi dasturda yarmidan ko'payib ketdi. Ba'zi bir standart va tizimli kutubxonalarni barcha dasturlar ishlatgani bois, samarasiz ishlatilgan xotira hajmi juda katta bo'lib ketishi mumkin. Shu sababli, amalda dinamik kutubxonalar keng ishlatiladi.

Dinamik yuklanuvchi kutubxonalar

Dinamik kutubxona komponentalari dasturning bajarish vaqtida unga ulanadi. Shunisi bilan u statik kutubxonadan farq qiladi.

Dinamik kutubxona komponentalari unga murojaat qilgan dasturlar bilan bog'liq emas va ular alohida tarqatiladi.

Ba'zi bir dinamik kutubxonalar dasturga uning bajarishidan oldin, operativ xotiraga yuklanish paytida ulanadi. Shunda bu kutubxona umuman aytganda, ishlatilmasligi ham mumkin. Boshqa kutubxonalar xotiraga faqat ularni komponentasiga murojaat qilganda yuklanadi. Xotirani bo'shatish ham turli shakldagi kutubxonalar uchun farq qiladi. Xotira, dastur bajarib bo'lgandan so'ng, yoki asosiy dastur buyrug'i bilan, yoki komponenta ishlatilib bo'lgandan keyin, bo'shatilishi mumkin.

Komponentaga murojaat qilganda yuklanish, dastur ish boshida yuklanishga nisbatan afzalroqdir. Bu holda xotira samarali ishlatiladi, lekin bu usulni amalga oshirish murakkabroqdir. Bu usul operatsion tizimni murakkablashtirishini talab qiladi, yoki bajaruvchi dasturga dinamik yuklovchi va bo'shatuvchi maxsus buyruqlarni kiritish kerak bo'ladi.

Dasturni bajarish oldidan kutubxonani yuklash, albatta xotirani samarasiz ishlatishga olib keladi, lekin bu usulni amalga oshirish osonroq. Bu masalani yuklovchi hal qilishi mumkin, buning uchun dastur sarlohasiga ishlatiladigan kutubxonalar ro'yxati jadvali qo'shiladi. Zamonaviy dasturlash tizimlarida ikkala usul bilan ishlash mumkin, qaysi birini qo'llash dastur yaratuvchiga bog'liq. Tanlash kutubxona hajmi, komponentalarga murojaatlar soniga bog'liq.

Dinamik kutubxonalarni xotiraga yuklash, bo'shatish va komponentalarni asosiy dasturga bog'lashni dinamik yuklovchi bajaradi. Bu yuklovchi operatsion tizim tarkibida bo'lib, barcha dinamik kutubxonalarga bo'lgan murojaatlarni nazorat qiladi. U kutubxonaga bo'lgan murojaatlar sonini hisoblab boradi. Agar bu son nolga teng bo'lsa kutubxonani xotiradan olib tashlaydi. Bitta yuklangan dinamik kutubxonadan bir nechta dasturlar foydalanishi mumkin. Agar kutubxona xotiraga yuklangan bo'lsa, u ikkinchi marotaba yuklanmaydi.

Dinamik kutubxonalar maxsus tuzimga ega. Zamonaviy dasturlash tizimlari bunday kutubxonalarni yaratishga imkoniyatlar beradi. Bu tuzim barcha kutubxonalar uchun bir hil va operatsion tizimga bog'liqdir.

Dinamik kutubxonani afzalligi shundaki, uning komponentalari bajaruvchi dastur tarkibiga kirmaydi.

Dinamik kutubxonalarning ma'lum kamchiligi, dastur mahsulotida bevosita kutubxona obyektlari ishlatilmaganiga u shu kutubxona obyektlariga bog'lanmoqda. Ya'ni foydalanuvchilarning dasturlari unga noma'lum bo'lgan dinamik kutubxonaga bog'lanib qolayapti.

Kutubxonalarni dinamik ulov asosidagi dasturni bajarishda, kerakli kutubxona komponentalari hisoblash tizimda mavjudligini ta'minlash uchun maxsus choralarni ko'rish kerak.

Undan tashqari, dastur mantig'i ulangan kutubxonalarga bog'liq bo'lib qoladi. Dinamik kutubxonalar komponentalariga o'zgartirishlar kiritish, foydalanuvchi dasturini to'g'ri ishlashiga ta'sir qiladi. Bu degani dinamik kutubxonalarni ishlatishda, foydalanuvchi hamda kutubxona yaratuvchi ma'lum majburiyatlarni o'z zimmasiga olishi kerak bo'ladi.

Dastur yaratuvchi kutubxona yaratuvchi tomondan taklif qilgan qoidalarga aniq rioya qilishi kerak. O'z navbatida kutubxona yaratuvchi komponentalarga o'zgartirishlarni kiritganda, komponentani oldingi naqlidagi mantiqni buzmasligi va bu o'zgarishlar minimal bo'lishi kerak. Albatta hotiraga dinamik kutubxonalarni samarali yuklash va bo'shatish foydalanuvchi dasturini sekin ishlashiga olib keladi.

Kutubxonalarni asosiy turlari

Har qanday zamonaviy dasturlash tizimi kutubxona vositalaridan foydalanadi. Dasturlash tizimlarda kutubxona vositalari keng tarqalganing kamida ikkita sababi bor:

- dasturlarni hisoblash jarayonida qo'llab quvvatlash;
- foydali dasturlarni saqlash va ularni keng tarqatish, bu jarayonda dasturni algoritmini ochish shart emas.

Zamonaviy dasturlash tizimlarida funksional nuqtayi nazardan kutubxonalarni quyidagicha sinflash mumkin:

- funksiyalar, protseduralar va makro ta'riflar kutubxonalari;
- sinflar kutubxonalari;
- komponentalar kutubxonalari.

Funksiyalar, protseduralar va makro ta'riflar kutubxonalari

Kompilyatsiyadan o'tgan barcha dasturlar bajarish jarayonida qo'llab

quvvatlashga muhtojdir va shu jumladan operatsion tizim bilan bog'lanish imkoniyati bo'lishi kerak. Masalan, ma'lumotni kiritish va chiqarish amallarini bajarishda.

Har bir dasturlash tilida kiritish va chiqarish amallari operatsion tizimning qism-dasturlarga murojaat qilish yo'li bilan bajariladi. Bu qism-dasturlar biror bir kutubxonaga obyekt modul shaklida yig'iladi.

Ular shu dasturlash tilida tuzilishi shart emas, lekin shunday tashkil qilinishi kerakki, ular dasturlash tilida va uning translyatori tomonidan to'g'ri chaqirilib ishlashi kerak.

Kutubxonalar nafaqat aniq hisoblash mashinaga bog'liq, kompilyatorga ham bog'liqdir.

Dasturlar oldin tizimli va na'munaviy kutubxonalardan foydalanardi. Bundan tashqari amaliy dasturlar kutubxonalari ham rivojlandi. Bu kutubxonalar amaliy dasturlar paketi deb nomlandi.

Har bir sohada dasturlar paketiga zarurat bor. Masalan, matematik hisoblar uchun juda ko'p paketlar yaratilgan. Shulardan biri Xalqaro sonli algoritm guruhi paketi (The Numerical Algorithms Group). Bu paketga kirgan amaliy dasturlar C va Fortran tillarda yozilgan dasturlar uchun mo'ljallangan. Paket turlicha operatsion tizimlarda (Microsoft Windows, Linux, Sun Solaris, Silicon Graphics IRIX) va turli kompyuterlarda (Intel x86-32, Intel x86-64, Compaq Alpha Tru64, IBM RS/6000) qo'llanilishi mumkin.

Buxgalteriya hisoboti uchun amaliy dasturlar paketi yaratilgan. Berilganlar bazasini boshqaradigan tizimlar (BBBT), nashriyot uchun paketlar ham juda ko'p.

Zamonaviy kutubxonalar obyekt modullardan tashqari, interfeys ma'lumotlarini ham alohida saqlaydi. Bu qismda kompilyatorlarga va dasturlarga kerak bo'lgan funksiyalarni hamda protseduralarni tavsiflari turadi. Bu holda funksiyalarni qayta tavsiflash hojati bo'lmaydi. Dasturga faqat interfeys faylini ulasa yetarlidir.

Sinflar kutubxonalari

Kutubxonalar rivojlanishining keyingi qadamida, dasturlash tizimlari uchun yaratilgan sinflar kutubxonasi bo'ldi. Bu kutubxonalar obyektga yo'naltirilgan dasturlash tillari (Si++, Java) asosida yaratilgan. Sinflar kutubxonasi quyidagi ko'rinishda bo'lishi mumkin:

- bog'lanmagan sinflar majmuasi:
- sinflar shajarasi;
- sinflar qoliplarining shajarasi.

Oddiy sinflar to'plami dasturlash tizimlarida kamdan-kam uchraydi, chunki ular bir-biriga bog'lanmagan va shajaralanmagan.

Shajaralangan sinflar kutubxonalari foydalanuvchiga ancha qulaydir.

Ko'pincha shajara asosida bitta umumiy sinf turadi. Qolgan sinflar vorislik asosida qo'shiladi.

Bu usulni kamchiligi ham bor – yuqori pog'onada umumiy hossalarga ega bo'lgan sinflar turishi kerak bo'ladi. Shunday bo'lib qoladiki barcha sinflarning umumiy xossasi yo'q bo'lganda, tayanch sinf bo'sh bo'lib qoladi, yoki juda ko'p virtual funksiyali abstrakt sinf qurish kerak bo'ladi.

Bo'sh tayanch sinf faqat shajarani tuzish uchun kerak bo'ladi.

Vorislikda hosil sinflarda, barcha virtual funksiyalar aniqlanishi kerak bo'ladi, aks holda u ham abstrakt bo'lib, amalda ishlatilmaydi.

Bunday vaziyatni oldini olish uchun bitta kutubxona ichida shajaralar tizimi tashkil qilinadi, ya'ni bir-biriga bog'lanmagan shajaralar daraxtlari quriladi (o'rmon). Bu usul bilan masalan STL kutubxonasi va Si++ tilni standart kutubxonasi qurilgan.

Qolipli kutubxonalar parametrik polimorfizm asosida qurilgan, ya'ni undagi sinflarda turlar parametrlangan. Har bir sinfni har hil turlar uchun ishlatish mumkin. STL kutubxona va Si++ tilining standart kutubxonasi sinf qoliplari shajaralari to'plamiga kiradi.

Komponentalar kutubxonalari

Komponentalar kutubxonalari bu sinf konsepsiyasi asosida tillar rivojlangan tillar kutubxonalari.

Mukammal dastur modullar kutubxonalar komponentasini tashkil qiladi. Bu komponentalar orqali ilovalar yaratish ancha yengillashadi. Bunday kutubxonalarda grafik va diagrammalar qurish, grafik interfeys asosida ko'p hujjatli darchalarni yaratish, internet sahifalar yaratish, berilgan bazasini boshqarish va boshqa masalarni yechish komponentalari joylashadi.

Kutubxona komponentalari umumiy dastur yadrosiga ega bo'lib, bir

hil arxitektura asosida loyihalangan. Shu sababli ularni o'rganish va baravariga ishlatish juda qulaydir.

Kutubxonaga kirgan komponentalar inkapsulyatsiya qoidalariga bo'ysunadi, ya'ni unda ochiq interfeys elementlari bor va ularni ichki algoritmi foydalanuvchidan berkitilgan. Bunday komponentalarni ko'pincha konteynerlar ichiga joylashtiradi. Konteynerlar alohida darchalarda joylashadi. Bunday texnologiya vizual dasturlash yoki komponent dasturlash deb nomlanadi ("drag & drop" stilida). Komponentalar kutubxonasi sifatida COM (Component Object Module) va DCOM (Distributed COM) kutubxonalarini keltirish mumkin.

Natijada shuni ta'kidlash mumkinki, barcha zamonaviy kutubxonalar ikkita kategoriyaga bo'linadi:

- aniq dasturlash tizimlarga bog'liq kutubxonalar;
- aniq yechilinayotgan masalaga bog'liq kutubxonalar.

Xulosa 11 bob

Dasturlash tizimlarning tarkibiga qism-dasturlar kutubxonalari kiradi. Kutubxonalar ikkita kategoriyaga bo'linadi: dastlabki tilning funksiyalar kutubxonasi va operatsion tizim funksiyalari. Kutubhonalar statik va dinamik bo'lishi mumkin. Statik kutubxonalar, bajaruvchi dastur ichiga qo'shiladigan protsedura va funksiyalardan iborat. Dasturga qo'shilgan komponentalar uning ajralmas qismi hisoblanib, kutubxonaga bog'liqmas bo'lib qoladi. Dinamik kutubxona komponentalari dasturning bajarish vaqtida unga ulanadi. Zamonaviy dasturlash tizimlarida funksional nuqtayi nazardan kutubxonalarni quyidagicha sinflash mumkin: funksiyalar, protseduralar va makro ta'riflar kutubxonalari; sinflar kutubxonalari; komponentalar kutubxonalari.

Nazorat uchun savollar

1. Kutubxona bilan ishlash texnikasi nimadan iborat?
2. Statik kutubxonalar to'g'risida ma'lumot keltiring.
3. Kutubxonalar turlari bo'yicha qisqa ma'lumot bering.

12-BOB. IDENTIFIKATOR JADVALLARI

Tayanch iboralar: *identifikatorlar jadvali, xesh-funksiya, xesh-manzillash, kolliziya, daraxt usuli, xesh-manzillash usuli, rexeshlash usuli, zanjir usuli.*

Identifikatorlar jadvallari (IJ) tuzilishi

Dastur semantikasi to'g'riligini aniqlashda va kod generatsiyasi jarayonida dasturda uchragan o'zgaruvchilar, o'zgarmaslar, fuksiyalar va boshqa elementlar xarakteristikalarini bilish kerak bo'ladi. Barcha elementlar identifikatorlar orqali belgilanadi. Identifikatorlar leksik tahlil fazasida ajratiladi. Ularni xossalari sintaktik, semantik tahlil va kod generatsiyaga tayyorgarlik fazalarida aniqlanadi. Bunday ma'lumotlar har bir fazada ishlatish uchun kompilyator ularni saqlash imkoniyatiga ega bo'lishi kerak. Shu maqsadda identifikatorlar jadvali ishlatiladi.

Identifikatorlar jadvalida satrlarda identifikatorlar ro'yxati turadi. Ustunlarda ularni tafsifnomalari yoziladi. Kompilyatorda bunday jadvallardan bir nechta bo'lishi mumkin, masalan o'zgaruvchilar uchun alohida, o'zgarmaslar uchun alohida, yoki har bir modul uchun alohida jadvallar.

Ustunlarda quyidagi ma'lumot yozilishi mumkin :

o'zgaruvchilar uchun -

- o'zgaruvchini nomi;
- turi;
- uzunligi;
- boshlang'ich qiymati;
- xotirada joylanish nisbiy manzili.

o'zgarmas uchun-

- o'zgarmas nomi(agar bo'lsa);
- qiymati;
- turi (kerak bo'lsa);

funksiya uchun-

- funksiya nomi;
- formal parametrlar soni;
- formal parametlar turi;
- qaytarish qiymat turi;
- funqtsiya kodini boshlang'ich manzili.

Albatta jadvallar ko'rinishi misol taraqasida keltirilgan. Aniq ko'rinishi kompilyatorga bog'liqdir. Undan tashqari ma'lumotlar birdaniga yozilmaydi, masalan o'zgaruvchilar nomi leksik tahlil fazasida uning turi sintaktik tahlilda va xotira manzili kod generatsiyaga tayyorgarlik fazalarida to'ldiriladi.

Kompilyator ish jarayonida IJga ko'p marta murojaat qiladi. Shu sababli bu jadvallar shunday tashkil qilinishi kerakki, identifikatorni maksimal tezlikda qidirish imkoniyati bo'lish kerak.

Identifikator jadvalini tuzish usullari

Eng sodda usuli, bu identifikatorlarni ajratish jarayonida ketma- ket jadvalga yozishdir. Bu holda oddiy tartiblanmagan massiv bo'ladi. Unda kerakli elementi qidirish uchun jadvaldagi har bir element bilan, to kerakli element topilmaguncha solishtirib chiqishi kerak bo'ladi. Shunda agarda elementlar sonini N desak, solishtirishlar soni o'rtacha N/2 bo'ladi. Umuman qidirish vaqti T=O(N) tartibda bo'ladi. Jadvalni bunday tashkil qilishda, jadvalga yangi element qo'shish uchun ketgan vaqt elementlar soniga bog'liq emas. Chunki u massivning oxiriga yoziladi. Lekin qidirish uchun ancha vaqt ketadi. Elementlar soni ko'p bo'lsa (10000), bunday usul samarasiz bo'ladi.

Agarda elementlar tartiblangan holda saqlansa, qidirishni ancha samarali tashkil qilish mumkin.

Buning uchun binar qidirish usulini qo'llasa bo'ladi. Qidirilayotgan element massivni o'rtadagi elementi bilan solishtiriladi. Agar teng bo'lsa, element topilgan hisoblanadi, aks holda katta bo'lsa o'ng qismi, kichik bo'lsa chap qismining o'rtadagi element bilan solishtiriladi. Bu jarayon toki element topilmaguncha, yoki interval birga teng bo'lmaguncha qaytariladi. 12.1- rasmda tartiblangan identifikatorlar jadvaliga joriy elementni qo'shish va 12.2- rasmda identifikatorni qidirish blok-chizmalari keltirilgan. Bu algoritm qidirish vaqti $T=O(\log_2 N)$. Agar N=128, solishtirishlar soni 8ga teng bo'ladi. Oddiy usulda esa 64ga teng. Bu usulni kamchiligi shundaki, jadval tartiblangan bo'lishi kerak. Ya'ni yangi elementni qo'shishga sarflangan vaqt elementlar soniga bog'liq bo'lib qoladi. Yangi elementni qo'shishda ham shu usulni qo'llash mumkin. Oldin yangi elementni tartiblangan massivda o'rnini topish kerak va

massivni chap qismini surib, shu o'ringa elementni yozish kerak. Yangi elementni qo'shish bir marta, lekin uni qidirish bir necha marotaba bajarilishi sababli bu usul ma'qul deb xisoblanadi.

Identifikator jadvalini daraxt shaklida tuzish

Identifikator jadvalida to'lgazish vaqtni oshirmasdan, qidirish vaqtini kamaytirish mumkin. Buning uchun jadvalni binar daraxt ko'rinishda tashkil qilish kerak. Har bir tugun element to'g'risidagi ma'lumotni saqlaydi. Daraxtni ildizida birinchi element yoziladi. Daraxtni har bir cho'qqisi ikkita shoxga ega bo'lishi mumkin, o'ng va chap shoxlar.

Kompyuterda daraxt tugunlarini tuzilma (struct) turida bo'lgan ko'rsatkich ko'rinishida ta'riflash mumkin .

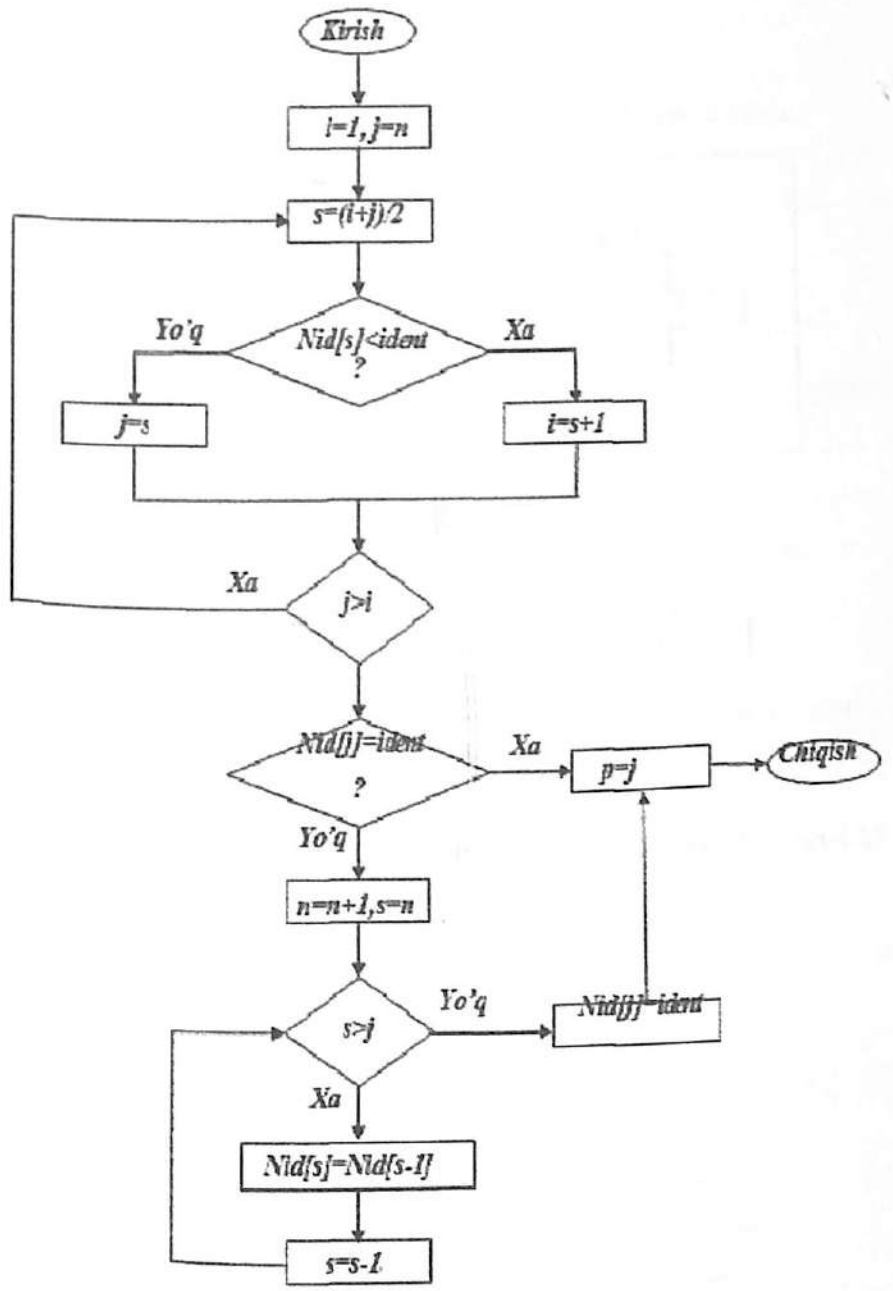
Masalan:

```
struct  daraht  {
   char Nid [20]; // identifikator
   daraht * Pshap; // chap tugunni manzili
   daraht * Pyng; // o'ng tugunni manzili
}
```

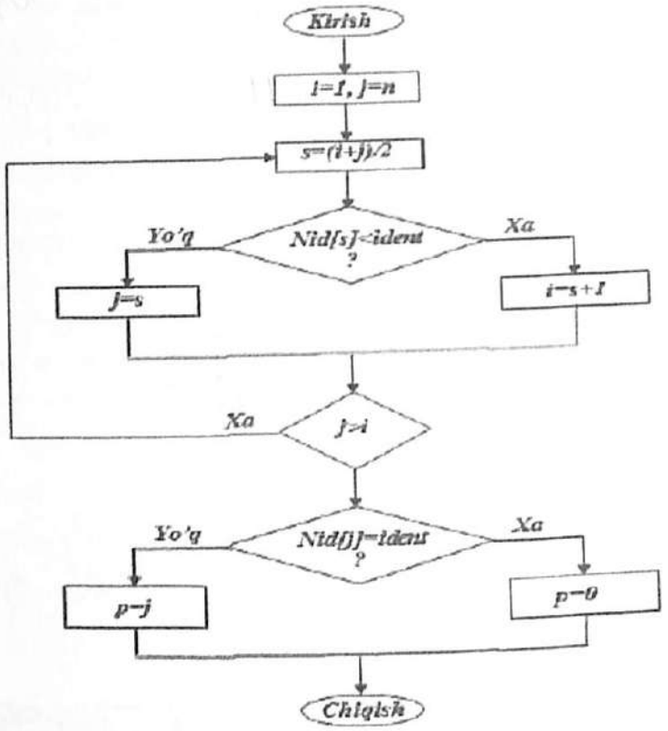
Daraxt tuzilishi 12.3- rasmda keltirilgan.

Daraxtni to'ldirish algoritmi 12.4- rasmda keltirilgan, dasturi esa quyidagicha:

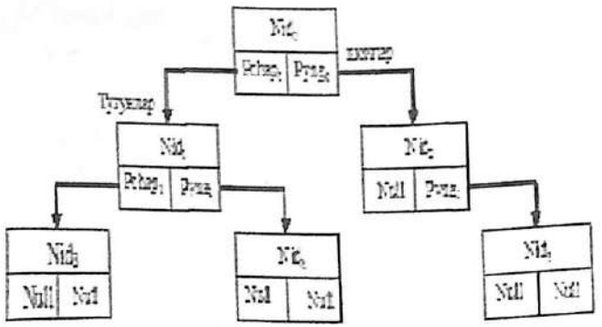
```
daraht * daraht_yaratish()
{
  int i;
daraht * Ildiz; // daraxtni ildizi
  daraht * Tugunj, *Tugun;   // daraxtni joriy
tuguni
  char ident[20];  // joriy identifikator
Ildiz = new daraht;  Ildiz->Pchap=NULL;Ildiz-
>Pyng=NULL;
  cin.getline(ident,80); strcpy(Ildiz-
>Nid,ident);
```

12.1-rasm. Tartiblangan IJga joriy elementni qo'shish blok chizmasi.



12.2-rasm. Tartiblangan identifikator jadvalida joriy elementni qidirish blok-chizmasi.

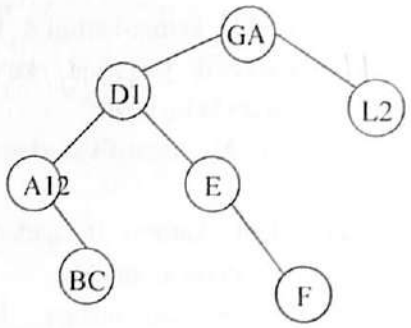


12.3-rasm. Daraxt tuzilishi.


```
m1:
     cin.getline(ident,80);
     if (strlen(ident)==0) return Ildiz;
     Tugunj=Ildiz;
m2:  i=strcmp(Tugunj->Nid,ident);
     if (i==0) goto m1;
     if (i<0)
       if ( Tugunj->Pyng==NULL)
         {  Tugun=new daraht;
            Tugun->Pchap=NULL;          Tugun-
>Pyng=NULL;
            strcpy(Tugun->Nid,ident);
            Tugunj->Pyng=Tugun;goto m1;}
       else {Tugunj=Tugunj->Pyng;goto m2;}
     else
       if ( Tugunj->Pchap==NULL)
        {Tugun=new daraht;
         Tugun->Pchap=NULL; Tugun->Pyng=NULL;
         strcpy(Tugun->Nid,ident);
         Tugunj->Pchap=Tugun; goto m1;}
       else {Tugunj=Tugunj->Pchap;goto m2;}
}
```

Daraxtda elementni qidirish algoritmi 12.5-rasmda keltirilgan, dasturi esa quyidagicha:

```
bool daraht_kidir(char * ident,daraht* Ildiz)
{
     int i;
     daraht * Tugunj;
     Tugunj=Ildiz;
  m2: i=strcmp(Tugunj->Nid,ident);
     if (i==0) return true;
     if (i<0)
```



12.6-rasm. Berilgan misol uchun tuzilgan daraxt.

Xesh-funksiya va xesh-manzillash. Xesh-funksiya ishlash tamoili

Jadvallarni to'ldirish va qidirish vaqtining logarifmik bog'lanishi bu juda yaxshi, lekin identifikatorlar soni judayam katta bo'lishi mumkin. U holda boshqa usullarning qo'llashga to'g'ri keladi.

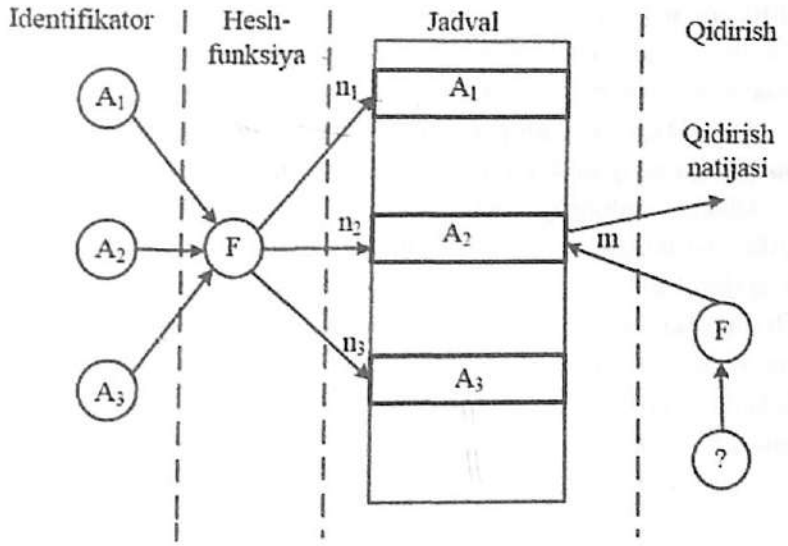
Xesh-funksiyalarni qo'llash usullari yaxshi natijaga olib keladi.

R elementlar to'plamini Z butun musbat sonlar to'plamiga aks ettiruvchi funksiya F(r) xesh-funksiya deb nomlanadi. Kiruvchi elementlar to'plami xesh funksiyani aniqlash soxasiga kiradi. Xesh funksiyaning qiymatlar sohasi qism to'plami M bo'ladi: $F:\forall r\in R:F(r)\in M;M\subseteq Z$.

Xesh-funksiyani aniqlash sohasini M qiymatlar to'plamiga aks ettirish jarayonini xeshlattirish deb taqdim qilinadi.

Identifikatorlar jadvallar ustida ishlaganda xesh-funksiya identifikatorlarni butun sonlarga aks ettirishi kerak. Shunda xesh-funksiyani aniqlash sohasi barcha identifikatorlar nomlari bo'ladi. Xesh-manzillashda xesh-funksiya qiymati, birorta berilganlar massivni elementlarining manzili deb hisoblanadi. Bundan ko'rinib turibdiki, massivni o'lchovi fuksiyani qiymatlar sohasiga teng bo'lishi kerak. Ya'ni xesh-funksiyani maksimal qiymati kompyuter xotirasidan oshib ketmasligi kerak. Jadvalni tuzish quyidagicha bo'ladi. Joriy elementni xesh-funksiyasi xisoblanadi. Hisoblangan qiymat elementni IJda joylashtiradigan yacheykani manzilini beradi.

Shunday qilib har qanday identifikator uchun uning xesh-funksiyasini xisoblash yetarli va shu yacheykaga identifikator yuklanadi.



12.7-rasm. IJni xesh-manzillash usuli bilan tashkil qilish.

Qidirishda ham elementni xesh-funksiyasi hisoblanadi va shu manzil bo'yicha yacheyka bo'shligi tekshiriladi. Agarda bo'sh bo'lmasa element jadvalda mavjud aks holda bunday element yo'q. 12.7-rasmda xesh-funksiyani ishlash tamoili ko'rsatilgan.

Bu usul juda yaxshi, chunki elementni jadvalga yuklashga va qidirishga ketgan vaqt funksiyani hisoblashi bilan o'lchanadi va bu juda kamdir.

Bu usulning ikkita kamchiligi bor:

- birinchidan, xotirani samarasiz ishlatish, chunki massivni o'lchovi funksiyani qiymatlar sohasiga teng bo'lishi shart, aksincha aniq identifikatorlar soni ancha kam bo'lishi mumkin;
- ikkinchidan, yaxshi natijaga olib keladigan xesh-funksiyani qurish bu katta muammodir;

Xesh-funksiyani tanlash usullari.

Xesh-funksiyaning turli variantlari mavjud. Xesh-funksiyada ko'pincha belgilar ketma-ketligi ustidan biror arifmetik yoki mantiqiy amallar bajariladi. Masalan, identifikatorning birinchi belgisining ASCII kodini natija qilib olish mumkin, masalan A harfni kodi 65. Mana shu qiymatni xesh-funksiyaning qiymati deb hisoblasa bo'ladi. Lekin bunday xesh-funksiyada yangi muammo paydo bo'ladi. Ikkita har xil

identifikatorlar bir xil harfdan boshlansa, ularni xesh-funksiya qiymati bir xil bo'lib qoladi. Bu holda bitta yacheykaga ikkita identifikatorlarni joylashtirish kerak bo'ladi, lekin bu mumkin emas.

Ikki yoki undan ko'p identifikatorlarning xesh-funksiyasi bir hil qiymatga ega bo'lgan holat kolliziya deb nomlanadi.

Albatta kolliziyaga ega bo'lgan, funksiyalarni to'g'ridan-to'g'ri ishlatib bo'lmaydi. Nazariy jihatdan kolliziyaga ega bo'lmagan funksiyalarni tuzish mumkin. Masalan har bir harfning kodi ketma-ket yozib chiqilsa, hosil bo'lgan son yagona manzilni ko'rsatishi mumkin. Lekin bunday funksiya uchun massiv uzunligi cheksiz bo'lib qoladi. Vaholanki kompyuter xotirasi cheklangan va bundan xulosa qilish mumkinki, kolliziyadan qutilish ilojisi yo'q. Shuning uchun kolliziyani ishlatadigan usullarni qo'llash kerak.

Rexeshlash usuli

Yuqorida ko'rsatilgan muommani yechish usullardan biri, rexeshlash usulidir.

Biror element A uchun $n=h(A)$ xisoblanadi. Agar shu manzil bo'yicha xotira yacheykasi band bo'lsa , $n_1=h_1(A)$ funksiya qiymati xisoblanadi. Agar bu manzilda ham element mavjud bo'lsa, $n_2=h_2(A)$ funksiya xisoblanadi va hokazo. Qachonki bo'sh joy topilmaguncha yoki $n_i=n$ teng bo'lgunuvcha. Bu holda jadval to'lgan deb hisoblanadi va hato beradi. Bu usulda identifikatorlar jadvali bo'sh belgisi bilan to'ldirilishi kerak. Masalan "" yoki NULL bilan.

h-funksiya quyidagi formula bilan yozilishi mumkin $h_i=(h(A) +p_i)mod N_m$, bunda p_i – qandaydir formula bilan xisoblanadigan son. N_m - identifikator jadvalidagi elementlar maksimal soni. Eng sodda usuli $p_i=i$.

Shunda formulamiz quyidagicha bo'ladi $h_i=(h(A)+i)mod N_m$. Bu holda xesh-funksiyani qiymati bir hil bo'lsa, $h(A)$ o'rindan boshlab ketma-ket yacheykalar bo'shligi tekshiriladi. Tanlangan formula uncha yaxshi emas, chunki elementlar bir joyga to'planib qoladi.

12.8-rasmda IJni rexeshlash usuli bilan to'ldirish blok-chizmasi keltirilgan. 12.9-rasimda biror-bir identifikatorni IJdan qidirish algoritmning blok-chizmasi ko'rsatilgan.

Misol taraqasida ketma-ket joylashgan n_1,n_2,n_3,n_4,n_5 jadval yacheykalarni ko'raylik va A_1,A_2,A_3,A_4,A_5 identifikatorlarni jadvalga

yuklaylik. Funksiyalar $h(A_1)=h(A_2)=h(A_5)=n_1$, $h(A_3)=n_2$, $h(A_4)=n_4$ga teng bo'lsin.

Eng sodda rexeshirlash usulini qo'llaganda, identifikatorlarni jadvalga joylashtirish ketma-ketligi 12.8-rasmda ko'rsatilgan.

Natijada A_1 elementni qidirish uchun faqat bitta solishtirish yetarli. A_2 uchun 2, A_3 uchun 2, A_4 uchun 1 va A_5 uchun 5 solishtirish kerak bo'ladi. Bundan ko'rinib turibdiki, solishtirishlar soni kolliziyaga bog'liqdir. Qancha kolliziya kam bo'lsa, shunchalik solishtirishlar ham kam bo'ladi. Undan tashqari shunday r qiymat topish kerakki, u kolliziyani jadval bo'yicha jipslashtirmasdan keng tarqatishi kerak.

Shunday usuldan biri r tasodifiy sonlar bo'lishi. Ko'paytma ko'rinishdagi xesh funksiya $h_i=(h(A)*i)mod N_m$ ham yaxshi natija beradi

Identifikatorlar jadvalini zanjir usuli bilan tuzish

Xotirani kam to'ldirilishi uni samarasiz ishlatishga olib keladi. Bu kamchilikni oldini olish uchun ko'shimcha xesh jadval ishlatiladi.

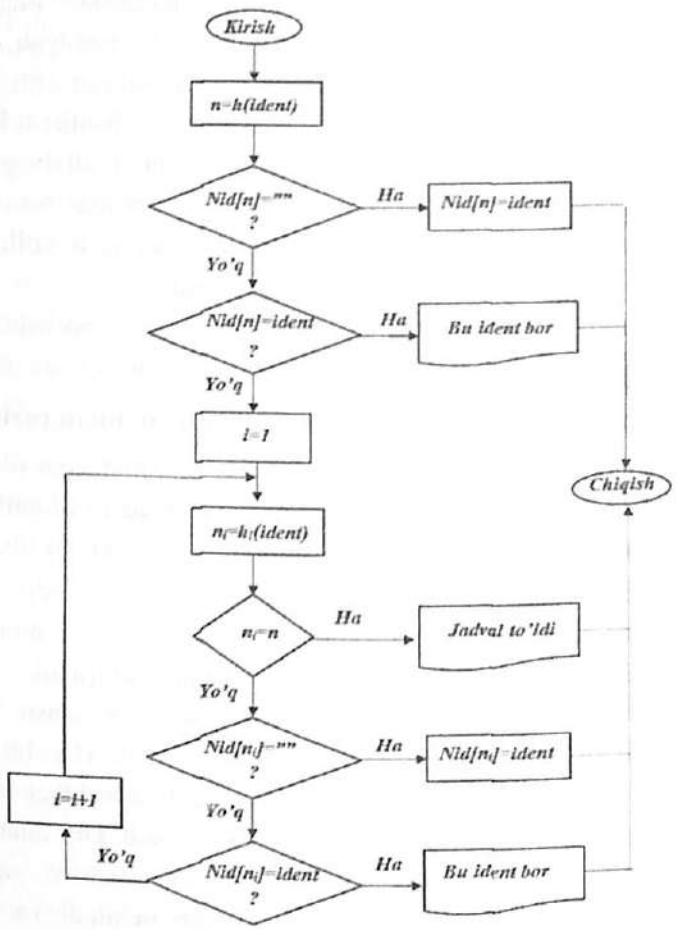
Bu jadval bir o'lchovli ko'rsatkichlar massivi bo'lib, uni qiymati Null yoki asosiy IJdagi elementning manziliga teng bo'ladi. Xesh-funksiya identifikatorning manzilini hisoblab, oldin xesh-jadvalga murojat qiladi va keyin jadvalda turgan manzil bo'yicha IJga murojaat qiladi.

Bu usulda birinchidan, IJni oldindan tozalash shart emas. Ikkinchidan, bu jadvalda bo'sh satrlar bo'lmaydi. Har bir identifikator uchun bitta xotira qismi ajratiladi. Agar xesh jadvaldagi element bo'sh bo'lsa, unga mos identifikator yo'q deb hisoblanadi. IJni dinamik struktura orqali tuzish mumkin. Bunday chizma asosida IJni tuzish – zanjir usuli deb nomlanadi. Bu usulda IJda yana bitta ustun ochiladi va unda boshqa elementga murojaat turadi. Dastavval bu ustun bo'sh bo'ladi.

Kompyuterda IJ quyidagi tuzimida berilishi mumkin:

```
// identifikator jadvali elementi ta'rifi
struct  ID
 {   string Nid ; // identifikator nomi
   ID * Next; // keyingi element manzili
 } ;
 ID * TID; // identifikator jadvali
 ID * TH[n]; // xesh jadvali
ID * JID; // maxsus ko'rsatkich;
```

12.11-rasmda indentifikator jadvalini to'ldirish ko'rsatilgan. 12.12-rasmda kiruvchi identifikatorni qidirish blok-chizmasi berilgan.

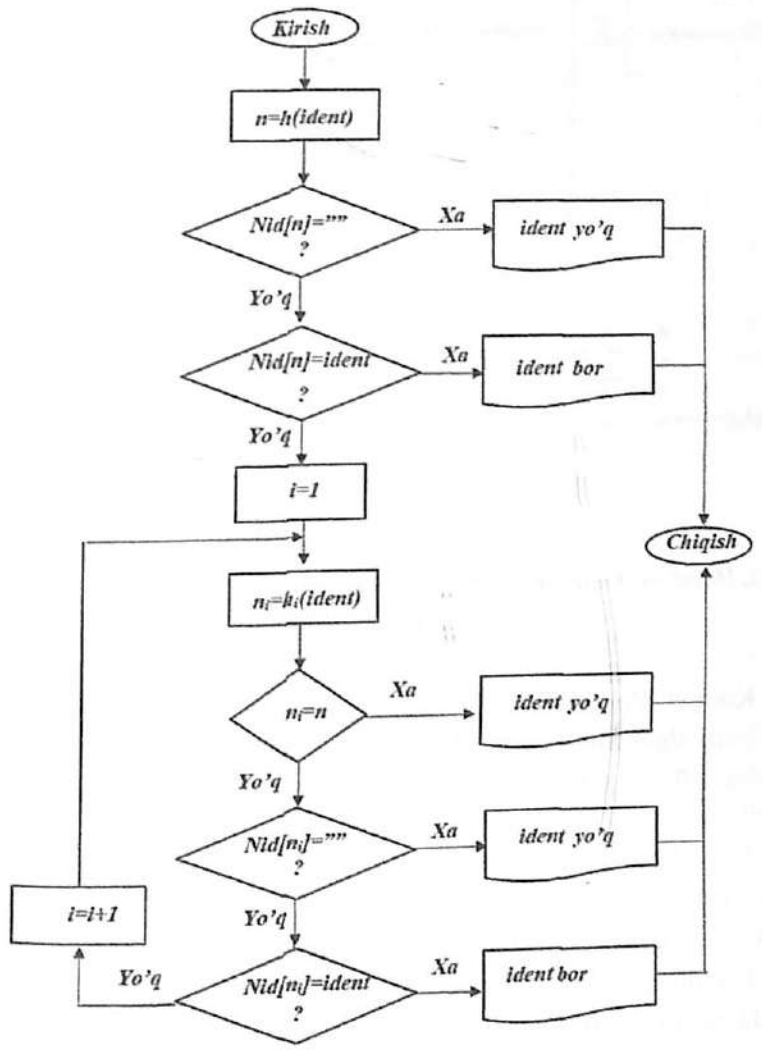


12.8-rasm. IJni rexeshlash usuli bilan to'ldirish blok-chizmasi.

Misol sifatida rexeshlash usulidagi berilganlarni olamiz. 12.13-rasmda identifikatorlar jadvali va xesh jadvali, zanjir usuli bilan to'ldirishi ko'rsatilgan.

Identifiktorlar jadvalini bunday tashkil qilishda uchragan kolliziya uchun har bir keyingi elementni manzili oldingi elementda ko'rsatiladi. Ya'ni zanjir bilan ulanib ketadi, shunung uchun zanjir usuli deb nomlangan.

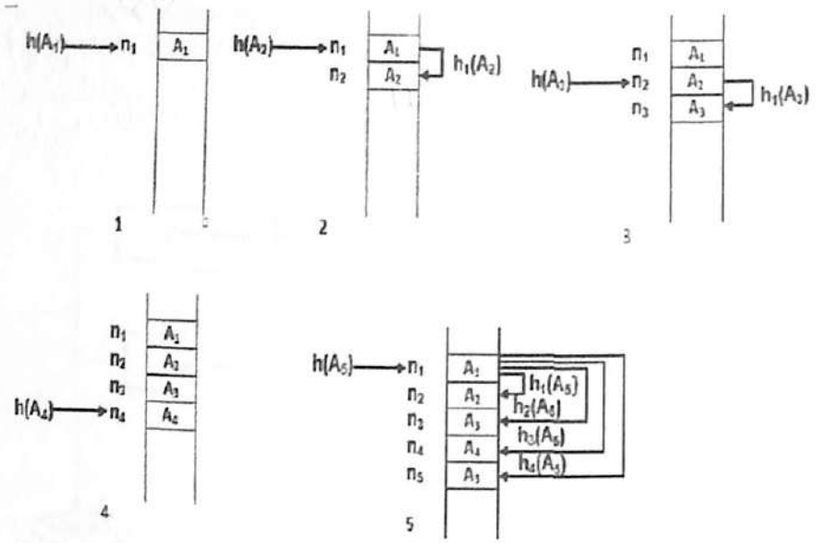
Bu usul juda ham samarali, chunki elementni jadvalga yuklash yoki qidirish vaqti o'rta xisobda xesh-funksiyani kolliziyasiga bog'likdir.



12.9-rasm. IJni rexeshlash usuli bilan qidirish algoritmning blok-chizmasi.

Usullarni kombinatsiya yordamida jadval tashkil qilish

Yuqorida ko'rsatilgan xesh-funksiyalar juda sodda hisoblanadi va ularni ishlatish talabga muvofiq emas. Yaxshi xesh-funksiya identifikatorlarni xotira bo'yicha teng taqsimlaydi va shu sababli kolliziyalar kam uchraydi. Amalda juda ko'p xesh-funksiyalar mavjud, lekin ideal funksiya haligacha topilmagan.



12.10-rasm. Oddiy rexeshirlash usuli bilan identifikator jadvalini to'ldirish misoli.

Kompilyatorlarda xesh-manzillash doimo uchrab turadi. Xesh-funksiyani algoritmi kompilyator yaratuvchining "nou-xau"si hisoblanadi. Ularning asosiy maqsadi kolliziyani kamaytirish. Yaxshi xesh-funksiya kompilyator ishini ancha tezlashtiradi.

Jadvallarni tashkil qilishda qaysi usulni ishlatish, kompilyatorni amalga oshirishiga bog'liq. Bitta kompilyator bir nechta jadvallar qurishi mumkin va ularni turli usul bilan tashkil qilishi mumkin.

Ko'pincha usullarni kombinatsiyasi ishlatiladi. Masalan zanjir usulida havola maydoniga qo'shimcha jadvalni manzili yoziladi. Agar kolliziya bo'lmasa, bu maydon bo'sh bo'ladi. Aks holda kolliziya ya'ni xesh-funksiyalar qiymati teng bo'lgan elementlar uchun, maydonga alohida jadvalni manzili yoziladi. Bu jadvalni
yuqorida aytilgan biror usul bilan tashkil qilish mumkin. Masalan tartibsiz, tartibli , binar usullar bilan.

Xulosa

Identifikatorlar jadvallarida dasturga tegishli bo'lgan barcha identifikatorlar o'z xarakteristikalar bilan saqlanadi. Bu jadvaldan

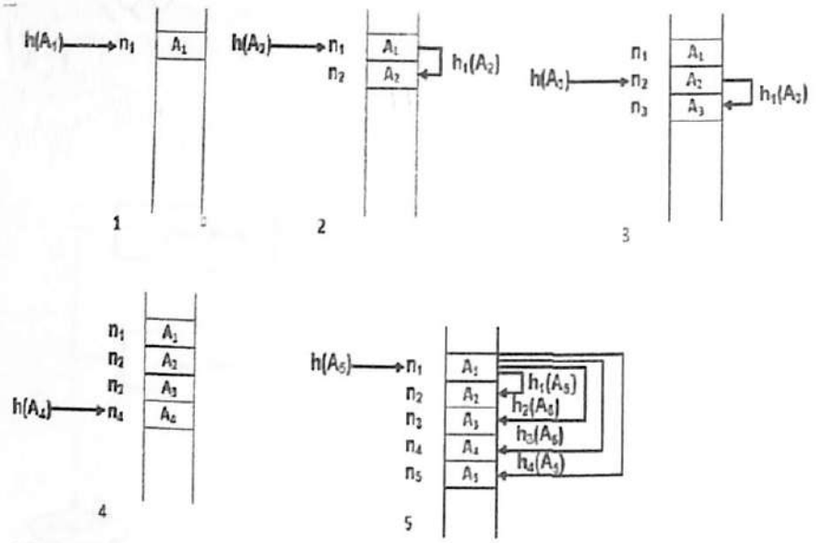
kompilyatorni barcha fazalarida ishlatiladi Identifikarolar jadvalini bir nechta usul bilan tuzish mumkin shu jumladan:tartiblangan massiv ko'rinishda, daraxt shaklida tuzish, rexeshlash usulli, zanjir usuli xamda ularning kombinatsiyalari. Daraxt usulida identifikatorlar binar daraxt tuzimida saqlanadi, bu esa identifikatorni qidirishga
ketadigan vaqt $T_k=O\ log_2(N)$) bo'ladi. Rexeshlash usuli xesh-funksiya va xesh manzillashga asoslangan. Kolliziya vaziyatlarda xesh funksiya qayta xisoblanadi. Rexeshlash usulida jadvalga element qo'shish va qidirish xisoblanadi. Rexeshlash usulida jadvalga element qo'shish va qidirish amallarga ketgan vaqt hesh-funksiyani hisoblash va kolliziya muammosini yechishga ketada.

Nazorat uchun savollar

1. Identifikator jadvalni tuzimi nimadan iborat?
2. IJ tuzish usullarini keltiring.
3. Xesh-funksiya xususiyati nimadan iborat?
4. Daraxt usulini keltiring.
5. Rexeshlash usuli nimadan iborat?
6. Zanjir usuli keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan matndan so'zlarni ajratib daraxt usuli bilan IJ tuzish dasturini tuzing.
2. Berilgan matndan so'zlarni ajratib rexeshlash usuli bilan IJ tuzish dasturini tuzing.
3. Berilgan matndan so'zlarni ajratib zanjir usuli bilan IJ tuzish dasturini tuzing.
4. Tartiblash va zanjir usuli kombinatsiyasi bilan IJ tuzish dasturini tuzing



12.10-rasm. Oddiy rexeshirlash usuli bilan identifikator jadvalini to'ldirish misoli.

Kompilyatorlarda xesh-manzillash doimo uchrab turadi. Xesh-funksiyani algoritmi kompilyator yaratuvchining "nou-xau"si hisoblanadi. Ularning asosiy maqsadi kolliziyani kamaytirish. Yaxshi xesh-funksiya kompilyator ishini ancha tezlashtiradi.

Jadvallarni tashkil qilishda qaysi usulni ishlatish, kompilyatorni amalga oshirishiga bog'liq. Bitta kompilyator bir nechta jadvallar qurishi mumkin va ularni turli usul bilan tashkil qilishi mumkin.

Ko'pincha usullarni kombinatsiyasi ishlatiladi. Masalan zanjir usulida havola maydoniga qo'shimcha jadvalni manzili yoziladi. Agar kolliziya bo'lmasa, bu maydon bo'sh bo'ladi. Aks holda kolliziya ya'ni xesh-funksiyalar qiymati teng bo'lgan elementlar uchun, maydonga alohida jadvalni manzili yoziladi. Bu jadvalni
yuqorida aytilgan biror usul bilan tashkil qilish mumkin. Masalan tartibsiz, tartibli , binar usullar bilan.

Xulosa

Identifikatorlar jadvallarida dasturga tegishli bo'lgan barcha identifikatorlar o'z xarakteristikalar bilan saqlanadi. Bu jadvaldan

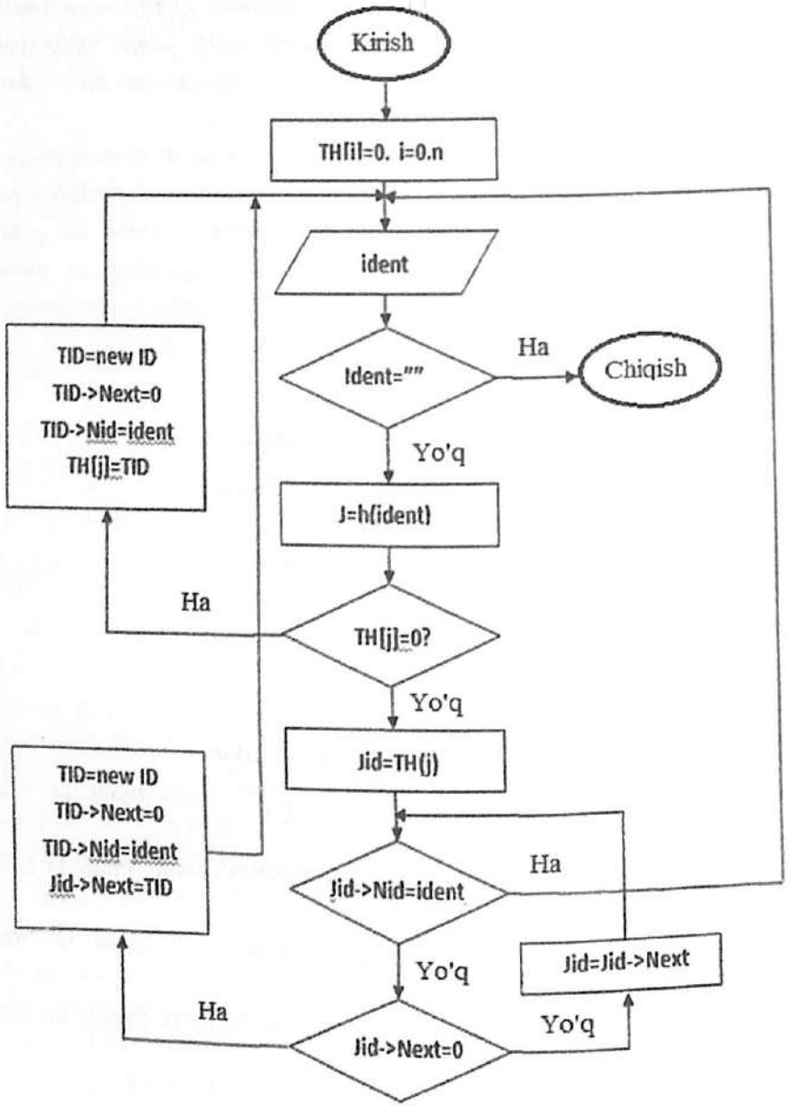
kompilyatorni barcha fazalarida ishlatiladi Identifikarolar jadvalini bir nechta usul bilan tuzish mumkin shu jumladan:tartiblangan massiv ko'rinishda, daraxt shaklida tuzish, rexeshlash usulli, zanjir usuli xamda ularning kombinatsiyalari. Daraxt usulida identifikatorlar binar daraxt tuzimida saqlanadi, bu esa identifikatorni qidirishga
ketadigan vaqt $T_k=O\ log_2(N)$) bo'ladi. Rexeshlash usuli xesh-funksiya va xesh manzillashga asoslangan. Kolliziya vaziyatlarda xesh funksiya qayta xisoblanadi. Rexeshlash usulida jadvalga element qo'shish va qidirish xisoblanadi. Rexeshlash usulida jadvalga element qo'shish va qidirish amallarga ketgan vaqt hesh-funksiyani hisoblash va kolliziya muammosini yechishga ketada.

Nazorat uchun savollar

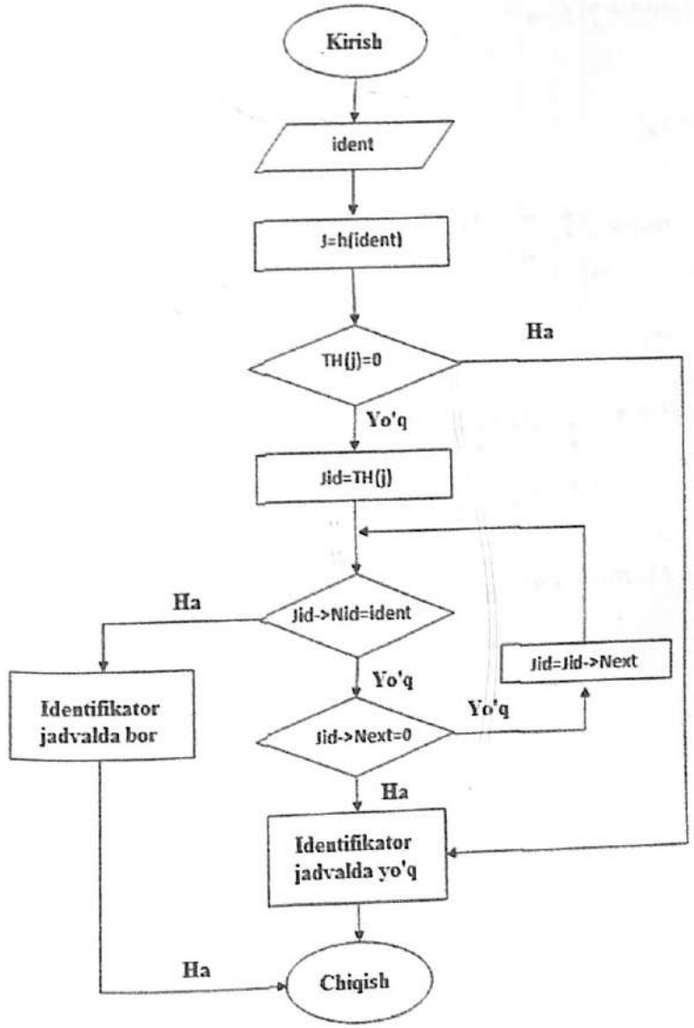
1. Identifikator jadvalni tuzimi nimadan iborat?
2. IJ tuzish usullarini keltiring.
3. Xesh-funksiya xususiyati nimadan iborat?
4. Daraxt usulini keltiring.
5. Rexeshlash usuli nimadan iborat?
6. Zanjir usuli keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan matndan so'zlarni ajratib daraxt usuli bilan IJ tuzish dasturini tuzing.
2. Berilgan matndan so'zlarni ajratib rexeshlash usuli bilan IJ tuzish dasturini tuzing.
3. Berilgan matndan so'zlarni ajratib zanjir usuli bilan IJ tuzish dasturini tuzing.
4. Tartiblash va zanjir usuli kombinatsiyasi bilan IJ tuzish dasturini tuzing

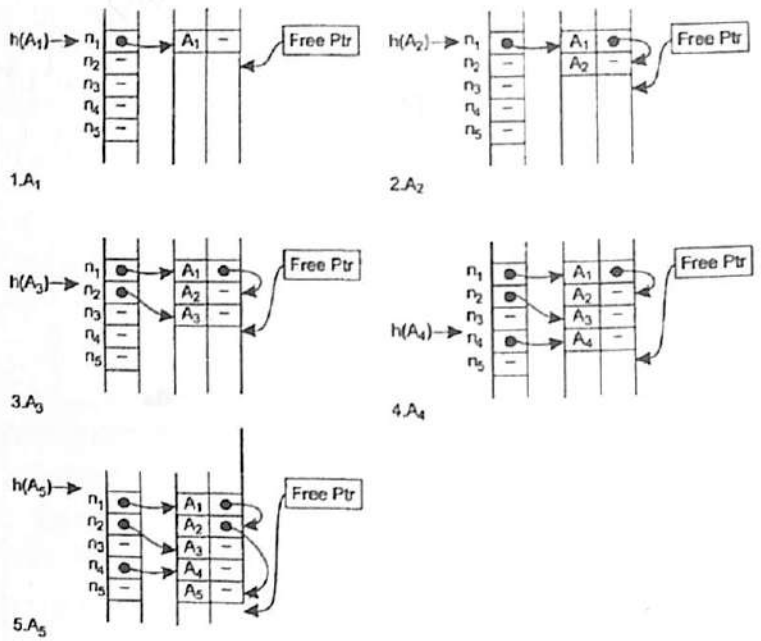


12.11-rasm. Zanjir usuli bilan IJni to'ldirish blok-chizmasi.



12.12-rasm. Zanjir usuli bilan identifikatorni qidirish blok-chizmasi.

Zanjir usulida qo'shimcha xesh jadvalidan foydalaniladi. Bu usul eng samarali xisoblanadi. Kombinatsiya usullari xotirani samarali foydalanush uchun qo'llanadi.



12.13- Rasm Zanjir usuli bilan identifikator jadvalini to'ldirish.

II-BO'LIM. ZAMONAVIY DASTURLASH TIZIMLAR TO'G'RISIDA QISQACHA MA'LUMOT

13- BOB. KOMPONENTALI VA VIZUAL DASTURLASH

Tayanch iboralar: *obyektga yo'naltirilgan dasturlash, komponentali dasturlash, vizual dasturlash, hodisali dasturlash.*

Hozirgi vaqda obyektga yo'naltirilgan dasturlash (OYD) asosida tuzilgan dasturlarni yaratish va kuzatish uchun qo'llanadigan dasturlash tizimlari ko'pdir. Undan tashqari bu tizimlarda nafaqat OYD tillari qo'shilgan, vaholanki OYD asosida yaratilgan vositalar ham qo'shilgan. Bunday vositalar dastur loyihalashda ancha qulayliklar yaratadi. Ular asosida dasturlar yaratish komponent dasturlash va vizual dasturlash usullari deb nomlanadi.

Komponentali dasturlash

Komponenta bu dasturni yaratishda ishlatiladigan abstakt jamlovchi elementidir. Komponenta alohida komponentalar kutubxonasida joylashadi. Komponentalar OYDdagi sinflardan farq qiladi. Sinf interfeysni ta'riflashdan tashqari, unin amalga oshirishi ham beradi. Komponentalarni ta'riflashda interfeyslarni amalga oshirishi ko'zda tutilmagan. Sinf biror bir tilda ta'riflanadi, komponentalar tilga bog'liq emas. Va nihoyat komponenta sinfga nisbatan kattaroq tuzim. Chunki u bir nechta bir-biriga bog'langan sinflardan iborat bo'lishi mumkin.

Komponent dasturlashda komponentalar kutubxonasida mavjud bo'lgan kerakli komponentalar dasturga qo'shiladi. Komponentalar dasturlash tizimiga alohida qo'shilishi yoki olib tashlanishi mumkin.

Vizual dasturlash

Tayyor vizual vositalardan foydalanib dastur ilova yaratish vizual dasturlash deb nomlanadi. Bunda dasturchi dastur kodini yaratmaydi. U faqat dastur shakli qanday bo'lishini ko'rsatadi. Masalan dastur darchasiga tugmachalar, matn darchalari, diagrammalar va hokazo. Vizual elementlarni ("komponentalarni") qulay ko'rinishda joylashtiradi. Ba'zi bir elementlar fokusga ega bo'lib, har hil hodisaga ega bo'lishi mumkin.

Shunda kerakli hodisalarga dastur kodi avtomatik ravishda, biror bir qolib bo'yicha generatsiya qilinadi. Dasturchi faqat shu hodisa uchun algoritm kodini yozsa yetarlikdir. Bunday dasturlash hodisali dasturlash deb nomlanadi. Hodisali va vizual dasturlashlar baravariga ishlatilishi mumkin. Vizual dasturlashni asosida ham OYD o'tiribdi. Chunki har bir viziual element uchun biror bir sinf yoki sinflar to'plami mavjud. Bu sinflarda elementni interfeysi va usullari ta'riflangan.

Xulosa

Zamonaviy dasturlash tizimlarida dasturlarni tezkor va oson yaratish uchun bir gurux komponentalar qo'shilgan. Bu komponentalar kutubhonalaridan keng foydalanish uchun komponent dasturlash va vizual dasturlash usullari qo'llanadi.tarqalgan. Identifikatorlar jadvallarida dasturga tegishli bo'lgan barcha identifikatorlar o'z xarakteristikalar bilan saqlanadi. Komponent dasturlashda komponentalar kutubxonasida mavjud bo'lgan kerakli komponentalar dasturga qo'shiladi. Vizual dasturlashda tayyor vizula vositalardan foydalaniladi.

Nazorat uchun savollar

1. Komponentali dasturlash nimadan iborat?
2. Vizual dasturlash nimadan iborat?

14-BOB. BORLAND KOMPANIYASINING DASTURLASH TIZIMLARI

Tayanch iboralar: *Turbo Pascal, Borland Pascal, Delphi, C++ Builder, RAD-texnologiya, BDE, RAD Studio.*

Shaxsiy kompyuterlar uchun zamonaviy dasturlash tizimlarni yaratishda Borland firmasi katta hissa qo'shdi. Bu firma boshida Paskal tili asosida dasturlash tizimini yaratib, keyin SI, SI++, Prolog tillari uchun ham tizimlar yaratdi.

Turbo Pascal tizimi

Eng mashhur bo'lgan birinchi dasturlash tizimi Paskal tili bilan bog'liq bo'lib, shahsiy kompyuterning MS DOS operatsion tizimida ishlardi.

Hozirgi vaqtda bu til asosida bir nechta tizimlar muvoffaqiyatli ishlatilib kelinmoqda. Shu jumladan MS DOSda Turbo Pascal 7.0, Borland Pascal, MS Windows uchun Delphi tizimlari.

Turbo Pascalda standart modullar qonsepsiyasi qo'llangan, bunda modul ikki qismdan iborat bo'lgan: interfeys qismi va aks ettirish qismi, OYD elementlari ham kiritilgan edi, lekin MS DOS uchun bular rivojlanmadi.

Bu tizim dasturlash jarayonini avtomatlashtirish uchun va dasturlarni sozlash uchun mo'ljallangan edi. Uning tarkibiga quyidagi vositalar kirgan edi:

- ko'p oynali matn tahrirchi. Unda quyidagi imkoniyatlar berilgan:
  - fayl tizimida qism-dasturlarini(modullarni) arhivdan tezkor qidirish;
  - matnni ekranga chiqarish va uni tahrirlash;
  - keyingi ishlarda foydalanish uchun matnni arhivda saqlash.
- Turbo Pascal dasturlash tilining translyatori. Unda matnda sintaktik hatolarni fiksirlaydigan va ko'rasatadigan qism-tizim;
- shahsiy kompyuterni tashqi qurilmalari bilan ishlaydigan tizimli modullar to'plami;

- modullarni jamlovchisi vositasi. Unda oldin translyatsiya qilingan qism-dasturlar va kutubxona modullari xamda dasturni o'zi yagona ishlovchi dasturga yig'iladi;
- dasturni qadam va qadam tartibotida ishlatuvchi va kompyuter hotirasidagi oraliq qiymatlarni nazorat qiluvchi sozlovchi dastur vositasi. Sozlovchi dastur, nazorat nuqtalar qo'yish imkoniyatlarini beradi. Bu nuqtalarda dastur to'htaydi va oraliq qiymatlarni ko'rish imkoniyati bo'ladi. So'ng ishni xuddi shu usul bilan davom etsa bo'ladi.

Turbo Pascal dasturlash tizimi MS DOS tarkibida bo'lgani uchun undagi standart modullarni qo'llash imkoniyati mavjud. Masalan: Graph – grafik qism-tizimi; Crt – konsol qism-tizim. Lekin bu imkoniyatlar tizim ma'lumotlari to'g'risida qo'shimcha bilimga ega bo'lishini talab qiladi.

Bunday kamchilikdan erkin bo'lgan Delphi dasturlash tizimini ko'ramiz.

Delphi tizimi

Delphi dasturlash muhiti Borland kompaniyaning Paskal tili uchun yakuniy muhit bo'ldi. Bu til qayta ishlandi va natijada unga modul dasturlashdan tashqari, obyektga-yo'naltirilgan tillarning namunalari qo'shildi va unga yangi Object Pascal nomi berildi. Keyinchalik tilga tarmoq texnologiyalari qo'shildi va yana qayta ishlanib, unga Delphi nomi berildi.

Paskal tili kabi Object Pascal va Delphi tillari universal tillar qatoriga kiradi. Bu tillarni barcha sohada ishlatsa bo'ladi. Bu tillar dunyoda keng tarqalgan obyektga-yo'naltirilgan SI++ dasturlash tiliga yaqindir. SI++ tili, Object Pascal va Delphi tillardan oldinroq chiqgan edi.

Lekin ulardagi OYD elementlari C++ tiliga nisbatan murakkabroq.

Delphi dasturlash tizimi vizual dasturlash texnologiyasini qo'llashga yo'naltirilgan. Bunday holat MS DOSdan WINDOWS operatsion tizimiga o'tishga olib keldi.

Delphi nafaqat integranlangan dasturlash tizimidir, balki dastur ishlab chiqaruvchi muhitdir (*IDE-Intrgrated Development Yenvironment*). Bu muhitda juda ko'p komponentalar mavjud va ular dasturchilarga keng imkoniyatlar yaratib beradi.

Foydalanuvchi interfeysini (muloqatini) loyihalash va dasturlash jarayonida, dasturchi faqatgina komponentalarni interfeys shaklnig zaruriy joyga va kerakli tarzda joylashtiradi. Loyihalash natijasi monitorda bevosita ko'rinib turadi. Hech qanday kompilyatsiyaning hojati bo'lmaydi.

Shu bilan birga tizim avtomatik ravishda Delphi tilida dastur kodini yasaydi va keyinchalik dasturchi tizimli qomponentalarning hossalarini o'zgartirish mumkin va ba'zi bir hodisalar uchun dastur kodini yaratishi mumkin. Masalan sichqon tugmachasi komponentasiga bosilgan hodisaga kerakli bo'lgan amalning algoritmining dastur kodini kiritishi mumkin.

Hullas dasturlash jarayoni tizimli komponentalarni hossalarin o'zgartirish va hodisalar uchun dastur kodini yaratishdan iborat bo'ladi.

Tayyor qomponentalar soni doimo kengayib boradi. Ular komponentalar kutubxonasida joylashadi (*VCL – Visual Component Library*). Bunday tehnologiya "ilovani tezkor ishlab chiqarish tehnologiyasi", RAD-texnologiya deb nomlanadi (*RAD - Rapid Application Development*).

Delphi tizimi yordamida turli-tuman sohadagi masalalarni yechish mumkin. Undagi berilganlarning turlarini jiddiy nazorati, rekursiv protseduralar, hamda ichki funksiyalarning mavjudligi, ihcham va ishonchli ilovalarni yaratishga imkoniyatlar beradi.

Delphi tizimiga qo'shimcha kiritilgan, berilganlar bazasiga ulanish va ishlash komponentalari imkoniyatlari yanada oshiradi. Unga kiritilgan berilganlar bazasini protsessori (*BDE - Borland Data Base*), tadbiqiy dasturchi va berilganlar bazasi o'rtasidagi vosita bo'ldi. BDE protsessori tarkibida keng tarqalgan berilganlar bazasini boshqaruv tizimlarning drayverlari mavjud. Shu jumladan: Microsoft Access, Foxpro, Paradox, dBase va boshqalar. Undan tashqari, alohida yaratilgan berilganlar bazasining boshqarish tizimi ham bor. Uning nomi INTERBASE keyinchalik FIRE BIERD deb nomlangan. Tashqi ba'zalar bilan bog'lanish uchun, ODBC (*Open Data Base Connective*) drayveri qo'shilgan.

C++ Builder tizimi

O'zining imkoniyatlari bilan C++ Builder tizimi Delphi tizimidan xech qolishmaydi. Ikkala tizimda vizual dasturlash usuli qo'llanadi. Ularning asosiy farqi, C++ Builder tizimning tayanch dasturlash tili sifatida C++ tili olingan. Bu ikkala tizim VCL vizual komponentalar

kutubxonasiga asoslangan. Bu kutubxona to'liq obyektga yo'naltirilgan dasturlash asosida qurilgan. Uning asosini TObject tayanch sinfi tashkil qiladi. VCLni barcha sinflari shu sinfning avlodlari bo'lib, vizual dasturlashning barcha komponentalarini o'ziga qamrab olgan.

Borland Studio - integranlangan dastur yaratish muhiti

Borland kompaniyasi 2002 avgust oyida, Borland Developer Studio 1.0 nomi bilan integranlargan dastur yaratish muxitini chiqargan. Undan so'ng 2003 yil dekabrida Borland Developer Studio, 2.0 2004 yil noyabrida Borland Developer Studio 3, 2005 yilning ohirida Borland Developer Studio 4 naqli chiqdi. 2006 yilning noyabrida yangi CodeGear kompaniya ochib, 2007 yilning mart oyida CodeGear RAD Studio 2007 muhitini chiqardi.

2008 yil 1 iyulida CodeGear Yembarcadero Technologies kompaniyasiga sotildi. Bu kompaniya RAD Studio XE 1-7 integranlangan muhitlarni chiqardi.

RAD Studio XE7 C++ va Delphi tilida ishlovchi dasturchilarga hilma-hil platforma va har hil ekran o'lchovi uchun yagona dastur kodini yaratish, imkoniyatlarini beradi. Bu imkoniyatlar tezkor ravishda smartfonlar uchun dastur ta'minotini yaratishga qulayliklar yaratadi. C++ yoki Delphi tilida yaratilgan dasturni Android yoki IOS platformaga bemalol o'tkazish mumkin bo'ladi.

Xulosa

Shaxsiy kompyuterlar uchun zamonaviy dasturlash tizimlarni yaratishda Borland firmasi katta hissa qo'shdi. Birinchi dasturlash tizim Paskal tili asosida bo'lgan, bu tizim MS DOS operatsionn tizimida ishlagan. Bu tizim Turbo Pascal deb nomlangan. Paskal tilida asoslangan ohirgi tizim Delphi deb nomlangan. Bu tizimda obyektga yo'naltirilgan dasturlash paradigmasi kiritilgan. Undan tashqari ko'p komponentali kutubxona (VCL) orqali komponentali va vizual dasturlash imkoniyatlarni oshirgan. Parallel ravishda S++ tili asosida S++ Builder tizimi yaratilgan. Natijada Borland kompaniyasi Borland Developer Studio nomli integranlangan dastur yaratish muxitini yaratgan. Bu muxit o'ziga ikkala tilni qoplab olgan. Hozirga vaqtda bu tizim ajdodi bo'lmish RAD

Studio XE 1-7 integranlangan muhitlarni chiqdi. Bu tizimda hilma-hil platformalarda dasturlar yaratish mumkin, shu jumlada mobil uskunalar uchun.

Nazorat uchun savollar

1. Turbo Pascal tizimi to'g'risida ma'lumot bering.
2. Delphi tizimi to'g'risida ma'lumot bering.
3. C++ tizimi to'g'risida ma'lumot bering.
4. RAD – texnologiya nimadan iborat?
5. RAD Studio XE7 muxit nimadan iborat?

15-BOB. MICROSOFT KOMPANIYASINI DASTURLASH TIZIMLARI

Tayanch iboralar: *Visual Studio,Visual Basic, VBA, Visual C++, .Net Framework, RAD-texnologiya, BDE, RAD Studio.*

Shaxsiy kompyuterlar uchun keng tarqalgan dasturlash tizimlariga, Microsoft kompaniyasining chiqargan tizimlari kiradi. Microsoft kompaniyasi chiqargan tizimlar Visual Studio integrallangan muhitni tashkil etib, yagona stil va mukammal interfeysi bilan ajralib turadi. Ko'p darchali interfeys yaratilgan, sozlanayotgan va bajarilayotgan dasturlar to'g'risida turli ma'lumotlarni birdaniga ko'rish imkoniyatini beradi. Tayanch dasturlash tillari uchun alohida rivojlangan sozlovchilar yaratilgan. Dasturchi har qanday daqiqada biror bir obyektni holatini ko'rishi va uning qiymatini o'zgartirish imkoniyatga ega. Kerakli nazorat nuqtalaridan dasturni qayta kompilyatsiya qilmasdan bajarishni davom etishi mumkin.

Visual Basic tizimi

Microsoft kompaniyasining eng mashhur bo'lgan va ko'p tarqalgan dasturlash tizimlari, bu Visual Basic tizimidir. Tizimning tayanch tili Basic o'zini boshlang'ich naqlidan ancha farq qiladi. Hozirgi vaqtda bu obyektga yo'naltirilgan tili bo'lib, boshqa yangi tillardagi imkoniyatlarini o'ziga olgan, lekin soddaligi jixatidan va uni o'rganish bo'yicha, boshqa tillardan ancha yuqori turadi. Bu tizimda vizual dasturlash texnologiyasi keng ishlatiladi.

Shakllarni yaratish jarayonida avtomatik ravishda, Visual Basic tilida dastur kodi yaratiladi. Faqat hodisa uchun protseduralarni yaratish yetarli.

Ish jarayonida dasturchi darcha qurish tartibotidan boshlab, dastur tahrirlash tartibotiga hech qanday qiyinchiliksiz o'tish imkoniyatlari bor. Dasturchi darchaga yangi shakllar, yangi boshqarish elementlarini qo'shish va ularni hossalarini o'zgartirishi mumkin. Dasturlash tizimiga kiritilgan sozlovchi, Visual Basic tilida ishlaydi, bu esa ancha qulayliklar yaratadi.

Umuman aytganda, Visual Basic tizimi bu har hil dasturiy mahsulotni yaratishdagi integranlangan muhitdir. Bu muhitda alohidagi mustaqil dasturni yaratishi va uni tizimga bog'lanmagan holda ishlatilishi mumkin.

Visual Basic tizimida, Microsoft Office ilovalar bilan ishlash juda qulaydir. Chunki ular uchun alohida sinflar kutubxonasi mavjuddir. Ushbu sinflardagi usullar orqali, dasturchi o'z dasturida bu ilovalarni yuritishi mumkin va ulardagi hujjatlarini bir ilovadan boshqasiga o'tkazishi mumkin.

VBA tizimi

Microsoft Office ilovalari (Word, Yexel, Access, PowerPoint va boshqalar) uchun kompaniya tomonidan, Visual Basic for Applications yoki VBA mahsus muhiti yaratilgan. Bu muhit ilovalar uchun yagona bo'lib, ular uchun dasturlash imkoniyatlarini beradi. VBA tizimi alohida dastur yaratmaydi. U faqat ilovalar bilan birgalikda ishlaydi va ilovani ichida ishga tushiriladi. Visual Basic tizimi bajaruvchi kodni yaratsa, aksincha VBA tizimi "translyator-interpretator" tartibotida ishlaydi. Ya'ni VBA tilidagi dastur uchun ichki ko'rinishdagi kod yaratiladi va bu qod hujjat bilan birga saqlanadi. Shu hujjat bilan ishlash jarayonida interpretator ichki kodni bajaradi. VBA qo'llashda hodisaviy dasturlash texnologiyasi ishlatiladi, ya'ni qanday hodisa uchun dastur kodi yoziladi va shu hodisa ro'y berganda kod ishga tushadi.

Visual C++ tizimi

Agar dasturchilarga Visual Basic tizimining imkoniyatlari kamlik qilsa, Visual C++ dasturlash tizimi taklif qilinadi. Bu tizimning tayanch tili Si++. Visual C++ tizimda har qanday ilovalarni yaratish uchun to'liq kutubxonalar to'plami mavjud. Bu tizimda dasturchilar uchun keng ma'lumotnoma bor. Unda har qanday vaziyatlar uchun yechimlar va misollar mavjud. Visual C++ tizimidagi sozlovchi bevosita Si++ tilidagi boshlang'ich kodida hatoliklarni ko'rsatadi. Xotira, registrlardagi qiymatlarni ko'rish va o'zgartirish imkoniyatlarni beradi. Hozirgi vaqtda Visual C++ va Visual Basic tizimlari eskirib qolgan deb, yangi dasturiy mahsulotlarda tarqatilishi to'htatilgan.

C# tili va .NET konsepsiyasi

Internet tarmog'i rivojlanishi bilan dasturlarga qo'shimcha talablar qo'yilmoqda. Dasturlar kodi kompyuterlarga va operatsion tizimlarga, bog'liq bo'lmasligi talab qilinadi. Bu esa Java konsepsiyasini yaratilishiga

olib keldi. Unda dastur kodi, bayt-kod nomli ichki oraliq tilga translyatsiya qilinadi. Oraliq koddagi dasturni, Java virtual mashina (JVM) interpretatori qayta ishlaydi.

Bu texnologiya orqali qayta ishlangan dastur, JVM mavjud bo'lgan har qanday platformada interpretatsiya qilinishi mumkin. Java texnologiyasi bo'yicha faqat yagona bajaruvchi kod hosil bo'ladiyu. Aksincha boshqa tillar har bir tizim uchun alohida bajaruvchi kodni hosil qiladi. Java tili dasturlarni nusxalash muammosini hal qildi. Yuqorida aytilgan tehnologiya yagona Java tili uchun qo'llanadi. Shu kamchiliklarni hisobga olib, Microsoft kompaniyasi yangi .NET texnologiyani yaratdi. Bu texnologiya bo'yicha bir nechta tillar yagona oraliq kodni yaratadi. Va shu orada kompaniya yangi C# tilini yaratdi. Java va C# tillari negizida C++ turadi. Ohirgi vaqtda Microsoft kompaniyasi yangi dasturlash tizimini yaratdi. Bu tizimning nomi .NET Framework bo'lib, unda umumiy tillar muhitini amalga oshirildi (*Common Language Runtime – CLR*) va sinflar kutubxonasi yaratildi. Bu tizimda Visual Basic .NET, Visual C# va Visual C++, VBScript va Jscript tillar ishlatiladi. .NET texnologiya bu yagona universal platforma bo'lib, har qanday dasturni yaratishga imkoniyatlar beradi – oddiy ilovalardan boshlab, berilganlar bazasi, tarmoq hizmati, mobil va ko'chma qurilmalarning ilovalarigacha. .NET Framework tizimning yadrosida (*Common Language Infrastructure – CLI*) tafsifi turibdi. Bu tafsifda yagona oraliq til (*Common Intermediate Language – CIL*) va barcha tillar uchun turlarni moslashtiruvchi, umumiy turlar tizimi (*Common Type System – CTS*) ta'riflangan. Java texnologiyasiga o'xshab xar qanday dasturlash tildan, CIL-yagona oraliq tilga dastur translyatsiya qilinadi. Lekin CLI tafsifi bo'yicha, oraliq tildagi dastur virtual mashina yordamida interpretatsiya qilinmaydi, aksincha bevosita bajarish jarayonida *just-in-time compilers*-JIT kompilyator bilan mashina tiliga o'tkaziladi. CILdagi dasturlarni translyatsiya qilishini va bajarishini qo'llab quvvatlaydigan tizim, virtual bajaruvchi tizim (*Virtual Yexecution System – VES*) deb nomlanadi. VES har qanday hisoblash tizimda va platformalarda qo'llanishi mumkin.

Windows operatsion tizimida amalga oshirilgan VES umumtillar bajaruvchi muxit CLR deb nomlanadi. .NET Framework tizimi aynan shunga asoslangan. Bu tizimda har qanday tildagi, har qanday elementar

tur uchun CILda mos tayanch turi mavjud. Bunday samarali imkoniyatlarga ega bo'lgan, CLR muxitiga barcha boshqa kompaniyalar o'z tizimlarini qo'shmoqda. Shu jumladan Delphi dasturlash tizimni 8 versiyasi ham .NET texnologiyani qo'llab quvvatlaydi. Hozirgi vaqtda CLIga .Net Compact Framework (mobil uskunalar va Xbox o'yin uskunalar uchun), Mono (OS Linux tizimi), Portable .NET (dotGNU loyihalar) va boshqalar kiradi.

Xulosa

Microsoft kompaniyasi chiqargan tizimlar Visual Studio integrallangan muhitni tashkil etib, yagona stil va mukammal interfeysi bilan ajralib turadi. Microsoft kompaniyasining eng mashhur bo'lgan va ko'p tarqalgan dasturlash tizimlari, bu Visual Basic tizimidir. Tizimning tayanch tili Basic o'zini boshlang'ich naqlidan ancha farq qiladi. Microsoft Office ilovalari (Word, Yexel, Access, PowerPoint va boshqalar) uchun kompaniya tomonidan VBA mahsus muhiti yaratilgan. Agar dasturchilarga Visual Basic tizimining imkoniyatlari kamlik qilsa, Visual C++ dasturlash tizimi taklif qilinadi. Hozirgi vaqtda Visual Basic va Visual C++ tizimlar kompaniya tomonidan tarqatilishi to'xtatilgan. Microsoft kompaniyasi yangi .NET texnologiyani yaratdi. Bu texnologiya bo'yicha bir nechta tillar yagona oraliq kodni yaratadi. Va shu orada kompaniya yangi SI# tilini yaratdi. Java va SI# tillari negizida Si++ turadi. Ohirgi vaqtda Microsoft kompaniyasi .NET Framework tizimini yaratdi. Bu tizimda Visual Basic .NET, Visual C# va Visual C++, VBScript va Jscript tillar ishlatiladi.

Nazorat uchun savollar

1. Visual Basic tizimi to'g'risida ma'lumot bering.
2. VBA tizimi to'g'risida ma'lumot bering.
3. Visual C++ tizimi to'g'risida ma'lumot bering.
4. C# tili va .Net konsepsiyasi to'g'risida ma'lumot bering.

16-BOB. UNIX , LINUX VA GNUDAGI DASTURLASH TIZIMLARI

Tayanch iboralar: *UNIX, Linux, GNU.*

UNIX operatsionn tizimi va uning Windows shakli Linux buyruqlar satiriga moslashgan edi. Hozirgi vaqtda ham bu tizimlar aktual hisoblanadi. Hozirgi vaqtda grafik interfeyslari chiqganiga qaramasdan, u o'zini samaradorligini isbotladi. Bu tizimlarda buyruqlar satri orqali barcha amallar bajariladi. Mahsus topshiriklarini boshqarish interpretator tili, buyruq satrini qayta ishlaydi va bajaradi. Bu til UNIXda buyruqlar interpretatori deb nomlanadi. Bunday interpretatorlardan bir nechta bo'lib, ulardan eng ko'p tarqalganlari - bu *shell* (*csh*), *Bourne shell* (*sh*), *Korn shell* (*ksh*) va *Bourne again shell* (*bash*).

Bu interpretatorlar faqat bitta buyruq emas, balki aksincha bir necha buyruqlardan iborat bo'lgan, komand faylini ham qayta ishlashi mumkin. Undan tashqari UNIX tizimida dasturni hayot siklini qo'llab quvvatlaydigan dasturlar majmuasi mavjud:

- matn tahrirchi ***vi*** (***visual editor*** – vizual tahrirchi). Bu tahrirchi matn va buyruqlarni muloqat tartibotida kiritishga imkon beradi;
- har xil tillarni kompilyatorlari, assembler tilidan boshlab, obyektga yo'naltirilgan C++ va Java tillargacha;
- dinamik kutubxonalar;
- jamlovchi dastur ***ld.*** Bu dastur oldindan translyatsiya qilingan modullar va kutubxonalar komponentalarini yagona bajaruvchi dasturga yig'adi;
- kompilyatsiyani va yig'ishni boshqarish tizimi (***Make***);
- boshlang'ich matnni versiyalarini boshqarish tizimi (*Source Code Control System*, ***SCCS***). Bu tizim har qanday o'zgarishlarni kuzatadi va saqlab qoladi. Har qanday vaqda oldingi naqllarga qaytish imkoniyatini beradi;
- dasturni qismlari qancha vaqt bajarilishini aniqlaydigan profilirovkachi ***prof;***
- C dasturlash tilida yozilgan dasturni, sintaktik tahlil qiluvchi dastur ***lint;***

- keng sozlovchi ***dbx***. Sozlovchi dasturni qadamma-qadam bajarish imkoniyatini beradi va sozlovchi ma'lumotlarni dasturlash tilida beradi, bu esa o'z navbatida sozlash jarayonini ancha soddalashtiradi.
- leksik tahlilchilarni tuzish dasturi ***Lex***. Bu dastur biror bir dasturlash tilida yozilgan, boshlang'ich matnni leksemalarga ajratib beradi;
- sintaktik tahlilchini tuzish dasturi ***Yacc***. Bu dastur, dasturlash tillarni sintaktik tahlilchisini yaratadi. Dasturga kiruvchi ma'lumot sifatida, tilning grammatikasi va semantik amallari beriladi. Dastur ishlash natijasida, C tilida sintaktik tahlilchi dasturini tuzib beradi.

GNU loyiha

Massachuses Texnologik Institutni suniy tafakkur laboratoriyasini xodimi Richard Stolmen 1983 yilda mobil, ya'ni ko'chma operatsion tizim GNU (***GNU's Not Unix***) loihasini taklif qildi. Bugun GNU tizimdan foydalanuvchilar soni o'n milliondan oshib ketdi. Bu tizimning afzalligi shundaki, dastur ochiq kod va elektron xujjati bilan tarqatiladi. GNU mutlaqo bepul dasturlash tizimi bo'lib, UNIX tizimni kengaytirilgan shakli bo'ladi. Shu jumladan, tizimda faylar nomi uzun bo'lishi ularning naqllari saqlanishi mumkin. Tizimning yadrosi GNU Hurd deb nomlanadi. Bu yadro ko'p oqimli serverlarga asoslangan. 1992 yilda GNU/Linux yadrosi yaratildi va keng tarqaldi. GNU loyihasidagi dasturlash tizimlari to'liq UNIXni dasturlash tizimini qaytaradi va undagi barcha dasturlarni o'z ichiga oladi. Shu jumladan:

- buyruqlar interperetatori ***bash***;
- ko'p tillarga mo'ljallangan kompilyator ***GCC*** va assembler ***GAS;***
- sozlanib kengayuvchi, ekranli tahrirchi ***EMACS;***
- leksik taxlilchi ***Flex;***
- sintaktik tahlilchini tuzuvchi, erkin tarqatiluvchi dastur ***Bison*** va boshqalarni.

Xulosa

UNIX operatsionn tizimi va uning Windows shakli Linux buyruqlar satiriga moslashgan edi. Hozirgi vaqtda ham bu tizimlar aktual

hisoblanadi. Bu tizimlarda buyruqlar orqali amallar bajariladi. Bu tizimlar viruslar tamonidan judya yahshi himoyalangan. Bu tizimlarda kompilyatorlar yaratish imkoniyatini beruvchi dasturlar mavjud – bu leksik tahlilchilarni tuzish dasturi ***Lex*** *va sintaktik tahlilchini tuzish dasturi* ***Yacc.***

Bu tizimlar tizimli administratorlar uchun juda qulaydir. Ko'pincha web-serverlarda internetni boshqarish Linux tizimiga asoslangan.

Richard Stolmen mobil, ya'ni ko'chma operatsion tizim GNU (GNU's Not Unix) loihasini taklif qildi. Bu tizimning afzalligi shundaki, dastur ochiq kod va elektron xujjati bilan tarqatiladi. GNU mutlaqo bepul dasturlash tizimi bo'lib, UNIX tizimni kengaytirilgan shakli bo'ladi.

Nazorat uchun savollar

1. UNIX va LINUX tizimlar to'g'risida ma'lumot bering.
2. GNU loyiha nimani nazarda tutadi.
3. C# tili va .Net konsepsiyasi to'g'risida ma'lumot bering.

III-BO'LIM. FORMAL TILLAR VA GRAMMATIKALAR

17- BOB.FORMAL TILLAR VA GRAMMATIKALAR

Tayanch iboralar: *belgilar zanjiri, konkatenatsiya, iteratsiya amali, bo'sh zanjir, alifbo, formal til, formal grammatika, aniqlovchi, tilni sintaksisi, tilni semantikasi, leksika, leksema, terminal belgi, noterminal belgi, boshlangich belgi, metatil, rekursiv qoidalar, Xomskiy shakli, Bekus-Naur shakli, Kengaytirilgan Bekus-Naur Shakli, Virt diagrammasi.*

Barcha zamonaviy yuqori bosqich dasturlash tillarining asosida formal tillar va grammatikalar nazariyasi turadi. Kompilyatorlarni yaratishda, formal tillar va grammatikalar nazariyasi ishlatilgan. Shu sababli biz shu nazariyani asoslarini o'rganib chiqamiz va uni asosida leksik, sintaktik va semantik tahlillarning qismlarini yaratishni o'rganamiz.

Belgilar zanjiri. Zanjirlar ustidagi amallar

Ketma-ket yozilgan ihtiyoriy belgilar ketma-ketligi - belgilar zanjiri (satr, so'z, jumla, gap) deb nomlanadi.

Belgi tushinchasi tayanch bo'lib, ta'rifga muhtoj emas.

Berilgan zanjirlarni grek harflari bilan belgilaymiz.

Misol: α="*a0103s*" , β="*aaaa*", γ=".".

Agarda mos zanjirlardagi belgilar to'plami va soni teng bo'lsa, hamda bir xil ketma-ketlikda kelsa, $\alpha=\beta$ bo'ladi.

Zanjirdagi belgilar soni zanjirning uzunligini bildiradi va quyidagicha yoziladi $|\alpha|$. Yuqorida ko'rsatilgan misolda $|\alpha|=6$, agar $\alpha=\beta$ bo'lsa, unda $|\alpha|=|\beta|$ bo'ladi.

Zanjirlar ustidagi asosiy amal konkatenatsiya (birlashtirish) amalidir. α va β zanjirlarni konkatenatsiyasi $\alpha\beta$ ko'rinishda yoziladi. Natijaviy zanjirda β zanjirning belgilari αni belgilaridan keyin yoziladi, masalan $\alpha\beta$= *"a0103saaaa"*.

Bu amal kommutativ emas, ya'ni $\alpha\beta\neq\beta\alpha$ lekin assotsiativ $\alpha(\beta\gamma)=(\alpha\beta)\gamma$.

Yana ikkita amal bilan tanishtiramiz:

Zanjirni teskari yozish amali - zanjir belgilarini teskari tartibda yozishdan iborat bo'lib, α^R ko'rinishda belgilanadi:

nomlanadi. Ko'pincha semantika erkin, norasmiy, tushuntirish usullari bilan beriladi.

Leksika – bu tildagi so'zlar to'plamidan iborat. Leksema yoki so'z – bu alifbo belgilaridan tuzilgan sodda bo'linmas konstruksiya. Tabiiy tillarda bu orfografik lug'atda keltirilgan so'zlardir. Dasturlash tillarida leksema - kalit so'zlar, identifikatorlar, o'zgarmaslar, amallar va ajratuvchilardan iborat. Leksemalar ham sodda grammatika orqali berilishi mumkin.

Til grammatikasi

Grammatika – bu tildagi gaplarni tuzish usullaridir.

Grammatika – tilni ta'riflovchi matematik tizimdir.

Grammatika– tilni ikkinchi usul bilan ta'riflaydi.

Til grammatikasi qoidalar to'plami yordamida beriladi.

Formal grammatikada qoidalar tartiblangan zanjirlar juftligi bilan beriladi – (α,β). Qoidalarda tartib juda muximdir, shuning uchun ko'pincha qoida α→β yoki α::=β ko'rinishda yoziladi va bu yozuv quyidagicha o'qiladi: α zanjir β zanjirni yaratadi, yoki ta'rif bo'yicha α zanjir βdir, shunda α zanjir qoidaning chap, β zanjir o'ng tarafi hisoblanadi.

Tilning grammatikasi G lotin harfi bilan belgilanadi.

G grammatika bilan berilgan til L(G) ko'rinishda yoziladi.

Agar ikkita G va G' grammatikalar bitta tilni ta'riflasa, ular ekvivalentdir. Ikkita G va G' grammatika bilan berilgan tillar faqat bo'sh zanjir bilan farqlansa, ular qisman ekvivalent bo'ladi.

Grammatikani rasmiy ta'rifi va uni berish usullari

Grammatikani rasmiy ta'rifi to'rtta obyekt bilan aniqlanadi va G(VT,VN,P,S) kabi belgilanadi, bunda :

VT – terminal belgilar chekli to'plami;

VN – noterminal belgilar chekli to'plami shunda VT∩ VN =Ø;

V=VN∪VT belgilar to'plami G grammatikani to'liq alifboi bo'ladi;

P – α→β shaklidagi qoidalar to'plami, bunda α∈V⁺; β∈V*, ya'ni α zanjir terminal va noterminal belgilardan iboratdir (bo'sh belgisidan tashqari); β zanjir terminal va noterminal belgilardan iboratdir ham da bo'sh zanjir ham bo'lishi mumkin;

S – grammatikani boshlang'ich belgisi, S∈ VN.

Grammatikadagi to'plamlar quyidagini bildiradi:

VT – bu terminal belgilar to'plami bo'lib, shu grammatika bilan tuzilgan tilni alifbosi yoki lug'atidir. Asosan bu belgilar qoidalarni o'ng tarafidagi zanjirda uchraydi. Agar ular qoidani chap tarafida uchrasa, albatta o'ng tarafdagi zanjirda ham bo'lishi shart. Faqat terminallardan tuzilgan zanjirlarni belgilash uchun ko'pincha *x,y,z,u,v,w* lotin xarflari ishlatiladi.

VN – bu noterminal belgilar to'plami bo'lib, tilni konstruksiyalarini ta'riflash uchun ishlatiladi. Bu belgilar tilni o'ziga kirmaydi. Ular faqat qoidalarni ta'riflashda ishlatiladi. Noterminal belgilar qoidani chap hamda o'ng tarafida uchrashishi mumkin. Lekin ular kamida bir marotaba, biror qoidani chap tarafida uchrashishi kerak. Har bir qoidani chap tarafida kamida bitta noterminal belgi bo'lishi shart. Hech bir belgining ham terminal, ham noterminal bo'lishi qat'iyan man etiladi. Noterminal belgilarni boshqa belgiga almashtirish mumkin, lekin terminal belgilarni almashtirib bo'lmaydi.

Grammatikani boshlang'ich belgisi doim noterminal belgi bo'ladi.

Bunday grammatika yaratuvchi grammatika deb nomlanadi, chunki u zanjirlarni yaratish qoidalaridan iborat bo'ladi.

Qoidalarni ta'riflash tili metatil deb nomlanadi. Har bir qoida alohida satrda yoziladi. Bunday yozish usuli Xomskiy metatili deb nomlanadi. Xomskiyning metatilida noterminal belgilar indeksli A xarf bilan belgilanadi. Masalan, butun ishorali sonning grammatikasi Xomskiy metatilida quyidagicha bo'ladi:

$G(\{0,1,2,3,4,5,6,7,8,9,+,-\},\{A_1,A_2,A_3\},P,A_1)$

P:

1. $A_1 \rightarrow A_2$
2. $A_1 \rightarrow +A_2$
3. $A_1 \rightarrow -A_2$
4. $A_2 \rightarrow A_3$
5. $A_2 \rightarrow A_2A_3$
6. $A_3 \rightarrow 0$
7. $A_3 \rightarrow 1$

8. $A_3 \rightarrow 2$

9. $A_3 \rightarrow 3$

10. $A_3 \rightarrow 4$

11. $A_3 \rightarrow 5$

12. $A_3 \rightarrow 6$

13. $A_3 \rightarrow 7$

14. $A_3 \rightarrow 8$

15. $A_3 \rightarrow 9$

Bu misolda terminal belgilar to'plami VT – o'nikkita belgidan iboratdir. Bu o'nta raqam hamda ishora belgilari. Noterminal belgilar to'plami VN - uchta belgidan tashkil topgan bu A_1, A_2 va A_3. Qoidalar to'plami 15ta qoidadan iborat bo'lib, 15ta satrga yozilgan. Grammatikaning boshlang'ich belgisi A_1.

Bir nechta qoidalarning chap tarafi bir xil, ya'ni $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ... ,$\alpha \rightarrow \beta_n$ bo'lsa, ularni bitta qoida shaklida yozish mumkin $\alpha \rightarrow \beta_1|\beta_2|...|\beta_n$ va bu ***α-qoidalar*** deb nomlanadi. Shunda har bir qoida muqobil qoida bo'ladi. Bunday yozuv shakli Bekus-Naur shakli (BNSH) deb nomlanadi, bu metatilda :

- "→" yoki "::=" belgilar qoidani chap tarafini o'ng tarafidan ajratish uchun ishlatiladi;
- noterminallar ixtiyoriy satr ko'rinishda yozilib "<" va ">" burchak qavslar ichiga olinadi;
- muqobil qoidalar " | " belgisi bilan ajratiladi.

Yuqorida berilgan butun sonning grammatikasi BNShda quyidagicha ta'riflanadi:

G({0,1,2,3,4,5,6,7,8,9,+,-}, {<butun son>, <natural son>, <raqam>}, P, <butun son>)

P:

<butun son>::=<natural son> | +<natural son> | -<natural son>

<natural son>::=<raqam> | <natural son><raqam>

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

Shuni aytib o'tish kerakki, noterminal so'zlarning nomlari va qoidalari tushinarli bo'lishi uchun mavjud bo'lgan, tushunchalardan

foydalaniladi. Vaholanki ularning boshqacha ham nomlash mumkin. Natijadagi grammatika bari bir butun sonni ta'riflaydi. Lekin terminal belgilarni o'zgartirib bo'lmaydi, aks holda butunlay boshqa til hosil bo'ladi. Masalan, yuqoridagi grammatikani quyidagicha berish mumkin:

G'({0,1,2,3,4,5,6,7,8,9,+,-}, {S,T,F}, P, S)

P:

S→T | +T | -T

T→F | TF

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

Bu grammatikada noterminallar sifatida katta lotin xarflar ishlatildi, ya'ni $VN=\{S,T,F\}$. G' va G grammatikalar ekvivalentdir chunki ular bitta tilni ta'riflaydi.

Formal grammatikalar chekli qoidalar orqali, cheksiz zanjirlar to'plamini ta'riflaydi. Bunday imkoniyatni rekursiv qoidalar beradi. Rekursiyali qoidada noterminal belgi o'zini o'zi ta'riflaydi. Bu bevosita rekursiya deb nomlanadi. Agar noterminal bir nechta qoidadan so'ng o'zi orqali ta'riflansa, bu vositali rekursiya deb nomlanadi.

Bizni misolda *<natural son>::=<natural son><raqam>* bevosita rekursiv qoida. Rekursiv qoida cheksiz bo'lib qolmasligi sababli shu noterminal uchun albatta rekursiv bo'lmagan, qoida ham bulishi kerak. Bizda bu:*<natural son>::=<raqam>*.

Grammatika qoidalarini yanada tushinarli va ihcham bo'lishi uchun boshqa usullar ham mavjud. Shunday usullardan ikkitasini keltiramiz, bu – metabelgilar usuli va grafik usuli.

Metabelgilar usulida boshqacha aytganda Kengaytirilgan Bekus-Naur Shakli (KBNSH)da, qoidalarni ta'riflashda ishlatilgan belgilar, ya'ni qo'shimcha metabelgilar kiritilgan bu:

(),**[ ]**,**{ }** ,",", "" .

Ularni ma'nosi quyidagicha:

()qavs ichida turgan zanjirlar ro'yxatidan faqat bittasi, shu o'rinda turishi mumkin;

[] qavs ichida turgan zanjir bo'lmasligi ham mumkin;

{} qavs ichidagi zanjir bo'lmasligi, yoki bir marta bo'lishi, yoki bir necha marta bo'lishi mumkin;

“,” vergul belgisi ro‘yxatdagi zanjirlarni ajratish uchun ishlatiladi:

“” qo‘shtirnoq belgisi ichidagi metabelgini terminal sifatida yozish uchun ishlatiladi.

Ba’zi bir tillarni ta’riflashda boshqa metabelgilar ham kiritilishi mumkin.

KBNShda yozilgan yuqoridagi grammatikamiz quydagicha bo‘ladi:

G({0,1,2,3,4,5,6,7,8,9,+,-},{<butun son>,<raqam>},P,<butun son>)

P:

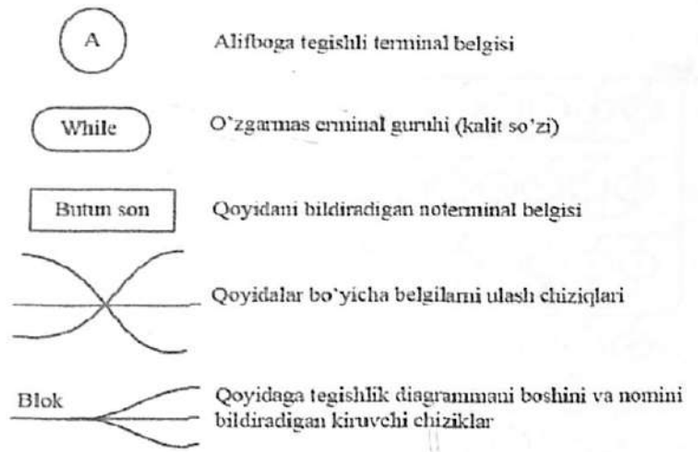
<butun son>→[(+,-)]<raqam>{<raqam>}

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

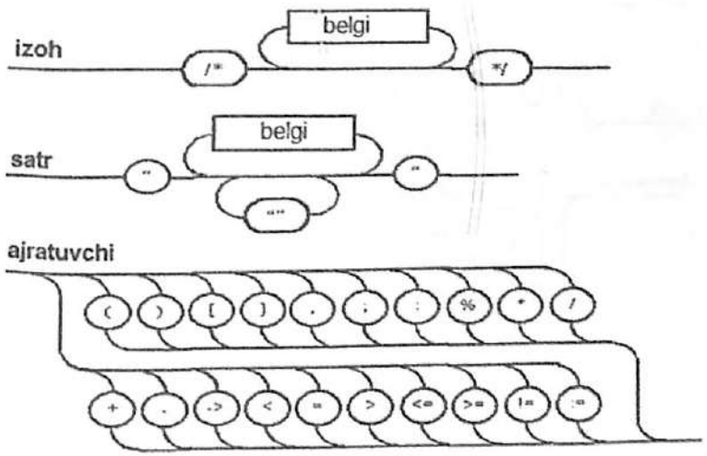
Bu grammatika qoidalaridan ko‘rinib turibdiki, birinchidan noterminal belgilar soni kamaydi, ikkinchidan rekursiv qoida o‘rniga iteratsiya (qaytarish) usuli qo‘llandi.

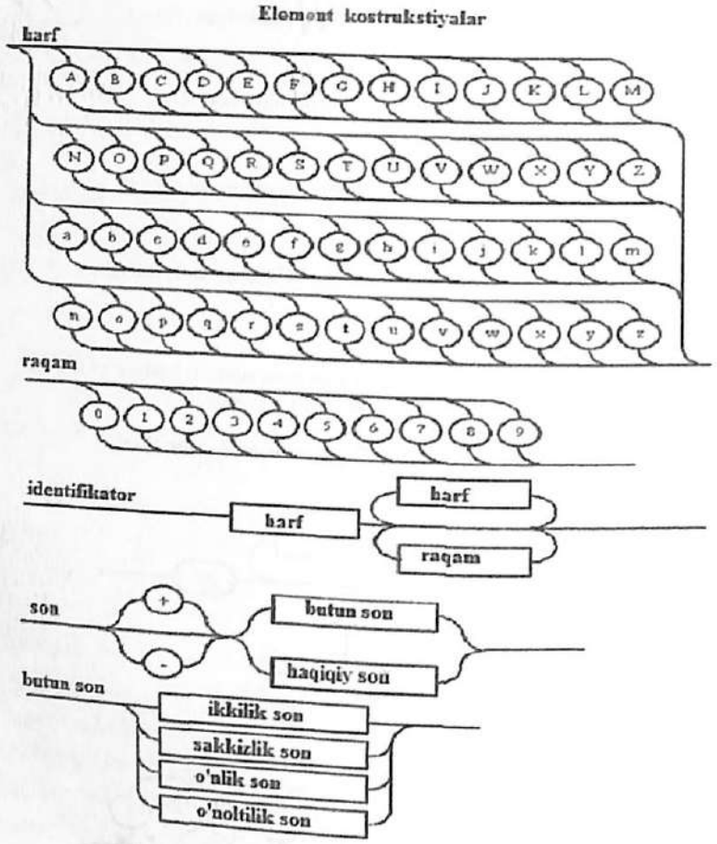
Grafik usulida har bir noterminal belgi uchun diagramma chiziladi. Bu diagramma grammatika qoidasini beradi. Bunday grafika Virt diagrammasi deb nomlanadi. Uning yozish qoidalari 17.1-rasmda keltirilgan:

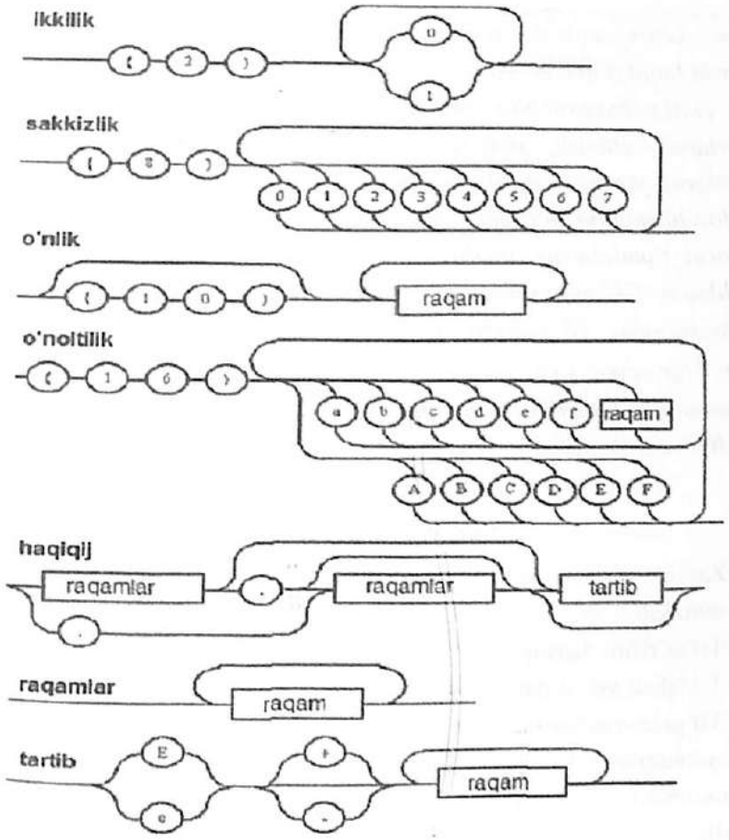
1. terminal belgilar va ulardan tuzilgan o‘zgarmas so‘zlar (leksemalar) aylanani yoki egri yon tomonli to‘g‘ri to‘rtburchak ichiga olinadi;
2. noterminal belgilar to‘g‘ri to‘rtburchak ichiga olinadi;
3. har bir grafik element bitta kiruvchi va bitta chiquvchi chiziqqa ega;
4. har bir qoida uchun alohida diagramma chiziladi va unda terminal va noterminal elementlar bir biri bilan yoylar orqali ulanadilar;
5. variantlar bir nechta yoylar bilan, iteratsiya (rekursiya, qaytarishlar) yoyni qayta ulanishi bilan beriladi;
6. qoidani boshlanishini bildiruvchi bitta noterminal bilan belgilangan kiruvchi yoy (chapda va ustida) va qoidani yakunlovchi bitta chiquvchi yoy (o‘ngda va ostida) bo‘lishi kerak ;



17.1-rasm. Virt diagramma elementlari.







17.2-rasm. Konstruksiyalar diagrammasi.

Misol taraqasida bir nechta sodda konstruqtsiyalarning diagrammalari 17.2- rasmda keltirilgan.

Xulosa

Barcha zamonaviy yuqori bosqich dasturlash tillarining asosida formal tillar va grammatikalar nazariyasi turadi. Kompilyatorlarni yaratishda, formal tillar va grammatikalar nazariyasi ishlatilgan.

Formal tillar nuktayi nazaridan til – bu belgilar to'plami asosida, ma'lum qoidalar bo'yicha belgilarning o'zaro ulanishidir. Har qanday til asosida alifbo yoki lug'at turadi. Til alifbosi– mumkin bo'lgan belgilarning chekli to'plamidir. Bu to'plam V harf bilan belgilanadi. V alifbodan iborat

α ketma-ketlik zanjir deb nomlanadi va α(V) ko'rinishda yoziladi. Bunda α zanjirda faqat alifbo belgilari mavjud bo'lishi kerak.

Tilni uchta usul bilan berish mumkin: barcha mumkin bo'lgan to'g'ri zanjirlarni keltirish; tilni grammatikasini (qoidalarini), ya'ni to'g'ri zanjirlarni yaratish usullarini berish; ixtiyoriy zanjirni tilga kirishini aniqlovchi mantiq qurilma (algoritmni) berish. Tilda mumkin bo'lgan gaplarni (jumlalarni) aniqlovchi qoidalar to'plami – til sintaksisi deb nomlanadi. Tildagi gaplarning ma'nosini aniqlovchi qism, til semantikasi deb nomlanadi. Til grammatikasi qoidalar to'plami yordamida beriladi. Formal grammatikada qoidalar α→β yoki α::=β ko'rinishda yoziladi. Grammatika qoidalarni ta'riflashda Homskiy metatilni, Bekus Naur Shaklini yoki Virt diagrammasini ishlatish mumkin.

Nazorat uchun savollar

1. Zanjir ta'rifini keltiring. Zanjir ustidan qanday amallar bajarish mumkin?
2. Til ta'rifini bering. Tilni berish usullarini keltiring. Misol keltiring.
3. Til leksikasi, sintaksisi va semantikasini izohlang.
4. Til grammatikasini ta'riflarini keltiring.
5. Grammatikani rasmiy ta'rifini keltiring.
6. Grammatikani berish usullarini keltiring va har bir usulga misol keltiring.

18-BOB.TIL VA GRAMMATIKA SINFLARI

Tayanch iboralar: *grammatika sinflari, erkin grammatika, kontekstga bog'liq grammatika, qisqarmas grammatika, kontekstdan erkin grammatika, qisqarmas kontektstan erkin grammatika, muntazam grammatika, chap chiziqli grammatika, o'ng chiziqli grammatika.*

Tilning sodda va murakkabligiga ko'ra kompilyator ham sodda yoki murakkab bo'ladi. Ba'zi bir tillar uchun umuman kompilyator yaratib bo'lmaydi. Masalan, xozirgacha tabiiy tillarda dastur yozish imkoniyati yo'q. Shu sababli, biz tillarning turi degan tushuncha kiritamiz va qaysi turdagi tillar uchun kompilyator yaratish mumkinligini aniqlaymiz.

Grammatika sinflari

Formal grammatika qoidalarini tuzilishiga qarab sinflash mumkin. Agar grammatikadagi barcha qoidalar biror kelishilgan qolipga tushsa, u holda bu grammatika qolipga mos turga kiradi. Agar grammatikani biror bir qoidasi qolipni qanoatlantirmasa, unda grammatika qolipga mos turga kirmaydi. Noam Xomskiy bo'yicha grammatikalar to'rtta turga ajratilgan:

0. Erkin grammatika. 0-chi tur grammatikasi.
1. Kontekstga bog'liq grammatika. Qisqarmas grammatika. 1-chi turdagi grammatikalar.
2. Kontekstdan erkin grammatika. 2-chi turdagi grammatikasi.
3. Muntazam grammatika. 3-turdagi grammatikasi

Erkin grammatika

Erkin grammatikada(EG) qoidalar tuzilishiga xech qanday chegaralar qo'yilmaydi: *G(VT,VN,P,S)* grammatika uchun qoidalar quyidagicha:

$\alpha \to \beta$, bunda $\alpha \in V^+$, $\beta \in V^*$.

Bu eng umumiy grammatika turi bo'lib, barcha grammatikalar bu turga kiradi. Lekin ulardan bir qismi boshqa turlarga ham kiradi.

Faqat bu turga kirgan grammatikalar eng murakkab tuzilishga ega va amalda ular ishlatilmaydi.

Bu grammatikaga asoslangan til ko'pincha tabiiy tillarga kiradi.

Bu tillarda har qanday so'zni ma'nosi nafaqat qaysi joyda turishi,

balki qaysi jumlada ishlatilishiga ham bog'liqdir.

Shu sababli bunday tillardagi jumlalarni tahlil qilish juda murakkab. Erkin grammatika 0-chi turga kiradi.

Kontekstga bog'liq va qisqarmas grammatika

Birinchi turga ikkita grammatika kiradi, kotekstga bog'liq va qisqarmas grammatikalar.

Kontekstga bog'liq grammatikada (KBG) $G(VT,VN,P,S)$ qoidalar quyidagicha yoziladi:

$\alpha_1 A\alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, bunda $\alpha_1,\alpha_2 \in V^*$, $A \in VN$, $\beta \in V^+$

Bu turdagi grammatikani tuzilishi shundayki, noterminal belgisi jumlada uchragan kontekstga qarab, har xil zanjir bilan almashtirilishi mumkin. Bunda α_1- chap qontekst va α_2- o'ng kontekst deb nomlanadi.

Qisqarmas grammatika (QG) $G(VT,VN,P,S)$ uchun qoidalar quyidagicha:

$\alpha \rightarrow \beta$, bunda $\alpha, \beta \in V^+$ va $|\beta| \geq |\alpha|$.

Qiskarmas grammatikada jumlaning uzunligi kamaymaydi. Bu ikkala grammatika o'zaro ekvivalentligi isbotlangan. Istisno tariqasida bu grammatikalarda $S \rightarrow \varepsilon$ qoida bo'lishi mumkin. Faqat bu holda S qolgan qoidalarning o'ng tarafida uchramasligi kerak. Bu hol faqat bo'sh zanjir mavjud bo'lgan, tillar grammatikasida uchraydi.

Kompilyatorlarni yaratishda bunday grammatikalar qo'llanmaydi. Chunki bunday grammatikalar noaniqlikka olib keladi. Bunday grammatikalar tarjima muxitlarida qo'llanilishi mumkin.

Kontekstdan erkin grammatika

Kontekstdan erkin grammatikada (KEG) $G(VT,VN,P,S)$ qoidalar quyidagicha:

$A \rightarrow \beta$, bunda $A \in VN, \beta \in V^*$

Shuni aytib o'tish kerakki, KEGning qoidalarni o'ng tarafi bo'sh qoidalar ham bo'lishi mumkin.

Agar grammatikani qoidalari quyidagicha bo'lsa:

$A \rightarrow \beta$, bunda $A \in VN, \beta \in V^+$,

Bunday grammatika qisqarmas kontekstdan erkin grammatika

(QKEG) deb nomlanadi. Chunki bunday grammatikada qoidani o'ng tarafida kamida bitta belgi turishi kerak va bo'sh zanjir bo'lishi man etiladi. Istisno tariqasida QKEGda $S \rightarrow \varepsilon$ qoida bo'lishi mumkin. Faqat bu holda S qolgan qoidalarni o'ng tarafida uchramasligi kerak. Har qanday KEGni QKEGga grammatikaga keltirish mumkinligi isbotlangan.

KEG va QKEG 2-tur grammatikalari hisoblanadi.

KEG dasturlash tillarini sintaktik konstruksiyalarini ta'riflashda keng ishlatiladi. Shu sababli biz shu turdagi grammatikani keyingi boblarda batafsil ko'rib chiqamiz.

Muntazam grammatika

Muntazam grammatikalar uchunchi turga kiradi. Unga ikkita ekvivalent grammatika kiradi:

Chap chiziqli muntazam grammatika (CHCHMG) $G(VT,VN,P,S)$ quyidagi ko'rinishdagi qoidalar bilan ifodalanadi:

$A \rightarrow Bx$ yoki $A \rightarrow x$, bunda $A,B \in VN$, $x \in VT^*$.

O'ng chiziqli muntazam grammatika (O'CHMG) $G(VT,VN,P,S)$ quyidagi qoidalar bilan ifodalanadi:

$A \rightarrow xB$ yoki $A \rightarrow x$, bunda $A,B \in VN$, $x \in VT^*$.

Muntazam grammatikalar dasturlash tillarni eng sodda konstruksiyalarni, ta'riflashda keng ishlatiladi. Bu asosan leksik konstruksiyalar ya'ni leksemalar. Shu jumladan identifikatorlar o'zgarmaslar, izohlar va xokazo. Shu sababli bu grammatika kompilyatorni leksik tahlil fazasida ishlatiladi.

Tillarni sinflari

Tillar - o'z grammatikasiga ko'ra sinflarga ajratiladi. Ma'lumki, bitta til bir nechta grammatika bilan berilishi mumkin. Shunda tilni sinfi, eng yuqori raqamli turdagi grammatika turi bilan aniqlanadi. Masalan, $L(\{0,1\})=\{0^n1^n, n>0\}$ tilni ikkita grammatika orqali berish mumkin:

0-tur:	*2-tur:*
$G_1(\{0,1\},\{A,S\},P1,S)$	$G_2(\{0,1\},\{S\},P2,S)$
P1:	*P2:*
$S \rightarrow 0A1$	$S \rightarrow 0S1$

$0A \to 00A1$ $\quad S \to 01$

$A \to \varepsilon$

Bu ikkala grammatika bitta tilni ta'riflaydi. Birinchisi erkin grammatika bo'lsa, ikkinchisi kontekstdan erkin grammatikadir. Shuning uchun bu til kontekstdan erkin tilga kiradi.

Grammatika turlariga misollar:

0-tur: $L(G)=\{a^2b^{n*n-1}, n>=1\}$, $n=2$: *aabbb*, $n=3$: a^2b^8

$G(\{a,b\},\{A,B,C,D,F,S\},P,S)$

P:

1. $S \to aaCFD$
2. $F \to AFB$
3. $F \to AB$
4. $AB \to bBA$
5. $Ab \to bA$
6. $AD \to D$
7. $Cb \to bC$
8. $CB \to C$
9. $bCD \to \varepsilon$

1-tur: $L(G)=\{a^nb^nc^n, n>=1\} n=2$: *aabbcc*,

$G(\{a,b,c\},\{A,B,C,S\},P,S)$

P:

1. $S \to aSBC$ *yoki* 1. $S \to aSBc$
2. $S \to abC$ 2. $S \to abc$
3. $CB \to BC$ 3. $cB \to Bc$
4. $bB \to bb$ 4. $bB \to bb$
5. $bC \to bc$
6. $cC \to cc$

2-tur: $L(G)=\{(ac)^n(cb)^n, n>0\}$

$G(\{a,b,c\},\{A,S\},P,S)$

P:

1. $S \to aAb$ *yoki* 1. $S \to acScb$
2. $S \to accb$ 2. $S \to accb$
3. $A \to cSc$

3-tur: $L(G)=\{w\perp \mid w \in \{a,b\}^*$, *bunda ikkita yonma yon turgan "a" xarfi yo'q*$\}$

$G(\{a,b,\perp\},\{A,B,S\},P,S)$

P:

1. $S \to A\perp \mid B\perp$
2. $A \to a \mid Ba$
3. $B \to b \mid Bb \mid Ab$

Xulosa

Formal grammatika qoidalarini tuzilishiga qarab sinflanadi. Noam Xomskiy bo'yicha grammatikalar to'rtta turga ajratilgan: erkin grammatika. 0-chi tur grammatikasi; kontekstga bog'liq grammatika, qisqarmas grammatika. 1-chi turdagi grammatikalar; kontekstdan erkin grammatika. 2-chi turdagi grammatikasi; muntazam grammatika. 3-turdagi grammatikasi.

Erkin grammatika qoydalariga xech qanday cheklovlar qo'yilmaydi va uning qoydalari $\alpha \to \beta$, bunda $\alpha \in V^+$, $\beta \in V^*$ *ko'rinishda bo'ladi. Bu eng umumiy grammatika bo'lib 0-chi turga kiradi.*

Kontekstga bog'liq grammatikada qoydalar $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, bunda $\alpha_1, \alpha_2 \in V^*$, $A \in VN$, $\beta \in V^+$ *ko'rinishda bo'ladi.*

Qisqarmas grammatikada qoidalar $\alpha \to \beta$, bunda α, $\beta \in V^+$ *bo'lib* $|\beta| \geq |\alpha|$ *shart bajarilishi kerak. Kontekstga bog'liq va qisqarmas grammatika ekvivalent bo'lib grammatikani 1-chi turiga kiradi.*

Kontekstdan erkin grammatikada (KEG) G(VT,VN,P,S) qoidalar quyidagicha : $A \to \beta$, *bunda* $A \in VN$, $\beta \in V^*$. *Bu grammatika 2-chi turga kiradi.*

Nixoyat 3-chi turga muntazam grammatika kiradi, u ikkita ekvivalent grammatikadan iborat.

Chap chiziqli muntazam grammatikada qoidalar quidagicha: $A \to Bx$ yoki $A \to x$, bunda $A, B \in VN$, $x \in VT^*$.

O'ng chiziqli muntazam grammatikaning qoidalari $A \to xB$ yoki $A \to x$, bunda $A,B \in VN$, $x \in VT^*$ *ko'rinishda bo'ladi.*

Tilning sinfi grammatika sinfiga bog'liq.

Nazorat uchun savollar

1. Grammatikani sinflashi kim tomondan kiritilgan?
2. Sinflash bo'yicha nechta tur bor?
3. Erkin grammatikani ta'rifini keltiring.
4. Kontekstga bog'liq grammatikani ta'rifini keltiring.
5. Kontektstdan erkin grammatikani ta'rifini keltiring.
6. Muntazam grammatika shakli qanday bo'ladi?
7. Til sinfi qanday aniqlanadi?

Amaliyot uchun topshiriqlar

1. Berilgan grammatika qaysi turga kiradi va qanday tilni yaratadi?

a) $S \to APA$ b) $S \to aQb \mid \varepsilon$ c) $S \to 1B$ d) $S \to A \mid SA \mid SB$

$P \to + \mid -$ $Q \to cSc$ $B \to B0 \mid 1$ $A \to c$

$A \to a \mid b.$ $B \to b$

2.Quyidagi qoidalar bilan berilgan grammatika qaysi turga kiradi:

a) $S \to a \mid Ba$
$B \to Bb \mid b$

b) $S \to Ab$
$A \to Aa \mid ba$

d) $S \to 0A1 \mid 01$
$0A \to 00A1$
$A \to 01$

d) $S \to AB$
$AB \to BA$
$A \to a$
$B \to b$

19-BOB. ZANJIRLAR USTIDA ISHLOVLAR

Tayanch iboralar: *zanjirni taxlil qilish, zanjirni keltirib chiqarish, bevosita keltirib chiqarish, sentensial shakl, sentensiya, chap tomonli keltirib chiqarish, o'ng tomonli keltirib chiqarish, keltirib chiqarish daraxti, yuqoridan pastga qarab daraxtni qurish, pastdan yuqoriga qarab daraxtni qurish, bir ma'nolik grammatika, ko'p ma'nolik grammatika.*

Zanjirlarni qayta keltirib chiqarish(tahlil qilish)

Tilning berilgan grammatikasi bo'yicha zanjir boshlang'ich belgi qoidasidan boshlab, to qolgan qoidalarga binoan keltirib chiqarilsa, bunday zanjir tilga kiradi.

Zanjirni keltirib chiqarish jarayoni tahlil qilish deb nomlanadi.

Bizni asosan KEG asosida qilingan tahlillar qiziqtiradi. Chunki uning kuvvati dasturlash tillarining sintaksisini ta'riflash uchun yetarlidir.

Agarda berilgan $G(VT,VN,P,S)$, $V=VN \cup VT$ ***grammatikada*** $\omega \to \gamma$ ***qoida mavjud bo'lsa,*** $\alpha=\delta_1\omega\delta_2$ ***zanjirdan bevosita*** $\beta=\delta_1\gamma\delta_2$ ***zanjir keltirib chiqariladi, bunda*** $\omega \in V^+$, $\delta_1,\gamma,\delta_2 \in V^*$, ***ya'ni*** δ_1 ,γ ,δ_2 ***zanjirlar bo'sh bo'lishi mumkin.*** Ya'ni berilgan zanjirda ko'rsatilgan qoidaga binoan ω zanjir qismi o'rniga γ zanjir qo'yiladi. ***Bevosita keltirib chiqarish*** $\alpha \Rightarrow \beta$ ***ko'rinishda yoziladi.***

Agarda berilgan $G(VT,VN,P,S)$, $V=VN \cup VT$ ***grammatikada shunday*** $\gamma_0, \gamma_1, \ldots, \gamma_n$ ***zanjirlar borki*** $\alpha \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_n \Rightarrow \beta$ ***keltirib chiqarilsa, ya'ni*** γ_i ***zanjiri bevosita*** γ_{i-1} ***zanjirdan keltirib chiqarilsa,*** β ***zanjir*** α ***zanjirdan keltirib chiqariladi va bu keltirib chiqarish*** $\alpha \Rightarrow^* \beta$ ***ko'rinishda yoziladi. Agar keltirib chiqarish kamida bitta qadamdan iborat bo'lsa, uni*** $\alpha \Rightarrow^+ \beta$ ***ko'rinishda yoziladi.***

Bunday ketma-ketlik, zanjirni keltirib chiqarish deb nomlanadi. Har bir bevosita chiqarish, keltirib chiqarish qadami deb nomlanadi. Agar chiqarish qadamlari ma'lum bo'lsa, uni * o'rniga yozish mumkin. Masalan qadamlar soni 4-ga teng bo'lsa $\alpha \Rightarrow^4 \beta$ ko'rinishda yoziladi.

Berilgan $G(VT,VN,P,S)$, $V=VN \cup VT$ ***grammatikada*** $\alpha \in V^*$ ***zanjir S boshlang'ich belgidan keltirib chiqarilsa, ya'ni*** $S \Rightarrow^* \alpha$ ***bo'lsa, bunday zanjir G grammatikada sentensial shakli deb nomlanadi. Agar sentensial shaklda faqat terminal belgilar bo'lsa, bunday zanjir chekli***

sentensial shakil yoki sentensiya deb nomlanadi.

Agar joriy sentensial shakl oldingi sentensial shakldan eng chap noterminal belgi qoidasini o'ng tarafiga almashtirish yo'li bilan chiqarilsa, bunday keltirib chiqarish chap tomonli keltirib chiqarish deb nomlanadi. Aksincha eng o'ng noterminal belgi almashtirilsa, bunday keltirib chiqarish o'ng tomonli keltirib chiqarish deb nomlanadi.

Masalan, $G(\{a,b,+\},\{T,S\}, \{S\to T \mid T+S; T\to a \mid b\},S)$ grammatikada $a+b+a$ zanjirni keltirib chiqaramiz:

1) $S\Rightarrow T+S\Rightarrow a+S\Rightarrow a+T+S\Rightarrow a+b+S\Rightarrow a+b+T\Rightarrow a+b+a$;

2) $S\Rightarrow T+S\Rightarrow T+T+S\Rightarrow T+T+T\Rightarrow T+T+a\Rightarrow T+b+a\Rightarrow a+b+a$;

3) $S\Rightarrow T+S\Rightarrow T+T+S\Rightarrow T+T+T\Rightarrow a+T+T\Rightarrow a+b+T\Rightarrow a+b+a$.

Bu misolda 1) chap tomonli keltirib chiqarish; 2) o'ng tomonli keltirib chiqarish; 3) chap tomonli ham emas, o'ng tomonli ham bo'lmagan keltirib chiqarish.

Grammatikada bitta zanjir uchun bir nechta keltirib chiqarishlar bo'lishi mumkin. Ularning hammasi ekvivalentdir. Kontekstdan erkin va muntazam grammatikalar uchun har doim chap tomonli, yoki o'ng tomonli keltirib chiqarishni tuzish mumkin. Lekin boshqa turdagi grammatikalarda bunday emas.

Keltirib chiqarishni daraxt ko'rinishda ham berish mumkin.

Daraxt ko'rinishda keltirib chiqarishda:

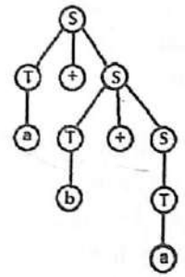
1) har bir cho'qqida ($V_N \cup V_T \cup \varepsilon$) to'plam belgisi turishi kerak;

2) agar daraxt tugunida A noterminal belgi turgan bo'lsa va uni avlodlari $X_1, X_2, \ldots, X_n$, $X_i \in (V_N \cup V_T \cup \varepsilon)$ belgilardan iborat bo'lsa, u holda $A\to X_1X_2\ldots X_n$ qoida grammatikada mavjud bo'ladi;

3) daraxtni ildizida boshlang'ich noterminal belgi turadi;

4) daraxtni barglarida terminal yoki bo'sh zanjir belgisi turadi.

Keltirib chiqarish daraxtni pastga, ildizdan barglarga qarab, yoki yuqoriga, ya'ni barglardan ildizga qarab tuzish mumkin. Undan tashqari ikkala usulda o'ngdan chapga, yoki chapdan o'nga qarab yurish mumkin. Misol sifatida yuqorida ko'rsatilgan grammatika bo'yicha $a+b+a$ zanjirni keltirib chiqarish daraxti 19.1-rasmda ko'rsatilgan.



19.1-rasm. a+b+a zanjirni keltirib chiqarish daraxti

Agar biror bir zanjir uchun ikkita yoki undan ortiq turli shaklda daraxtlar qurish mumkin bo'lsa, bunday grammatika ko'p ma'nolik, aks holda bir ma'nolik grammatika bo'ladi.

Masalan, arifmetik ifodalar grammatikasi berilsin:

$G(\{a,b,+,-,/,*,(,)\},\{S\},P,S)$. P: $S\to S+S \mid S-S \mid S*S \mid S/S \mid (S) \mid a \mid b$

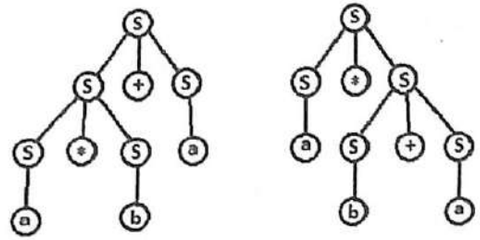
Bu tilda $a*b+a$, $b+(a-b)$ kabi zanjirlar yozish mumkin.

$a*b+a$ zanjir uchun keltirib chiqarishni ishlatamiz, bunday keltirib chiqarish ikkita bo'ladi:

1) $S\Rightarrow S+S\Rightarrow S*S+S\Rightarrow a*S+S\Rightarrow a*b+S\Rightarrow a*b+S\Rightarrow a*b+a$;

2) $S\Rightarrow S*S\Rightarrow a*S\Rightarrow a*S+S\Rightarrow a*b+S\Rightarrow a*b+a$.

Shu keltirib chiqarishlar uchun tuzilgan daraxtlar ham ikki xil bo'ladi. (19.2-rasm)



19.2-rasm. a*b+a zanjirni ikki hil keltirib chiqarish daraxti

Formal tillar nuqtayi nazaridan zanjirni qaysi yo'l bilan keltirib chiqganligining axamiyati yo'q, lekin dasturlash tillarida qaysi ketma-ketligda chiqarishi muhimdir. Ya'ni "oldin qo'shish amalinimi yoki ko'paytirish amalini chiqarish kerakmi?", degan savolga javob berish kerak. Ko'p ma'nolik bu grammatikani xossasi, tilga bog'liq emas. Ya'ni ba'zi bir tillar uchun ekvivalent bir ma'nolik grammatika tuzish mumkin.

Umumiy holda ikkita muammoni yechish kerak bo'ladi:

Birinchidan ikki grammatika ekvivalentligini aniqlash ya'ni bitta tilni ta'riflashi, ikkinchidan yangi tuzilgan grammatika bir ma'nolik grammatika bo'lishini.

Ekvivalentlik muammosi umumiy holda quyidagicha bo'ladi: bizga ikkita ixtiyoriy grammatika berilgan, ularni ekvivalentligini aniqlaydigan algoritm tuzish kerak. Afsuski, bu muammoni aloritmik yechimi yo'q va umumiy holda bunday algoritmni tuzib bo'lmasligi isbotlangan.

Huddi shunday umumiy holda har qanday grammatikani bir ma'noligini aniqlab beradigan, algoritm yo'q va tuzib bo'lmaydi. Undan tashqari umumiy holda, ko'p ma'nolik grammatikani bir ma'nolik grammatikaga qayta o'zgartirish algoritmi ham yo'q.

Bu muammolar umumiy holda yechilmasa ham, muayyan hollarda yechimini topish mumkin.

Masalan, yuqorida ko'rsatilgan ko'p ma'nolik grammatika uchun ekvivalent bir ma'nolik grammatika quyidagicha bo'ladi:

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

P:

$S \to S+T \mid S-T \mid T$

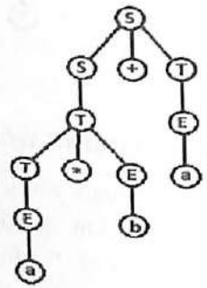
$T \to T*E \mid T/E \mid E$

$E \to (S) \mid a \mid b.$

Bu grammatikaga asosan $a*b+a$ zanjir uchun chap tomonli keltirib chiqarish quyidagicha:

$S \Rightarrow S+T \Rightarrow T+T \Rightarrow T*E+T \Rightarrow E*E+T \Rightarrow a*E+T \Rightarrow a*b+T \Rightarrow a*b+a$

Bu keltirib chiqarishga yagona daraxt mos keladi (19.3-rasm).



19.3-rasm a*b+a zanjirni yagona keltirib chiqarish daraxti.

Agar grammatikamiz bir ma'noli bo'lsa, har qanday zanjirni keltirib chiqarish daraxt shakli barcha usullar uchun bir xil bo'ladi.

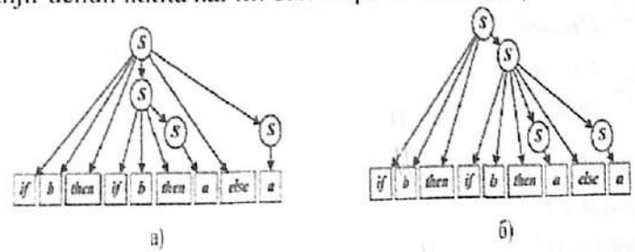
Yana bir misol,

$G(\{$***if, then, else,*** $a, b\}, \{S\}, P, S)$va

P:

$S \to$ ***if*** b ***then*** S ***else*** $S \mid$ ***if*** b ***then*** $S \mid a$

bo'lgan grammatika ko'p ma'nolikdir. Chunki ***if*** b ***then if*** b ***then*** a ***else*** a zanjir uchun ikkita har xil daraxt qurish mumkin (19.4-rasm).



а) б)

19.4-rasm. if b then if b then a else a zanjirni ikki xil daraxti.

Bizni misolda ***else*** *a zanjir qismi*, birinchi ***then***ga ham, ikkinchi ***then***ga tegishli bo'lishi mumkin. Agar ***else*** eng yaqin ***then***ga tegishli deb kelishsak, unda bir ma'nolik grammatikani tuzish mumkin:

$S \to$ ***if*** b ***then*** $S \mid$ ***if*** b ***then*** S' ***else*** $S \mid a$

$S' \to$ ***if*** b ***then*** S' ***else*** $S' \mid a$

Ma'lumki, grammatikada quyidagi shakldagi qoidalar biri uchrasa:

1. $A \to AA \mid \alpha$
2. $A \to A\alpha A \mid \beta$
3. $A \to \alpha A \mid A\beta \mid \gamma$
4. $A \to \alpha A \mid \alpha A\beta A \mid \gamma$

hamda $A \in VN$; $\alpha,\beta,\gamma \in (VN \cup VT)^*$ bo'lsa, bunday grammatika ko'p ma'noli bo'ladi.

Misol taraqasida ba'zi bir tillar uchun keltirib chiqarishlarni ko'rsatamiz:

0-tur:

$L(G)=\{a^2b^{n*n-1}, n>=1\}$,

$n=1: a^2$; $n=2: a^2b^3$, $n=3: a^2b^8$

$G(\{a,b\},\{A,B,C,D,F,S\},P,S)$

P:

1. $S \to aaCFD$
2. $F \to AFB$
3. $F \to AB$
4. $AB \to bBA$
5. $Ab \to bA$
6. $AD \to D$
7. $Cb \to bC$
8. $CB \to C$
9. $bCD \to \varepsilon$

$n=1$: a^2

$S \Rightarrow_1 aaCFD \Rightarrow_3 aaCABD \Rightarrow_4 aaCbBAD \Rightarrow_7 aabCBAD$
$\Rightarrow_8 aabCAD \Rightarrow_6 aabCD \Rightarrow_9 aa$

$n=2$: a^2b^3

$S \Rightarrow_1 aaCFD \Rightarrow_2 aaCAFBD \Rightarrow_3 aaCAABBD \Rightarrow_4 aaCAbBABD$
$\Rightarrow_5 aaCbABABD \Rightarrow_7 aabCABABD \Rightarrow_4 aabCbBAABD$
$\Rightarrow_7 aabbCBAABD \Rightarrow_8 aabbCAABD \Rightarrow_4 aabbCAbBAD$
$\Rightarrow_5 aabbCbABAD \Rightarrow_7 aabbbCABAD \Rightarrow_4 aabbbCbBAAD$
$\Rightarrow_5 aabbbbCBAAD \Rightarrow_8 aabbbbCAAD \Rightarrow_6 aabbbbCAD$
$\Rightarrow_6 aabbbbCD \Rightarrow_9 aabbb$
$S \Rightarrow^{18} a^2b^3$

1-tur: $L(G)=\{a^n b^n c^n, n>=1\}$

$G(\{a,b,c\},\{A,B,C,S\},P,S)$

P:

1. $S \to aSBC$
2. $S \to abC$
3. $CB \to BC$
4. $bB \to bb$
5. $Ab \to bA$
6. $bC \to bc$
7. $cC \to cc$

$n=1$: abc

$S \Rightarrow_2 abC \Rightarrow_5 abc$.

$n=2$: $a^2b^2c^2=aabbcc$

$S \Rightarrow_1 aSBC \Rightarrow_2 aabCBC \Rightarrow_3 aabBCC \Rightarrow_4 aabbCC \Rightarrow_5 aabbcC \Rightarrow_6 aabbcc$

2-tur:

$L(G)=\{(ac)^n(cb)^n, n>0\}$

$G(\{a,b,c\},\{A,S\},P,S)$

P:

1. $S \to aAb$
2. $S \to accb$
3. $A \to cSc$

$n=1$: $accb$ $S \Rightarrow_2 accb$

$n=2$: $acaccbcb$

$S \Rightarrow_1 aAb \Rightarrow_3 acScb \Rightarrow_2 acaccbcb$

3-tur:

$L(G)=\{w\perp \mid w \in \{a,b\}^*\}$, bunda ikkita yonma-yon turgan a xarfi yo'q)

$G(\{a,b,\perp\},\{A,B,S\},P,S)$

P:

1. $S \to A\perp$
2. $S \to B\perp$
3. $A \to a$
4. $A \to Ba$
5. $B \to b$
6. $B \to Bb$
7. $B \to Ab$

Shu tildagi $abbab\perp$ zanjirni keltirib chiqaramiz:

$S \Rightarrow_2 B\perp \Rightarrow_7 Ab\perp \Rightarrow_4 Bab\perp \Rightarrow_6 Bbab\perp \Rightarrow_7 Abbab\perp \Rightarrow_3 abbab\perp$, ya'ni $S \Rightarrow^6 abbab\perp$

Xulosa

Tilning berilgan grammatikasi bo'yicha zanjir boshlang'ich belgi qoidasidan boshlab, to qolgan qoidalarga binoan keltirib chiqarilsa, bunday zanjir tilga kiradi deb hisoblanadi. Berilgan G(VT,VN,P,S) , V=VN∪VT grammatikada $\alpha \in V^*$ zanjir S boshlang'ich belgidan keltirib chiqarilsa, ya'ni $S \Rightarrow^* \alpha$ bo'lsa, bunday zanjir G grammatikada sentensial shakli deb nomlanadi. Agar sentensial shaklda faqat terminal belgilar bo'lsa, bunday zanjir chekli sentensial shakil yoki sentensiya deb nomlanadi. Agar joriy sentensial shakl oldingi sentensial shakldan eng chap noterminal belgi qoidasini o'ng tarafiga almashtirish yo'li bilan chiqarilsa, bunday keltirib chiqarish chap tomonli keltirib chiqarish deb nomlanadi. Aksincha eng o'ng noterminal belgi almashtirilsa, bunday keltirib chiqarish o'ng tomonli keltirib chiqarish deb nomlanadi.

Nazorat uchun savollar

1. Zanjirni tilga kirish ta'rifini bering.
2. Bevosita keltirib chiqarishni ta'rifini keltiring.
3. Zanjirni keltirib chiqarish ta'rifini bering.
4. Sentensial shakli bu nima?
5. Sentensiya nimadan iborat?
6. Chaptomonli keltirib chiqarish ta'rifini bering.
7. O'ngtomonli keltirib chiqarish ta'rifini bering.
8. Keltirib chiqarishni daraxt ko'rinishini bering.
9. Bir ma'nolik va ko'p ma'nolik grammatikani farqi nimada?
10. Grammatikani qoidalari qaysi shaklda bo'lganda, ko'p ma'nolik grammatika bo'ladi.

Amaliyot uchun topshiriqlar

1. Quyidagi qoidalar bilan berilgan grammatikada:

$S \to T \mid T+S \mid T-S$

$T \to F \mid F*T$

$F \to a \mid b$

$a-b*a+b$ zanjirni keltirib chiqaring.

2. Berilgan qoidalar bo'yicha barcha sentensial shakllarni tuzing:

$S \to A+B \mid B+A$

$A \to a$

$B \to b$

3. Quyidagi tillarni grammatikasini tuzing:

1) $L = \{a^n b^m \mid n, m \geq 1\}$
2) $L = \{a^n b^m \mid n \neq m ; n, m \geq 0\}$
3) $L=\{a^n b^m c^k \mid n, m, k>0\}$
4) $L=\{(ab)^n (cb)^m \mid n, m \geq 0\}$
5) $L=\{0^n (10)^m \mid n, m \geq 0\}$
6) $L=\{(10)^{n-1} (01)^{n-1} \mid n>0\}$
7) $L=\{c^{2n} d^n \mid n>0\}$
8) $L = \{\alpha c \beta c \gamma c \mid \alpha, \beta, \gamma \in \{a,b\}^*\}$
9) $L = \{wcwcwc \mid w \in \{a,b\}^*\}$
10) $L=\{l+l-l \mid l \in \{a, b\}^*\}$
11) $L = \{a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \mid a_i \in \{0,1\}, n \geq 1\}$
12) $L = \{w \mid w \in \{0,1\}^*$, *bunda 0 va 1 soni bir xil*$\}$
13) $L = \{xx \mid x \in \{a,b\}^*\}$
14) $L = \{w \mid w \in \{0,1\}^*$ *bunda 0 va 1 soni teng emas*$\}$
15) $L = \{(a^{2m} b^m)^n \mid m \geq 1, n \geq 0\}$
16) $L = \{a^{n} \perp \mid n \geq 1\}$
17) $L = \{a^{n} \mid n \geq 1\}$
18) $L = \{a^{n} \mid n \geq 1\}$
19) $L=\{(ac)^n \mid n>0, a \in \{b, d\}, c \in \{+, -\}\}$
20) $L=\{\perp (010)^n \perp \mid n>0\}$
21) $L=\{ib.b \mid i \in \{+, -\}, b \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*\}$

4. Berilgan grammatikalar ekvivalentmi? :

a) $S \to AB$	va	$S \to AS \mid SB \mid AB$
$A \to a \mid Aa$		$A \to a$
$B \to b \mid Bb$		$B \to b$

b) $S \to aSL \mid aL$ va $S \to aSBc \mid abc$

$L \to Kc$

$cK \to Kc$

$cB \to Bc$

$K \to b$

$bB \to bb$

5. Berilgan grammatikalarga ekvivivalent KE-grammatika tuzing:

a) $S \to aAb$

$aA \to aaAb$

$A \to \varepsilon$

b) $S \to AB \mid ABS$

$AB \to BA$

$BA \to AB$

$A \to a$

$B \to b$

6. Berilgan grammatikalarga ekvivivalent muntazam grammatika tuzing:

a) $S \to A \mid AS$

$A \to a \mid bb$

b) $S \to A.A$

$A \to B \mid BA$

$B \to 0 \mid 1$

7.Quyidagi qoidalar bilan grammatika berilgan:

a) $S \to S0 \mid S1 \mid D0 \mid D1$

$D \to H.$

$H \to 0 \mid 1 \mid H0 \mid H1$

b) $S \to$ ***if*** B ***then*** $S \mid B = E$

$E \to B \mid B + E$

$B \to a \mid b$

quyidagi zanjirlar uchun:

a) *10.1001*

b) ***if*** *a* ***then*** $b = a+b+b$

yuqoridan pastga tushish va pastdan yuqoriga chiqish usullari orqali daraxt tuzing.

8. Quyidagi grammatikani turini aniqlang. Shu grammatika bo'yicha yaratilgan tilni ta'riflang. Shu til uchun KEGni keltiring:

$S \to P\perp$

$P \to 1P1 \mid 0P0 \mid T$

$T \to 021 \mid 120R$

$R1 \to 0R$

$R0 \to 1$

$R\perp \to 1\perp$

9. $\{a, b\}$ alifboda tuzilgan a xarfi ketma-ket ikki marta uchramaydigan zanjirlar uchun muntazam grammatika tuzing.

10. Quyidagi L = $\{a^{2n} b^m c^{2k} \mid m=n+k, m>1\}$ til uchun KEG yozing va *aabbbccccc* zanjirni chap tomonli keltirib chiqarish usul bilan keltirib chiqaring xamda daraxt quring.

11. Quyidagi L = $\{a^n cb^m ca^n \mid n, m>0\}$ til uchun KE-grammatika yozing va aacbbbcaa zanjirni keltirib chiqaring.

12. Quyidagi L = $\{1^n 0^m 1^p \mid n+p>m;\ n, p, m>0\}$ til uchun KE-grammatika yozing va *110000111* zanjirni keltirib chiqaring.

13. Berilgan G grammatikani turini aniqlang,va tilni ta'riflang.

P:$S \to 0A1$

$0A \to 00A1$

$A \to \varepsilon$

14. Berilgan L = $\{1^{3n-2} 0^n \mid n \geq 0\}$ tilni turini aniqlang va grammatikasini tuzing . *1111111100* zanjirni chap va o'ng tomonli usullar bilan keltirib chiqaring.

20- BOB. MUNTAZAM TILLAR VA GRAMMATIKALAR

Tayanch iboralar: *chap chiziqli avtomat grammatika, o'ng chiziqli avtomat grammatika, muntazam ifodalar.*

Chap chiziqli va o'ng chiziqli grammatikalar. Avtomat grammatikalar

Muntazam grammatikalarga ikkita grammatika kiradi:

$G(VT,VN,P,S)$, $V=VN \cup VT$ chap chiziqli grammatikada ikki xil qoidalar bo'lishi mumkin:

$A \to Bx$ yoki $A \to x$, bunda $A,B \in VN$, $x \in VT^*$ terminal belgilardan tuzilgan zanjir, bo'sh ham bo'lishi mumkin.

O'ng chiziqli grammatikada qoidalar quyidagicha:

$A \to xB$ yoki $A \to x$, bunda $A,B \in VN$, $x \in VT^*$.

Bunday grammatikalar bilan berilgan tillar muntazam tillar deb nomlanadi.

Har qanday chap chiziqli grammatika uchun o'ng chiziqli ekvivalent grammatika tuzish mumkinligi isbotlangan.

Barcha muntazam grammatikalar ichidan alohida avtomat grammatikasini ajratamiz, bu grammatikada qoidalar quyidagicha :

Chap chiziqli avtomat grammatika qoidalari quyidagicha: $A \to Bt$ yoki $A \to t$, bunda $A,B \in VN$, $t \in VT$ va t yagona terminal belgisi.

O'ng chiziqli avtomat grammatika qoidalari quyidagicha: $A \to tB$ yoki $A \to t$, bunda $A,B \in VN$, $t \in VT$ va t yagona terminal belgisi.

Har qanday avtomat grammatika muntazam grammatikadir, lekin teskarisi noto'g'ri. Muntazam va avtomat grammatikalar qisman ekvivalentdir. Ular to'liq ekvivalent bo'lish uchun avtomat grammatikaga $S \to \varepsilon$ qoidasi qo'shiladi va S boshlang'ich noterminal belgisi boshqa qoidalarni o'ng tarafida uchramasligi kerak. Lekin xaqiqiy tillarda bo'sh belgisi uchramaganligi sababli, biz $S \to \varepsilon$ qoidani ishlatmaymiz.

Chap chiziqli muntazam grammatikani chap chiziqli avtomat grammatikaga keltirish

$G(VT,VN,P,S)$ chap chiziqli muntazam grammatika berilgan, $G'(VT',VN',P',S')$ chap chiziqli avtomat grammatikani tuzish kerak.

1-qadam. Barcha noterminal belgilar VN to'plamdan G' grammatikani VN' to'plamiga o'tkaziladi. VT'=VT;

2-qadam. G grammatikani P qoidalar to'plami ko'rib chiqiladi.

Agar unda $A \to Bt$ yoki $A \to t$, $A,B \in VN$, $t \in VT$ qoidalar uchrasa ularni o'zgartirmasdan G' grammatikani qoidalar to'plamiga o'tkaziladi.

Agar G grammatikada $A \to Bt_1t_2 \ldots t_n$, $n>1$, $A,B \in VN$, $t_i \in VT$ uchrasa, VN' to'plamiga $A_1,A_2,\ldots,A_{n-1}$ noterminal belgilar qo'shiladi va P' to'plamiga quyidagi qoidalar kiritiladi:

$A \to A_{n-1}t_n$

$A_{n-1} \to A_{n-2}t_{n-1}$

...

$A_2 \to A_1t_2$

$A_1 \to Bt_1$

Agar G grammatikada $A \to t_1t_2 \ldots t_n$, $n>1$, $A \in VN$, $t_i \in VT$ uchrasa, VN' to'plamiga $A_1,A_2 \ldots A_{n-1}$ noterminal belgilar qo'shiladi va quyidagi qoidalar kiritiladi:

$A \to A_{n-1}t_n$

$A_{n-1} \to A_{n-2}t_{n-1}$

...

$A_2 \to A_1t_2$

$A_1 \to t_1$

Agar $A \to B$ yoki $A \to \varepsilon$ qoidalar uchrasa ular o'zgartirilmasdan P' to'plamiga ko'chiriladi.

3-qadam. G' grammatikani P' qoidalar to'plami ko'rib chiqiladi.

Agar $A \to B$ qoida uchrasa , barcha $B \to C$, $B \to Ct$, $B \to t$, $B \to \varepsilon$ kabi qoidalar o'rniga $A \to C$, $A \to Ct$, $A \to t$, $A \to \varepsilon$ qoidalar qo'shiladi, A→B qoida P' to'plamdan olib tashlanadi.

Agar $A \to \varepsilon$ qoida uchrasa, barcha $B \to A$, $B \to At$ kabi qoidalar o'rniga $B \to \varepsilon$ va $B \to t$ qoidalar qo'shiladi, $A \to \varepsilon$ qoida P' to'plamdan olib tashlanadi.

Almashtirish natijasida bir hil qoidalar uchrasa ulardan bittasi qoldiriladi.

4-qadam. G' grammatikada boshlang'ich belgi sifatida S olinadi.

Misol:

$G(\{a,*,/\},\{S,C\},P,S)$ chap chiziqli muntazam grammatika

P:

$S \to C*/$

$C \to /* \mid Ca \mid C* \mid C/$

qoidalar bilan berilgan. Bu grammatika izohlarni ta'riflaydi.va uni avtomat grammatikaga o'tkazamiz.

1-qadam. $VN' = \{S,C\}$

2-qadam. P qoidalarni ko'rib chiqamiz.

$S \to C*/$ qoida uchun VN' to'plamga S_1 belgi qo'shamiz va P' to'plamga $S \to S_1/$ va $S_1 \to C*$ qoidalarni qo'shamiz.

$C \to /*$qoida uchun VN' to'plamga C_1 belgi qo'shamiz va P' to'plamga $C \to C_1*$ va $C_1 \to /$ qoidalarni qo'shamiz.

Qolgan $C \to Ca \mid C* \mid C/$ qoidalarni o'zgartirmasdan P' to'plamiga ko'chiramiz.

3-qadam. Bu misolda $A \to B$ yoki $A \to \varepsilon$ qoidalar yo'q.

4-qadam. G' grammatikada boshlang'ich belgi sifatida Sni olamiz.

Natijada chap chiziqli avtomat grammatika hosil bo'ladi.

$G'(\{a,/,*\},\{S,S_1,C,\},P',S)$;

P':

$S \to S_1/$

$S_1 \to C*$

$C \to C_1*$

$C_1 \to /$

$C \to Ca \mid C* \mid C/$

Bu grammatikada S_1 va C_1 noterminallar o'rniga A va B belgilarni ishlatish natijasida quyidagi ekvivalent grammatikaga ega bo'lamiz:

$G'(\{a,/,*\},\{A,B,C,S,P',S)$;

P':

$S \to A/$

$A \to C*$

$C \to B* \mid Ca \mid C* \mid C/$

$B \to /$

O'ng chiziqli muntazam grammatikani o'ng chiziqli avtomat grammatikaga keltirish

$G(VT,VN,P,S)$ o'ng chiziqli muntazam grammatika berilgan $G'(VT,VN',P',S)$ o'ng chiziqli avtomat grammatikani tuzish kerak.

1-qadam. Barcha noterminal belgilar VN to'plamdan G' grammatikani VN' to'plamiga o'tkaziladi;

2-qadam. G grammatikani P qoidalar to'plami ko'rib chiqiladi.

Agar unda $A \to tB$ yoki $A \to t$, $A,B \in VN$, $t \in VT$ qoidalar uchrasa ularni o'zgartirmasdan G' grammatikani qoidalar to'plamiga o'tkaziladi.

Agar G grammatikada $A \to t_1t_2$ ..... t_nB, $n>1$, $A,B \in VN$, $t_i \in VT$ uchrasa, VN' to'plamiga $A_1,A_2 \ldots A_{n-1}$ noterminal belgilar qo'shiladi va P' to'plamiga quyidagi qoidalar kiritiladi:

$A \to t_1 A_1$

$A_1 \to t_2 A_2$

...

$A_{n-2} \to t_{n-1}A_{n-1}$

$A_{n-1} \to t_nB$

Agar G grammatikada $A \to t_1t_2$ ..... t_n, $n>1$, $A \in VN$, $t_i \in VT$ uchrasa, VN' to'plamiga $A_1,A_2 \ldots A_{n-1}$ noterminal belgilar qo'shiladi va quyidagi qoidalar kiritiladi:

$A \to t_1 A_1$

$A_1 \to t_2 A_2$

...

$A_{n-2} \to t_{n-1}A_{n-1}$

$A_{n-1} \to t_n$

Agar $A \to B$ yoki $A \to \varepsilon$ qoidalar uchrasa ular o'zgartirilmasdan P' to'plamiga ko'chiriladi.

3-qadam. G' grammatikani P' qoidalar to'plamini ko'rib chiqiladi.

Agar $A \to B$ qoida uchrasa , barcha $B \to C$, $B \to tC$, $B \to t$, $B \to \varepsilon$ kabi qoidalar o'rniga $A \to C$, $A \to tC$, $A \to t$, $A \to \varepsilon$ qoidalar qo'shiladi, $A \to B$ qoida P' to'plamdan olib tashlanadi.

Agar $A \to \varepsilon$ qoida uchrasa , barcha $B \to A$, $B \to tA$ kabi qoidalar o'rniga $B \to \varepsilon$ va $B \to t$, qoidalar qo'shiladi, $A \to \varepsilon$ qoida P' to'plamdan olib tashlanadi.

Almashtirish natijasida bir xil qoidalar uchrasa ulardan bittasi qoldiriladi.

4-qadam. G'grammatikada boshlang'ich belgi sifatida S olinadi.

Misol:

$G(\{a,*,/\},\{S,C\},P,S)$ o'ng chiziqli muntazam grammatika

P:

$S \rightarrow /*C$

$C \rightarrow aC \mid *C \mid /C \mid */$

qoidalar bilan berilgan. Bu grammatikani o'ng chiziqli avtomat grammatikaga o'tkazamiz.

1-qadam. $VN' = \{S,C\}$

2-qadam. P qoidalarni ko'rib chiqamiz.

$S \rightarrow /*C$ qoida uchun VN' to'plamga S_1 belgi qo'shamiz va P' to'plamga $S \rightarrow /S_1$ va $S_1 \rightarrow *C$ qoidalarni qo'shamiz.

$C \rightarrow */$ qoida uchun VN' to'plamga C_1 belgi qo'shamiz va P' to'plamga $S \rightarrow *C_1$ va $C_1 \rightarrow /$ qoidalarni qo'shamiz.

Qolgan $C \rightarrow aC \mid *C \mid /C$ qoidalarni o'zgartirmasdan P' to'plamiga qo'shamiz.

3-qadam. Bu misolda $A \rightarrow B$ yoki $A \rightarrow \varepsilon$ kabi qoidalar yo'q.

4-qadam. G'grammatikada boshlang'ich belgi sifatida S ni olamiz.

Natijada o'ngchiziqdi avtomat grammatika hosil bo'ladi.

$G'(\{a,/,*\},\{S,S_1,C,C_1\},P',S)$:

P':

$S \rightarrow /S_1$

$S_1 \rightarrow *C$

$C \rightarrow *C_1 \mid aC \mid *C \mid /C$

$C_1 \rightarrow /$.

Bu grammatikada S_1 va C_1 noterminallar o'rniga A va B noterminal belgilar ishlatish natijasida quyidagi ekvivalent grammatika ega bo'lamiz:

$G'(\{a,/,*\},\{A,B,C,S\},P',S)$ bunda P':

$S \rightarrow /A$

$A \rightarrow *C$

$S \rightarrow *B \mid aC \mid *C \mid /C$

$B \rightarrow /$.

Muntazam ifodalar

Muntazam ifodalar - bu tilni berish turlaridan biri. Muntazam ifoda zanjirlar to'plamini maxsus ko'rinishda belgilaydi. Muntazam ifodalar berilgan alifbo ustidan uchta amal yordamida muntazam tilni ta'riflash imkoniyatini beradi. Bu birlashtirish "+" , konkantenatsiya (ulash) "." (nuktani yozmasa ham bo'ladi) va iteratsiya (qaytarish) "*" amallari.

Ta'rif: Bizga V alifboda L, L_1,L_2 tillar berilsin. Unda til $L_1 \cup L_2$: L_1 va L_2 tillarni birlashmasi deb nomlanadi va uning zanjirlari yoki L_1 yoki L_2 kiradi; til L_1L_2 : L_1 va L_2 tillarni konkontenatsiyasi (ulamasi) deb nomlanadi va uning zanjirlari L_1 tilning ihtiyoriy zanjirini L_2 tilning ihtiyoriy zanjiriga ulash natijasida hosil bo'ladi; Tilni i-darajasi deb $L^i = L^{i-1}L$, i>0 va $L^0 = \{\varepsilon\}$ xisoblanadi. $L^* = \bigcup_{n=0}^{\infty} L^n$ til Lni iteratsiyasi deb nomlanadi. Buni adabiyotda Klini iteratsiyasi deb nomlaydi.

V alifbo ustidagi muntazam ifodalar quyidagi qoidalar bilan tuziladi (rekursiv ta`rifi):

1. $\varnothing$ - bo'sh ifodani bildiradi. $L(\varnothing) = \varnothing$
2. ε – bo'sh zanjir, bu muntazam ifoda, va $L(\varepsilon) = \{\varepsilon\}$.
3. Alifboni alohidagi belgisi "a"– muntazam ifoda, va $L(a) = \{a\}$ ko'rinishda belgilanadi, ya'ni bitta belgili zanjir.
4. Agar α va β, muntazam ifodalar bo'lsa :

a) $\alpha+\beta$ - muntazam ifoda va $L(\alpha+\beta) = L(\alpha) \cup L(\beta)$

b) $\alpha\beta$ –muntazam ifoda va $L(\alpha\beta) = L(\alpha)\, L(\beta)$

c) β^* –muntazam ifoda va $L(\beta^*) = (L(\beta))^*$

5. Agar β muntazam ifoda bo'lsa, (β) ham muntazam ifoda.
6. Boshqa xech qanday ifoda muntazam emas

Ifodada uchragan amallarni o'z ustuvorligi bor. Birinchi navbatda iteratsiya (*) amali bajariladi, ikkinchi navbatda konkantenatsiya va oxirida yig'im(+) amali bajariladi,. Amallar chapdan o'nga qarab bajariladi. Ifodani yozishda qavslar ishlatishi mumkin, shunda qavs ichidagi ifoda birinchi navbatda hisoblanadi.

Agar α, β, γ – muntazam ifodalar bo'lsa, ular uchun quyidagi formulalar to'g'ridir:

1. $\varepsilon+\alpha\alpha^* = \varepsilon+\alpha^*\alpha = \alpha^*$
2. $\alpha+\beta = \beta+\alpha$

3. $\alpha+(\beta+\gamma) =(\alpha+\beta) + \gamma$
4. $\alpha(\beta+\gamma) = \alpha\beta + \alpha\gamma$
5. $(\beta+\gamma)\alpha = \beta\alpha + \gamma\alpha$
6. $\alpha(\beta\gamma) =(\alpha\beta)\gamma$
7. $\alpha+\alpha = \alpha$
8. $\alpha+\alpha^* = \alpha^*$
9. $\varepsilon+\alpha^* = \alpha^*+\varepsilon= \alpha^*$
10. $\varnothing^* =\varepsilon$
11. $\varnothing\alpha = \alpha\varnothing= \varnothing$
12. $\varnothing+\alpha = \alpha+\varnothing= \alpha$
13. $\varepsilon\alpha = \alpha\varepsilon= \alpha$
14. $(\alpha^*)^* = \alpha^*$

Misollar:

L $(a+b)$ ifoda tilda "a" yoki "b" bo'lishi mumkinligini ko'rsatadi

L$((a+b)^*)$ ifoda $\{a,b\}^*$ zanjirlar to'plamini bildiradi (ya'ni "a" va "b" belgilardan ixtiyoriy ketma-ketlikdagi zanjirlar to'plami)

L(a^*b^*) ifoda $L=\{a^n b^m \quad n\geq 0, m\geq 0\}$ tildagi zanjirlar to'plamini bildiradi.

Ishorali butun sonlar tili muntazam ifoda yordamida quyidagicha yoziladi:

$\alpha=$"+" +"-" $+\varepsilon$

$\beta=$"0"+"1"+ "2"+"3"+"4"+"5"+"6"+"7"+"8"+"9"

$L_{son}(\alpha\beta\beta^*)$

Bunda amallar bilan adashtirmaslik uchun ishora qo'shtirnoq ichiga olingan.

Muntazam ifodalar chekli avtomatlarni qurishda keng ishlatiladi.

Har qanday muntazam ifoda bilan berilgan til uchun, chekli avtomat tuzish mumkin.

Har qanday chekli avtomat bilan berilgan til uchun, shu tilni aniqlaydigan muntazam ifoda tuzish mumkin.

O'ng chiziqli muntazam grammatikalar xossalari

Agar L_1 va L_2 tillar o'ng chiziqli grammatikaga asoslangan bo'lsa, $L_1 \cup L_2$, $L_1.L_2$ va L_1^* tillar ham o'ng chiziqlikdir.

$L_1(G_1)$ tilni o'ng chiziqli grammatika $G_1(VT_1,VN_1,P_1,S_1)$ aniqlasin. $L_2(G_2)$ tilni o'ng chiziqli grammatika $G_2(VT_2,VN_2,P_2,S_2)$ aniqlasin.

$VN_1 \cap VN_2 = \varnothing$ bo'lsin. Unda:

I. $L=L_1 \cup L_2$ tilni aniqlovchi o'ng chiziqli grammatika $G(VT,VN,P,S)$ quyidagicha bo'ladi:

$VT=VT_1 \cup VT_2$, $VN=VN_1 \cup VN_2 \cup \{S\}$, $P=P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$, bunda S yangi boshlang'ich belgisi.

II. $L=L_1L_2$ tilni tafsiflovchi o'ng chiziqli grammatika $G(VT,VN,P,S)$ quyidagicha aniqlanadi :

$VT=VT_1 \cup VT_2$, $VN=VN_1 \cup VN_2$, $S=S_1$

P :

(1) P_1dagi $A \to xB$ qoidalar o'zgarmasdan Pga ko'chiriladi;

(2) P_1dagi $A \to x$ qoidalar $A \to xS_2$ ko'rinishda Pga ko'chiriladi;

(3) P_2dagi qoidalar o'zgarmasdan Pga ko'chiriladi.

III. $L=L_1^*$ tilni aniqlovchi o'ng chiziqli grammatika $G(VT,VN,P,S)$ quyidagicha aniqlanadi:

$VT=VT_1$, $VN=VN_1 \cup \{S\}$, bunda S yangi boshlang'ich belgisi

P :

(1) P_1dagi $A \to xB$ qoidalar o'zgarmasdan Pga ko'chiriladi;

(2) P_1dagi $A \to x$ qoidalar Pga ko'chiriladi va mos $A \to xS_1$ qoidalar Pga qo'shiladi;

(3) $S \to S_1 \mid \varepsilon$ qoidalar Pga qo'shiladi.

Xulosa

Chekli avtomatlarni qo'llash uchun muntazam grammatikani ma'lum algoritmlardan foydalanib muntazam avtomat grammatikaga o'tkazish kerak. Muntazam til muntazam ifodalar orqali berilishi mumkin. Muntazam ifodalarda zanjirlar ustidan uchta amal bajariladi. Bu birlashtirish, konkontenatsiya (ulash) va qaytarish amallari. Har qanday muntazam ifoda orqali berilgan til uchun chekli avtomat qurish mumkin, va teskarisi ham to'g'ri.O'ng chiziqli grammatika ajoib hossaga ega. Agar L_1 va L_2 tillar o'ng chiziqli grammatikaga asoslangan bo'lsa, $L_1 \cup L_2$, $L_1.L_2$ va L_1^* tillar ham o'ng chiziqlikdir.

Nazorat uchun savollar

1. Chap chiziqli avtomat grammatikani ta'rifini keltiring.
2. O'ng chiziqli avtomat grammatikani ta'rifini keltiring.
3. Chap chiziqli muntazam grammatikani chap chiziqli avtomat grammatikaga o'tkazish algoritmini izohlang.
4. O'ng chiziqli muntazam grammatikani o'ng chiziqli avtomat grammatikaga o'tkazish algoritmini izohlang.
5. Muntazam ifodalar ta'rifini bering.
6. Muntazam ifodalar formulalarni keltiring.
7. O'ng chiziqli grammatikani hossalarini keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan til uchun chap chiziqli muntazam va avtomat grammatika hamda

o'ng chiziqli muntazam va avtomat grammatika
muntazam ifoda tuzilsin:

a) $L = \{ x\alpha y\perp \mid \alpha \in \{x, y\}^* \}$;

b) $L = \{ (xy^3)^n \perp \mid n \geq 1 \}$;

s) $L = \{ (abb)^k \perp \mid k \geq 1 \}$;

d) $L = \{\omega\perp \mid \omega \in \{0,1\}^*$, bunda har 1 dan keyin 0 keladi$\}$

e) $L = \{1\omega 1\perp \mid \omega \in \{0,1\}^+$, bunda ketma-ket turgan 0 uzunligi tog'q son$\}$;

f) L – o'nlik tizimdagi ishorali butun son;

g) L – o'nlik tizimdagi ishorali fiksirlangan nuqtali haqiqiy son;

h) L – o'nlik tizimdagi ishorali tartibli haqiqiy son;

i) L – satr;

j) L – identifikator.

2. Berilgan til uchun muntazam ifodalar tuzilsin:

a) $L = \{ x\alpha y\perp \mid \alpha \in \{x, y\}^* \}$;

b) $L = \{ (xy^3)^n \perp \mid n \geq 1 \}$;

s) $L = \{ (abb)^k \perp \mid k \geq 1 \}$;

d) $L = \{\omega\perp \mid \omega \in \{0,1\}^*$, bunda har 1 dan keyin 0 keladi$\}$

e) $L = \{1\omega 1\perp \mid \omega \in \{0,1\}^+$, bunda ketma ket turgan 0 uzunligi tog' son$\}$;

f) L – o'nlik tizimdagi ishorali butun son;

g) L – o'nlik tizimdagi ishorali fiksirlangan nuqtali haqiqiy son;

h) L – o'nlik tizimdagi ishorali tartibli haqiqiy son;

i) L – satr;

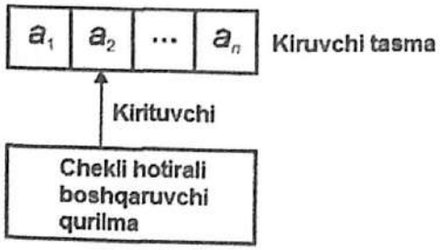
j) L – identifikator.

21-BOB.CHEKLI AVTOMATLAR

Tayanch iboralar: *chekli avtomat, to'liq aniqlangan avtomat, kirituvchi, kiruvchi tasma, boshqarish qurilma, avtomat holatlari, boshlang'ich holat, chekli holatlar, o'tish funksiya, o'tish jadvali, holatlar diagrammasi, determinantlangan chekli avtomat, nodeterminantlangan chekli avtomat, minimallashtirilgan chekli avtomat, chekli o'zgartirgich, kengayish lemma.*

Chekli avtomat ta'rifi

Berilgan zanjirning muntazam grammatika qoidalariga qanoatlanishini aniqlovchi mantiq qurilma chekli avtomat deb nomlanadi. Chekli avtomat takt (qadam) bilan ishlaydi. Chekli avtomatni modeli 21.1-rasmda keltirilgan:



21.1-rasim Chekli avtomat modeli.

Bu modelda kiruvchi tasmada kiruvchi dastlabki zanjir joylashgan va tasmadan joriy belgini kirituvchi o'qiydi.

Boshqarish qurilma holatlar to'plamidan iborat bo'lib , har bir takda joriy holatni o'zgartiradi (yoki o'zgartirmaydi) va kirituvchini bitta belgiga o'nga suradi.

Chekli avtomat (CHA) beshta obyekt bilan ta'riflanadi: $M(Q,V,\delta,H,F)$,

bunda

- ***Q – avtomatdagi holatlarning chekli to'plami;***
- ***V – avtomatni alifbosi (terminal belgilarning chekli to'plami);***
- ***δ – o'tish funksiyalarining chekli to'plami, bu funksiya QxV dekart ko'paytma to'plamini R holatlar to'plamiga aks ettiradi,***

ya'ni $\delta(q,t)=R$, $q \in Q$, $t \in V$, $R \subseteq Q$.

- ***H – avtomatning boshlang'ich holati $H \in Q$***
- ***F – avtomatning chekli holatlar to'plami $F \subseteq Q$.***

Agar $\forall t \in V$ va $\forall q \in Q$, $\exists \delta(q,t)=R$, $R \subseteq Q$ bo'lsa, chekli avtomat to'liq aniqlangan deb hisoblanadi.

CHA konfiguratsiyasi deganda q va ω juftlik, yani $(q, \omega) \in QxV^*$ bo'lgan vaziyat tushiniladi. Bunda q-holat, ω – hali o'qilmagan zanjir.

(q_0,ω)-boshlang'ich konfiguratsiya, bunda q_0=H- boshlangich xolatga teng, ω - dastlabki zanjir.

(q, ε)- chekli konfiguratsiya, $q \in F$.

Chekli avtomat muntazam tillar sinfning aniqlovchisidir.

Avtomatni ish jarayoni quyidagidan iborat.

CHA ishni boshlang'ich qonfiguratsiyasidan boshlaydi va kiruvchi zanjirning bosh belgisini kiritishini kutadi. Birinchi taktda zanjirning birinchi belgisi qabul qilinadi. Keyin shu belgi va boshlang'ich holat uchun o'tish funksiyasi xisoblanadi. So'ngra yangi konfiguratsiyaga o'tiladi va natijada kirituvchi keyingi belgida turadi, ω-dastlabki zanjir bosh belgiga qisqaradi. O'tish funksiyani qiymatiga ko'ra, joriy holat o'zgaradi yoki o'zgarmaydi. Keyingi taktda navbatdagi belgi o'qiladi, va yuqoridagi amallar qaytariladi. Agar barcha belgilar o'qib bo'linsa, avtomat ishni to'xtatadi.

Natijada CHA boshlang'ich konfiguratsiyadan biror bir chekli konfiguratsiyaga o'tsa, zanjir qabul qilingan deb hisoblanadi va u tilga tegishli bo'ladi. Aks holda zanjir qabul qilinmaydi va bunday zanjir tilga kirmaydi.

CHA ikki vaziyatda zanjirni qabul qilmaydi: birinchidan, joriy xolat va o'qilgan belgi uchun o'tish funksiyani qiymati no'malum bo'lsa, ikkinchidan, zanjir to'liq o'qilib, avtomat cheklik holatda bo'lmasa.

Chekli avtomatni boshqa usullar bilan ham ta'riflash mumkin:

- o'tishlar jadvali;
- holatlar diagrammasi, ya'ni o'tishlar grafi.

O'tishlar jadvali, bu o'tish funksiyasining jadval shaklida berilishidir. Bu jadvalda satrlar holatlarni, ustunlar esa terminallarni bildiradi. Birinchi

satrda boshlang'ich holati turadi, ohirgi satrda chekli holat joylashadi. Satr va ustun kesishgan katakda o'tish funksiyaning qiymati (holatlar to'plami) turadi.

Holatlar diagrammasining tugunlarida holat belgilar turadi va yoylari terminallar bilan belgilanadi. Boshlang'ich va chekli holatlar ikki marta aylanaga olinadi.

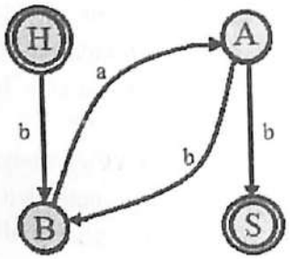
Masalan, $M(\{H,A,B,S\},\{a,b\}, \delta, H,\{S\})$ chekli avtomat berilgan bo'lsin, bunda

δ: $\delta(H,b)=B$, $\delta(B,a)=A$, $\delta(A,b)=\{B,S\}$.

Bu CHA uchun o'tishlar jadvali quyidagicha bo'ladi:

	a	b
H	--	B
A	--	$\{B,S\}$
B	A	--
S	--	--

Holat diagrammasi 21.2-rasmda keltirilgan:



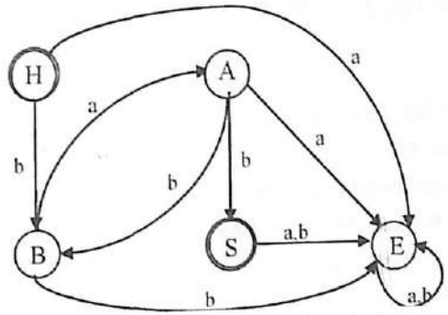
21.2-rasm. Holat diagrammasi.

Chekli avtomatni dasturlashtirish uchun uni to'liq aniqlangan shaklga keltirish ma'quldir, ya'ni joriy holatdan barcha terminal belgilari uchun boshqa holatga o'tish imkoniyati bo'lishi kerak. Buning uchun CHA ga yana bir holat qo'shamiz.Bu holat xatolikni bildiradi va unga barcha aniqlanmagan o'tishlarni uzatamiz. Xato holatdan o'ziga o'tishni bajaramiz. Natijada biz to'liq aniqlangan avtomatga ega bo'lamiz:

$M(\{H,A,B,E,S\},\{a,b\}, \delta, H,\{S\})$:

δ: $\delta(H,a)=E$,$\delta(H,b)=B$, $\delta(B,a)=A$, $\delta(B,b)=E$, $\delta(A,a)=E$,$\delta(A,b)=\{B,S\}$, $\delta(E,a)=E$, $\delta(E,b)=E$, $\delta(S,a)=E$, $\delta(S,b)=E$

To'liq aniqlangan avtomatning holatlar diagrammasi 21.3-rasmda keltirilgan.



21.3.- rasm To'liq aniqlangan avtomatning holatlar diagrammasi.

Holatlar diagrammasi bilan qulay ishlash uchun bir nechta kelishuvlar kiritamiz:

- Agar bir holatdan boshqa holatga o'tishda har hil belgilar uchun bir nechta yoylar chiqsa, ular o'rniga bitta yoy chizib, uni belgilar ro'yhati bilan belgilaymiz.
- Belgilanmagan yoyni ayni holat uchun boshqa yoylarga kirmagan terminallar ro'yhati, deb faraz qilamiz.

Determinantlangan va nodeterminantlangan chekli avtomatlar

Agar CHAning har bir holati va har qanday kiruvchi belgisi uchun o'tish funksiyaning qiymatlar to'plami faqat bitta holatdan iborat bo'lsa yoki umuman to'plam bo'sh bo'lsa, ya'ni $\forall t \in V$ va $\forall q \in Q$, $\exists\, \delta(q,t)=\{r\}$, $r \in Q$ yagona holat yoki $\delta(q,t)=\varnothing$, unda biz determinantlangan chekli avtomatga (DCHA) egamiz.

Aks holda, chekli avtomat nodeterminantlagan deb xisoblanaymiz (NCHA), ya'ni $\exists t \in V$ va $\exists q \in Q$, $\exists \delta(q,t)=\{R\}$, $R \subseteq Q$- holatlar chekli to'plami va bu to'plamda holatlar soni bittadan ortiq bo'lishi mumkin.

Isbotlanganki har qanday NCHAni ekvivalent DCHAga o'tkazish mumkin. Kompilyator tuzishda asosan to'liq aniqlangan DCHA qo'llanadi.

NCHA avtomatni DCHAga o'tkazish algoritmi

Agar har qanday kiruvchi zanjir uchun boshlang'ich holatdan hech bir yo'l bilan A holatga o'tib bo'lmasa , bunday holat erishib bo'lmaydigan deb nomlanadi.

Kirish: $M\ (Q, V, \delta, H, F)$ – nodeterminantlangan chekli avtomat.

Chiqish: $M'\ (Q', V, \delta', H', F')$ – ekvivalent determinantlangan chekli avtomat.

Algoritm:

1. Q' holatlar to'plami Q to'plamni to'plamlar qismidan tashkil topadi. Q'da har bir holat $[A_1A_2,...,A_n]$ bilan belgilanadi, bunda $A_i \in Q$, ya'ni Q'dagi har bir holat Q to'plamdagi holatlar kombinatsiyasi bo'ladi.

2. O'tish funksiyasi $\delta'([A_1A_2,...,A_n],t)=[B_1B_2,...,B_m]$ ko'rinishda aniqlaymiz, bunda har qanday $1 \le j \le m$ va biror bir $1 \le i \le n$ uchun $\delta(A_i, t) = B_j$ o'tish funksiyasi mavjuddir.

3. $H' = H$.

4. F' chekli holatlar to'plami, u F to'plamida uchragan barcha holatlar qatnashgan kombinatsiyalardan iborat.

Agar Q'da erishib bo'lmaydigan holatlar bo'lsa ularni olib tashlaymiz va ular qatnashgan o'tish funksiyalarni ham olib tashlaymiz.

Misol.

Bizga NCHA berilgan: $M = (\{H, A, B, S\}, \{0, 1\}, \delta, \{H\}, \{S\})$,bunda

$\delta(H, 1) = B\delta(B, 0) = A$

$\delta(A, 1) = B\delta(A, 1) = S$,ya'ni $\delta(A, 1) = \{B,S\}$

mos DCHA holatlar to'plami quyidagicha bo'ladi:

$Q' = \{[H], [A], [B], [S], [HA], [HB], [HS], [AB], [AS], [BS], [HAB], [HAS], [ABS], [HBS], [HABS]\}$, ya'ni Qdagi holatlar kombinatsiyalaridan tuzilgan yangi holatlar.

O'tish funksiyalari quyidagicha bo'ladi:

$\delta'([H], 1) = [B]$
$\delta'([A], 1) = [BS]$
$\delta'([HB], 1) = [B]$
$\delta'([HS], 1) = [B]$
$\delta'([AB], 0) = [A]$
$\delta'([BS], 0) = [A]$
$\delta'([HAB], 1) = [BS]$
$\delta'([B], 0) = [A]$
$\delta'([HA], 1) = [BS]$
$\delta'([HB], 0) = [A]$
$\delta'([AB], 1) = [BS]$
$\delta'([AS], 1) = [BS]$
$\delta'([HAB], 0) = [A]$
$\delta'([HAS], 1) = [BS]$

$\delta'([ABS], 1) = [BS]$
$\delta'([HBS], 1) = [B]$
$\delta'([HABS], 1) = [BS]$
$\delta'([ABS], 0) = [A]$
$\delta'([HBS], 0) = [A]$
$\delta'([HABS], 0) = [A]$

$F' = \{[S], [HS], [AS], [BS], [HAS], [ABS], [HBS], [HABS]\}$

Shunda erishib bo'ladigan holatlar quyidagilar $[H]$, $[B]$, $[A]$ va $[BS]$, kolgan holatlarga erishib bo'lmaydi, shuning uchun ularni va ularga tegishli o'tish funksiyalarni ham olib tashlaymiz.

Shunday qilib, $M' = (\{[H], [B], [A], [BS]\}, \{0, 1\}, \delta', H, \{[BS]\})$, bunda

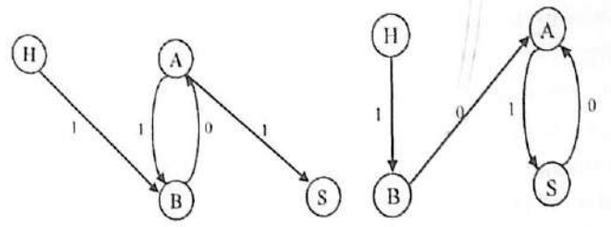
$\delta'([H], 1) = [B]$, $\delta'([B], 0) = [A]$, $\delta'([A], 1) = [BS]$, $\delta'([BS], 0) = [A]$

$[BS]$ holatni S bilan belgilaymiz.

Natijada NCHAni DCHAga o'tkazdik:

$M' = (\{H, B, A, S\}, \{0, 1\}, \delta', H,S)$, bunda $\delta'(H, 1) = B$,
$\delta'(B, 0) = A$, $\delta'(A, 1) = S$, $\delta'(S, 0) = A$.

NCHA diagrammasi 21.4a-rasmda va DCHA diagrammasi 21.4b-rasmda keltirilgan.



a) b)

21.4-rasm. NCHA va DCHA diagrammalar

DCHAni tuzish jarayoniga extiyotkorlik bilan qarash kerak, chunki yangi holatlar soni ko'payib ketishi mumkin. Natijada DCHA qurishga ketgan harajatlar, dastlabki CHA ni modellashtirishdan ancha murakkab bo'lib ketishi mumkin.

Chekli avtomatni minimallashtirish

Chekli avtomatlarni minimallashtirish mumkin. Bu jarayonda holatlar soni kamaygan ekvivalent CHA quriladi. Minimallashtirishda ekvivalent holatlar to'plamni qurish algoritmi qo'llanadi.

CHAning holatlar to'plamida ekvivalent munosabat sinflari tushinchasini kiritamiz:

1) 0-ekvivalentlik munosabati: har qanday q va q' holatlar 0-ekvivalent xisoblanadi $q\equiv^0 q'$, agar ularning ikkalasi chekli holatlar bo'lsa yoki ikkalasi chekli holatlar bo'lmasa. Yani 0-ekvivalent holatlar to'plami ikkita, bu F va Q/F. Bu to'plamlar 0-ekvivalent munosabatning sinflari hisoblanadi $R(0)=\{Q/F, F\}$

2) n-ekvivalentlik munosabati: har qanday holatlar q va q' n-ekvivalent xisoblanadi $q\equiv^n q'$, agar $q\equiv^{n-1} q'$ bo'lsa va har qanday kiruvchi t belgi uchun $\delta(q, t) \equiv^{n-1} \delta(q', t)$, ya'ni $\delta(q, t)$, $\delta(q', t)$ holatlar $(n-1)$-ekvivalentni bitta sinfga kirsa. Shunda q va q'-$R(n)$ ekvivalent munosabatning bitta sinfga kiradi, aks holda ular har hil sinfga kiradi.

Yekvivalent holatlar to'plamini qurish algoritmi:

1 - qadam. $R(0)=\{Q/F, F\}$ quriladi, $n=0$;

2 - qadam. $n=n+1$ va $R(n-1)$ asosida $R(n)$ quriladi: $R(n)=\{r_i(n): \{q_{ij}\in Q;\ \forall t\in V\ \delta(q_{ij},t)\subseteq r_j(n-1)\}\ \forall i,j<=m\}$, bunda m dastlabki avtomatning holatlar soni. Ya'ni n-ekvivalentlik holatlar sinfiga, bir hil belgi uchun bitta sinfdagi holatlarga o'tadiganlar kirittiladi.

3- qadam. Agar $R(n)=R(n-1)$ bo'lsa sinflar tuzilgan bo'ladi, aks holda 2-qadamga o'tiladi.

CHA minimallashtirish algoritmi:

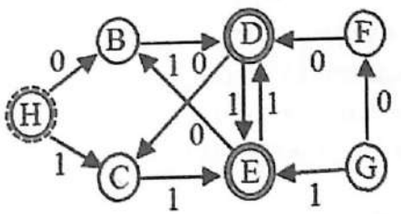
1. Agar avtomat NCHA bo'lsa, uni DCHAga o'tkaziladi
2. Yerishib bo'lmaydigan holatlar olib tashlanadi.
3. Avtomatning ekvivalentlik sinflari tuziladi.
4. Dastlabki bo'lmish CHA ekvivalentlik sinflari, minimallashtirilgan natijaviy CHAni holatlari, deb hisoblanadi.
5. Natijaviy CHAni o'tish funksiyalari, dastlabki bo'lmish CHAning o'tish funksiyalari asosida quriladi.

Algoritmning ishlashi natijasida hosil bo'lgan CHAdagi holatlar soni minimallashtirilgan bo'ladi.

Misol. CHA berilsin: $M(\{H,B,C,D,E,F,G\},\{0,1\},\delta,H,\{D,E\})$

δ: $\delta(H, 0)=B$, $\delta(H, 1)=C$,$\delta(B, 1)=D$,$\delta(C, 1)=E$,$\delta(E, 0)=B$,
$\delta(E, 1)=D$,$\delta(D, 0)=C$,$\delta(D, 1)=E$,$\delta(F, 0)=D$, $\delta(G, 0)=F$,
$\delta(G, 1)=E$

Holatlar diagrammasi 17.5-rasmda keltirilgan.



Rasm- 21.5. Minimallashtirilmagan CHAning holat diagrammasi.

Minimallashtirish algoritmi quyidagicha bo'ladi:

1-qadam. CHA determinantlangan, shu sababli hech narsa qilmaymiz

2-qadam. F va G holatlar erishib bo'lmaydigan, ularni olib tashlaymiz.

3-qadam. Yekvivalent sinflar tuzamiz:

$R(0)=\{\{H,B,C\},\{D,E\}\}$, $n=0$;
$R(1)=\{\{H\}\{B,C\},\{D,E\}\}$, $n=1$;
$R(2)=\{\{H\}\{B,C\},\{D,E\}\}$, $n=2$;

Natijada uchta ekvivalentlik sinflar bo'ldi: $\{H\},\{B,C\},\{D,E\}$

4-qadam. Minimal CHAni holatlarini belgilab chiqamiz:

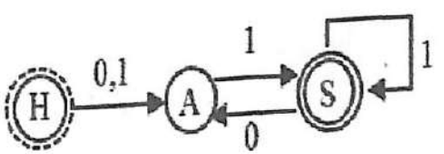
$H=H, A=BC, S=DE$;

5-qadam. Natijada minimal CHA hosil bo'ldi:

$M'(\{H,A,S\},\{0,1\}, \delta', H,\{S\})$, bunda

δ': $\delta'(H, 0)=A$, $\delta'(H, 1)=A$,$\delta'(A, 1)=S$,$\delta'(S, 0)=A$,$\delta'(S, 1)=S$

Minimallashtirilgan CHA holatlar diagrammasi 21.6-rasmda keltirilgan:



21.6-rasm. Minimallashtirilgan CHAning holat diagrammasi.

Chap chiziqli avtomat grammatika asosida chekli avtomatni tuzish algoritmi

Kirish: Chapchizikli avtomat grammatika $G(VT,VN,P,S)$;

Chiqish: Chekli avtomat $M(Q, V, \delta, H, F)$;

Algoritm:

1-qadam: $Q=VN\cup\{H\}$; $V=VT$; $F=\{S\}$;

2-qadam: Qoidalarni ko'rib chiqiladi va $A\to t$ kabi qoidalar mavjud bo'lsa, ularga mos $\delta(H,t) = \{A\}$ o'tish funksiyasining qiymati kiritiladi.

3-qadam: Har bir $A\to Bt$ qoida uchun $\delta(B, t) = \{A\}$ o'tish funksiya qiymati aniqlanadi.

Misol, quyidagi grammatika $G = (\{a,b, \perp\}, \{S,A,B,C\}, P, S)$ berilgan bundagi qoidalar esa quyidagicha bo'lsin:

$P: S \to C\perp$

$S \to Ab \mid Ba$

$A \to a \mid Ca$

$B \to b \mid Cb$

To'liq chekli avtomat $M(Q, V, \delta, H, F)$ tuzamiz.

Yuqoridagi algoritmga asosan:

$Q=\{A,B,C,S,H,ER\}$, $V=\{a,b, \perp\}$,$F=\{S\}$

δ:

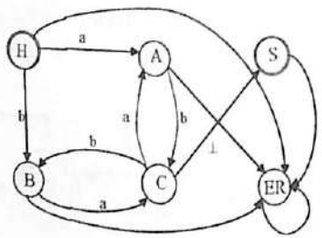
$\delta(H,a)=A$, $\delta(H,b)=B$, $\delta(H, \perp)=ER$, $\delta(A,b)=C$, $\delta(A,a)=ER$,

$\delta(A,\perp)=ER$,$\delta(B,a)=C$, $\delta(B,b)=ER$, $\delta(B,\perp)=ER$,

$\delta(C,a)=A$, $\delta(C,b)=B$, $\delta(C,\perp)=S$,$\delta(S,a)=ER$, $\delta(S,b)=ER$,

$\delta(S,\perp)=ER$,$\delta(ER,a)=ER$, $\delta(ER,b)=ER$, $\delta(ER,\perp)=ER$

Yendi holatlar diagrammasini chizamiz (21.7-rasm)



21.7.-rasm. Chap chiziqli grammatika asosida qurilgan CHAning holatlar diagrammasi.

Chizilgan diagramma bo'yicha CHAning dasturini tuzamiz:

```
#include <fstream.h>
#include <iostream.h>
using namespace std;
ifstream fin; // oqimnomi
char c;
void gc(){fin>>c;} // joriybelgi
int scan_G()
{
enum state {H, A, B, C, S, ER};  //holatlar
to'plami
state CS;  // CS - joriy holat
CS=H;
gc ();  /* birinchi belgi o'qilmoqda */
do switch (CS)
{
case H: if (c == 'a') {gc(); CS = A;}
      else if (c == 'b') {gc(); CS = B;}
            else CS = ER;
        break;
case A: if (c == 'b') {gc(); CS = C;}
        else CS = ER;
       break;
case B: if (c == 'a') {gc(); CS = C;}
       else CS = ER;
      break;
case C: if (c =='a') {gc(); CS = A;}
     else if (c == 'b') {gc(); CS = B;}
          else if (c == ' | ') CS = S;
                else CS = ER;
} while (CS != S && CS != ER);
  if (CS == ER) return -1; else return 0;
}
void main ()
{
int i;
```

```
    fin.open ("data.txt"); /* zanjir fayli
ochiladi */
    i=scan_g();
      if (i) then cout << "zanjir
noto'g'ri"<<endl;
      else cout << "zanjir to'g'ri";
    }
```

Zanjirni holat diagrammasi asosida tahlil qilish misoli

Yuqorida ko'rsatilgan aniqlovchini *abba⊥* zanjir uchun ishlashini ko'rib chiqamiz. HD bo'yicha tahlil qilish jarayonida quyidagi o'tishlar ketma-ketligini ko'ramiz:

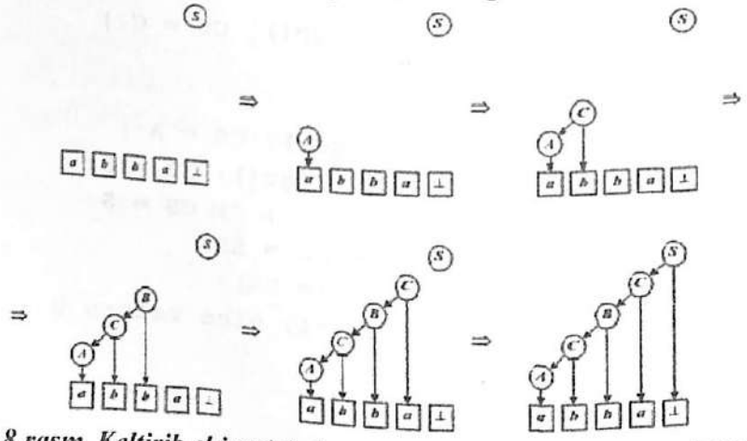
$$H \xrightarrow{a} A \xrightarrow{b} C \xrightarrow{b} B \xrightarrow{a} C \xrightarrow{\perp} S$$

Agar biz holatlarni noterminal deb bilsak, ko'rinib turibdiki har bir "noterminal-terminal" juftlik (*Nt*) noterminal *L*ga almashtirilmoqda, ya'ni $L \rightarrow Nt$ qoida teskari qo'llanilmoqda. Bunday qo'llanish teskari mil bilan yoziladi $Nt \leftarrow L$, shunda HDga mos bo'lgan quyidagi ketma-ketlik hosil bo'ladi:

$$abba\perp \leftarrow Abba\perp \leftarrow Cba\perp \leftarrow Ba\perp \leftarrow C\perp \leftarrow S$$

Bu ketma-ketlik *abba⊥* zanjirni o'ng teskari keltirib chiqarishga to'g'ri keladi.

Unga pastdan yuqoriga tuzilgan daraxt to'g'ri keladi (21.8-rasm).



21.8-rasm. Keltirib chiqarish daraxtini pastdan yuqoriga qarab tuzish.

Chekli avtomat asosida chap chiziqli avtomat grammatikasini tuzish algoritmi

Kirish: Chekli avtomat *M (Q, V, δ, H, S);*

Chiqish: Chapchizikli avtomat grammatika *G(VT,VN,P,S);*

Algoritm:

1-qadam: *VN=Q /{H} VT=V;*

Har bir t belgi uchun o'tish funksiyalar ko'rib chiqiladi.

2-qadam: Agar $\delta(H, t) = \{B_1 \ldots B_n\}$ bo'lsa, P qoidalar to'plamiga $B_i \rightarrow t$qoidalar qo'shiladi.

3-qadam: Agar $\delta(A, t) = \{B_1 \ldots B_n\}$ bo'lsa, P qoidalar to'plamiga $B_i \rightarrow At$qoidalar qo'shiladi.

4-qadam: Chekli holatlar to'plami bitta S_0 holatdan iborat bo'lsa, boshlang'ich noterminal sifatida $S = S_0$ olinadi,. Aksincha chekli holatlar to'plami bittadan ortiq bo'lsa, ya'ni $S=\{S_1 \ldots S_n\}$, grammatikaga yangi boshlang'ich *S* noterminal qiritiladi, va Pga $S \rightarrow S_1 \mid \ldots \mid S_n$ yangi qoidalar ko'shiladi.

Misol, bizga chekli avtomat $M(\{A,B,C,S,H\}, \{a,b, \perp\}, \delta, H,\{S\})$

δ: $\delta(H,a)=A$, $\delta(H,b)=B$, $\delta(A,b)=C$,$\delta(B,a)=C$,$\delta(C,a)=A$, $\delta(C,b)=B$, $\delta(C,\perp)=S$

berilgan, va chap chiziqli avtomat grammatika*G(VT,VN,P,S)* tuzamiz.

Bunda $VN=\{A,B,C,S\}$, $VT=\{a,b, \perp\}$.

P qoidalar to'plamini algoritmga asosan quyidagicha aniqlaymiz:

$P: S \rightarrow C\perp$

$S \rightarrow Ab \mid Ba$

$A \rightarrow a \mid Ca$

$B \rightarrow b \mid Cb.$

O'ng chiziqli avtomat grammatika asosida chekli avtomatni tuzish algoritmi

Kirish: O'ngchizikli avtomat grammatika *G(VT,VN,P,S);*

Chiqish: Chekli avtomat *M(Q, V, δ, H, F);*

Algoritm:

1-qadam: $Q=VN \cup \{E\}$; $V=VT$; Bunda *E* yangi qo'shilgan holat;

2-qadam: Qoidalarni ko'rib chiqiladi va $A \to t$ kabi qoidalar mavjud bo'lsa, ularga mos $\delta(A, t) = \{E\}$ o'tish funksiyasining qiymati kiritiladi.

3-qadam: Har bir $A \to tV$ qoida uchun $\delta(A, t) = \{B\}$ o'tish funksiyasining qiymati aniqlanadi.

4-qadam: Agar $S \to \varepsilon \in P$, unda $F = \{S, E\}$, aks holda $F = \{E\}$

5-qadam: Boshlang'ich holat sifatida $H = S$ noterminalni olamiz.

Misol, quyidagi grammatika $G = (\{a,b, \perp\}, \{S,A,B\}, P, S)$ berilgan va bunda qoidalar quyidagicha bo'lsin:

$P: S \to aS \mid bA$

$A \to bA \mid \perp$

Bu grammatika $L(aa^*bb^*\perp)$ yoki $L = \{a^n b^m \perp \mid n, m \geq 1\}$ tilni ta'riflaydi.

To'liq chekli avtomat $M (Q, V, \delta, H, F)$ tuzamiz: $Q=\{A,S,E,ER\}$, $V=\{a,b, \perp\}$,

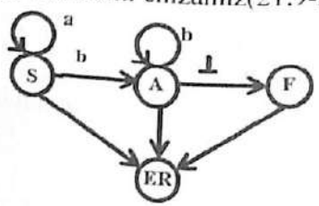
δ:

$\delta(S,a)=S$, $\delta(S,b)=A$, $\delta(A,b)=A$, $\delta(A,\perp)=E$

$\delta(A, a)=ER$, $\delta(S, \perp)=ER$, $\delta(E, a)=ER$, $\delta(E, b)=ER$, $\delta(E, \perp)=ER$

$H=S$, $F=\{E\}$

Yendi holatlar diagrammasini chizamiz(21.9-rasm.)



21.9-rasm. O'ng chiziqli grammatika asosida qurilgan CHAning holatlar diagrammasi.

Chekli avtomat asosida o'ng chiziqli avtomat grammatikani tuzish algoritmi

Kirish: Chekli avtomat $M(Q, V, \delta, H, F)$;

Chiqish: O'ngchizikli avtomat grammatika $G(VT, VN, P, S)$;

Algoritm:

1-qadam: $VN=Q$, $VT=V$;

2-qadam: Boshlang'ich noterminal sifatida boshlang'ich holat olinadi, ya'ni $S=H$.

3-qadam: Agar $A \in F$, unda P qoidalar to'plamiga $A \to \varepsilon$ qoida qo'shiladi $A \in VN$.

4-qadam: Agar $B \in \delta(A,t)$, unda P to'plamiga $A \to tB$ qoida qo'shiladi,va bunda $A,B \in VN$, $t \in VT$.

Misol, bizga chekli avtomat $M(\{A,B,C,S,H\}, \{a,b, \perp\}, \delta, H,\{S\})$

δ: $\delta(H,a)=A$, $\delta(H,b)=B$, $\delta(A,b)=C$,$\delta(B,a)=C$,$\delta(C,a)=A$, $\delta(C,b)=B$, $\delta(C,\perp)=S$

berilgan bo'lsin, o'ng chiziqli avtomat grammatika $G(VT,VN,P,S)$ tuzamiz.

Algoritm 1-qadami bo'yicha:

$VN=\{A,B,C,S,H\}$, $VT=\{a,b, \perp\}$.

2-qadam bo'yicha $S=H$:

3-qadam bo'yichaP qoidalar to'plamiga $S \to \varepsilon$ qoida qo'shamiz,

4-qadam asosida har bir o'tish funksiya uchun qoida qo'shamiz, va natijada quyidagicha bo'ladi:

$H \to aA \mid bB$

$A \to bC$

$B \to aC$

$C \to aA \mid bB \mid \perp S$

$S \to \varepsilon$

Bo'sh qoidani olib tashlaymiz va H noterminalni S bilan belgilaymiz. Natijada o'ng chiziqli avtomat grammatikamiz quyidagicha bo'ladi: $G(\{a,b, \perp\}, \{S,A,B,C\},P,S)$, bu yerda

P:

$S \to aA \mid bB$

$A \to bC$

$B \to aC$

$C \to aA \mid bB \mid \perp$

Muntazam ifoda bilan berilgan til uchun chekli avtomat tuzish

Muntazam ifodalar rekursiv ta'riflangan. Shu ta'rifga ko'ra ketmaket CHA tuziladi.

1. Agar muntazam ifoda $\varnothing$ bo'lsa, uning CHA $M(\{H,F\},V, \delta,H,\{F\})$ bo'ladi, bunda $\forall t \in V, \forall q \in Q$, $\delta(q,t)= \varnothing$.

2. Agar muntazam ifoda ε bo'lsa, uning CHA $M(\{F\},V, \delta,F,\{F\})$ bo'ladi, bunda $\forall t \in V$ $\delta(F,t)= \varnothing$, ya'ni $H=F$.

3. Agar muntazam ifoda terminal belgisi bo'lsa, $t \in V$ uning CHA $M(\{H,F\},V, \delta,H,\{F\})$ bo'ladi, bunda $\delta(H,t)= \{F\}$

4. Bizga muntazam ifodalar α va β, ularni tillari L_1 va L_2 va ularga mos bo'lgan chekli avtomatlar $M_1(Q_1,V, \delta_1,H_1,F_1)$ va $M_2(Q_2,V, \delta_2,H_2,F_2)$ berilsin, ular asosida quyidagi ifodalarga chekli avtomat tuziladi:

- $\alpha+\beta$ ifoda uchun til $L=L_1 \cup L_2$, CHA $M(Q,V, \delta,H,F)$ bo'ladi , bunda $Q=(Q_1 \cup Q_2 \cup \{H\})/\{N_1,N_2\}$

$\forall t \in V$ uchun $\delta(N,t)= \delta_1(N_1,t) \cup \delta_2(N_2,t)$,

$\forall t \in V, \forall q \in (Q_1/\{H_1\})$ uchun $\delta(q,t)= \delta_1(q,t)$,

$\forall t \in V, \forall q \in (Q_2/\{H_2\})$ uchun $\delta(q,t)= \delta_2(q,t)$,

Agar $\alpha+\beta$ ifodada ε mavjud bo'lsa $F=F_1 \cup F_2 \cup \{H\}$, aks holda $F=F_1 \cup F_2$, chekli avtomatni boshlang'ich holati N bo'ladi.

- $\alpha\beta$ ifoda uchun til $L=L_1 L_2$, CHA $M(Q,V, \delta,H,F)$ bo'ladi , bunda $Q=Q_1 \cup (Q_2/\{H_2\})$,

$\forall t \in V$, $\forall q \in (Q_1/F_1)$ uchun $\delta(q,t)= \delta_1(q,t)$

$\forall t \in V$, $\forall q \in F_1$, $\delta(q,t)= \delta_1(q,t) \cup \delta_2(N_2,t)$,

$\forall t \in V, \forall q \in (Q_2/\{H_2\})$ uchun $\delta(q,t)= \delta_2(q,t)$.

Agar $N_2 \in F_2$, unda $F=F_1 \cup F_2$, aks holda $F=F_2$, chekli avtomatni boshlang'ich holati $N=H_1$ bo'ladi.

- α^* ifoda uchun til $L=L_1^*$, CHA $M(Q,V, \delta,H,F)$ bo'ladi , bunda $Q= (Q_1/\{H_1\}) \cup \{H\}$,

$\forall t \in V$ uchun $\delta(H,t)= \delta_1(H_1,t)$,

$\forall t \in V$, $\forall q \in (Q_1/\{H_1,F_1\})$ uchun $\delta(q,t)= \delta_1(q,t)$

$\forall t \in V$, $\forall q \in F_1$, $\delta(q,t)= \delta_1(q,t) \cup \delta_1(N_1,t)$,

$F=F_1 \cup \{H\}$, boshlang'ich holati N bo'ladi.

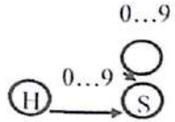
Misol,bizga ikkita til berilgan bo'lsin:

$\alpha = (0+1+2+3+4+5+6+7+8+9)$ –raqamlar.

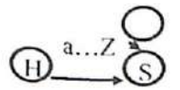
$\beta = (a+b+...+y+z+A+B+...+Y+Z)$- harflar.

$L= (\alpha\alpha^* + \beta(\beta+\alpha)^*) \perp$ ifoda orqali berilgan til uchun CHA tuzamiz.

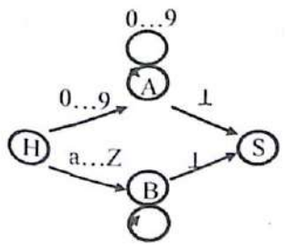
$\alpha\alpha^*$ -(natural son) XD quyidagicha:



$\beta(\beta+\alpha)^*$ - (identifikator) XD quyidagicha :



$L= (\alpha\alpha^* + \beta(\beta+\alpha)^*) \perp$XD quyidagicha:

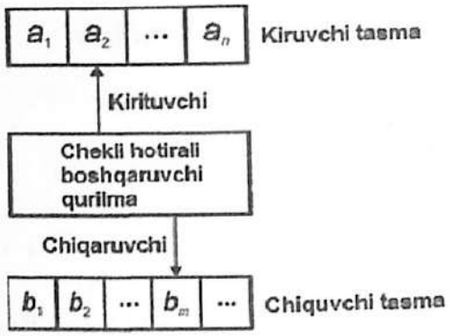


Chekli o'zgartirgich

Chekli avtomat yordamida berilgan kiruvchi zanjir muntazam tilga kirishi yoki kirmasligini oson aniqlash mumkin. Lekin faqatgina aniqlash maqsadga muvofiq emas. Ko'pincha qo'yilgan masalada, aniqlashdan tashqari chiquvchi tilda chiquvchi zanjir hosil qilish kerak bo'ladi.

Kiruvchi zanjirdan chiquvchi zanjirga tarjima qiladigan chekli avtomat asosida tuzilgan mantiq qurilma chekli o'zgartirgich (CHO') deb nomlanadi.

Chekli o‘zgartirgich modeli 21.10-rasmda keltirilgan.



21.10-rasm. Chekli o‘zgartirich modeli.

Noderminantlangan chekli o‘zgartirgich oltita obyektdan iborat P(Q,V,W,δ, H,F), bunda

- ***Q – o‘zgartirgichdagiholatlarning chekli to‘plami;***
- ***V – o‘zgartirgichga kiruvchi zanjirlar alifbosi (terminal belgilarning chekli to‘plami)***
- ***W – o‘zgartirgichdagi chiquvchi zanjirlar alifbosi (terminal belgilarning chekli to‘plami)***
- ***δ – o‘tish funksiyalarning chekli to‘plami, bu funksiya QxV dekart ko‘paytma to‘plamini R to‘plami va w chiquvchi zanjir qismi juftlikka aks ettiradi, ya’ni δ(q,t)=Y{(r,w)| r∈R⊆Q,w∈W*}, q∈Q , t∈V, Y⊆QxW .***
- ***H – o‘zgartirgichning boshlang‘ich holati N∈ Q***
- ***F – o‘zgartirgichning chekli holatlar to‘plami F⊆ Q.***

Chekli o‘zgartirgichning konfiguratsiyasi deb uchtalik $(q,\omega,\gamma)\in QxV^*xW^*$ hisoblanadi, bunda q-holat, ω – hali o‘qilmagan zanjir, γ - shu vaktga hosil bo‘lgan chiquvchi zanjir.

$(q_0, \omega,\varepsilon,)$ –boshlang‘ich konfiguratsiya, q_0=H, bunda ω–to‘liq kiruvchi zanjir.

(q, ε, γ) – chekli konfiguratsiya, q∈F, bunda γ–hosil bo‘lgan chiquvchi zanjir.

O‘zgartirgichni ish jarayoni quyidagidan iborat:

CHO‘ ishni boshlang‘ich konfiguratsiyasidan boshlaydi va kiruvchi zanjirni kiritishini kutadi, chiquvchi zanjir bo‘sh bo‘ladi. Birinchi taktda zanjirning birinchi belgisi qabul qilinadi. Keyin shu belgi va boshlang‘ich holat uchun o‘tish funksiyasi xisoblanadi,. So‘ngra yangi konfiguratsiyaga o‘tiladi, natijada kirituvchi keyingi belgida turadi-dastlabki zanjir ω bosh belgiga qisqaradi. Joriy holat o‘zgaradi yoki o‘zgarmaydi. Chiquvchi γ zanjirga uning mos qismi qo‘shiladi. Keyingi taktda kirituvchi navbatdagi belgini o‘qiydi va yuqoridagi amallar qaytariladi. Agar barcha belgilar o‘qib bo‘linsa, o‘zgartirgich ishini to‘xtatadi.

Natijada CHO‘ boshlang‘ich konfiguratsiyadan biror bir chekli konfiguratsiyaga o‘tsa, zanjir qabul qilingan deb xisoblanadi va u tilga kiradi. Qayta o‘zgartirish natijasida esa chiquvchi zanjir hosil bo‘ladi. Aks holda kiruvchi zanjir qabul qilinmaydi va chiquvchi zanjir bo‘shatiladi.

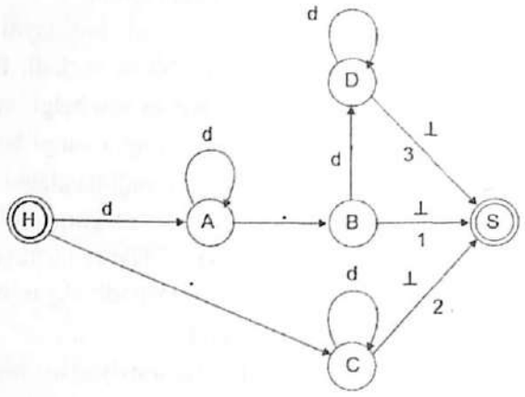
Misol, haqiqiy fiksirlangan sonlar ko‘rinishdagi zanjirlar berilgan va son *dd...d. ⊥*uchun *1*, . *dd...d⊥*uchun 2 va *dd...d.dd...d⊥* uchun 3 chiqarilsin. Bunda *d*-o‘nli raqam deb faraz qilamiz va shunda CHO‘ ni quyidagicha beramiz:

P({H,A,B,C,D,S}{d,., ⊥},{1,2,3}, δ,H,{S}).

bu yerda *δ:*

δ(H,d)=(A, ε);
δ(H,.)=(C, ε);
δ(A,d)= (A, ε);
δ(A,.)=(B, ε)
δ(B, ⊥)= (S, 1)
δ(B,d)= (D, ε)
δ(D,d)= (D, ε);
δ(D, ⊥)= (S, 3)
δ(C,d)=(C, ε)
δ(C, ⊥)=(S, 2)

Holatlar diagrammasidagi chiquvchi belgilarni yoyning ostida yozamiz, shunda (21.11-rasm):



21.11-rasm. Chekli o'zgartirgichni holatlar diagrammasi.

234.17⊥ kiruvchi zanjir uchun qurilgan CHO' quyidagi taktlar ketma-ketligini bajaradi:

(H, 234.17⊥,ε)|—
(A, 34.17⊥,ε)|—
(A,4.17⊥,ε)|—
(A, .17⊥,ε)|—
(B, 17⊥,ε)|—
(D, 7⊥,ε)|—
(D, ⊥,ε)|—
(S, ε,3) –chekli konfiguratsiya.

Muntazam tillar uchun kengayish lemmasi

Ba'zi bir hollarda biror bir til muntazamligini isbotlash kerak bo'ladi. Albatta tilni yuqorida ko'rsatilgan biror bir usul bilan aniqlash mumkin (muntazam grammatika, muntazam ifoda va chekli avtomatlar). Agar xhech bo'lmaganda bitta usul bilan til aniqlansa, u muntazam til bo'ladi. Lekin xhech qaysi ko'rsatilgan usullar tilni aniqlamasa, biz berilgan tilni muntazam yoki muntazam emas deb aytolmaymiz.

Ammo lekin tilni muntazamligini tekshiruvchi sodda usul bor. Bu usul tilni kengayish lemmasiga asoslangan. Agar biror bir berilgan til uchun muntazam tilni kengayish lemmasi bajarilsa, bunday til muntazam aks holda muntazam emasligi isbotlangan.

Muntazam tilni kengayib ketish lemmasi quyidagicha:

Agar muntazam tilga kiruvchi biror bir uzun zanjirda, uning qandaydir qismi ixtiyoriy marta qaytarilsa hosil bo'lgan zanjir (kengaytirilgan) muntazam tilga kiriladi. Matematik nuktayi nazardan lemmani quyidagicha ifodalash mumkin: L muntazam til berilgan, unda shunday $\exists r>0$ mavjud bo'lsinki, agar $\alpha\in L$ va $|\alpha|\geq p$ bo'lsa, u holda α zanjirni $\alpha=\gamma\beta\omega$ shaklda yozish mumkin, bu yerda $0<|\beta|\leq p$ va $\forall i\geq 0$ $\alpha'=\gamma\beta^i\omega$ uchun $\alpha'\in L$ kiradi.

Ushbu lemmadan foydalanib,$L=\{\ a^nb^n|\ n>0\}$ tilni muntazam emasligini isbotlaymiz. Faraz qilaylik, u muntazam til bo'lsin. Unda biror bir $\alpha=a^nb^n$ zanjirni olamiz va uni $\alpha=\gamma\beta\omega$ ko'rinishda yozamiz. Agar $\beta\in a^k$ yoki $\beta\in b^k$ $(0<k<n)$ bo'lsa, unda $i=0$ uchun zanjir $\gamma\beta^0\omega=\gamma\omega$ L tilga kirmaydi, chunki α va b soni teng bo'lmaydi,. Ya'ni lemmaning shartlari bajarilmaydi. Agarda $\beta\in a^kb^k$, unda $i=2$ uchun zanjir $\gamma\beta^2\omega=\gamma\beta\beta\omega$ L tilga kirmaydi, chunki "b"dan keyin "a"kelib qoladi. Shunday qilib, L til muntazam emasligini isbotladik.

Yendi shu lemma asosida $L=\{\ (ab)^n|\ n>0\}$ tilning muntazamligini isbotlaymiz. Unda biror bir $\alpha=(ab)^n$ zanjirni olamiz va uni $\alpha=\gamma\beta\omega$ ko'rinishda yozamiz. Agar $\beta=ab$ $(p=2)$ bo'lsa,unda har qanday $i\geq 0$ uchun zanjir $\gamma\beta^i\omega L$ tilga kiradi, ya'ni lemmani shartlari bajariladi. Shunday qilib, L til muntazam ekanligini isbotladik.

Xulosa

Berilgan zanjirning muntazam grammatika qoidalariga qanoatlanishini aniqlovchi mantiq qurilma chekli avtomat deb nomlanadi.

Chekli avtomat beshta obyekt bilan ta'riflanadi bu xolatlar chekli to'plami, avtomat alifbosi, o'tish funksiyalar, avtomatning boshlang'ich holati xamda uning chekli holatlar. Chekli avtomat xolatlar diagrammasi orqali berilishi mumkin. Chekli avtomat determinantlangan va nodeterminantlangan bo'lishi mumkin. Agar CHAning har bir holati va har qanday kiruvchi belgisi uchun o'tish funksiyaning qiymatlar to'plami faqat bitta holatdan iborat bo'lsa yoki umuman to'plam bo'sh bo'lsa bunday avtomat determinantlangan hisoblanadi. Aks holda nodeterminantlangan bo'ladi. Har qanday nodeterminantli chekli avtomatni determinantli avtomatga o'tkazish mumkin. Chekli avtomatda holatlar sonini kamaytirish mumkin.

Har qanday muntazam grammatika uchun chekli avtomat tuzish mumkin va teskarisi ham to'g'ri. Kiruvchi zanjirdan chiquvchi zanjirga tarjima qiladigan chekli avtomat asosida tuzilgan mantiq qurilma chekli o'zgartirgich (CHO') deb nomlanadi.

Muntazam tillar uchun kengayish lemmasi isbotlangan, agar u bajarilsa bunday til muntazam grammatika bo'ladi.

Nazorat uchun savollar.

1. Chekli avtomat qanday qurilmalardan iborat?
2. Chekli avtomatni rasmiy ta'rifini keltiring.
3. O'tish funksiyaning ma'nosi nimada?
4. Chekli avtomatni o'tish jadvali bilan bering.
5. Chekli avtomatni holatlar diagrammasi orqali bering.
6. To'liq chekli avtomat qanday shakllantiriladi?
7. Determinantlangan va nodeterminantlangan chekli avtomatlarni ta'rifini keltiring. Ularning farqi nimada?
8. NCHAni DCHAga o'tkazish algoritmini misolda izohlang.
9. Chekli avtomatni minimallashtirish yo'lini ko'rsating.
10. Chap chiziqli avtomat grammatikasi asosida chekli avtomatni tuzish algoritmini keltiring.
11. Chekli avtomat asosida chap chiziqli avtomat grammatikani tuzish algoritmini keltiring.
12. O'ng chiziqli avtomat grammatikasi asosida chekli avtomatni tuzish algoritmini keltiring.
13. Chekli avtomat asosida o'ng chiziqli avtomat grammatikasini tuzish algoritmini keltiring.
14. Muntazam ifodalar asosida chekli avtomatni tuzish algoritmini keltiring.
15. Chekli o'zgartirgichni izohlang.
16. Muntazam tillarni kengayish lemmasini izohlang.

Amaliyot uchun topshiriqlar

1. Muntazam grammatika quyidagiqoidalar bilan berilgan:

$S \to S0 \mid S1 \mid P0 \mid P1$

$P \to N.$

$N \to 0 \mid 1 \mid N0 \mid N1.$

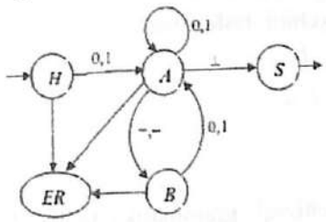
Holatlar diagrammasini tuzing va u yordamida *11.010, 0.1, 01., 100* zanjirlarni tahlil qiling. Bu grammatika qanday tilni yaratadi?

2. 17.12-rasmda XD berilgan.

a) 1011⊥, 10 + 011⊥va 0 −101+1⊥zanjirlarni tahlil qiling.

b) Ko'rsatilgan HD bo'yicha muntazam grammatikani tiklang.

c) Hosil bo'lgan grammatika qaysi tilni yaratadi?



Rasm 21.12 Holat diagramma.

3. Berilgan til uchun chap chiziqlimuntazam avtomat grammatika tuzilsin va u orqali chekli avtomat yasalsin (va) xamda HD bo'yicha tahlilchi dastur tuzilsin:

a) $L = \{ x\alpha y\perp \mid \alpha \in \{x, y\}^*\}$;

b) $L = \{ (xy^3)^n\perp \mid n\geq 1\}$;

s) $L = \{ (abb)^k\perp \mid k\geq 1\}$;

d) $L = \{\omega\perp \mid \omega \in \{0,1\}^*$, bunda har 1 dan keyin 0 keladi}

e) $L = \{1\omega 1\perp \mid \omega \in \{0.1\}^+$, bunda ketma-ket turgan 0 uzunligi tog'q son};

4. Muntazam grammatika berilgan:

$S \to A\perp$

$A \to Ab \mid Bb \mid b$

$B \to Aa$

Bu grammatika yaratiladigan til aniqlansin, uning uchun HD tuzilsin va aniqlovchinidasturi tuzilsin.

5. Quyidagi o'ng chiziqli grammatika uchun:

a) $S \to 0S \mid 0B$
$B \to 1B \mid 1C$
$C \to 1C \mid \perp$

b) $S \to 0B$
$B \to 1C \mid 1S$
$C \to \perp$

c) $S \to aB$
$B \to aC \mid aD \mid dB$
$C \to aB$
$D \to \perp$

Yekvivalent determinantlangan xossaga ega bo'lgan chap chiziqli muntazam grammatika tuzilsin.

6. Berilgan grammatika uchun:

a) turi aniqlansin;

b) yaratilgan tili aniqlansin;

c) ekvivalent avtomat grammatika tuzilsin;

d) HD va aniqlovchini dasturi tuzilsin.

$S \to 0S \mid S0 \mid D$

$D \to DD \mid 1A \mid \perp$

$A \to 0B \mid \perp$

$B \to 0A \mid 0$

7. Berilgan chap chiziqli grammatika uchun HD tuzilsin. Agar HD NCHAga to'g'ri kelsa, ekvivalent DCHA tuzilsin, uning uchun HD va aniqlovchini dastursi tuzilsin. DCHA grammatikasi tuzilsin.

a) $S \to Sa \mid Ab \mid b$
$A \to Ab \mid Sa \mid a$

b) $S \to Sb \mid Aa \mid a$
$A \to Sb \mid a \mid b$

c) $S \to C\perp$
$C \to A1 \mid B1 \mid 1$
$A \to A1 \mid C0 \mid 0$
$B \to C0 \mid 0$

d) $S \to A\perp$
$A \to Bb \mid a$
$B \to Bb \mid b$

e) $S \to B0 \mid C0$
$B \to B0 \mid 0$
$C \to C1 \mid A1$
$A \to 0$

f) $S \to Bb \mid Cc$
$B \to Bb \mid Ab$
$C \to Cc \mid Ab$
$A \to a$

g) $S \to S1 \mid A0$
$A \to B1 \mid C1$
$B \to A0$
$C \to C0 \mid 0$

h) $S \to Sa \mid Cc \mid a$
$C \to Bb$
$B \to Sa \mid a$

i) $S \to C\perp$
$C \to A1 \mid B1 \mid 1$

j) $S \to A\perp$
$A \to Bb \mid a$

$A \to A1 \mid C0 \mid 0$
$B \to C0 \mid 0$

$B \to Bb \mid b$

k) $S \to C\perp$
$B \to B1 \mid 0 \mid D0$
$C \to B1 \mid C1$
$D \to D0 \mid 0$

l) $S \to C\perp$
$C \to B1$
$B \to 0 \mid D0$
$D \to B1$

m) $S \to A0$
$A \to A0 \mid S1 \mid 0$

n) $S \to B0 \mid 0$
$B \to B0 \mid C1 \mid 0 \mid 1$
$C \to B0$

o) $S \to A0 \mid A1 \mid B1 \mid 0 \mid 1$
$A \to A1 \mid B1 \mid 1$
$B \to A0$

p) $S \to S0 \mid A1 \mid 0 \mid 1$
$A \to A1 \mid B0 \mid 0 \mid 1$
$B \to A0$

r) $S \to Sb \mid Aa \mid a \mid b$
$A \to Aa \mid Sb \mid a$

8. Berilgan o'ng chiziqli grammatika uchun ekvivalent DCHA tuzilsin, uning uchun HD va aniqlovchini dasturi tuzilsin, kaysi tilni anglatishi aniqlansin.

a) $S \to aA \mid bB$
$A \to aA \mid bB \mid \perp$
$B \to aC$
$C \to aC \mid \perp$

b) $S \to aA \mid bS \mid a$
$A \to aA \mid bB$
$B \to bS \mid aC$
$C \to aC \mid bC \mid \perp$

c) $S \to 1A$
$A \to 0S \mid 1B$
$B \to 0C$
$C \to 0C \mid 1D \mid \perp$
$D \to 0D \mid \perp$

d) $S \to \varepsilon \mid 1S \mid 0A$
$A \to 1S \mid 0B$
$B \to 0B \mid 1B \mid \perp$

e) $S \to aA \mid bA$
$A \to aB \mid bB$
$B \to aB \mid bB \mid a \mid b$

f) $S \to aB$
$B \to aB \mid aC \mid bB \mid bC$
$C \to \perp$

g) $S \to 0A \mid 1B$
$A \to 0S \mid 0$
$B \to 1S \mid 1$

h) $S \to +A \mid -A \mid d \mid dC$
$A \to dC \mid d$
$C \to dC \mid d \mid .D$
$D \to dD \mid d$

i) $S \to aA \mid \perp$
$A \to bB \mid bS \mid c$
$B \to bC$
$C \to bA$

j) $S \to 0A \mid 0 \mid 1B$
$B \to 1S \mid 1$
$C \to 0A \mid 0$

k) $S \to 1C \mid 0D$
$C \to 0D \mid 0S \mid 1$
$D \to 1C \mid 1S \mid 0$

l) $S \to aA \mid bB \mid aC$
$A \to bA \mid bB \mid c$
$B \to aA \mid cC \mid b$
$C \to bB \mid bC \mid a.$

9. Ikkilik tizimga asoslangan xaqiqiy ishorali sonlar grammatikasini tuzib uning CHA tuzing xamda holat diagrammasini va dasturini yozing.

10. Sakkizlik tizimga asoslangan xaqiqiy ishorali sonlar grammatikasini tuzib uning CHA tuzing xamda holat diagrammasini va dasturini yozing.

11. O'nlik tizimga asoslangan xaqiqiy ishorali sonlar grammatikasini tuzib uning CHA tuzing xamda holat diagrammasini va dasturini yozing.

12. O'noltilik tizimga asoslangan xaqiqiy ishorali sonlar grammatikasini tuzib uning CHA tuzing xamda holat diagrammasini va dasturini yozing.

13. Ikkilik tizimga asoslangan butun sonlar hamda ko'shish va ayrish amallardan tuzilgan ifodalarni grammatikasini tuzib uning CHAni tuzing xamda holat diagrammasini va dasturini yozing.

14. Sakkizlik tizimga asoslangan butun sonlar hamda ko'shish va ayrish amallardan tuzilgan ifodalarni grammatikasini tuzib uning CHAni tuzing xamda holat diagrammasini va dasturini yozing.

15. O'nlik tizimga asoslangan butun sonlar hamda ko'shish va ayrish amallardan tuzilgan ifodalarni grammatikasini tuzib uning CHAni tuzing xamda holat diagrammasini va dasturini yozing.

16. O'noltilik tizimga asoslangan butun sonlar hamda ko'shish va ayrish amallardan tuzilgan ifodalarni grammatikasini tuzib uning CHAni tuzing xamda holat diagrammasini va dasturini yozing.

17. Determinantlangan chekli o'zgartirgich quring:

a) Berilgan ishorasiz haqiqiy fiksirlangan sonlar(son nuqta bilan boshlanmaydi va tugallanmaydi) ketma-ketligini va kasr qismi olib tashlangan butun sonlarga qayta o'zgartiring. Sonlar vergul bilan ajratiladi. Ohirgi belgi #

b) Berilgan haqiqiy va butun sonlar ketma-ketligini 1 va 0 ketma-ketligiga qayta o'zgartiring va bunda butun son uchun 0, haqiqiy son uchun 1 mos keladi. Sonlar vergul bilan ajratiladi. Ketma-ketlik # belgisi bilan tugaydi.

c) Berilgan ishorasiz butun sonlar ketma-ketligini 1^n ketma-ketligiga qayta o'zgartiring, bunda n – juft sonlar soni. Sonlar vergul bilan ajratiladi. Ketma-ketlik # belgisi bilan tugaydi.

d) Berilgan ishorasiz butun sonlar ketma-ketligini 1^n ketma-ketligiga qayta o'zgartiring, bunda n – tog' sonlar soni. Sonlar vergul bilan ajratiladi. Ketma-ketlik # belgisi bilan tugaydi.

e) Berilgan ishorasiz butun sonlar ketma-ketligini 1^n ketma-ketligiga qayta o'zgartiring, bunda n – 5 ga bo'linadigan sonlar soni. Sonlar vergul bilan ajratiladi. Ketma-ketlik # belgisi bilan tugaydi.

f) Berilgan ishorasiz butun sonlar ketma-ketligini 1^n ketma-ketligiga qayta o'zgartiring, bunda n – 10 ga bo'linadigan sonlar soni. Sonlar vergul bilan ajratiladi. Ketma-ketlik # belgisi bilan tugaydi.

g) a^n zanjirlar ketma-ketligi vergul bilan ajratilgan va nuqta bilan tugagan, a= "!" , bu ketma-ketlikni 1^m qayta o'zgartiring, bunda m uzunligi tog' son bo'lgan zanjirlar soni.

h) a^n zanjirlar ketma-ketligi vergul bilan ajratilgan va nuqta bilan tugalgan, a= "!" , bu ketma-ketlikni 1^m qayta o'zgartiring, bunda m uzunligi juft son bo'lgan zanjirlar soni.

22-BOB.KONTEKSTDAN ERKIN TILLAR VA GRAMMATIKALAR

Tayanch iboralar: *keltirilgan grammatika,erishib bo'lmaydigan belgilar, qatnamaydigan belgilar, bo'sh qoidalar, zajirli qoidalar, taqroriy keltirib chiqarish, Xomskiy normal shakli, Greybax normal shakli, chap faktorizatsiya chap rekursiyasiz grammatika.*

Kontekstdan erkin grammatikada (QEG) $G(VT,VN,P,S)$ qoidalar quyidagicha: $A\to\beta$, bunda $A\in VN$, $\beta\in(VT\cup VN)^*$. Kontekstdan erkin grammatikaga asoslangan tilla, kontekstdan erkin tillar deb nomlanadi.

Bunday tillar gapdagi jumla kontekstga bog'liq emasdir. Kontekstdan erkin tillar, dasturlash tillarni sintaksisini tasvirlaydi. KEG uchun umumiy holda aniqlovchini tuzish murakkabdir. Shuning uchun KEG qoidalarini sodda aniqlovchisi mavjud bo'lgan ko'rinishga qayta o'zgartirish mumkin.

Qayta o'zgartirishlarni ikki guruhga ajratish mumkin:

Birinchi guruxga grammatikadagi ba'zi bir qoidalar va belgilarni olib tashlashlar kiradi (grammatikani soddalashtirish).

Ikkinchi guruxga grammatikadagi qoidalarning tuzilishini biror bir qolipga keltirishlar kiradi. Bu holda grammatikaga yangi qoidalar va noterminal belgilari qo'shilishi mumkin.

Bu qayta o'zgartirishlar natijasida hosil bo'lgan grammatika dastlabki sifatida keluvchi grammatikaga ekvivalent bo'lishi kerak.

Keltirilgan grammatika

Keltirilgan grammatika– bu erishib bo'lmas belgilar, qatnashmaydigan belgilar va bo'sh qoidalari bo'lmagan kontekstdan erkin grammatikadir. Bunday grammatika - kanonik grammatika deb ham nomlanadi.

Ihtiyoriy KEGni kanonik grammatikaga qayta o'zgartirish uchun quyida keltirilgan ketma-ketlikdagi amallarni bajarish kerak:

- ***barcha qatnashmaydigan belgilarni olib tashlash;***
- ***barcha erishib bo'lmaydigan belgilarni olib tashlash;***
- ***bo'sh qoidalarni olib tashlash;***
- ***zanjirli qoidalarni olib tashlash.***

Yerishib bo'lmaydigan belgilarni olib tashlash

Agar terminal yoki noterminal belgi grammatikaning xech qaysi sentensial shaklida uchramasa, bunday belgi erishib bo'lmaydigan belgi deb hisoblanadi.

Albatta bunday belgilarni olib tashlash uchun barcha sentensial shakllarni ko'rib chiqish shart emas. Buning uchun erishib bo'lmaydigan belgilarni olib tashlovchi maxsus algoritmni qo'llash yetarlikdir. Bu algoritm erishib bo'ladigan belgilar to'plamini qurish asosida tuziladi.

Dastlab bu to'plamga boshlang'ich belgi kiradi va keyin qoidalar asosida qolgan belgilar qo'shiladi. Natijada shu to'plamga kirmagan belgilar erishib bo'lmaydigan belgilarni tashkil qiladi va grammatikadan olib tashlanadi. Undan tashqari aynan shu belgilar qatnashgan qoidalar ham olib tashlanadi.

Yerishib bo'lmaydigan belgilarni olib tashlash algoritmi quyidagicha:

Kirish:KEG$G(VT,VN,P,S)$;

Chiqish: Yerishib bo'lmaydigan belgilar qatnashmagan KEG$G'(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritmi:

1-qadam: $V_0=\{S\}; i=1;$

2-qadam: $V_i=V_{i-1}\cup\{x|x\in VT\cup VN, A\to\alpha x\beta\in P, A\in V_{i-1}, \alpha,\beta\in(VT\cup VN)^*\}$

3-qadam: Agar $V_i\neq V_{i-1}$, unda $i=i+1$ va 2-qadamga o'tiladi, aks holda 4-qadamga o'tiladi

4-qadam: $VN'=V_i\cap VN$; $VT'=V_i\cap VT$; P' qoidalar to'plamiga, faqat V_i belgilaridan tuzilgan R to'plamning qoidalari kiradi . $S'=S$.

Qatnashmaydigan belgilarni olib tashlash

Agar noterminal $A\in VN$ belgidan xech qanday sentensiya keltirib chiqarilmasa, ya'ni $\{\alpha|A\Rightarrow^*\alpha,\ \alpha\in VT^*\}=\varnothing$ bo'lsa, bunday belgi qatnashmaydigan belgi deb hisoblanadi.

Agar noterminal belgi qatnashgan barcha qoidalarda chap va o'ng tarafida birdaniga qatnashsa, bunday belgi ham qatnashmaydigan belgi bo'ladi.

Quyida keltirilgan algoritm qatnashuvchi noterminal belgilar to'plamini qurish asosida tuzilgan. Dastlab bu to'plam bo'sh deb hisoblanadi, va keyin qoidalar asosida qolgan belgilar qo'shiladi. Natijada shu to'plamga kirmagan belgilar, qatnashmaydigan belgilarni tashkil qiladi va grammatikadan olib tashlanadi. Undan tashqari shu belgilar qatnashgan qoidalar ham olib tashlanadi.

Qatnashmaydigan belgilarni olib tashlash algoritmi quyidagicha:

Kirish:KEG $G(VT,VN,P,S)$;

Chiqish: Qatnamaydigan belgilar kirmagan KEG $G'(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

1-qadam: $N_0=\varnothing; i=1;$

2-qadam: $N_i=N_{i-1}\cup\{A|A\in VN, A\to\alpha\in P, \alpha\in(N_{i-1}\cup VT)^*\}$

3-qadam: Agar $N_i\neq N_{i-1}$, unda $i=i+1$ va 2 qadamga o'tiladi, aks holda 4-qadamga o'tiladi

4-qadam: $VN'=N_i$; $VT'=VT$; P' qoidalar to'plamiga, faqat to'plamiga kirgan belgilaridan tuzilgan R to'plamning qoidalari kiradi . $S'=S$.

Yerishib bo'lmaydigan va katnashmaydigan, ya`ni befoyda bo'lgan belgilarni olib tashlashga doir misol.

KEG berilsin: $G(\{a,b,c\},\{A,B,C,D,E,F,G,S\},P,S)$ bu yerda

P:

$S\to aAB \mid E$

$A\to aA \mid bB$

$B\to ASb \mid b$

$C\to A \mid bA \mid cC \mid aE$

$E\to cE \mid aE \mid Yeb \mid ED \mid FG$

$D\to a \mid c \mid Fb$

$F\to BC \mid EC \mid AC$

$G\to Ga \mid Gb$

I.Avval qatnashmaydigan belgilarni olib tashlaymiz:

1. $N_0=\varnothing$ $i=1$
2. $N_1=\{B,D\}$ $N_1\neq N_0$ $i=2$
3. $N_2=\{B,D,A\}$ $N_2\neq N_1$ $i=3$

4. $N_3=\{B,D,A,S,C\}$ $N_3\neq N_2$ $i=4$
5. $N_4=\{B,D,A,S,C,F\}$ $N_4\neq N_3$ $i=5$
6. $N_5=\{B,D,A,S,C,F\}$ $N_5=N_4$

E va G belgilar qatnashmaydigan belgilar.

7. $VN'=\{A,B,C,D,F,S\}$ $VT'=\{a,b,c\}$ bu yerada

P':

$S\to aAB$

$A\to aA \mid bB$

$B\to ACb \mid b$

$C\to A \mid bA \mid cC$

$D\to a \mid c \mid Fb$

$F\to BC \mid AC$

II. Yendi erishib bo'lmaydigan belgilarni olib tashlaymiz:

1. $V_0=\{S\}$; $i=1$
2. $V_1=\{S,a,A,B\}$ $V_1\neq V_0$ $i=2$
3. $V_2=\{S,a,A,B,b,C\}$ $V_2\neq V_1$ $i=3$
4. $V_3=\{S,a,A,B,b,C,c\}$ $V_3\neq V_2$ $i=4$
5. $V_4=\{S,a,A,B,b,C,c\}$ $V_4=V_3$

D va F belgilar erishib bo'lmaydigan belgilar.

6. $VN'=\{A,B,C,S\}$, $VT'=\{a,b,c\}$

$S'=S$, va G' grammatikani P'qoidalar to'plami quyidagicha bo'ladi:

$S\to aAB$

$A\to aA \mid bB$

$B\to ACb \mid b$

$C\to A \mid bA \mid cC$

Bo'sh qoidalarni olib tashlash

Bo'sh qoidalar (ε-qoidalar) deb $A\to\varepsilon$ shakldagi barcha qoidalar nomlanadi.

Kanonik $G(VT,VN,P,S)$ grammatikada biror bir bo'sh qoida bo'lmasligi kerak, faqatgina bitta $S\to\varepsilon$ bo'sh qoida bo'lishi mumkin. Bu holda S qolgan qoidalarni o'ng tarafida uchramasligi kerak. Shuni eslatib

o'tish kerakki, yuqorida aytilgan holat faqat tilda bo'sh zanjir bo'lgandagina mavjud bo'ladi.

Quyida keltirilgan algoritm bo'sh zanjir keltirib chiqaradigan noterminal belgilar to'plamini qurish asosida tuzilgan. Boshida bu to'plam bo'sh qoidalardagi chap tarafiga kirgan noterminallardan iborat bo'ladi.

Bo'sh qoidalarni olib tashlash algoritmi.

Kirish:KEG$G(VT,VN,P,S)$:

Chiqish: Bo'sh qoidalar bo'lmagan KEG$G(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

1-qadam: $W_0=\{A \mid A\to\varepsilon\in P\}; i=1;$

2-qadam: $W_i= W_{i-1}\cup\{A \mid A\to\alpha\in P,\ \alpha\in W_{i-1}^{*}\};$

3-qadam: Agar $W_i\neq W_{i-1}$, unda $i=i+1$ va 2 qadamga o'tiladi, aks holda 4-qadamga o'tiladi;

4-qadam: $VN'=VN;\ VT'=VT;$ P' qoidalar to'plamiga, $A\to\varepsilon$ kabi qoidalardan tashqari Pdagi barcha qoidalar ko'chiriladi.

5-qadam: $A\to\alpha$ qoidada α zanjirga W_i to'plamdan biror bir belgi kirsa, α zanjirdan W_i dagi belgilardan tuzilgan barcha kombinatsiyalarni ketma ket o'chirish orqali hosil bo'lgan barcha $A\to\alpha'$ kabi qoidalar P'ga qo'shiladi.

6-qadam: Agar $S\in W_i$ unda $\varepsilon\in L(G)$ va VN'ga yangi boshlang'ich belgi S' qo'shiladi va P' ga $S'\to\varepsilon \mid S$ qoidalar qo'shiladi, aks holda $S'=S$.

Ko'pincha bu algoritm qoidalar sonini ko'paytirishiga olib keladi, lekin aniqlovchi qurilma soddalashadi.

Bo'sh qoidalarni olib tashlashga doir misol.

KEG berilsin: $G(\{a,b,c\},\{A,B,C,S\},P,S\}$ bu yerda

P:

$S\to AaB \mid aB \mid cC$

$A\to AB \mid a \mid b \mid B$

$B\to Ba \mid \varepsilon$

$C\to AB \mid c$

I. Avval bo'sh zanjir keltirib chiqaradigan noterminal belgilar to'plamini quramiz:

1. $W_0=\{B\}$, $i=1$

2. $W_1=\{B,A\}$ $W_1\neq W_0$ $i=2$
3. $W_2=\{B,A,C\}$ $W_2\neq W_1$ $i=3$
4. $W_3=\{B,A,C\}$ $W_3\neq W_2$
5. $VN'=\{A,B,C,S\}$ $VT'=\{a,b,c\}$ va P'ga $B\to\varepsilon$, qoidadan tashqari barcha qoidalarni qo'shamiz.
6. $S\to AaB \mid aB \mid cC$ qoidalardan A, B S va ularni kombinatsiyalarini olib tashlaymiz. Natijada yangi qoidalar hosil bo'ladi. $S\to Aa|aB|a|a|c$, nusxa muqobil qoidalarni olib tashlab, P'ga qo'shamiz: $S\to AaB|aB|cC|Aa|a|c$.

$A\to AB|a|b|B$ qoidalardan A va B qkombinatsiyalarini olib tashlaymiz. Yangi qoidalar hosil bo'ladi $A\to A|B$ ularni P'ga qo'shishni xojati yo'q, chunki $A\to B$ qoidasi bor, $A\to A$ qoidani ma'nosi yo'q.

$B\to Ba$ qoidadan Vni o'chiramiz va P'ga $B\to a$ qoida qo'shamiz natijada $B\to Ba|a$.

$C\to AB|c$ qoidalardan A va B kombinatsiyalarini o'chiramiz va yangi $C\to A|B$ qoidalarni P' qo'shamiz: $C\to AB|A|B|c$.

7. $S'=S$ chunki $S\notin W_3$

Natijada quyidagi grammatika hosil bo'ladi:

$G(\{a,b,c\},\{A,B,C,S\},P',S\}$ bunda

P':

$S\to AaB|aB|cC|Aa|a|c$

$A\to AB|a|b|B$

$B\to Ba|a$

$C\to AB|A|B|c$

Zanjir qoidalarni olib tashlash.

Grammatikada $A\Rightarrow^{*}A$, $A\in VN$ ko'rinishdagi keltirib chiqarishlar, taqroriy keltirib chiqarish(tsikllar) deb nomlanadi.

Bunday keltirib chiqarishlarni xech qanday foydasi yo'q va bunday sikllardan qutilish kerak.

KEGda $A\to B$, $A,B\in VN$ kabi zanjir qoidalar qatnashsa, sikllar

bo‘lishi mumkin. Sikllarni yo‘qotish uchun grammatikadan zanjir qoidalarni olib tashlash yetarlikdir.

Zanjir qoidalarni olib tashlashda har bir $X \in VN$ noterminal belgi uchun zanjir qoidalarga tegishli maxsus N^x noterminallar to‘plami tuziladi va shu to‘plamlar orqali qoidalar qayta ishlanib chiqiladi. Shu sababli zanjir qoidalarini olib tashlash algoritmni grammatikadagi barcha noterminal belgilari uchun qo‘llash kerak bo‘ladi. Barcha noterminallar soni n bo‘lgan holda, zanjir qoidalarni olib tashlash algoritmni keltiramiz.

Kirish:KEG$G(VT,VN,P,S)$;

Chiqish:Zanjir qoidalar bo‘lmagan KEG$G(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

1. $j=1$; Birinchi noterminal belgi olinadi va u X bo‘lsin.
2. $N^X_0=\{X\}$, $i=1$.
3. $N^X_i = N^X_{i-1} \cup \{B:(A\to B)\in P, A\in N^X_{i-1}\}$.
4. Agar $N^X_i = N^X_{i-1}$, unda $i=i+1$ va 3 qadamga o‘tiladi, aks holda $N^X = N^X_i - \{X\}$,
5. $j=j+1$, agar $j \le n$ bo‘lsa, joriy noterminal belgini olib 2-qadamga o‘tiladi.
6. $VN'=VN, VT'=VT$, P' to‘plamiga $A\to B$ qoidalardan tashqari Pdan barcha qoidalar ko‘chiriladi. $S'=S$.
7. Xar bir $A\to\alpha\in P'$ qoidalar uchun , $A\in N^B$ga kirsa va $A\neq B$, unda P'ga $B\to\alpha$ qoidani qo‘shiladi.

Bu algoritm qoidalar sonini qo‘paytirishga olib keladi, lekin aniqlovchi qurilma soddalashadi.

Zanjir qoidalarni olib tashlashga doir misol.

KEG berilsin:

$G(\{a,b,c\},\{A,B,C,S\},P,S\}$

P:

$S \to AaB \mid aB \mid cC \mid Aa \mid a \mid c$

$A \to AB \mid a \mid b \mid B$

$B \to Ba \mid a$

$C \to AB \mid A \mid B \mid c$

I. Avval zanjir belgilar to‘plamini tuzamiz:

1. $N^S_0=\{S\}$, $i=1$
2. $N^S_1=\{S\}$, $N^S_1 = N^S_0, N^S = \varnothing$
3. $N^A_0=\{A\}$, $i=1$
4. $N^A_1=\{A,B\}$, $N^A_i \neq N^A_0$, $i=2$
5. $N^A_2=\{A,B\}$, $N^A_2 = N^A_1$ $N^A=\{B\}$
6. $N^B_0=\{B\}$, $i=1$
7. $N^B_1=\{B\}$, $N^B_1 = N^B_0$ $N^B=\varnothing$
8. $N^C_0=\{C\}$, $i=1$
9. $N^C_1=\{C,A,B\}$, $N^C_1 \neq N^C_0$ $i=2$
10. $N^C_2=\{C,A,B\}$, $N^C_2 = N^C_1$ $N^C=\{A,B\}$
11. Natijada: $N^S=\varnothing$, $N^A=\{B\}$, $N^B=\varnothing$, $N^C=\{A,B\}$, $VN'=\{A,B,C,S\}$, $VT'=\{a,b,c\}$, P' ga $A\to B$, $S\to A$, $S\to B$qoidalardan tashqari Pdan barcha qoidalarni ko‘chiramiz,$S'=S$

12. P' dagi barcha qoidalarni ko‘rib chiqamiz, bizni faqat A va V noterminallar uchun qoidalar qiziqtiradi, chunki $N^A=\{B\}$, $N^C=\{A,B\}$,

a) $A\to AB|a|b$qoidalar uchun $S\to AB|a|b$ yangi qoidalar qo‘shamiz chunki $A\in N^S$ga , bulardan $S \to AB$qoida oldindan mavjud bo‘lgani uchun uni qo‘shmaymiz.

b) $B \to Ba|a$qoidalar uchun $A\to Ba|a$ va $S\to Ba|a$qoidalar qo‘shamiz, chunki $V\in N^A$va $V\in N^S$, bulardan $A\to a$ va S$\to$aqoidalar P’da mavjud.

Natijada quyidagi grammatika hosil bo‘ladi:

$G'(\{a,b,c\},\{A,B,C,S\},P',S\}$bu yerda

P':

$S \to AaB \mid aB \mid cC \mid Aa \mid a \mid c$

$A \to AB \mid a \mid b \mid Ba$

$B \to Ba \mid a$

$C \to AB \mid a \mid Ba \mid c \mid b$.

Ko‘rinib turibdiki, bu grammatikada zanjir qoidalari yo‘q.

Dasturlash tillariga yaqin bo‘lgan arifmetiki fodalar grammatikasini ko‘ramiz:

$G(\{+,-,/,*, a,b\}, \{S,T,E\}, P, S)$ bunda

P:

bo'lishi mumkin. Sikllarni yo'qotish uchun grammatikadan zanjir qoidalarni olib tashlash yetarlikdir.

Zanjir qoidalarni olib tashlashda har bir $X \in VN$ noterminal belgi uchun zanjir qoidalarga tegishli maxsus N^x noterminallar to'plami tuziladi va shu to'plamlar orqali qoidalar qayta ishlanib chiqiladi. Shu sababli zanjir qoidalarini olib tashlash algoritmni grammatikadagi barcha noterminal belgilari uchun qo'llash kerak bo'ladi. Barcha noterminallar soni n bo'lgan holda, zanjir qoidalarni olib tashlash algoritmni keltiramiz.

Kirish:KEG$G(VT,VN,P,S)$;

Chiqish:Zanjir qoidalar bo'lmagan KEG$G(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

1. $j=1$; Birinchi noterminal belgi olinadi va u X bo'lsin.
2. $N^X_0=\{X\}$, $i=1$.
3. $N^X_i= N^X_{i-1} \cup \{B:(A\to B)\in P, A\in N^X_{i-1}\}$.
4. Agar $N^X_i= N^X_{i-1}$,unda $i=i+1$va 3 qadamga o'tiladi, aks holda $N^X= N^X_i-\{X\}$,
5. $j=j+1$, agar $j\le n$ bo'lsa, joriy noterminal belgini olib 2-qadamga o'tiladi.
6. $VN'=VN, VT'=VT$, P' to'plamiga $A\to B$ qoidalardan tashqari Pdan barcha qoidalar ko'chiriladi. $S'=S$.
7. Xar bir $A\to\alpha\in P'$ qoidalar uchun , $A\in N^B$ga kirsa va $A\neq B$, unda P'ga $B\to\alpha$ qoidani qo'shiladi.

Bu algoritm qoidalar sonini qo'paytirishga olib keladi, lekin aniqlovchi qurilma soddalashadi.

Zanjir qoidalarni olib tashlashga doir misol.

KEG berilsin:

$G(\{a,b,c\},\{A,B,C,S\},P,S)$

P:

$S \to AaB \mid aB \mid cC \mid Aa \mid a \mid c$

$A \to AB \mid a \mid b \mid B$

$B \to Ba \mid a$

$C \to AB \mid A \mid B \mid c$

I. Avval zanjir belgilar to'plamini tuzamiz:

1. $N^S_0=\{S\}$, $i=1$
2. $N^S_1=\{S\}$, $N^S_1= N^S_0, N^S=\varnothing$
3. $N^A_0=\{A\}$, $i=1$
4. $N^A_1=\{A,B\}$, $N^A_1\neq N^A_0, i=2$
5. $N^A_2=\{A,B\}$, $N^A_2= N^A_1$ $N^A=\{B\}$
6. $N^B_0=\{B\}$, $i=1$
7. $N^B_1=\{B\}$, $N^B_1= N^B_0$ $N^B=\varnothing$
8. $N^C_0=\{C\}$, $i=1$
9. $N^C_1=\{C,A,B\}$, $N^C_1\neq N^C_0$ $i=2$
10. $N^C_2=\{C,A,B\}$, $N^C_2= N^C_1$ $N^C=\{A,B\}$
11. Natijada: $N^S=\varnothing$, $N^A=\{B\}$, $N^B=\varnothing$, $N^C=\{A,B\}$, $VN'=\{A,B,C,S\}$, $VT'=\{a,b,c\}$, P' ga $A\to B$, $S\to A$, $S\to B$qoidalardan tashqari Pdan barcha qoidalarni ko'chiramiz,$S'=S$
12. P' dagi barcha qoidalarni ko'rib chiqamiz, bizni faqat A va V noterminallar uchun qoidalar qiziqtiradi, chunki $N^A=\{B\}$, $N^C=\{A,B\}$,

a) $A\to AB|a|b$qoidalar uchun $S\to AB|a|b$ yangi qoidalar qo'shamiz chunki $A\in N^S$ga , bulardan $S \to AB$qoida oldindan mavjud bo'lgani uchun uni qo'shmaymiz.

b) $B \to Ba|a$qoidalar uchun $A\to Ba|a$ va $S\to Ba|a$qoidalar qo'shamiz, chunki $V\in N^A$va $V\in N^S$, bulardan $A\to a$ va S→aqoidalar P'da mavjud.

Natijada quyidagi grammatika hosil bo'ladi:

$G'(\{a,b,c\},\{A,B,C,S\},P',S)$bu yerda

P':

$S \to AaB \mid aB \mid cC \mid Aa \mid a \mid c$

$A \to AB \mid a \mid b \mid Ba$

$B \to Ba \mid a$

$C \to AB \mid a \mid Ba \mid c \mid b.$

Ko'rinib turibdiki, bu grammatikada zanjir qoidalari yo'q.

Dasturlash tillariga yaqin bo'lgan arifmetiki fodalar grammatikasini ko'ramiz:

$G(\{+,-,/,*, a,b\}, \{S,T,E\}, P, S)$ bunda

P:

$S \rightarrow S+T \mid S-T \mid T$

$T \rightarrow T*E \mid T/E \mid E$

$E \rightarrow (S) \mid a \mid b.$

Bu grammatikada ikkita qoida $S \rightarrow T$ va $T \rightarrow E$ zanjir qoidalardir, ularni olib tashlaymiz.

I. Birinchi navbatda zanjir qoidalarga tegishli noterminallar to'plamini tuzamiz:

1. $N^S_0 = \{S\}, \quad i=1$
2. $N^S_1 = \{S,T\}, \quad N^S_1 \neq N^S_0 \quad i=2$
3. $N^S_2 = \{S,T,E\}, \quad N^S_2 \neq N^S_1 \quad i=3$
4. $N^S_3 = \{S,T,E\}, \quad N^S_3 = N^S_2, \quad N^S = \{T,E\}$
5. $N^T_0 = \{T\}, \quad i=1$
6. $N^T_1 = \{T,E\}, \quad N^T_1 \neq N^T_0 \quad i=2$
7. $N^T_2 = \{T,E\}, \quad N^T_2 = N^T_1, \quad N^T = \{E\}$
8. $N^E_0 = \{E\}, \quad i=1$
9. $N^E_1 = \{E\}, \quad N^E_1 = N^E_0, \quad N^E = \varnothing$

Natijada: $N^S = \{T,E\}$, $N^T = \{E\}$, $N^E = \varnothing$, $VN' = \{T,E,S\}$, $VT' = \{+,-,*,/,a,b\}$, P'ga $S \rightarrow T$, $T \rightarrow E$ qoidalardan tashqari Pdan barcha qoidalarni ko'chiramiz, $S' = S$

10. P' dagi barcha qoidalarni ko'rib chiqamiz.. Bizni faqat T va E noterminallar uchun qoidalar qiziqtiradi, chunki $N^S = \{T,E\}$, $N^T = \{E\}$,

a) $T \rightarrow T*E \mid T/E$ qoidalar uchun $S \rightarrow T*E \mid T/E$ yangi qoidalar qo'shamiz chunki $T \in N^S$.

b) $E \rightarrow (S) \mid a \mid b$qoidalar uchun $S \rightarrow (S) \mid a \mid b$ va $T \rightarrow (S) \mid a \mid b$qoidalar qo'shamiz, chunki $E \in N^S$ va $E \in N^T$.

Natijada quyidagi grammatika hosil bo'ladi:

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

P':

$S \rightarrow S+T \mid S-T \mid T*E \mid T/E \mid (S) \mid a \mid b$

$T \rightarrow T*E \mid T/E \mid (S) \mid a \mid b$

$E \rightarrow (S) \mid a \mid b$

Bu misolda natijaviy grammatikada qoidalar soni ancha ko'paydi. Shuni aytib o'tish kerakki, yuqorida ko'rsatilgan dastlabki bo'lmish

grammatikadagi zanjir qoidalar takroriy keltirib chiqarishga olib kelmaydi. Shu sababli ularni olib tashlamasa ham bo'ladi.

Xomskiy normal shakldagi grammatika

Xomskiy normal shakli (XNSH) bu keltiririlgan shakllarni bir turi. Har qanday KEGni XNShga kayta o'zgartirish mumkin. XNShga qayta o'zgartirishdan oldin grammatikani keltirilgan shakliga qayta o'zgartirish kerak.

Xomskiy normal shaklini ta'rifi:

Agar KEGni qoidalarida faqat quyidagi qoidalar mavjud bo'lsa:

1. $A \rightarrow BS, A,V,S \in VN,$

2. $A \rightarrow a$, $a \in VT,$

3. $S \rightarrow \varepsilon$, agar $\varepsilon \in L(G)$ va S boshqa qoidalarni o'ng tarafida uchramasligi kerak.

Bunday grammatikalar Xomskiy normal shakli deb nomlanadi.

Qoidalarni boshqa xhech qanday shakli XNShda uchramasligi kerak.

Xomskiy normal shakliga grammatikani qayta o'zgartirish algoritmi.

Kirish:KEG$G(VT,VN,P,S)$:

Chiqish:XNSH grammatikaKEG$G(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

Birinchi qadamda dastlabki grammatikani keltirilgan grammatikaga qayta o'zgartirish kerak, faraz qilaylik dastlabki grammatika keltirilgan shaklda.

Algoritm ishlashdan oldin VNdagi barcha noterminal belgilar VN'ga va terminal belgilar VT' o'tkaziladi, ya'ni $VN'=VN$ va $VT'=VT$.

Keltirilgan dastlabki grammatikani barcha qoidalari ko'rib chiqiladi.

1. Agar $A \rightarrow a$ kabi qoidalar uchrasa, ular o'zgartirilmasdan P' ga ko'chiriladi.
2. Agar $A \rightarrow BC$ kabi qoidalar uchrasa, ular o'zgartirilmasdan P' ga ko'chiriladi.
3. Agar $S \rightarrow \varepsilon$ qoida uchrasa, u o'zgartirilmasdan P' ga ko'chiriladi.
4. Agar $A \rightarrow aB$ kabi qoidalar uchrasa, unda P' ga $A \rightarrow <AaB>B$ va $<AaB> \rightarrow a$ qoidalar qo'shiladi va VN' to'plamiga $<AaB>$

$S \to S+T \mid S-T \mid T$

$T \to T*E \mid T/E \mid E$

$E \to (S) \mid a \mid b.$

Bu grammatikada ikkita qoida $S \to T$ va $T \to E$ zanjir qoidalardir, ularni olib tashlaymiz.

I. Birinchi navbatda zanjir qoidalarga tegishli noterminallar to'plamini tuzamiz:

1. $N^S_0 = \{S\}, \quad i=1$
2. $N^S_1 = \{S,T\}, \quad N^S_1 \neq N^S_0 \quad i=2$
3. $N^S_2 = \{S,T,E\}, \quad N^S_2 \neq N^S_1 \quad i=3$
4. $N^S_3 = \{S,T,E\}, \quad N^S_3 = N^S_2 \quad N^S = \{T,E\}$
5. $N^T_0 = \{T\}, \quad i=1$
6. $N^T_1 = \{T,E\}, \quad N^T_1 \neq N^T_0 \quad i=2$
7. $N^T_2 = \{T,E\}, \quad N^T_2 = N^T_1 \quad N^T = \{E\}$
8. $N^E_0 = \{E\}, \quad i=1$
9. $N^E_1 = \{E\}, \quad N^E_1 = N^E_0 \quad N^E = \varnothing$

Natijada: $N^S = \{T,E\}$, $N^T = \{E\}$, $N^E = \varnothing$, $VN' = \{T,E,S\}$, $VT' = \{+,-,*,/,a,b\}$, P'ga $S \to T$, $T \to E$ qoidalardan tashqari Pdan barcha qoidalarni ko'chiramiz, $S' = S$

10. P' dagi barcha qoidalarni ko'rib chiqamiz,. Bizni faqat T va E noterminallar uchun qoidalar qiziqtiradi, chunki $N^S = \{T,E\}$, $N^T = \{E\}$,

a) $T \to T*E \mid T/E$ qoidalar uchun $S \to T*E \mid T/E$ yangi qoidalar qo'shamiz chunki $T \in N^S$.

b) $E \to (S) \mid a \mid b$qoidalar uchun $S \to (S) \mid a \mid b$ va $T \to (S) \mid a \mid b$qoidalar qo'shamiz, chunki $E \in N^S$ va $E \in N^T$.

Natijada quyidagi grammatika hosil bo'ladi:

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

P':

$S \to S+T \mid S-T \mid T*E \mid T/E \mid (S) \mid a \mid b$

$T \to T*E \mid T/E \mid (S) \mid a \mid b$

$E \to (S) \mid a \mid b$

Bu misolda natijaviy grammatikada qoidalar soni ancha ko'paydi. Shuni aytib o'tish kerakki, yuqorida ko'rsatilgan dastlabki bo'lmish

grammatikadagi zanjir qoidalar takroriy keltirib chiqarishga olib kelmaydi. Shu sababli ularni olib tashlamasa ham bo'ladi.

Xomskiy normal shakldagi grammatika

Xomskiy normal shakli (XNSH) bu keltiririlgan shakllarni bir turi. Har qanday KEGni XNShga kayta o'zgartirish mumkin. XNShga qayta o'zgartirishdan oldin grammatikani keltirilgan shakliga qayta o'zgartirish kerak.

Xomskiy normal shaklini ta'rifi:

Agar KEGni qoidalarida faqat quyidagi qoidalar mavjud bo'lsa:

1. $A \to BS$, $A,V,S \in VN$,

2. $A \to a$, $a \in VT$,

3. $S \to \varepsilon$, agar $\varepsilon \in L(G)$ va S boshqa qoidalarni o'ng tarafida uchramasligi kerak.

Bunday grammatikalar Xomskiy normal shakli deb nomlanadi.

Qoidalarni boshqa xhech qanday shakli XNShda uchramasligi kerak.

Xomskiy normal shakliga grammatikani qayta o'zgartirish algoritmi.

Kirish:KEG$G(VT,VN,P,S)$;

Chiqish:XNSH grammatikaKEG$G(VT',VN',P',S')$ va $L(G)=L(G')$.

Algoritm:

Birinchi qadamda dastlabki grammatikani keltirilgan grammatikaga qayta o'zgartirish kerak, faraz qilaylik dastlabki grammatika keltirilgan shaklda.

Algoritm ishlashdan oldin VNdagi barcha noterminal belgilar VN'ga va terminal belgilar VT' o'tkaziladi, ya'ni $VN'=VN$ va $VT'=VT$.

Keltirilgan dastlabki grammatikani barcha qoidalari ko'rib chiqiladi.

1. Agar $A \to a$ kabi qoidalar uchrasa, ular o'zgartirilmasdan P' ga ko'chiriladi.
2. Agar $A \to BC$ kabi qoidalar uchrasa, ular o'zgartirilmasdan P' ga ko'chiriladi.
3. Agar $S \to \varepsilon$ qoida uchrasa, u o'zgartirilmasdan P' ga ko'chiriladi.
4. Agar $A \to aB$ kabi qoidalar uchrasa, unda P' ga $A \to <AaB>B$ va $<AaB> \to a$ qoidalar qo'shiladi va VN' to'plamiga $<AaB>$

noterminal belgisi qo'shiladi.

5. Agar $A \to Ba$ kabi qoidalar uchrasa, unda P' ga $A \to B<ABa>$ va $<ABa> \to a$ qoidalar qo'shiladi va VN' to'plamiga $<ABa>$ noterminal belgisi qo'shiladi.
6. Agar $A \to ab$kabi qoidalar uchrasa, unda P' ga $A \to <Aa><Ab>, <Aa> \to a$ va $<Ab> \to b$ qoidalar qo'shiladi va VN' to'plamiga $<Aa>$ va $<Ab>$ noterminal belgilar qo'shiladi.
7. Agar $A \to X_1, ..., X_k, k>2, A \in VN, X_i \in VT \cup VN$ qoidalar uchrasa, unda P' ga quyidagi qoidalar qo'shiladi:

$A \to <X_1'><X_2..., X_k>$

$<X_2..., X_k> \to <X_2'><X_3..., X_k>$

...

$<X_{k-1}, X_k> \to <X_{k-1}'><X_k'>$

$<X_2..., X_k>, <X_3..., X_k>, ... <X_{k-1}, X_k>$ noterminal belgilar VN' to'plamiga qo'shiladi, undan tashqari, agar $X_i \in VN$, unda $X_i' \equiv X_i$, aks holda (agar $X_i \in VT$) P'ga $<X_i'> \to X_i$ qoida qo'shiladi va VN' to'plamiga $<X_1'>$ - yangi noterminal qo'shiladi.

8. $S'=S$

Xomskiy normal shakliga qayta o'zgartirishga doir misol.

Misol taraqasida $G(\{a,b,c\},\{A,B,C,S, P,S)$ grammatikani ko'ramiz bunda P:

$S \to AaB \mid Aa \mid bc$

$A \to AB \mid a \mid aC$

$B \to Ba \mid b$

$C \to AB \mid c$

Ko'rinib turibdiki, bu grammatika keltirilgan shaklda. Unga ekvivalent XNShdagi $G'(VT', VN', P', S')$ grammatika tuzamiz. $VT'=VT$, $VN'=VN$ deb olamiz.

Yendi har bir qoidani qayta ko'rib chiqamiz.

Birinchi qoidamiz $S \to AaB$ algoritmni 7-chi variantiga to'g'ri keladi, unga ko'ra qoidani quyidagi qoidalarga almashtirib P'ga qo'shamiz:

$S \to <A'><aB>$

$<aB> \to <a'><B'>$

Bunda $A' \equiv A$, $B' \equiv B$ va P'ga $<a'> \to a$ yangi qoida qo'shamiz, shunda P'da quyidagi qoidalar ketma-ketligi hosil bo'ladi:

$S \to A<aB>$

$<aB> \to <a'>B$

$<a'> \to a$

VN' to'plamiga $<aB>$ va $<a'>$ noterminal belgilar qo'shiladi.

Ikkinchi $S \to Aa$ qoidamiz algoritmni 5-chi variantiga to'g'ri keladi,va unga ko'ra ikkita qoida qo'shamiz:

$S \to A<SAa>$

$<SAa> \to a$

Yangi noterminal $<SAa>$ VN' to'plamiga qo'shiladi.

Uchinchi $S \to bc$ qoidamiz algoritmni 6-chi variantiga to'g'ri keladi,va unga ko'ra uchta qoida qo'shamiz:

$S \to <Sb><Sc>$

$<Sb> \to b$

$<Sc> \to c$

Yangi noterminallar $<Sb>$ va $<Sc>$ VN' to'plamiga qo'shiladi.

To'rtinchi $A \to AV$ qoidamiz algoritmni 2-chi variantiga to'g'ri keladi. Qoidani o'zgartirmasdan yangi grammatikaga ko'chiramiz.

Beshinchi $A \to a$ qoidamiz algoritmni 1-chi variantiga to'g'ri keladi. Qoidani o'zgartirmasdan yangi grammatikaga ko'chiramiz.

Oltinchi $A \to aS$ qoidamiz algoritmni 4-chi variantiga to'g'ri keladi, unga ko'ra ikkita qoida qo'shamiz:

$A \to <AaC>C$

$<AaC> \to a$

Yangi noterminal $<AaC>$ VN' to'plamiga qo'shiladi.

Yettinchi $B \to Ba$ qoidamiz algoritmni 5-chi variantiga to'g'ri keladi,va unga ko'ra ikkita qoida qo'shamiz:

$B \to B<BAa>$

$<BAa> \to a$

Yangi noterminal $<BAa>$ VN' to'plamiga qo'shiladi.

Sakkizinchi $V \to b$ qoidamiz algoritmni 1-chi variantiga to'g'ri keladi. Qoidani o'zgartirmasdan yangi grammatikaga ko'chiramiz.

To'qqizinchi $C \to AV$ qoidamiz algoritmni 2-chi variantiga to'g'ri

keladi. Qoidani o'zgartirmasdan yangi grammatikaga ko'chiramiz.

O'ninchi $S \to s$ qoidamiz algoritmni 1-chi variantiga to'g'ri keladi. Qoidani o'zgartirmasdan yangi grammatikaga ko'chiramiz.

Barcha qoidalarni ko'rib chiqdik va yangi grammatikani P' qoidalar va VN' noterminallar to'plamini tuzib bo'ldik.

$S'=S$ yangi grammatikani boshlang'ich belgisi S bo'ladi.

Natijada Xomskiy normal shaklidagi yangi ekvivalent grammatikani hosil qildik:

$G'(\{a,b,c\},\{A,B,C,S,<aB>,<a'>,<SAa>,<Sb>,<Sc>,<AaC>,<BBa>\},P',S)$

Bu yerda

P':

$S \to A<aB> \mid A<SAa> \mid <Sb><Sc>$

$<aB> \to <a'>B$

$<a'> \to a$

$<SAa> \to a$

$<Sb> \to b$

$<Sc> \to c$

$A \to AB \mid a \mid <AaC>C$

$<AaC> \to a$

$B \to B<BBa> \mid b$

$<BBa> \to a$

$C \to AB \mid c$

Yendi noterminal belgilarni ketma-ket lotin xarflarga o'zgartirib chiqsak, quyidagi grammatikaga ega bo'lamiz.

$G'(\{a,b,c\},\{A,B,C,S,D,E,F,G,H,I,J\},P',S)$ bunda

P':

$S \to AD \mid AF \mid GH$

$D \to EB$

$E \to a$

$F \to a$

$G \to b$

$H \to c$

$A \to AB \mid a \mid IC$

$I \to a$

$B \to BJ \mid b$

$J \to a$

$C \to AB \mid c$

Ko'rinib turibdiki, grammatikani Xomskiy normal shakliga qayta o'zgartirish natijasida qoidalar va noterminallar soni ko'paydi, natijada grammatika murakkablanishib ketdi,. Lekin bu grammatika yordamida aniqlovchi qurilmani tuzish ancha yengillashadi.

Ixtiyoriy qoidani olib tashlash algoritmi

Bunday qayta o'zgartirishlar grammatikani biror bir berilgan sinfga keltirish uchun qo'llanadi.

Kirish:$G(VT,VN,P,S)$KEG berilgan, va unda $A \to \alpha B\beta \in P$, $A,B \in VN$, va zanjirlar $\alpha,\beta \in (VN \cup VT)^*$

Chiqish:$G'(VT,VN,P',S)$KEG, $L(G')=L(G)$ va $A \to \alpha B\beta \in P'$.

Algoritm:

1. P'ga $A \to \alpha B\beta$ tashqari barcha qoidalar o'tkaziladi.
2. P to'plamida $B \to \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_k$ qoidalar bo'lsin, va unda P' ga $A \to \alpha\gamma_1\beta \mid \alpha\gamma_2\beta \mid \dots \mid \alpha\gamma_k\beta$ qoidalar qo'shiladi.

Misol. G({a,b, +,-,/, *,(,)},{T,F,S},P,S) grammatika berilgan bo'lsin,va unda P:

$S \to S+T \mid S-T \mid T$

$T \to T*E \mid T/E \mid E$

$E \to (S) \mid a \mid b$

Bu to'plamdan $E \to (S)$ qoidani olib tashlaymiz,va unda P'ga $E \to (S+T) \mid (S-T) \mid (T)$ qoidalar qo'shamiz.

Natijada P' quyidagicha bo'ladi:

$S \to S+T \mid S-T \mid T$

$T \to T*E \mid T/E \mid E$

$E \to (S+T) \mid (S-T) \mid (T) \mid a \mid b$

Chap faktorizatsiyalash algoritmi

Noterminalga tegishli o'ng tarafi bir hil prefiksdan boshlangan qoidalarni olib tashlash algoritmi chap faktorizatsiyalash deb nomlanadi.

Chap faktorizatsiyalash algoritmi quyidagicha.

Kirish: $G(VT,VN,P,S)$ KEG berilgan, va unda bir hil prefiksli qoidalar mavjud.

Chiqish: $G'(VT,VN,P',S)$ KEG, $L(G')=L(G)$ va u chap faktorizatsiyalangan.

Algoritm:

1. Grammatikada A-qoidalar $A\to\alpha\beta_1|\alpha\beta_2|\dots|\alpha\beta_n|\gamma_1|\gamma_2\dots|\gamma_m$ ko'rinishda bo'lib, unda $\alpha\in(VT\cup VN)^*$, $\beta_i\in(VT\cup VN)^*$ barcha $i=1,2\dots n$ uchun; $\gamma_j\in(VT\cup VN)^*$ va γ_j barcha $j=1,2\dots m$ uchun α-prefiksdan boshlanmagan bo'lsin. Bu holda yangi grammatikaga yangi A'noterminal qo'shiladi va yuqoridagi qoida o'rniga

$A\to\alpha A'|\gamma_1|\gamma_2\dots|\gamma_m$

$A'\to\beta_1|\beta_2|\dots\beta_n$

qoidalar qo'shiladi.

2. 1-qadamni bir hil prefiksli qoidalar tugagunicha bajariladi.

Misol, bizga $G(\{a, b\}, \{S, A\}, P, S)$ berilgan bo'lsin, bu yerda P:

$S\to abSa|\ aaAb\ |\ b$

$A\to baAb|\ b.$

Bunda S-qoidalar uchta muqobil qoidadan iborat va ulardan ikkisi bir hil prefiksga ega. Shu sababli S-qoidalarni quidagi qoidalarga almashtiramiz:

$S\to aS'|\ b$

$S'\to bSa|\ aAb.$

A-qoidalarni ham almashtiramiz:

$A\to bA'$

$A'\to aAb|\varepsilon.$

Natijada chap faktorizatsiyalashgan grammatikaga ega bo'lamiz:

$G(\{a, b\}, \{S, S', A, A'\}, P', S)$bu yerda$P'$:

$S\to aS'|\ b$

$S'\to bSa|\ aAb$

$A\to bA'$

$A'\to aAb|\varepsilon.$

Chap rekursiyani olib tashlash

KEGdagi A noterminal uchun $A\Rightarrow\alpha A\beta$ keltirib chiqarish mavjud bo'lsa, noterminal A rekursiv deb nomlanadi. Bunda $\alpha,\beta\in(VN\cup VT)^*$.

Agar $\alpha=\varepsilon$ va $\beta\neq\varepsilon$, unda rekursiya chap hisoblanadi va grammatika chap rekursiyali deb nomlanadi.

Agar $\alpha\neq\varepsilon$ va $\beta=\varepsilon$, unda rekursiya o'ng hisoblanadi va grammatika o'ng rekursiyali deb nomlanadi.

Agar $\alpha=\varepsilon$ va $\beta=\varepsilon$, unda rekursiya zanjirli, agar grammatika keltirilgan bo'lsa unda zanjirli qoidalar bo'lmaydi, shuning uchun biz ularni etiborga olmaymiz.

Grammatikada ham chap, ham o'ng rekursiyalar bo'lishi mumkin.

Agar grammatikada chap rekursiyali qoidalar bo'lmasa, bunday grammatika, chap rekursiyasiz grammatika deb nomlanadi.

Agar grammatikada o'ng rekursiyali qoidalar bo'lmasa, bunday grammatika, o'ng rekursiyasiz grammatika deb nomlanadi.

Ba'zi bir aniqlovchilar chap rekursiyali grammatikalar bilan ishlamaydi, shuning uchun grammatikada chap rekursiyali qoidalarni olib tashlash kerak bo'ladi.

Har qanday KEGni chap rekursiyasiz yoki o'ng rekursiyasiz grammatikaga qayta o'zgartirish mumkin.

Chap rekursiyani olib tashlash algoritmi.

Kirish: Keltirilgan KEG$G(VT,VN,P,S)$

Chiqish: Chaprekursiyasiz ekvivalent KEG$G'(VT',VN',P',S)$

Usuli:

1. $VT'=VT$, grammatikani barcha noterminallarini quyidagicha belgilab VN'ga kiritamiz:$VN'=\{A_1,A_2,\ \dots\ A_n\}$, Pdan barcha qoidalarni yangi noterminal orqali yozilgan holda P'ga ko'chiriladi, $i=1$;
2. G'grammatikada A_i–qoidalarni ko'rib chiqiladi.

Agar A_i-qoidalar quyidagicha bo'lsa $A_i\to A_i\alpha_1|A_i\alpha_2|\dots|A_i\alpha_m|\beta_1|\beta_2|\dots|\beta_p$, bunda $\alpha_l\in(VN'\cup VT')^+$, $1\le l\le m$, $\beta_j\in(VN'\cup VT')^+$, $1\le j\le p$,

va hech qaysi β_j , A_k dan boshlanmagan, $k \le i$. Bu qoidalar o'rniga P' to'plamiga:

$A_i \to \beta_1 | \beta_2 | \ldots | \beta_p | \beta_1 A_i' | \beta_2 A_i' | \ldots | \beta_p A_i'$,

$A_i' \to \alpha_1 | \alpha_2 | \ldots | \alpha_m | \alpha_1 A_i' | \alpha_1 A_i' | \ldots | \alpha_m A_i'$

qoidalar qo'shiladi, A_i' VN' to'plamiga kiritiladi. Yendi A_i–qoidalar terminal belgidan yoki noterminal $A_k$dan boshlanadi, bunda $k>i$.

3. Agar $i<n$ bo'lsa, $i=i+1$ va $j=1$ qiymat berib 4-qadamga o'tiladi, aks holda G' grammatika tuzildi va 6-qadamga o'tiladi.
4. A_j belgi uchun P' qoidalar ko'rib chiqiladi va $A_i \to A_j\alpha$ qoida mavjud bo'lsa va A_j–qoidalar $A_j \to \gamma_1 | \gamma_2 | \ldots | \gamma_q$ ko'rinishda bo'lsa, unda A_i–qoida $A_i \to \gamma_1\alpha \mid \gamma_2\alpha \mid \ldots \mid \gamma_q\alpha$ga almashtiriladi, bunda $\alpha \in (VN' \cup VT')^*$, $\gamma_l \in (VN' \cup VT')^*$, $1 \le l \le q$,.
5. Agar $j<i-1$ bo'lsa, $j=j+1$qiymat berib 4-kadamga, aks holda 2-qadamga o'tiladi.
6. Sga mos kelgan A_1 boshlang'ich belgi bo'ladi.

Misol:

$G(\{a,b, +,-,/, *,(,)\},\{T,F,S\},P,S)$ grammatika berilsin, unda

P:

$S \to S+T \mid S-T \mid T$

$T \to T*E \mid T/E \mid E$

$E \to (S) \mid a \mid b$

Bu chap rekursiyali grammatika,va ekvivalent G' chap rekursiyasiz grammatikani tuzamiz

1-chi qadam. $VN'=\{A_1,A_2,A_3\}$, $i=1$;

ShundaP'qoidalar to'plami quyidagicha yoziladi:

$A_1 \to A_1+A_2 \mid A_3-A_2 \mid A_2$

$A_2 \to A_2*A_3 \mid A_2/A_3 \mid A_3$

$A_3 \to (A_1) \mid a \mid b$

2.1-chi qadam. A_1-qoidalarni $A_1 \to A_1\alpha_1 | A_1\alpha_2 | \beta_1$ko'rinishda yozish mumkin, bunda $\alpha_1=+A_2$, $\alpha_2=-A_2$, $\beta_1=A_2$

P'ga bu qoidalar o'rniga yangi qoidalarni qo'shamiz:

$A_1 \to A_2 \mid A_2A_1'$

$A_1' \to +A_2 \mid -A_2 \mid +A_2A_1' \mid -A_2A_1'$

$VN'=\{A_1,A_2,A_3,A_1'\}$

3.1-chi qadam. $i=i+1=2$; $j=1$; 4-chi qadamga o'tamiz.

4.1-chi qadam. Grammatikada $A_2 \to A_1\alpha$qoidalar bo'lmaganligi uchun xhech qanday amal bajarmaymiz.

5.1-chi qadam. $j=1$; $j=i-1$teng bo'lgan uchun algoritmni 2-qadamiga o'tamiz.

2.2-chi qadam. A_2-qoidalarni $A_2 \to A_2\alpha_1 | A_2\alpha_2 | \beta_1$ko'rinishda yozish mumkin, bunda $\alpha_1=*A_3$, $\alpha_2=/A_3$, $\beta_1=A_3$

P'ga yangi qoidalarni qo'shamiz:

$A_2 \to A_3 \mid A_3A_2'$

$A_2' \to *A_3 \mid /A_3 \mid *A_3A_2' \mid /A_3A_2'$

$VN'=\{A_1,A_1', A_2,A_2',A_3\}$

3.2-chi qadam. $i=i+1=3$; $j=1$; 4-chi qadamga o'tamiz.

4.2-chi qadam. Grammatikada $A_3 \to A_1\alpha$qoidalar bo'lmaganligi uchun xhech qanday amal bajarmaymiz.

5.2-chi qadam. $j<i-1$ bo'lgan uchun, $j=j+1=2$vaalgorimni 4-qadamga o'tamiz.

4.3-chi qadam. Grammatikada $A_3 \to A_2\alpha$qoidalar bo'lmaganligi uchun xhech qanday amal bajarmaymiz.

5.3-chi qadam. $j=i-1$teng bo'lgan uchun algoritmni 2 qadamga o'tamiz.

2.4-chi qadam. A_3-qoidalar o'zgarmaydi.

3.4-chi qadam. $i$uchga teng bo'lgan uchun G' tuzildi 6 qadamga o'tamiz.

6-chi qadam. Boshlang'ich belgisi - A_1.

Algoritm bajarilish natijasida chap rekursiyasiz grammatikani tuzdik:

$G'(\{a,b, +,-,/, *,(,)\},\{A_1,A_1',A_2,A_2',A_3\},P',A_1)$

P':

$A_1 \to A_2 \mid A_2A_1'$

$A_1' \to +A_2 \mid -A_2 \mid +A_2A_1' \mid -A_2A_1'$

$A_2 \to A_3 \mid A_3A_2'$

$A_2' \to *A_3 \mid /A_3 \mid *A_3A_2' \mid /A_3A_2'$

$A_3 \to (A_1) \mid a \mid b$

Noterminallarni qayta o'zgartirilgan holda :

$G'(\{a,b, +,-,/, *,(,)\},\{S,H,T,F,E\},P,S)$

P':

$S \to T \mid TH$

$H \to +T \mid -T \mid +TH \mid -TH$

$T \to E \mid EF$

$F \to *E \mid /E \mid *EF \mid /EF$

$E \to (S) \mid a \mid b$

grammatikaga ega bo'lamiz.

Greybax normal shaklidagi grammatika

Chap rekursiyasiz grammatika asosida Greybax normal shakldagi grammatikani (GNSH) tuzish mumkin.

Greybax normal shaklni ta'rifi:

Agar KEG chap rekursiyasiz bo'lib uni qoidalarida faqat quyidagi qoidalar mavjud bo'lsa:

1. $A \to t\alpha$, $t \in VT$, $\alpha \in VN^*$,

2. $S \to \varepsilon$, agar $\varepsilon \in L(G)$ va S boshqa qoidalarni o'ng tarafida uchramasa,

bunday grammatikalar Greybax normal shakli deb nomlanadi.

Qoidalarni boshqa xech qanday shakli GNShda uchramasligi kerak.

Grammatikani Greybax normal shakliga o'tkazishdan oldin noterminallarni chiziqli tartibga keltiramiz.

Chap rekursiyasiz grammatikani noterminallar to'plamida qisman tartib o'rnatish mumkin. Agar $A \Rightarrow^+ B\alpha$ kelib chiqsa, ikki noterminal o'rtasida qisman tartibli R munosabati mavjud hisoblanadi, ya'ni ARB rost bo'ladi. Qisman tartibli munosabat $R=\{(A,V)\}$ ko'rinishda yoziladi, bunda $A,V \in VN$. Bunday juftliklar R munosabatda bir nechta bo'lishi mumkin.

Agar bevosita $A \to B\alpha \in R$ qoida mavjud bo'lsa, unda A va V orasida chiziqli tartib hisoblanadi va A<B ko'rinishda yoziladi.

Qisman tartibni chiziqli tartibga keltirish algoritmi:

Kirish: Qisman tartibli **R** munosabatlar to'plami berilgan.

Chiqish: Chiziqli tartibli **P'** munosabat, bunda $\mathbf{R} \subseteq \mathbf{P'}$.

Algoritm:

Bizga $A=(N_1,N_2, \dots N_n)$ berilsin. Unda chizikli **P'** tartibni $N_1<N_2< \dots <N_n$, ko'rinishda yozish mumkin, ya'ni $N_i \mathbf{P'} N_j$, $i<j$ uchun rost.

Bunday tartibni o'rnatish uchun quyidagi algoritmni ishlatiladi:

1. i=1, $A_1=A$, $\mathbf{R_1}=\mathbf{R}$ deb olinadi;
2. Agar $A_i \neq \varnothing$, shu to'plamdan shunday N_i tanlash kerakki, barcha $N \in A$ uchun $N\mathbf{R}_1N_i$ yolg'on bo'lishi kerak, ya'ni $\mathbf{R}_1$ to'plamga (N,N_i) juftlik kirmasligi kerak. So'ng 3-qadamga o'tiladi, aks holda (N_i topilmasa) 4-qadamga o'tiladi.
3. $A_{i+1}=A_i-\{N_i\}$ va $\mathbf{R}_{i+1}=\mathbf{R}_i \cap (A_{i+1} \times A_{i+1})$, $i=i+1$ va 2-qadamga o'tish kerak bo'ladi.
4. Qidiruvchi chiziqli tartib quyidagicha: $N_1,N_2, \dots N_{i-1}$ bo'ladi.

Shuni aytib o'tish kerakki N_{i-1} noterminalga tegishli qoidalarni o'ng tarafi faqat terminallardan boshlanadi.

Grammatikani Greybax normal shakliga qayta o'zgartirish algoritmi.

Kirish:Chap rekursiyasiz keltirilgan KEG $G(VT,VN,P,S)$.

Chiqish: Greybax normal shaklidagi ekvivalent $G'(VT,VN,P',S)$ grammatika.

Algoritm:

1. Noterminal belgilar to'plamini, yuqorida ko'rsatilgan algoritm bo'yicha, chiziqli tartiblab chiqiladi: $VN=\{A_1,A_2,\dots A_n\}$, bunda $A_1<A_2<\dots<A_n$ va $A_n$ga tegishli qoidalar o'zgartirmasdan P'ga qo'shiladi.
2. i=n-1 beriladi.
3. Agar i=0 bo'lsa, 5-qadamga o'tiladi, aks holda agar A_i tegishli qoidani o'ng tarafi terminaldan boshlansa u o'zgartirilmasdan P'ga qo'shiladi. So'ng har bir $A_i \to A_j\beta$ qoida ($i<j$) uchun qoidani olib tashlash algoritmi qo'llaniladi, ya'ni agar $A_j \to \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_k$ qoidalar bo'lsa, unda P' ga $A_i \to \gamma_1\beta \mid \gamma_2\beta \mid \dots \mid \gamma_k\beta$ qoidalar qo'shiladi
4. $i=i-1$ qiymat berib 3-qadamga o'tiladi.
5. Yendi har bir qoidani o'ng tarafi terminaldan boshlanadi.

Har bir $A \to tX_1,...,X_m$ qoidada, agar $X_k \in VT$ kirsa, X_k terminal belgisi yangi X'_k noterminal belgiga almashtiriladi, bunda $1 \le k \le m$.

6. 5-qadamda kiritilgan yangi X'_k noterminallar uchun P'ga $X'_k \to X_k$ qoidalar qo'shiladi.

Misol, bizga $G(\{a,b, +,-,/, *,(,)\},\{S,H,T,F,E\},P,S)$ berilsin, va bu yerda

P:

S→ T | TH

$H \to +T \mid -T \mid +TH \mid -TH$

T→E | EF

$F \to *E \mid /E \mid *EF \mid /EF$

$E \to (S) \mid a \mid b.$

Bu arifmetik ifodalar tili uchun chap rekursiyasiz grammatika.

Bu grammatikani noterminallar to'plami $VN=\{S,H,T,F,E\}$ uchun qisman tartib $R=\{(S,T),(S,E), (T,E)\}$ mavjud. Yuqorida keltirilgan algoritm bo'yicha chiziqli tartib hosil qilamiz.

1 qadam. $i=1$, $A_1=VN=\{S,H,T,F,E\}$, $\mathbf{R_1}=\mathbf{R}=\{(S,T),(S,E), (T,E)\}$

2.1 qadam. $A_1 \neq \varnothing$. Berilgan hossaga ega bo'lgan noterminal $N_1 = N$ bo'ladi, chunki $(S,H) \notin \mathbf{R_1}$, $(H,H) \notin \mathbf{R_1}$, $(T,H) \notin \mathbf{R_1}$, $(F,H) \notin \mathbf{R_1}$, $(E,H) \notin \mathbf{R_1}$

3.1 qadam. $A_2=\{S,T,F,E\}$, $\mathbf{R_2}=\mathbf{R_1}$, $i=2$ berib, algoritmni 2-qadamiga o'tamiz.

2.2 qadam. $A_2 \neq \varnothing$. Berilgan hossaga ega bo'lgan noterminal $N_2 = S$ bo'ladi, chunki $(S,S) \notin \mathbf{R_1}$, $(T,S) \notin \mathbf{R_1}$, $(F,S) \notin \mathbf{R_1}$, $(E,S) \notin \mathbf{R_1}$

3.2 qadam. $A_3=\{T,F,E\}$, $\mathbf{R_3}=\{(T,E)\}$, $i=3$ berib, algoritmni 2-qadamiga o'tamiz.

2.3 qadam. $A_3 \neq \varnothing$. Berilgan hossaga ega bo'lgan noterminal $N_3 = F$ bo'ladi, chunki $(T,F) \notin \mathbf{R_1}$, $(F,F) \notin \mathbf{R_1}$, $(E,F) \notin \mathbf{R_1}$

3.3 qadam. $A_4=\{T,E\}$, $\mathbf{R_4}=\{(T,E)\}$, $i=4$ berib, algoritmni 2-qadamiga o'tamiz.

2.4 qadam. $A_4 \neq \varnothing$. Berilgan hossaga ega bo'lgan noterminal $N_4 = T$ bo'ladi, chunki $(T,T) \notin \mathbf{R_4}$, $(E,T) \notin \mathbf{R_4}$

3.4 qadam. $A_5=\{E\}$, $\mathbf{R_5}=\varnothing$, $i=5$ berib ,algoritmni 2-qadamiga o'tamiz.

2.5 qadam. $A_5 \neq \varnothing$. Berilgan hossaga ega bo'lgan noterminal $N_5 = E$ bo'ladi, chunki $(E,E) \notin \mathbf{R_5}$

3.5 qadam. $A_6=\varnothing$., $\mathbf{R_6}=\varnothing$, $i=6$ berib,algoritmni 2-qadamga o'tamiz.

2.6 qadam. $A_6=\varnothing$. 4-qadamga o'tamiz.

4 qadam. Chiziqli tartib : H,S,F,T,E

Natijada berilgan grammatika uchun $H<S<F<T<E$.

Grammatikani Greybax normal shakliga qayta o'zgartirish:

1 qadam. Yuqorida ko'rsatilgan misol uchun noterminallar quyidagicha chiziqli tartiblandi: $H<S<F<T<E$, $n=5$, YE noteminalga tegishli qoidalarni o'zgartirmasdan P'ga ko'chiramiz, chunki ularni o'ng tarafi terminaldan boshlangan.

2qadam. $i=n-1=4$.

3.1 qadam. $i \neq 0$. Grammatikani faqat ikkita qoidasida T noterminal chap tarafida qatnashgan: $T \to E \mid EF$. Ularni quyidagi qoidalarga almashtirib $T \to (S) \mid a \mid b \mid (S)F \mid aF \mid bF$ P'ga ko'chiramiz.

4.1qadam. $i=i-1=3$. 3-qadamga o'tamiz.

3.2qadam. $i \neq 0$. Grammatikani to'rta qoidasida F noterminal chap tarafida qatnashgan: $F \to *E \mid /E \mid *EF \mid /EF$. Ularni o'ng tarafi terminallardan boshlangan, shuning uchun ularni o'zgartirmasdan P'ga ko'chiramiz.

4.2qadam. $i=i-1=2$. 3 qadamga o'tamiz.

3.3 qadam. $i \neq 0$. Grammatikani ikkita qoidasida S noterminal chap tarafida qatnashgan: $S \to T \mid TH$. Ularni quyidagi qoidalarga almashtirib $S \to (S) \mid a \mid b \mid (S)F \mid aF \mid bF \mid (S)H \mid aH \mid bH \mid (S)FH \mid aFH \mid bHF$ P'ga ko'chiramiz.

4.3 qadam. $i=i-1=1$. 3-qadamga o'tamiz.

3.4 qadam. $i \neq 0$. Grammatikani to'rta qoidasida H noterminal chap tarafida qatnashgan:$H \to +T \mid -T \mid +TH \mid -TH$. Ularni o'ng qismi terminallardan boshlangan, shuning uchun ularni o'zgartirmasdan P'ga ko'chiramiz.

4.4 qadam. $i=i-1=0$. 5-qadamga o'tamiz.

5 qadam. Grammatikada qoidalarida faqat bitta ")" belgisi o'ng qismini uchinchi o'rnida turibdi, uni A noterminal belgisiga o'zgartiramiz. Shunda mos qoidalar quyidagicha o'zgartiriladi.

$S \rightarrow (SA \mid (SAF \mid (SAH \mid (SAFH$

$T \rightarrow (SA \mid (SAF$

$E \rightarrow (SA$

6 qadam. Grammatikaga yana bitta qoida qo'shamiz: $A \rightarrow)$

Natijada GNSH yangi grammatikamiz quyidagicha bo'ladi:

$G(\{a,b, +,-,/, *,(,)\},\{S,H,T,F,E,A\},P',S)$ bunda

P':

$S \rightarrow (SA \mid a \mid b \mid (SAF \mid aF \mid b\,F \mid (SAH \mid aH \mid bH \mid (SAFH \mid aFH \mid bHF$

$H \rightarrow +T \mid -T \mid +TH \mid -TH$

$T \rightarrow (SA \mid a \mid b \mid (SAF \mid aF \mid b\,F$

$F \rightarrow *E \mid /E \mid *EF \mid /EF$

$E \rightarrow (SA \mid a \mid b$

$A \rightarrow)$

Bu grammatika qoidalari dastlabki grammatikaga nisbatan 15-qoidaga qo'shildi va 1 noterminal ham qo'shildi.

Xulosa

Kontekstdan erkin grammatika dasturlash tillarini sintaksisini tasvirlaydi. Kontekstdan erkin grammatikani aniqlovchisini tuzish uchun uni qayta o'zgartiril sodda holga olib kelish kerak. Sodda holga keltirilgan grammatika kanonik grammatika deb nomlanadi. Grammatikani kanonik shaklga keltirish uchun quyidagi amallarni bajarish kerak: barcha qatnashmaydigan belgilarni olib tashlash; barcha erishib bo'lmaydigan belgilarni olib tashlash;bo'sh qoidalarni olib tashlash;zanjirli qoidalarni olib tashlash.Xomskiy va Greybax keltirilgan turlardan biri. Undan tashqari bazi bir aniqlovchilar chap rekursiya bo'lmasligini talab qiladi.

Nazorat uchun savollar

1. Kontekstdan erkin grammatikani qanday qayta o'zgartirish guruxlari mavjud?
2. Qatnashmaydigan belgilar ta'rifini keltiring.
3. Qatnashmaydigan belgilarni olib tashlash algoritmini keltiring.

4. Yerishib bo'lmaydigan belgilar ta'rifini keltiring.
5. Yerishib bo'lmaydigan belgilarni olib tashlash algoritmini keltiring.
6. Bo'sh qoidalarni olib tashlash algoritmini keltiring.
7. Zanjir qoidalarni olib tashlash algoritmini keltiring.
8. Ihtiyoriy qoidani olib tashlash algoritmi qanday?
9. Chap faktorizatsiyalash algoritmini keltiring.
10. Chap rekursiyasiz grammatikani ta'rifini keltiring.
11. Chap rekursiyasiz grammatikaga keltirish algoritmini izohlang.
12. Xomskiy normal shaklini ta'rifini keltiring
13. Xomskiy shakliga keltirish algoritmini keltiring.
14. Greybax grammatikani ta'rifini keltiring.
15. Chiziqli tartib tuzish algoritmini keltiring.
16. Greybax normal shakliga o'tkazish algoritmni keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan $G(VT,VN,P,S)$ KEGni befoyda belgilarni olib tashlangan ekvivalent grammatikaga qayta o'zgartiring:

a) $S \rightarrow b \mid C \mid cCB$
$A \rightarrow Ab \mid e$
$B \rightarrow Bb \mid cB$
$C \rightarrow Yef \mid d$

b) $S \rightarrow aC \mid bA$
$A \rightarrow cAB$
$B \rightarrow aC$
$C \rightarrow bA \mid d$

s) $S \rightarrow aABC \mid aE$
$A \rightarrow SCD \mid c$
$B \rightarrow bFD \mid b$
$C \rightarrow aE$
$D \rightarrow aD$
$E \rightarrow aCE \mid a$
$F \rightarrow AB$

d) $S \rightarrow aA$
$A \rightarrow aA \mid b \mid cC$
$B \rightarrow cB \mid a$
$C \rightarrow bAC$

e) $S \rightarrow aA \mid bA$
$A \rightarrow aB \mid bB$
$B \rightarrow aB \mid bB \mid a \mid b$

f) $S \rightarrow aB$
$B \rightarrow aB \mid aC \mid bB \mid bC$
$C \rightarrow \varepsilon$

2. Berilgan $G(VT,VN,P,S)$ KEGni bo'sh qoidalar olib tashlangan ekvivalent grammatikaga qayta o'zgartiring:

a) $S\to AB$
$A\to SA|BB|bB$
$B\to b|aA|\varepsilon$

b) $S\to Aa \mid bB$
$A\to cAaA|a|\varepsilon$
$B\to cBdd|\varepsilon$

s) $S\to bA$
$A\to bA|aB|\varepsilon$
$B\to bB|\varepsilon$

d) $S\to aAB|bA|\varepsilon$
$A\to aAB|\varepsilon$
$B\to bB|c$

e) $S\to aAB \mid bBS \mid \varepsilon$
$A\to cBS \mid \varepsilon$
$B\to dB \mid \varepsilon$

3. Berilgan $G(VT,VN,P,S)$ KEGni bir hil o'ng qismi bo'lmagan ekvivalent grammatikaga qayta o'zgartiring:

a) $S\to AC$
$A\to B|AaB$
$B\to i$
$C\to D|DaC$
$D\to i$

b) $S\to 1A|B0$
$A\to 1A|C$
$B\to B0|C$
$C\to 1C0|\varepsilon$

s) $S\to SS|1A0$
$A\to 1A0|\varepsilon$

d) $S\to aC|bA$
$A\to cAB$
$B\to aC|c$
$C\to bA|d$

e) $S\to abSa \mid aaAb \mid b$
$A\to baAb \mid b$

f) $S\to aAB \mid bBS \mid \varepsilon$
$A\to cBS \mid \varepsilon$
$B\to dB \mid \varepsilon$

4. Berilgan $G(VT,VN,P,S)$KEGni zanjir qoidalarni olib tashlangan ekvivalent grammatikaga qayta o'zgartiring:

a) $S\to LA|LB$
$L\to P:=|Q:=$
$P\to i$
$A\to F$
$Q\to i$
$B\to F$
$F\to Q(i)$

b) $S\to AC$
$A\to B|AaB$
$B\to i$
$C\to D|DaC|i$

s) $S\to 1A|B0$
$A\to 1A|C$
$B\to B0|C$
$C\to 1C0|\varepsilon$

d) $S\to T+P|T$
$T\to T*P|P$
$P\to C$
$C\to /C|/$

e) $S\to A \mid B \mid b$
$A\to 1A0|1a0$
$B\to 1B00|1b00$

5. Berilgan $G(VT,VN,P,S)$KEGni chap rekursiyadan erkin bo'lgan ekvivalent grammatikaga qayta o'zgartiring:

a) $S\to Ba|Ab$
$A\to Sa|AAb|a$
$B\to Sb|BBa|b$

b) $S\to AB|a$
$A\to BS|Sb$
$B\to SA|BB|a$

c) $S\to Ab$
$A\to Sa|cB$
$B\to bS|c$

d) $S\to AB$
$A\to SA|BB|bB$
$B\to b|aA|\varepsilon$

e) $S\to SaA|AA|b$
$A\to ASa|Ad|c$

6. Berilgan $G(VT,VN,P,S)$KEGni ekvivalent Xomskiy normal shakl grammatikaga qayta o'zgartiring:

a) $S\to AB|Ab$
$A\to SA|BB|bB$
$B\to b|aA|\varepsilon$

b) $S\to Aa|bB$
$A\to cAaA|a|\varepsilon$
$B\to cBdd|\varepsilon$

c) $S\to SS|1A0$
$A\to 1A0|\varepsilon$

d) $S\to aC|bA$
$A\to cAB$
$B\to aC$
$C\to bA|d$

7. Berilgan $G(VT,VN,P,S)$KEGni ekvivalent Greybax normal shakl grammatikaga qayta o'zgartiring:

a) $S\to A|B$
$A\to 1A0|1a0$
$B\to 1B00|1b00$

b) $S\to Aa|bB$
$A\to cAaA|a|\varepsilon$
$B\to cBdd|\varepsilon$

c) $S\to abSa|aaAb|b$
$A\to baAb|b$

d) $S\to AB$
$A\to SA|$
$BB|bB$
$B\to aA|b|\varepsilon$

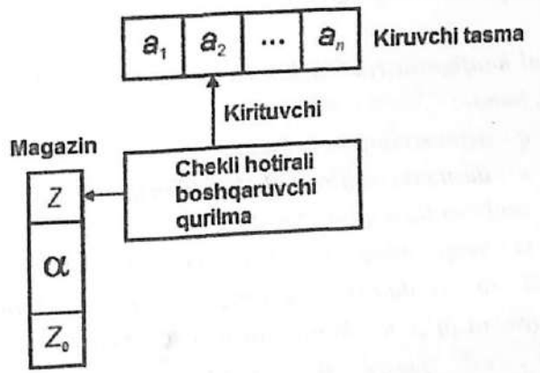
23-BOB. KONTEKSTDAN ERKIN TILLARNING ANIQLOVCHILARI. MAGAZIN XOTIRALI AVTOMATLAR

Tayanch iboralar: *magazin hotirali avtomat, kengaytirilgan magazin hotirali avtomat, determinantlangan magazin hotirali avtomat, nodeterminantlangan, nodeterminantlangan magazin hotirali avtomat, zanjirni chap o'ralishi, zanjirni qismini negizi, magazin hotirali qayta o'zgartirgich.*

Magazin hotirali avtomatlarni ta'rifi

P to'plamdagi qoidalar $A\to\beta$, shaklida bo'lib, bunda $A\in VN$ va $\beta\in(VN\cup VT)^*$, shaklida bo'lgan $G(VT,VN,P,S)$ grammatika asosidagi til kontekstdan erkin til deb nomlanadi.

Kontekstdan erkin tillarni aniqlovchisi ***magazin hotirali avtomat***(MXA) bo'ladi. Uning modeli 23.1-rasmda keltirilgan.



23.1-rasm. Magazin xotirali avtomatning modeli.

Bu bir tomonli nodeterminantlangan qo'shimcha hotirali aniqlovchi. Qo'shimcha hotira magazin (stek) tamoilda tashkil qilingan, ya'ni ohirgi yozilgan belgi birinchi o'qiladi. Magazinni belgilar zanjiri deb hisoblasak bo'ladi, bu zanjirning eng chap belgisi magazin ustidagi elementga to'g'ri keladi. Magazin o'ngdan chapga qarab to'ldiriladi.

Magazin xotirali avtomat deb yettita obyekt hisoblanadi:

R(Q,V,Z,δ,q_0,z_0,F), bunda:

- ***Q – avtomatdagi holatlarning chekli to'plami;***
- ***V – avtomat alifbosi (terminal belgilarning chekli to'plami);***
- ***Z – magazinni maxsus alifbosi (grammatikaning terminal va noterminal belgilardan hosil bo'lgan chekli to'plami) , bunga maxsus belgi z_0 qo'shiladi;***
- ***δ – boshqaruv qurilmasidagi o'tish funksiyalarning chekli to'plami. Bu funksiya δ :Q×(V∪{ε})×Z→ Q×Z*, ya'ni δ (q,t,z)={(r,μ)| r∈Q, μ∈Z^*}, q∈Q , t∈V, z∈Z;***
- ***q_0 – avtomatning boshlang'ich holati, q_0∈ Q;***
- ***z_0– magazinning tag(osti) belgisi z_0∈ Z;***
- ***F – avtomatning chekli holatlar to'plami F⊆Q.***

MXAni CHAdan farqi shundaki, unda magazin xotirasi qo'shilgan bo'lib, bir konfiguratsiyadan boshqa konfiguratsiyaga o'tishi uchun kiruvchi zanjirning joriy belgisidan tashqari, magazin hotirani ustida turgan belgilarga ham bog'liq bo'ladi.

MXAni konfiguratsiyasi uchta element bilan aniqlanadi (q,x,ω) ∈ Q×V^*×Z^*, bunda:

- ***q – avtomatning joriy holati;***
- ***x – dastlabki zanjirninghali o'qilmagan qismi.Agar dastlabki zanjir to'liq o'qilsa, x= ε;***
- ***ω - magazindagi zanjir,agar magazin bo'sh bo'lsa ω= ε.***

Agarda (q',γ)∈δ(q,t,z) to'plamiga kirsa, avtomat (q,tx,zω) konfiguratsiyadan (q',x,γω) konfiguratsiyaga o'tadi, bunda q, q'∈Q, t∈ V∪{ε}, x∈V^*, z∈Z, γ,ω∈Z^*. Bunday o'tish avtomat takti(qadami) deb nomlanadi va (q,tx,zω) |— (q',x,γω) munosabat ko'rinishda yoziladi.

Bir taktni bajarishda, magazin ustidagi belgi, o'tish funksiyasiga ko'ra, zanjirga almashtiriladi. Almashtirilgan zanjirning eng chap (birinchi) belgisi, magazinning ustida bo'lib qoladi. Avtomatda kirituvchi qurilma siljimasligi ham mumkin, ya'ni joriy belgini o'qish holatida qolishi mumkin. Bu vaziyat o'tish funksiyada $t=\varepsilon$ bo'lganda bo'ladi. Bunday taqtlarε-taktlar deb nomlanadi. ε-taktlarda boshqarish qurilmaning holati va magazindagi zanjir o'zgarilishi mumkin, lekin kiruvchi zanjirning

qolgan qismi o'zgarmaydi. Huddi shunday, avtomat magaziniga belgilar qo'shilishi shart emas, bu vaziyat o'tish funksiyani qiymatidagi $\gamma=\varepsilon$, ya'ni bo'sh zanjirga teng bo'lganda bo'ladi. Shuni aytib o'tish kerakki, kiruvchi tasma bo'sh bo'lganda ham taktlar bajariladi, lekin magazin bo'sh bo'lsa avtomat ta'rif bo'yicha, ishini tamomlaydi.

MXAda biror bir konfiguratsiyadan bir takda mobaynida bir nechta konfiguratsiyalarga o'tish imkoniyati mavjud bo'lsa, avtomat nodeterminantlangan deb hisoblanadi.

Avtomatning boshlang'ich konfiguratsiyasi (q_0, x, z_0), $x \in V^*$.

Avtomatning chekli konfiguratsiyalar to'plami - (q, ε, ω), $q \in F$, $\omega \in Z^*$.

Agar biror bir zanjir kiruvchiga uzatilib, avtomat boshlang'ich konfiguratsiyadan, biror bir chekli konfiguratsiyaga o'tsa, va bunda kiruvchi zanjir to'liq o'qilgan. Boshqarish qurilma biror bir chekli holatga o'tgan va magazinda biror bir zanjir qolgan bo'lsa, kiruvchi zanjir aniqlovchia vtomat tomondan qabul qilindi, aks holda qabul qilinmadi deb hisoblanadi.

R(Q,V,Z,δ,q_0,z_0,F) MXA bilan aniqlangan til deb,
L(R) ={x|x∈VT^*, (q_0,x,z_0) |— * (q,ε ,ω) , q∈F, ω∈Z^*} to'plam nomlanadi.

MXA bilan aniqlangan til, bu qabul qilish mumkin bo'lgan barcha zarjirlar to'plamidir. Agar ikkita R_1 va R_2 MXAlar bitta tilni aniqlasa, ular ekvivalent hisoblanadi, ya'ni $L(R_1)= L(R_2)$.

Agar zanjirni tahlil qilish natijasida avtomat biror bir chekli holatda bo'lib, zanjir to'liq o'qilgan va magazin bo'sh bo'lsa, ya'ni$(q, \varepsilon, \varepsilon)$, konfiguratsiyada bo'lsa, MXA zanjirni bo'sh magazin bilan qabul qildi, deb hisoblanadi. Bunday avtomat bilan aniqlangan til $L_\varepsilon(R)$ kabi yoziladi.

Har qanday MXA uchun doimo ekvivalent magazini bo'sh MXA qurish mumkin.

Misol, $L=\{a^n b^n | n>0\}$ til zanjirlarini qabul qiladigan MHA avtomatni quramiz:

$R=(\{q_0,q_1,q_2\},\{a,b\},\{z_0,a\},\delta,q_0,z_0,\{q_2\})$, bunda δ quyidagicha aniqlanadi:

1) $\delta(q_0,a,z_0)=\{(q_1,az_0)\}$;
2) $\delta(q_1,a,a)=\{(q_1,aa)\}$;
3) $\delta(q_1,b,a)=\{(q_2,\varepsilon)\}$;

4) $\delta(q_2, b, a)=\{(q_2, \varepsilon)\}$;

5) $\delta(q_2, \varepsilon, z_0)=\{(q_2, \varepsilon)\}$;

aaabbb kiruvchi zanjir uchun MHA quyidagi taktlar ketma-ketligini bajaradi:

1. $(q_0, aaabbb, z_0)\vdash^1$
2. $(q_1, aabbb, az_0)\vdash^2$
3. $(q_1, abbb, aaz_0)\vdash^2$
4. $(q_1, bbb, aaaz_0)\vdash^3$
5. $(q_2, bb, aaz_0)\vdash^4$
6. $(q_2, b, az_0)\vdash^4$
7. $(q_2, \varepsilon, z_0)\vdash^5$
8. $(q_2, \varepsilon, \varepsilon)$

bunda $\vdash$ belgi ustidagi raqam δ-funksiyani tartib raqamini ko'rsatadi.

Avtomatning ishlashidan ko'rinib turibdiki, u oldin barcha "*a*" belgilarni magazinga yuklaydi, keyin har bir "*b*" belgisi uchun magazindan "*a*" belgisini olib tashlaydi.

Oddiy MXAdan tashqari ***kengaytirilgan magazin hotirali avtomat***(KMXA) tushunchasi mavjud. KMXAda magazin ustidagi chekli uzinlikdagi zanjirni, boshqa zanjir bilan almashtirish mumkin. Bu degani bir takt ichida KMXA faqat bitta emas, balki bir nechta belgilar ketma-ketligini almashtirishi mumkin. Bunday avtomatda o'tish funksiyalar $Q\times V\times Z^*$ dekart to'plamini $Y(Q\times Z^*)$ to'plamiga aks ettiradi.

KMHAda magazinning usti elementi deb zanjirni eng o'ng belgisi hisoblanadi va magazin chapdan o'nga qarab to'ldiriladi.

Bir taktni bajarishda, magazin ustidagi bir nechta belgi (zanjir), o'tish funksiyasiga ko'ra, boshqa zanjirga almashtiriladi. Almashtirilgan zanjirning eng o'ng (ohirgi) belgisi, magazinning ustida bo'lib qoladi.

Agarda $(q',\psi)\in\delta(q,t,\gamma)$ bo'lib, bunda $q,q'\in Q$, $t\in V\cup\{\varepsilon\}$, $x\in V^*,\gamma$, $\omega,\psi\in Z^*$ bo'lsa, bir taktda kengaytirilgan avtomat $(q,tx,\omega\gamma)$ konfiguratsiyadan $(q',x,\omega\psi)$ konfiguratsiyaga o'tadi.

Kengaytirilgan avtomat, magazin bo'sh bo'lganda ham taktlarni bajarishi mumkin, ya'ni $\omega\gamma=\varepsilon$.

Misol, $L=\{\omega\omega^R \mid \omega\in\{a,b\}^*\}$ til zanjirlarini qabul qiladigan KMHAni

quramiz:

$R=(\{q_0,q_1\},\{a,b\},\{z_1,z_0,a,b\},\delta,q_0,z_0,\{q_1\})$, bunda $\delta(q,t,\gamma)$ quyidagicha aniqlanadi:

1) $\delta(q_0,a,\varepsilon)=\{(q_0,a)\}$;
2) $\delta(q_0,b,\varepsilon)=\{(q_0,b)\}$;
3) $\delta(q_0,\varepsilon,\varepsilon)=\{(q_0,z_1)\}$;
4) $\delta(q_0,\varepsilon,az_1a))=\{(q_0,z_1)\}$;
5) $\delta(q_0,\varepsilon,bz_1b))=\{(q_0,z_1)\}$;
6) $\delta(q_1,\varepsilon,z_0z_1)=\{(q_1,\varepsilon)\}$.

aabbaa kiruvchi zanjir uchun KMHA quyidagi taktlar ketma-ketligini bajaradi:

1. $(q_0, aabbaa, z_0)\vdash^1$
2. $(q_0, abbaa, z_0a)\vdash^1$
3. $(q_0, bbaa, z_0aa)\vdash^2$
4. $(q_0, baa, z_0aab)\vdash^3$
5. $(q_0, baa, z_0aabz_1)\vdash^2$
6. $(q_0, aa, z_0aabz_1b)\vdash^5$
7. $(q_0, aa, z_0aaz_1)\vdash^1$
8. $(q_0, a, z_0aaz_1a)\vdash^4$
9. $(q_0, a, z_0az_1)\vdash^1$
10. $(q_0, \varepsilon, z_0az_1a)\vdash^4$
11. $(q_0, \varepsilon, z_0z_1)\vdash^6$
12. $(q_1, \varepsilon, \varepsilon)$

Avtomatning ishlashidan ko'rinib turibdiki, u oldin kiruvchi zanjirning o'rtasigacha boshidagi belgilarni magazinga yuklaydi, keyin o'rtani nishonlovchi "z_1" belgini yozadi. So'ng keyingi belgi tasmadan o'qiladi va az_1a yoki bz_1b zanjirlarni "z_1"ga almashtiradi, bu jarayon kiruvchi zanjirning oxirigacha o'qiguncha bajariladi. Natijada magazinda z_0z_1 qolsa, keyingi takda avtomat chekli konfiguratsiyaga o'tadi, aks holda zanjir qabul qilinmaydi.

Har qanday KMHAni ekvivalent bo'lgan oddiy MHAga o'tkazish mumkinligi isbotlangan.

Kontekstdan erkin grammatika asosida magazin hotirali avtomat

tuzish algoritmi

Kirish:$G(VT,VN,P,S)$ KEG;

Chiqish:$R(Q,V,Z,\delta,q_0,z_0,F)$ nodeterminantlangan magazin xotirali avtomat (NMHA).

Algoritm:

1-qadam. Q holatlar to'plami bitta holatdan iboratdir $Q=\{q\}$, $V=VT$, $Z=VT\cup VN$, $q_0=q$, $z_0=S$, $F=\{q\}$.

2-qadam. Agar $A\to\alpha$ grammatikaning qoidasi bo'lsa, unda $(q, \alpha) \in \delta(q, \varepsilon, A)$ bo'ladi.

3-qadam. Barcha terminallar uchun $\delta(q, t, t)=\{(q, \varepsilon)\}$, $\forall t \in V$.

Misol, arifmetik ifodalarni ta'riflaydigan grammatika $G(\{a,b,+,*,(,)\},\{S,T,E\},P,S)$ berilgan va bunda P:

$S\to S+T|T$

$T\to T*E|E$

$E\to(S)|a|b$bo'lsin.

Shunda NMHA tuzilishi quyidagicha bo'ladi:

$R(\{q\},\{a,b,+,*,(,)\},\{a,b,+,*,(,),S,T,E\}, \delta,q,S,\{q\})$

δ:

1) $\delta(q, \varepsilon,S)=\{(q,S+T),(q,T)\}$;
2) $\delta(q, \varepsilon,T)=\{(q,T*E),(q,E)\}$;
3) $\delta(q, \varepsilon,E)=\{(q,(S)),(q,a),(q,b)\}$;
4) $\delta(q, a,a)=\{(q, \varepsilon)\}$;
5) $\delta(q, b,b)=\{(q, \varepsilon)\}$;
6) $\delta(q, +,+)=\{(q, \varepsilon)\}$;
7) $\delta(q, *,*)=\{(q, \varepsilon)\}$;
8) $\delta(q, (,()=\{(q, \varepsilon)\}$;
9) $\delta(q,),))=\{(q, \varepsilon)\}$.

NMHA avtomatning kiruvchi tasmasiga $a*(a+b)$zanjir uzatilsa, shunda avtomatning taktlar ketma-ketligi quyidagicha bo'ladi:

1. $(q, a*(a+b), S)|-^1$
2. $(q, a*(a+b), T)|-^2$
3. $(q, a*(a+b), T*E)|-^2$
4. $(q, a*(a+b), E*E)|-^3$
5. $(q, a*(a+b), a*E)|-^4$

6. $(q, *(a+b), *E)|-^7$
7. $(q, (a+b), E)|-^3$
8. $(q, (a+b), (S))|-^8$
9. $(q, a+b), S))|-^1$
10. $(q, a+b), S+T))|-^1$
11. $(q, a+b), T+T))|-^1$
12. $(q, a+b), E+T))|-^2$
13. $(q, a+b), a+T))|-^3$
14. $(q, +b), +T))|-^6$
15. $(q, b), T))|-^2$
16. $(q, b), E))|-^3$
17. $(q, b), b))|-^5$
18. $(q,),))|-^9$
19. $(q, \varepsilon, \varepsilon)$

avtomat chekli konfiguratsiyaga o'tadi va zanjir qabul qilinadi.

Avtomatni bajargan taktlar ketma-ketligi, grammatikadagi chap keltirib chiqarishga to'g'ri keladi:

$S\Rightarrow T\Rightarrow T*E\Rightarrow E*E\Rightarrow a*E\Rightarrow a*(S)\Rightarrow a*(S+T)\Rightarrow a*(T+T)\Rightarrow a*(E+T)\Rightarrow a*(a+T)\Rightarrow a*(a+E)\Rightarrow a*(a+b)$

Zanjirni tahlili, daraxtni yuqoridan pastga va chapdan o'ngga qarab tuzilishiga to'g'ri kelmoqda.NMXA asosida daraxtni tuzish, pasayuvchi sintaktik tahlil deb nomlanadi,. Chunki keltirib chiqarish daraxti ildizdan boshlab barglarigacha quriladi, va har bir taktda eng chap belgi, ya'ni magazinning ustida turgan belgi qoidaning o'ng qismiga almashtiriladi.

Kontekstdan erkin grammatika asosida kengaytirilgan magazin hotirali avtomat tuzish algoritmi

Kirish:$G(VT,VN,P,S)$KEG;

Chiqish:$R(Q,V,Z,\delta,q_0,S,F)$nodeterminantli kengaytirilgan xotirali avtomat (NKMHA).

Algoritm:

1-qadam. Q holatlar to'plami ikkita holatdan iboratdir $Q=\{q,r\}$ $V=VT$; $Z=VT\cup VN\cup\{S\}$,$q_0=q$, $F=\{r\}$, S magazinning tag belgisi.

2-qadam. Barcha terminallar uchun $\delta(q, t, \varepsilon)=\{(q, t)\}$ $\forall t \in V$.

3-qadam.Agar $A \to \alpha$ grammatikani qoidasi bo'lsa, unda$(q, A) \in \delta(q, \varepsilon, \alpha)$ bo'ladi.

4-qadam. $\delta(q, \varepsilon, \$S) = \{(r, \varepsilon)\}$

Misol, yuqorida ko'rsatilgan grammatika uchun NKMHA quramiz.

NKMHA qurilishi quyidagicha:

$R(\{q,r\},\{a,b,+,*,(,)\},\{a,b,+,*,(,),S,T,E,\$\},\delta,q,\$,\{r\})$

δ:

1) $\delta(q, a, \varepsilon) = \{(q, a)\}$;
2) $\delta(q, b, \varepsilon) = \{(q, b)\}$;
3) $\delta(q, +, \varepsilon) = \{(q, +)\}$;
4) $\delta(q, *, \varepsilon) = \{(q, *)\}$;
5) $\delta(q, (, \varepsilon) = \{(q, ()\}$;
6) $\delta(q,), \varepsilon) = \{(q,)\}$;
7) $\delta(q, \varepsilon, S+T) = \{(q,S)\}$;
8) $\delta(q, \varepsilon, T) = \{(q,S)\}$;
9) $\delta(q, \varepsilon, T*E) = \{(q,T)\}$;
10) $\delta(q, \varepsilon, E) = \{(q,T)\}$;
11) $\delta(q, \varepsilon, b) = \{(q,E)\}$;
12) $\delta(q, \varepsilon, a) = \{(q,E)\}$;
13) $\delta(q, \varepsilon, (S)) = \{(q,E)\}$;
14) $\delta(q, \varepsilon, \$S) = \{(r, \varepsilon)\}$.

Yendi, NKMHA avtomatning kiruvchi tasmasiga $a+a*b$ zanjir uzatilsa, avtomatning taktlar ketma-ketligi quyidagicha bo'ladi:

1. $(q, a+a*b, \$) \vdash^{1}$
2. $(q, +a*b, \$a) \vdash^{12}$
3. $(q, +a*b, \$E) \vdash^{10}$
4. $(q, +a*b, \$T) \vdash^{8}$
5. $(q, +a*b, \$S) \vdash^{3}$
6. $(q, a*b, \$S+) \vdash^{1}$
7. $(q, *b, \$S+a) \vdash^{12}$
8. $(q, *, \$S+E) \vdash^{10}$
9. $(q, *, \$S+T) \vdash^{8}$
10. $(q, b, \$S+T*) \vdash^{4}$
11. $(q, \varepsilon, \$S+T*b) \vdash^{11}$
12. $(q, \varepsilon, \$S+T*E) \vdash^{9}$

13. $(q, \varepsilon, \$S+T) \vdash^{7}$
14. $(q, \varepsilon, \$S) \vdash^{14}$
15. $(r, \varepsilon, \varepsilon)$.

ya'ni avtomat chekli konfiguratsiyaga o'tdi va zanjir qabul qilindi.

Kengaytirilgan avtomatning bajargan taktlar ketma-ketligi grammatikada o'ng keltirib chiqarishga to'g'ri keladi, faqat teskari tartibda bajariladi.

KMHA asosida qurilgan sintaktik tahlilchilar ko'tariluvchi tahlilchi deb nomlanadi

$G=(VT,VN,P,S)$ KEG berilsin va unda $S \Rightarrow_r^* \alpha Aw \Rightarrow_r \alpha\beta w \Rightarrow_r^* xw$ zanjirning o'ng keltirib chiqarishi bo'lsin. U holda $A \to \beta$ qoida asosida o'ng keltirib chiqarishli $\alpha\beta w$ zanjirning o'ng keltirib chiqarishli αAw zanjirga almashtirish, chap o'rash deb nomlanadi. $\alpha\beta w$ zanjirning β negiz qismi deb nomlanadi.

Ta'rifdan ko'rinib turibdiki zanjirning negiz qismi, bu o'ng keltirib chiqarishda qatnashgan biror bir qoidaning o'ng tarafidir. Berilgan $a+a*b$ zanjirni o'ng keltirib chiqarishini ko'ramiz:

$$S \Rightarrow S+T \Rightarrow S+T*E \Rightarrow S+T*b \Rightarrow S+E*b \Rightarrow S+a*b \Rightarrow T+a*b \Rightarrow E+a*b \Rightarrow a+a*b$$

Yendi uni teskari tartibda yozib chiqamiz:

$$a+a*b \Leftarrow E+a*b \Leftarrow T+a*b \Leftarrow S+a*b \Leftarrow S+E*b \Leftarrow S+T*b \Leftarrow S+T*E \Leftarrow S+T \Leftarrow S$$

bundan ko'rinib turibdiki, masalan zanjir $S+T*E$ grammatikani $T \to T*E$ qoidasiga ko'ra $S+T$ zanjirga chap o'ralmoqda.

Zanjirni tahlil qilish uchun, daraxt pastdan yuqoriga va chapdan o'nga qarab tuzilish kerak.

Magazin hotirali avtomat asosida kontekstdan erkin grammatikani tuzish algoritmi

Kirish:$R(Q,V,Z,\delta,q_0,z_0,F)$ MHA;

Chiqish:$G(VT,VN,P,S)$KEG;

Algoritm:

1-qadam. $VT=V$; $VN=\{[qzp] \mid q,p \in Q, z \in Z\} \cup \{S\}$

2-qadam. P qoidalar to'plamiga har bir $q \in Q$uchun$S \to [q_0 z_0 q]$ qoida qo'shiladi.

3-qadam. Agar $(r,\varepsilon)\in\delta$ (q,a,z), P to'plamiga $[qzr]\rightarrow a$ qoida qo'shiladi.

4-qadam. Agar $(r, X_1X_2...X_n) \in \delta$ (q,a,z)bo'lsa, P to'plamga har bir $q_i\in Q$ uchun $[qzp]\rightarrow a[rX_1q_1][q_1X_2q_2]...[q_{n-1}X_nq_n]$ qoidalar qo'shiladi,. Bu yerda $z,X_i\in Z$, $n>0, a\in V\cup\{\varepsilon\}$, $p=q_n$, $q,r\in Q$.

5-qadam. Noterminal belgilar nomi o'zgartiriladi va grammatika qanonik shaklga keltiriladi.

Misol, $L=\{a^nb^n|\ n>0\}$ til zanjirlarini qabul qiladigan MHA avtomat $R=(\{q_0,q_1,q_2\},\{a,b\},\{z_0,a\},\delta,q_0,z_0,\{q_2\})$ berilsin, bundagi δ o'tish funksiyasi quyidagicha aniqlanadi:

1) $\delta(q_0,a,z_0)=\{(q_1,az_0)\}$;
2) $\delta(q_1,a,a)=\{(q_1,aa)\}$;
3) $\delta(q_1,b,a)=\{(q_2,\varepsilon)\}$;
4) $\delta(q_2, b, a)=\{(q_2, \varepsilon)\}$;
5) $\delta(q_2, \varepsilon,z_0)=\{(q_2, \varepsilon)\}$.

$G(VT,VN,P,S)$KEG tuzamiz:

1 qadam. $VT=\{a,b\}$, $VN=\{[q_0z_0q_0],[q_0z_0q_1],\ldots,[q_2aq_2],S\}$.

2 qadam. P to'plamga quyidagiqoidalarni qo'shamiz:

0.1)$S\rightarrow[q_0z_0q_0]$; 0.2)$S\rightarrow[q_0z_0q_1]$; 0.3)$S\rightarrow[q_0z_0q_2]$.

3 qadam.(3-5) o'tish funksiyalar asosida quyidagi qoidalarni qo'shamiz:

1.1)$[q_1aq_2]\rightarrow b$; 1.2)$[q_2aq_2]\rightarrow b$; 1.3) $[q_2z_0q_2]\rightarrow\varepsilon$.

4 qadam.

a) 0.1 qoidada o'ng qismidagi $[q_0z_0q_0]$ noterminal uchun qoidalarni 1-chi o'tish funksiyasiga asosan, yozib chiqamiz:

2.1)$[q_0z_0q_0]\rightarrow a[q_1aq_0]$ $[q_0z_0q_0]$;
2.2) $[q_0z_0q_0]\rightarrow a[q_1aq_1]$ $[q_1z_0q_0]$;
2.3) $[q_0z_0q_0]\rightarrow a[q_1aq_2]$ $[q_2z_0q_0]$.

Bu qoidalarda $[q_1z_0q_0]$va$[q_2z_0q_0]$ noterminallar keltirib chiqarishda qatnashmaydi, chunki ular uchun mos o'tish funksiyalari yo'q va ular 1.1-1.3 qoidalarda qatnashmaydi, shu sababli 2.2 va 2.3 qoidalar olib tashlanadi. $[q_1aq_0]$ noterminal uchun qoidalarni tuzamiz.

2 o'tish funksiyasiga ko'ra:

3.1)$[q_1aq_0]\rightarrow a[q_1aq_0]$ $[q_0aq_0]$;
3.2) $[q_1aq_0]\rightarrow a[q_1aq_1]$ $[q_1aq_0]$;

3.3) $[q_1aq_0]\rightarrow a[q_1aq_2]$ $[q_2aq_0]$.

Bu qoidalarda $[q_0aq_0]$va$[q_2aq_0]$ noterminallar qatnashmaydi, chunki ular uchun mos o'tish funksiyalari yo'q, shu sababli 3.1 va 3.3 qoidalar olib tashlanadi. $[q_1aq_1]$ noterminal uchun qoidalarni tuzamiz.

2 o'tish funksiyasiga ko'ra:

4.1) $[q_1aq_1]\rightarrow a[q_1aq_0]$ $[q_0aq_1]$;
4.2) $[q_1aq_1]\rightarrow a[q_1aq_1]$ $[q_1aq_1]$;
4.3) $[q_1aq_1]\rightarrow a[q_1aq_2]$ $[q_2aq_1]$.

Bu qoidalarda $[q_0aq_1]$va $[q_2aq_1]$ noterminallar qatnashmaydi, chunki ular uchun mos o'tish funksiyalar yo'q va ular boshqa qoidalarda qatnashmaydi, shu sababli 4.1 va 4.3 qoida olib tashlanadi. $[q_1aq_1]$ noterminal ham keltirib chiqishda qatnashmaydi, ya'niboshqa qoidalarda chiqmaydi, shu sababli 4.2 va 3.2 qoidalar ham olib tashlanadi va $[q_1aq_0]$ qatnashmaydi.

b)0.2qoidada o'ng qismidagi $[q_0z_0q_1]$noterminal uchun qoidalarni 1-(chi) o'tish funksiyasiga asosan, yozib chiqamiz:

5.1) $[q_0z_0q_1]\rightarrow a[q_1aq_0]$ $[q_0z_0q_1]$;
5.2) $[q_0z_0q_1]\rightarrow a[q_1aq_1]$ $[q_1z_0q_1]$;
5.3) $[q_0z_0q_1]\rightarrow a[q_1aq_2]$ $[q_2z_0q_1]$.

Bu qoidalarda $[q_1aq_0]$va$[q_1aq_1]$ noterminallar yuqorida ko'rsatilgan bo'yicha qatnashmaydi, $[q_2z_0q_1]$ chiqarishda ham qatnashmaydi, chunki unga mos o'tish funksiyalari yo'q va boshqa qoidalarda qatnashmaydi, shu sababli 5.1-5.3qoidalar olib tashlanadi.

v)0.3qoidada o'ng qismidagi $[q_0z_0q_2]$noterminal uchun qoidalarni 1- o'tish funksiyasiga asosan, yozib chiqamiz:

6.1) $[q_0z_0q_2]\rightarrow a[q_1aq_0]$ $[q_0z_0q_2]$;
6.2) $[q_0z_0q_2]\rightarrow a[q_1aq_1]$ $[q_1z_0q_2]$;
6.3) $[q_0z_0q_2]\rightarrow a[q_1aq_2]$ $[q_2z_0q_2]$.

Bu qoidalarda $[q_1aq_0]$va$[q_1aq_1]$ noterminallar yuqorida ko'rsatilgan bo'yicha qatnashmaydi, shu sababli 6.1 va 6.2qoidalar olib tashlanadi. $[q_1aq_2]$ noterminal uchun qoidalarni tuzamiz.

2- o'tish funksiyasiga ko'ra:

7.1) $[q_1aq_2] \to a[q_1aq_0]\ [q_0aq_2]$;

7.2) $[q_1aq_2] \to a[q_1aq_1]\ [q_1aq_2]$;

7.3) $[q_1aq_2] \to a[q_1aq_2]\ [q_2aq_2]$.

Bu qoidalardan faqat 7.3 qoida qoladi va qolgan qoidalar yuqorida ko'rsatilgan sabablarga ko'ra olib tashlanadi.

Natijada quyidagi noterminallar va qoidalar qoladi:

$S \to [q_0z_0q_2]$

$[q_0z_0q_2] \to a[q_1aq_2]\ [q_2z_0q_2]$

$[q_1aq_2] \to a[q_1aq_2]\ [q_2aq_2]$

$[q_1aq_2] \to b$

$[q_2aq_2] \to b$

$[q_2z_0q_2] \to \varepsilon$.

Noterminallarni qayta belgilab chiqamiz:

$[q_0z_0q_2]=A, [q_1aq_2]=B,\ [q_2aq_2]=C,\ [q_2z_0q_2]=D$.

Shunda qoidalar quyidagicha bo'ladi:

$S \to A$

$A \to aVD$

$B \to aBC|b$

$C \to b$

$D \to \varepsilon$.

Qoidalarni oldin ko'rsatilgan usullar bilan qayta o'zgartirib Greybax shaklidagi grammatikani hosil qilamiz:

$G(\{a,b\},\{S,B,C\},P,S)$ bunda P:

$S \to aB$

$B \to aBC|b$

$C \to b$

Determinantlangan magazin hotirali avtomat

Amalda NMXA kam ishlatiladi chunki, bunday avtomat uchun aniklovchi tuzish juda murakkabdir. Shu sababli ko'pincha determinantlangan magazin xotirali avtomatlar ishlatiladi.

Agar $R(Q,V,Z,\delta,q_0,z_0,F)$ MXAda har bir $q \in Q$ va $z \in Z$ uchun quyidagi shartlar bajarilsa:

1) ***Har bir $t \in (V \cup \{\varepsilon\})$ uchun o'tish funksiyasi $\delta(q,t,z)$ bo'sh yoki bitta elementga teng bo'lsa,***

2) ***Har bir $\delta(q,\varepsilon,z) \neq \varnothing$ uchun, $\forall t \in V$, $\delta(q,t,z) = \varnothing$ bo'lsa, bunday avtomat, determinantlangan magazin hotirali avtomat (DMHA) deb nomlanadi***

Bu shartlar avtomatning har qanday konfiguratsiyasidan o'zidan faqat bitta keyingi konfiguratsiyasiga o'tish mumkinligini bildiradi. Shu sababli bu avtomatning aniqlovchisini qurish ancha yengil bo'ladi.

O'tish funksiyani to'plami bitta elementdan tashkil topganligi sababli, uni $\delta(q, a, z) = (q', \gamma)$ shaklda yozsak ham bo'ladi.

DMHA yordamida aniqlangan til determinantlangan kontekstdan erkin til deb nomlanadi.

Misol:

$L=\{wcw^R \mid w \in \{a,b\}^*\}$ til uchun DMHA quraylik.

DMHA – avtomatni quyidagicha yozamiz

$R(\{q_0,q_1,q_2\},\{a,b,c\},\{S,a,b\},\ \delta,q_0,S,\{q_2\})$, bunda

δ:

1) $\delta(q_0, a, S) = (q_0, aS)$;
2) $\delta(q_0, a, a) = (q_0, aa)$;
3) $\delta(q_0, a, b) = (q_0, ab)$;
4) $\delta(q_0, b, S) = (q_0, bS)$;
5) $\delta(q_0, b, a) = (q_0, ba)$;
6) $\delta(q_0, b, b) = (q_0, bb)$;
7) $\delta(q_0, c, a) = (q_1, a)$;
8) $\delta(q_0, c, b) = (q_1, b)$;
9) $\delta(q_1, a, a) = (q_1, \varepsilon)$;
10) $\delta(q_1, b, b) = (q_1, \varepsilon)$;
11) $\delta(q_1, \varepsilon, S) = (q_2, \varepsilon)$

Bu avtomat dastlab q_0 holatida bo'lib, kiruvchi zanjir belgilarini s belgisigacha o'qiydi va magazinga yozadi. Bu belgini o'qigandan keyin q_1 holatga o'tadi, so'ngra magazindagi belgilarni kiruvchi zanjirni qolgan belgilari bilan solishtiradi va magazin ustidagi belgi olib tashlanadi. Agar belgilar soni va ketma-ketligi bir hil bo'lsa, avtomat q_2 holatiga o'tib o'z ishini tugallaydi, aks holda avtomat zanjirni qabul qilmaydi.

aabacabaa zanjir uchun DMHAquyidagi taktlar ketma-ketligini bajaradi:

1. $(q_0, aabacabaa, S) \vdash^1$
2. $(q_0, abacabaa, aS) \vdash^1$
3. $(q_0, bacabaa, aaS) \vdash^5$
4. $(q_0, acabaa, baaS) \vdash^3$

5. $(q_0, \quad cabaa, abaa\$) \vdash^{7}$
6. $(q_1, \quad abaa, abaa\$) \vdash^{9}$
7. $(q_1, \quad baa, baa\$) \vdash^{12}$
8. $(q_1, \quad aa, aa\$) \vdash^{9}$
9. $(q_1, \quad a, a\$) \vdash^{9}$
10. $(q_1, \quad \varepsilon, \$) \vdash^{13}$
11. $(q_2, \quad \varepsilon, \varepsilon).$

Agar $R(Q,V,Z,\delta,q_0,z_0,F)$ KMXAda quyidagi shartlar bajarilsa:

1) ***Har qanday $q \in Q, \gamma \in Z^*$, $t \in (V \cup \{\varepsilon\})$ uchun o'tish funksiyasi $\delta(q, t, \gamma)$ bo'sh yoki bitta elementga teng bo'lsa;***
2) ***Har qanday $q \in Q$, $\alpha, \beta \in Z^*$, $t \in (V \cup \{\varepsilon\})$ uchun, $\delta(q, t, \alpha) \neq \varnothing$ va $\delta(q, t, \beta) \neq \varnothing$ unda $\forall \lambda \in Z^*$ uchun $\lambda\alpha \neq \beta$ va $\lambda\beta \neq \alpha$ bo'lsa;***
3) ***Har qanday $q \in Q$, $\gamma \in Z^*$, $t \in V$ uchun $\delta(q, t, \gamma) \neq \varnothing$ unda $\delta(q, \varepsilon, \gamma) = \varnothing$ bo'lsa bunday avtomat, determinantlangan kengaytirilgan magazin hotirali avtomat (DKMHA) deb nomlanadi***

Masalan, arifmetik ifodalar tilini KMHAsi quyidagicha beriladi:

$R(\{q,r\},\{a,b,+,*,(,)\},\{a,b,+,*,(,),S,T,E,S\}, \delta, q, S, \{r\})$

δ:

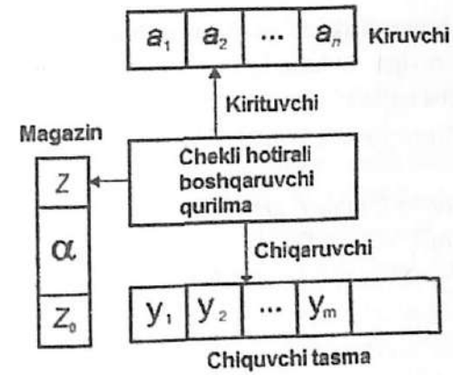
$\delta(q, t, \varepsilon) = \{(q, t)\}$ barcha $t \in V$ uchun; $\delta(q, \varepsilon, S+T) = \{(q,S)\}$;
$\delta(q, \varepsilon, T) = \{(q,S)\}$; $\delta(q, \varepsilon, T*E) = \{(q,T)\}$; $\delta(q, \varepsilon, E) = \{(q,T)\}$;
$\delta(q, \varepsilon, b) = \{(q,E)\}$; $\delta(q, \varepsilon, a) = \{(q,E)\}$; $\delta(q, \varepsilon, (S)) = \{(q,E)\}$;
$\delta(q, \varepsilon, \$S) = \{(r, \varepsilon)\}$.

Bu KMHAda, 1 va 3 shartlar bajariladi, lekin 2 shart bajarilmaydi. Chunki, masalan ikkita bo'sh bo'lmagan funksiyani olsak: $\delta(q, \varepsilon, S+T) = \{(q,S)\}$, $\delta(q, \varepsilon, T) = \{(q,S)\}$, ko'rinib turibdiki T zanjir $S+T$ zanjirni suffiksi bo'lmoqda, shu sababli u nodeterminantlangan avtomat bo'ladi.

Magazin hotirali qayta o'zgartirgichlar

Magazin hotirali qayta o'zgartirgichlar (MHQO') MHAga chiquvchi tasmani qo'shish asosida quriladi. Bu tasmaga chiquvchi zanjir yoziladi.

MHQO'ni modeli 23.2-rasmda keltirilgan.



23.2-rasm. Magazin hotirali qayta o'zgartirgich.

MHQO' deb sakkizta obyekt $D(Q,V,Z,U, \delta,q_0,z_0,F)$ hisoblanadi,. Bunda barcha obyektlarin ma'nosi MHA ta'rifida keltirilgan, faqat U- bu chiquvchi tilnining alifbosi, δ – o'tish funksiyasi $Q \times (V \cup \{\varepsilon\}) \times Z \rightarrow Q \times Z^* \times U^*$ to'plamiga aks ettiradi.

MHQO'ni konfiguratsiyasi to'rta element bilan aniqlanadi $(q,x,\omega,y) \in Q \times V^* \times Z^* \times U^*$ bunda:

- ***q – avtomatning joriy holati,;***
- ***x – hali o'qilmagan dastlabki zanjirining qismi,;***
- ***ω– magazindagi zanjir,;***
- ***y –shu vaktda hosil bo'lgan chiquvchi zanjir.***

Agar $(r,\gamma,\mu) \in \delta(q,t,z)$ bo'lsa, konfiguratsiya o'tishlarini quyidagicha yozamiz $(q, tx, z\omega, y) \vdash (r, x, \gamma\omega, \mu y)$, bunda $q, r \in Q$, $t \in V \cup \{\varepsilon\}$, $x \in V^*$, $z \in Z$, $\gamma, \omega \in Z^*$ va $y, \mu \in U^*$.

O'zgartirgichni boshlang'ich konfiguratsiyasi (q_0,x,z_0,ε) bo'ladi, bu yerda x - kiruvchi zanjir.

O'zgartirgichni chekli konfiguratsiyalar to'plami (q,ε,ω,y), $q \in F$, $\omega \in Z^*$ va $y \in U^*$.

$D(Q,V,Z,U,\delta,q_0,z_0,F)$ MHQO' bilan aniqlangan tarjima deb $\tau(D) = \{(x,y) \mid (q_0,x,z_0,\varepsilon) \vdash^* (q,\varepsilon,\omega,y)$, bunda $q \in F$, $\omega \in Z^*\}$ to'plam nomlanadi.

Agar $\omega = \varepsilon$ bo'lsa, unda MHQO' magazini bo'sh o'zgartirgich bo'ladi.

Misol, bizga infiks shaklida yozilgan arifmetik ifodani, ekvivalent prefiks polyak yozuviga o'tkazadigan, MHQO' berilgan va uning ko'rinishi quyidagicha bo'lsin:

$D(\{q\},\{a,b,+,*,(,)\},\{a,b,+,*,(,),S,T,E\},\{a,b,+,*\},\delta,q,S,\{q\})$,

bunda δ:

1) $\delta(q, \varepsilon, S) = \{(q, S+T, +), (q, T, \varepsilon)\}$;
2) $\delta(q, \varepsilon, T) = \{(q, T*E, *), (q, E, \varepsilon)\}$;
3) $\delta(q, \varepsilon, E) = \{(q, (S), \varepsilon), (q, a, a), (q, b, b)\}$;
4) $\delta(q, a, a) = \{(q, \varepsilon, \varepsilon)\}$;
5) $\delta(q, b, b) = \{(q, \varepsilon, \varepsilon)\}$;
6) $\delta(q, +, +) = \{(q, \varepsilon, \varepsilon)\}$;
7) $\delta(q, *, *) = \{(q, \varepsilon, \varepsilon)\}$;
8) $\delta(q, (, () = \{(q, \varepsilon, \varepsilon)\}$;
9) $\delta(q,),)) = \{(q, \varepsilon, \varepsilon)\}$.

MHQO' o'zgartirgichni kiruvchi tasmasiga $a*(a+b)$ zanjir uzatilsin,va shunda o'zgartirgichning taktlar ketma-ketligi quyidagicha bo'ladi:

1. $(q, a*(a+b), S, \varepsilon)\vdash^{1}$
2. $(q, a*(a+b), T, \varepsilon)\vdash^{2}$
3. $(q, a*(a+b), T*E, *)\vdash^{2}$
4. $(q, a*(a+b), E*E, *)\vdash^{3}$
5. $(q, a*(a+b), a*E, *a)\vdash^{4}$
6. $(q, *(a+b), *E, *a)\vdash^{7}$
7. $(q, (a+b), E, *a)\vdash^{3}$
8. $(q, (a+b), (S), *a)\vdash^{8}$
9. $(q, a+b), S), *a)\vdash^{1}$
10. $(q, a+b), S+T), *a+)\vdash^{1}$
11. $(q, a+b), T+T), *a+)\vdash^{1}$
12. $(q, a+b), E+T), *a+)\vdash^{2}$
13. $(q, a+b), a+T), *a+a)\vdash^{3}$
14. $(q, +b), +T), *a+a)\vdash^{6}$
15. $(q, b), T), *a+a)\vdash^{2}$
16. $(q, b), E), *a+a)\vdash^{3}$
17. $(q, b), b), *a+a)\vdash^{5}$

18. $(q,),), *a+ab)\vdash^{9}$
19. $(q, \varepsilon, \varepsilon, *a+ab)$

O'zgartirgich chekli konfiguratsiyaga o'tdi, zanjir qabul qilindi va $*a+ab$ chiquvchi zanjir tuzildi.

Yukorida ko'rsatilgan o'zgartirgich pasayuvchi tarjimani amalga oshiradi. Ko'taruluvchi tarjimani kengaytirilgan magazin xotirali o'zgartirgich (KMHQO') yordamida modellashtirish mumkin. KMHQO'ni MHQO'dan farqi shundaki uning o'tish funksiyasi, KMHA o'tish funksiyasiga asoslangan.

Misol, bizga infiks shaklida yozilgan arifmetik ifodani, ekvivalent postfiks polyak yozuviga o'tkazadigan, KMHQO' berilgan va uning ko'rinishi quyidagicha bo'lsin:

$D(\{q,r\},\{a,b,+,*,(,)\},\{a,b,+,*,(,),S,T,E,S\},\{a,b,+,*\},\delta,q,S,\{r\})$,

bunda δ:

1) $\delta(q, a, \varepsilon) = \{(q, a, \varepsilon)\}$;
2) $\delta(q, b, \varepsilon) = \{(q, b, \varepsilon)\}$;
3) $\delta(q, +, \varepsilon) = \{(q, +, \varepsilon)\}$;
4) $\delta(q, *, \varepsilon) = \{(q, *, \varepsilon)\}$;
5) $\delta(q, (, \varepsilon) = \{(q, (, \varepsilon)\}$;
6) $\delta(q,), \varepsilon) = \{(q,), \varepsilon)\}$;
7) $\delta(q, \varepsilon, S+T) = \{(q, S, +)\}$;
8) $\delta(q, \varepsilon, T) = \{(q, S, \varepsilon)\}$;
9) $\delta(q, \varepsilon, T*E) = \{(q, T, *)\}$;
10) $\delta(q, \varepsilon, E) = \{(q, T, \varepsilon)\}$;
11) $\delta(q, \varepsilon, b) = \{(q, E, b)\}$;
12) $\delta(q, \varepsilon, a) = \{(q, E, a)\}$;
13) $\delta(q, \varepsilon, (S)) = \{(q, E, \varepsilon)\}$;
14) $\delta(q, \varepsilon, SS) = \{(r, \varepsilon, \varepsilon)\}$.

KMHQO' o'zgartirgichning kiruvchi tasmasiga $a+a*b$ zanjir uzatilsin, shunda o'zgartirgichning taktlar ketma-ketligi quyidagicha bo'ladi:

1. $(q, a+a*b, \$, \varepsilon)\vdash^{1}$
2. $(q, +a*b, \$a, \varepsilon)\vdash^{12}$
3. $(q, +a*b, \$E, a)\vdash^{10}$
4. $(q, +a*b, \$T, a)\vdash^{8}$

5. (q, +a*b, $S , a)|— 3
6. (q, a*b, $S+ , a)|— 1
7. (q, *b, $S+a , a)|— 12
8. (q, *b, $S+E , aa)|— 10
9. (q, *b, $S+T , aa)|— 8
10. (q, b, $S+T* , aa)|— 4
11. (q, ε, $S+T*b, aab)|— 11
12. (q, ε, $S+T*E, aab)|— 9
13. (q, ε, $S+T , aab*)|— 7
14. (q, ε, $S , aab*+)|— 14
15. (r, ε, ε , aab*+)

o‘zgartirgich chekli konfiguratsiyaga o‘tdi, zanjir qabul qilindi va *aab*+* chiquvchi zanjir tuzildi.

Haqiqiy til protsessorini yaratishda determinantlangan magazin xotirali qayta o‘zgartirgichlarni (DMHO‘) va determinantlangan kengaytirilgan magazin xotirali qayta o‘zgartirgichlarni (DKMHO‘) tarjima sifatida qo‘llash ma’quldir. Albatta bunday o‘zgaritirgichlar DMHA va DKMHA avtomatlarga asoslangan.

Misol, bizga prefiks polyak yozuvidagi arifmetik ifodani, ekvivalent postfiks polyak yozuviga o‘tkazadigan, DMHO‘ berilgan va uning ko‘rinishi quyidagicha bo‘lsin:

D({q},{a,b,+,},{+,*,S},{a,b,+,*}, δ,q,S,{q}).*

bunda δ:

1) $\delta(q, a, S) = \{(q, \varepsilon, a)\}$;
2) $\delta(q, b, S) = \{(q, \varepsilon, b)\}$;
3) $\delta(q, +, S) = \{(q, SS+, \varepsilon)\}$;
4) $\delta(q, *, S) = \{(q, SS*, \varepsilon)\}$;
5) $\delta(q, \varepsilon, +) = \{(q, \varepsilon, +)\}$;
6) $\delta(q, \varepsilon, *) = \{(q, \varepsilon, *)\}$;

DMHO‘ o‘zgartirgichning kiruvchi tasmasiga *a+ab zanjir uzatilsin,va shunda o‘zgartirgichning taktlar ketma-ketligi quyidagicha bo‘ladi:

1. (q, *a+ab, S, ε)|— 4
2. (q, a+ab, SS*, ε)|— 1
3. (q, +ab, S*, a)|— 3

4. (q, ab, SS+*, a)|— 1
5. (q, b, S+*, aa)|— 2
6. (q, ε, +*, aab)|— 5
7. (q, ε, *, aab+*)|— 6
8. (q, ε, ε, aab+*)

o‘zgartirgich chekli konfiguratsiyaga o‘tdi, zanjir qabul qilindi va *aab+** chiquvchi zanjir tuzildi.

Xulosa

Kontekstdan erkin grammatika aniqlovchisi magazin hotirali avtomat bo‘ladi. Chekli avtomatdan farqi shundaki unda yana bitta obyekt qo‘shilgan bu magazin va o‘tish funksiyasi nafaqat joriy holat va joriy belgidan, balki magazin ustidagi belgiga bog‘liq. Magazin hotirali avtomat determinantlig va nodeterminantlik bo‘lishi mumkin. Undan tashqari bazi bir aniqlovchilar kengaytirilgan magazin hotirali avtomatlarga asoslangan. Kontekstdan erkin grammatika uchun magazin hotirali avtomat yoki kengaytirilgan magazin hotirali avtomat tuzish mumkin. Berilgan magazin hotirali avtomat asosida kontekstdan erkin grammatikani tiklash mumkin.

Chekli qayta o‘zgartiruvchilar kabi magazin hotirali qayta o‘zgartigichlar xam mavjud. Ular ishlash natijasida chiquvchi zanjir hosil bo‘ladi.

Nazorat uchun savollar

1. Nodeterminantlangan magazin hotirali avtomatning ta’rifini keltiring.
2. MHAda konfiguratsiya, takt, ε-takt iboralarning izohini bering.
3. Nodeterminantlangan kengaytirilgan magazin hotirali avtomatni ta’rifini keltiring.
4. KMHAda konfiguratsiya, takt, ε-takt iboralarning izohini bering.
5. KEG asosida MHAni tuzish algoritmini keltiring.
6. KEG asosida KMHAni tuzish algoritmini keltiring.
7. NMHA asosida KEGni tuzish algoritmini keltiring.
8. Determinantlangan MHAning ta’rifini bering.
9. Determinantlangan KMHAning ta’rifini bering.

10. Nodeterminantlangan magazin xotirali qayta o‘zgartirgichning ta’rifini keltiring.
11. Nodeterminantlangan kengaytirilgan magazin xotirali qayta o‘zgartirgichning ta’rifini keltiring.
12. Determinantlangan magazin xotirali o‘zgartirgichning ta’rifini keltiring.
13. Determinantlangan kengaytirilgan magazin xotirali qayta o‘zgartirgichni ta’rifini keltiring.
14. MHQO‘ orqali tarjimaning ta’rifini keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan $G(VT,VN,P,S)$KEG uchun MHA va KMHA tuzilsin.

a) $S\to L=B|B$
$L\to {}^*B|a$
$B\to L|F$
$F\to Q(i)$

b) $S\to S\vee T|T$
$T\to T\wedge E|E$
$E\to\neg E|a|(S)$

c) $S\to AB|PQx$
$A\to xy|m$
$B\to bC$
$C\to bC|\varepsilon$
$P\to pP|\varepsilon$
$Q\to qQ|\varepsilon$

d) $S\to Aa|bB$
$A\to cAaA|a|\varepsilon$
$B\to cBdd|\varepsilon$

e) $S\to S+T|T$
$T\to T^*E|E$
$E\to .C.$
$C\to /C|/$

f) $S\to$ ***begin*** $D;$ ***Rend***
$D\to dX$
$X\to ,dX|\varepsilon$
$R\to rY$
$Y\to ,rY|\varepsilon$

g) $S\to bAb$
$A\to (L|a$
$L\to Aa)$

h) $S\to SaA|AA|b$
$A\to ASa|Ad|c$

i) $S\to Ab$
$A\to Sa|cB$
$B\to bS|c$

j) $S\to aAa|\varepsilon$
$A\to aSB|dSc$
$B\to b|\varepsilon$

2. Berilgan til uchun DMHA tuzilsin:

a) $L=\{a^nb^mc^n|\ n>0,m\geq 0\}$
b) $L=\{a^nb^mc^md^n|\ n>0,m\geq 0\}$
c) $L=\{w|\ w\in\{0,1\}^*va\ |w_0|=|w_1|\}$
d) $L=\{0^n1^n0^m1^m|\ n,m>0\}$
e) $L=\{0^n1^n|\ n>0\}$
f) $L=\{0^n1^n|\ n>0\}\cup\{1^n0^n|\ n>0\}$
g) $L=\{0^n10^n|\ n>0\}$
h) $L=\{1^{3n-2}0^n|\ n\geq 0\}$
i) $L=\{0^n1^m|\ n>m>0\}$
j) $L=\{0^n1^m|\ n\geq m>0\}$

3. Ko‘rsatilgan tilda berilgan ixtiyoriy zanjirni tarjima qiladigan DMHO‘ tuzilsin:
a) $L=\{a^nb^mc^n|\ n>0,m\geq 0\}$ tildagi zanjirni 1^{n+m} zanjirga;
b) $L=\{a^nb^mc^md^n|\ n>0,m\geq 0\}$ zanjirni 1^n0^{n+m} zanjirga;
c) $L=\{w|\ w\in\{0,1\}^*\}$ zanjirni 1^n0^mzanjirga, bunda n – 1lar soni, m – 0lar soni;
d) $L=\{0^n1^n0^m1^m|\ n,m>0\}$zanjirni 1^{n+m} zanjirga;
e) $L=\{0^n1^n|\ n>0\}$ zanjirni a^{2n} zanjirga;
f) $L=\{0^n1^n|\ n>0\}\cup\{1^n0^n|\ n>0\}$ zanjirni $1^{2n}0^{2n}$ zanjirga;
g) $L=\{1^m0^n|\ n,m>0,\ m\neq n\}$ zanjirni $1^{|n-m|}$ zanjirga;
h) $L=\{1^{3n-2}0^n|\ n\geq 0\}$ zanjirni 1^n0^n zanjirga;
i) $L=\{0^n1^m|\ n,m>0\}$ zanjirni 1^n0^{2m} zanjirga;
j) ixtiyoriy $b_i$zanjirni $(b_{i-1})^r$ zanjirga, bunda b_i i-sonning ikkilik sanoq tizimidagi ko‘rinishi.

24-BOB. ZANJIRLARNI SINTAKTIK TAHLILI

Tayanch iboralar: *pasayuvchi chap tahlil usuli, ko'tariluvchi o'ng tahlil usuli, chap tahlil, o'ng tahlil, yuqoridan pastga tushuvchi tahlilchi, pastdan yuqoriga ko'tariluvchi tahlilchi, pasayuvchi qaytish bilan ishlovchi aniqlovchi, ko'tariluvchi qaytish bilan ishlovchi aniqlovchi.*

Sintaktik tahlil usullari

Barcha sintaktik tahlil usullari, KEG qoidalarini tahlil jarayonida ishlatadi. Sintaktik tahlilni asosiy vazifasi berilgan zanjir uchun, grammatik qoidalar asosida keltirib chiqarish daraxtini qurishdir.

Biz ko'rib chiqadigan tahlilchilarimiz berilgan zanjirlarni chapdan o'nga qarab ko'rib chiqadi. Shu jarayonda daraxtni ikki asosiy usullar bilan qurish mumkin.

Birinchi usulda berilgan zanjir uchun chap keltirib chiqarish bajariladi, ya'ni ishlatilgan grammatik qoidalar yuqoridan pastga qarab aniqlanadi. Bunday jarayon chap tahlil usuli deb nomlanadi. Shu asosda tuzilgan mantiq qurilma esa yuqoridan pastga tushuvchi tahlilchi, deb nomlanadi.

Ikkinchi usulda berilgan zanjir uchun teskari o'ng keltirib chiqarish bajariladi, ya'ni ishlatilgan grammatik qoidalar pastdan yuqoriga qarab aniqlanadi. Bunday jarayon o'ng tahlil usuli, deb nomlanadi. Shu asosda tuzilgan mantiq qurilma pastdan yuqoriga ko'tariluvchi tahlilchi, deb nomlanadi.

Ko'pincha kompilyatorlar sintaktik tahlilni MHQO'ni modellashtirish yo'li bilan amalga oshiradi.

MHQO' kiruvchi zanjir uchun chap keltirib chiqarishni (pasayuvchi tahlil, ya'ni chap tahlilni) amalga oshiradi.

KMHQO' kiruvchi zanjir uchun teskari o'ng keltirib chiqarishni (ko'tariluvchi tahlil, yoki o'ng tahlil) aks ettiradi.

Bizga G(VT,VN,P,S) KEG berilsin va undagi qoidalar 1,2,...,r raqamlab chiqilsin va $\alpha \in (VT \cup VN)^*$ zanjir berilsin.

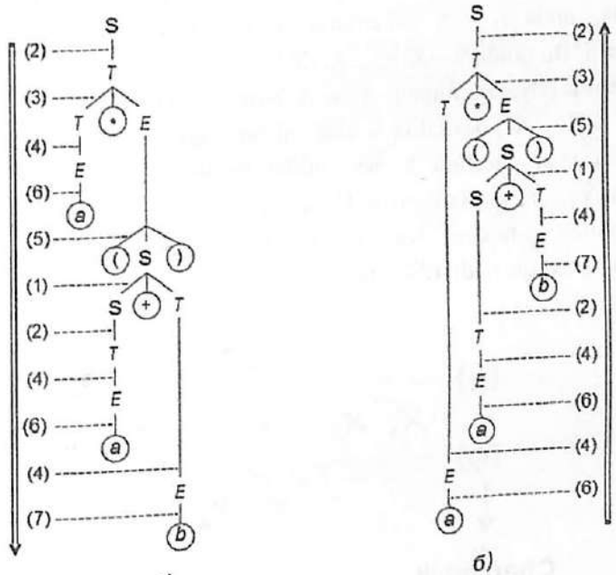
Bunda α zanjirning chap tahlili deb, S noterminal belgidan boshlab, chap keltirib chiqarishda ishlatilgan qoida raqamlarining ketma-ketligiga aytiladi.

Shu qatorda α zanjirning o'ng tahlili deb, S noterminal belgisidan boshlab, o'ng keltirib chiqarishda ishlatilgan qoida raqamlarining teskari ketma-ketligiga aytiladi.

Misol, bizga arifmetik ifodalarni ta'riflaydigan KEG grammatika berilsin: $G(\{a, b, +,*,(,)\},\{S,T,E\},P,S)$ va undagi qoidalar quyidagicha tartiblansin:

P:

1. $S \to S+T$
2. $S \to T$
3. $T \to T*E$
4. $T \to E$
5. $E \to (S)$
6. $E \to a$
7. $E \to b$



24.1-rasm. Keltirib chiqarish daraxtlar: a) chap tahlil uchun, b) o'ng tahlil uchun.

Berilgan $a*(a+b)$ zanjir uchun chap keltirib chiqarishni quramiz:

$S_2 \Rightarrow T_3 \Rightarrow T*E_4 \Rightarrow E*E_6 \Rightarrow a*E_5 \Rightarrow a*(S)_1 \Rightarrow a*(S+T)_2 \Rightarrow a*(T+T)_4 \Rightarrow a*(E+T)_6 \Rightarrow a*(a+T)_4 \Rightarrow a*(a+E) \Rightarrow a*(a+b)$

Shunda berilgan zanjirni chap tahlili *23465124647* ketma-ketlikka to'g'ri keladi.

Xuddi shu zanjir uchun o'ng keltirib chiqarishni tuzamiz:

$S \Rightarrow_2 T \Rightarrow_3 T*E \Rightarrow_5 T*(S) \Rightarrow_1 T*(S+T) \Rightarrow_4 T*(S+E) \Rightarrow T*(S+b) \Rightarrow_2 T*(T+b) \Rightarrow_4 T*(E+b) \Rightarrow_6 T*(a+b) \Rightarrow_1 E*(a+b) \Rightarrow_6 a*(a+b)$

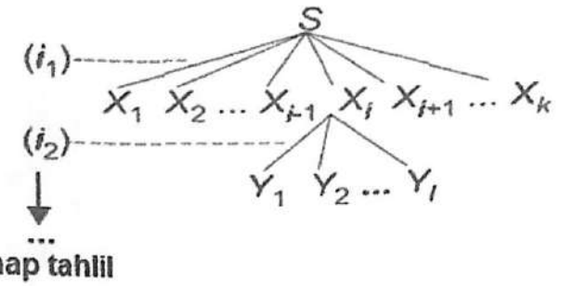
Shunda berilgan zanjirni o'ng tahlili *64642741532* ketma-ketlikka to'g'ri keladi.

Bu tahlillarni daraxtlari 24.1-rasmda keltirilgan:

Pasayuvchi chap tahlil

Agar bizga *w*zanjirni chap tahlili π ketma-ketlik bilan berilgan bo'lsa, keltirib chiqarish daraxti quyidagicha tuziladi.

Daraxtni ildiziga S belgisi qo'yiladi. Faraz qilaylik, $\pi=i_1,i_2,\ldots i_n$ bo'lsin, unda i_1 - S noterminalga qo'llanadigan qoidaning raqamini bildiradi. Bu qoida $S \rightarrow X_1X_2\ldots X_{l-1}X_lX_{l+1}\ldots X_k$ qo'rinishda bo'lsin. Bunda S ildizdan k poyalar chiqadi. Agar X_l birinchi chap noterminal belgi bo'lsa, unda $X_1X_2\ldots X_{l-1}$ dastlabki *w* zanjirni bosh chap qismi bo'ladi. Keyingi i_2 qoidani chap tarafida X_l noterminal belgisi turadi. Faraz qilaylik, bu $X_l \rightarrow Y_1Y_2\ldots Y_m$ qoida bo'lsin. Unda yuqorida ko'rsatilgan amalni X_l uchun qaytaramiz va hokazo. Natijada chapdan o'ngga, yuqoridan pastga tushish orqali daraxt quriladi (24.2-rasm).



24.2-rasm. Chap tahlil uchun daraxt qurish usuli.

Chap tahlilda bizga daraxt ildizi va shoh barglari berilgan. Faqat oradagi cho'qqilarni qurish kerak bo'ladi.

Chap tahlilchi NMHQO' asosida quriladi.

Bizga G(VT,VN,P,S) KEG berilsin va undagi qoidalar 1,2,...,r raqamlab chiqilsin, unda chap tahlilchi deb, $D_l(\{q\},V,VT\cup VN,\{1,2,\ldots p\},\delta,q,S,\{q\})$ NMHQO'ni hisoblaymiz, bunda:

1. ***Agar $A \rightarrow \alpha$ grammatikani qoidasi bo'lsa, u holda $(q, \alpha, i) \in \delta$ (q, ε, A) bo'ladi.***
2. ***Barcha terminallar uchun $\delta(q, a, a)=\{(q, \varepsilon, \varepsilon)\}$ bo'ladi.***

Bunday tahlilchi faqat chap keltirib chiqarishni aniqlaydi va quyidagi ikki amalni bajaradi:

- Magazin ustida turgan noterminal biror bir qoida asosida ***kengaytiriladi***, ya'ni magazin ustiga qoidani chap tarafi noterminal o'rniga yoziladi va bir vaqtda 1-punkt asosida qurilgan δ-funksiyani qiymatiga ko'ra chiquvchi zanjirga qoida raqami yoziladi.
- Kiruvchi zanjirning joriy belgisi magazin ustidagi belgi bilan ***solishtiradi***, agar belgilar bir hil bo'lsa, 2-punkt asosida qurilgan δ-funksiyaga ko'ra belgi magazin ustidan olib tashlanadi. Avtomat kiruvchi zanjirni keyingi belgisiga o'tadi, chiquvchi zanjirga hech narsa yozilmaydi.

Misol, yuqorida ko'rsatilgan arifmetik ifodalar grammatikasi uchun chap tahlilchini, ya'ni nodeterminantlangan magazin hotirali o'zgartirgichni quramiz:

$D_l(\{q\}, \{a,b,+,*,(,)\}, \{a,b,+,*,(,),S,T,E\}, \{1,2,3,4,5,6,7\}, \delta, q, S, \{q\})$, bunda δ:

1) $\delta(q, \varepsilon, S)=\{(q,S+T,1),(q,T, 2)\}$;
2) $\delta(q,\varepsilon,T)=\{(q,T*E,3),(q,E, 4)\}$;
3) $\delta(q, \varepsilon,E)=\{(q,(S), 5),(q,a,6), (q,b,7)\}$;
4) $\delta(q, a,a)=\{(q, \varepsilon, \varepsilon)\}$;
5) $\delta(q, b,b)=\{(q, \varepsilon, \varepsilon)\}$;
6) $\delta(q, +,+)=\{(q, \varepsilon, \varepsilon)\}$;
7) $\delta(q, *,*)=\{(q, \varepsilon, \varepsilon)\}$;
8) $\delta(q, (,()=\{(q, \varepsilon, \varepsilon)\}$;
9) $\delta(q,),))=\{(q, \varepsilon, \varepsilon)\}$;

NMHQO'nikiruvchitasmasiga $a*(a+b)$ zanjirni uzatamiz, shunda qayta o'zgartirgichning taktlar ketma-ketligi quyidagicha bo'ladi:

1.	(q,	a*(a+b),	S, ε	)|— 1
2.	(q,	a*(a+b),	T, 2	)|— 2
3.	(q,	a*(a+b),	T*E, 23	)|— 2
4.	(q,	a*(a+b),	E*E, 234	)|— 3
5.	(q,	a*(a+b),	a*E, 2346	)|— 4
6.	(q,	*(a+b),	*E, 2346	)|— 7
7.	(q,	(a+b),	E, 2346	)|— 3
8.	(q,	(a+b),	(S), 23465	)|— 8
9.	(q,	a+b),	S), 23465	)|— 1
10.	(q,	a+b),	S+T), 234651	)|— 1
11.	(q,	a+b),	T+T), 2346512	)|— 1
12.	(q,	a+b),	E+T), 23465124	)|— 2
13.	(q,	a+b),	a+T), 234651246	)|— 3
14.	(q,	+b),	+T), 234651246	)|— 6
15.	(q,	b),	T), 234651246	)|— 2
16.	(q,	b),	E), 2346512464	)|— 3
17.	(q,	b),	b), 23465124647	)|— 5
18.	(q,	),	), 23465124647	)|— 9
19.	(q,	ε,	ε, 23465124647	)

Chiquvchi zanjir ***23465124647*** kiruvchi "$a*(a+b)$" zanjirni chap tahliliga teng uning daraxti 24.1, a)-rasmda keltirilgan.

Ko'tariluvchi o'ng tahlil

Bizga quyidagi $G(\{a,b,(,)\},\{S,A\},P,S)$ grammatika berilgan bo'lsin:

$(1) S \to (AS)$

$(2) S \to (b)$

$(3) A \to (SaA)$

$(4) S \to (a)$

$((a)(((b)a(a))(b)))$ zanjirni o'ng keltirib chiqarishini ko'rib chiqamiz:

$S \Rightarrow_1 (AS) \Rightarrow_1 (A(AS)) \Rightarrow_2 (A(A(b))) \Rightarrow_3 (A((SaA)(b))) \Rightarrow_4 (A((Sa(a))(b)))$
$\Rightarrow_2 (A(((b)a(a))(b))) \Rightarrow_4 ((a)(((b)a(a))(b)))$

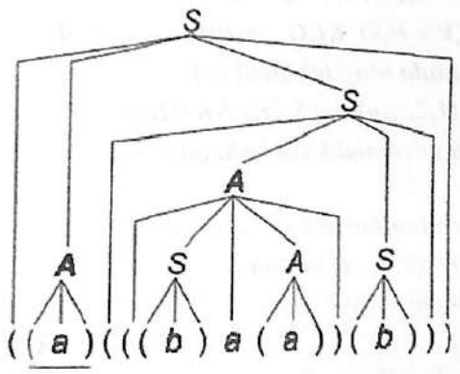
Bunda o'ng tahlil ro'yhatini ***424321*** tashkil qiladi.

S	$\Rightarrow_{(1)}$	-1-	((a)(((b)a(a))(b)))	$\succ_{(4)}$
(AS)	$\Rightarrow_{(1)}$	-2-	(A((b)a(a))(b)))	$\succ_{(2)}$
(A(AS))	$\Rightarrow_{(2)}$	-3-	(A((Sa(a))(b)))	$\succ_{(4)}$
(A(A(b)))	$\Rightarrow_{(3)}$	-4-	(A((SaA)(b)))	$\succ_{(3)}$
(A((SaA)(b)))	$\Rightarrow_{(4)}$	-5-	(A(A(b)))	$\succ_{(2)}$
(A((Sa(a))(b)))	$\Rightarrow_{(2)}$	-6-	(A(AS))	$\succ_{(1)}$
(A(((b)a(a))(b)))	$\Rightarrow_{(4)}$	-7-	(AS)	$\succ_{(1)}$
((a)(((b)a(a))(b)))		-8-	S	

a *б*

24.3-rasm. ((a)(((b)a(a))(b)) zanjirni a) o'ng keltirib chiqarishi, b) o'ng tahlili.

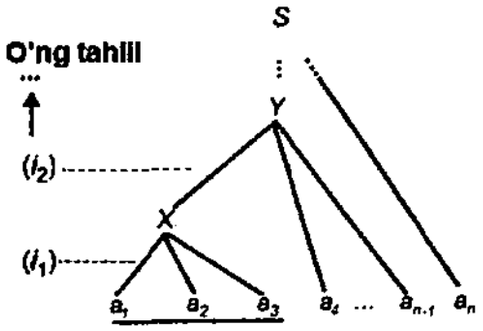
Buning daraxti quyidagicha bo'ladi:



24.4-rasm. ((a)(((b)a(a))(b)) zanjirni keltirib chiqarish daraxti

Agar bizga w zanjirni o'ng tahlili π ketma-ketlik bilan berilgan bo'lsa, keltirib chiqarish daraxti quyidagicha tuziladi. O'ng tahlilda daraxtni qurish, barglaridan boshlanadi. Daraxtni barglariga dastlabki zanjir qo'yiladi. $\pi=i_1,i_2,\ldots i_n$ bo'lsin, unda i_1–qoidani o'ng tarafiga to'g'ri keladigan, chapdan o'ngga qarab, dastlabki zanjirda zanjir qismi ajratiladi va uni o'rniga qoidani chap tarafi (noterminal belgisi) daraxtni yuqori cho'qqisiga qo'yiladi. Va zanjir qismi barglari bilan ulanadi, dastlabki

zanjirdagi zanjir qismi noterminalga o'raladi. Keyingi qadamda i_2 -qoidani o'ng tarafi yana chapdan o'nga qarab zanjir qismi qidiriladi va bu jarayon ketma-ket qaytariladi. Natijada daraxt ildizi qo'yiladi va dastlabki zanjirimiz S noterminalga aylanadi (24.5-rasm).



24.5-rasm. O'ng tahlil uchun daraxt qurish usuli.

O'ng tahlilchi NKMHQO' asosida quriladi uning ta'rifi quyidagicha:

Bizga G(VT,VN,P,S) KEG berilsin va undagi qoidalar 1,2,...,r tartibla chiqilsin, unda o'ng tahlilchi deb, D_r({q},V,VT∪VN,{1,2,...p},δ,q,$,{q}) NKMHQO'ni hisoblaymiz, bunda:

1. ***Agar A→α grammatikani i-chi qoidasi bo'lsa, (q, A, i) ∈δ (q, ε, α) bo'ladi.***
2. ***Barcha terminallar uchun, δ (q, a, ε)={(q, a, ε)} bo'ladi.***
3. ***δ (q, ε, $S)={(q, ε, ε)} bo'ladi.***

Bunday tahlilchi faqat o'ng keltirib chiqarishni aniqlaydi va quyidagi amallarni bajaradi:

- D_r kiruvchi belgilarni 2-punkt bo'yicha hisoblangan δ-funksiya asosida magazin ustiga ketma-ket ***ko'chiradi***.
- Magazin ustida turgan zanjir qismi biror bir qoidani o'ng tarafiga to'g'ri kelsa u noterminalga ***o'raladi***, ya'ni magazin ustiga zanjir qismi o'rniga qoidani chap tarafi yoziladi, va bir vaqtda 1-punkt asosida qurilgan δ-funksiya qiymatiga asosan chiquvchi zanjirga qoidaning raqami yoziladi.
- D_r kiruvchi belgilarni, magazin ustida biror bir qoidaning o'ng tarafi hosil bo'lgunicha, ko'chirishni davom etadi.

- Magazin ustida faqat marker belgisi bilan boshlang'ich belgi qolguncha yuqoridagi jarayon qaytariladi. Natijada D_r ta'rifidagi 3-punkt bo'yicha, o'zgartirgich ohirgi konfiguratsiyaga o'tadi va magazin bo'shatiladi.

Misol, yuqorida ko'rsatilgan arifmetik ifodalar grammatikasi uchun o'ng tahlilchini, ya'ni nodeterminantlangan kengaytirilgan magazin hotirali qayta o'zgartirgichni quramiz:

D_r({q}, {a,b,+,*,(,)}, {a,b,+,*,(,),S,T,E,$}, {1,2,3,4,5,6,7}, δ, q, $, {q}), bunda δ:

1) δ (q, ε,S+T)= {(q,S,1)};
2) δ (q, ε,T)={(q,S, 2)};
3) δ (q, ε,T*E)= {(q,T,3);
4) δ (q, ε,E)={(q,T, 4)};
5) δ (q, ε,(S))= {(q,(E), 5)};
6) δ (q, ε,a)={(q,E,6)};
7) δ (q, ε,b)={(q,E,7)};
8) δ (q, a, ε)= {(q, a, ε)};
9) δ (q, b, ε)= {(q, b, ε)};
10) δ (q, +, ε)= {(q, +, ε)};
11) δ (q, *, ε)= {(q, *, ε)};
12) δ (q, (,ε)= {(q, (, ε)};
13) δ (q,), ε)= {(q,), ε)};
14) δ (q, ε, $S)= {(q, ε, ε)};

NKMHQO' qayta o'zgartirgichni kiruvchi tasmasiga "a*(a+b)"zanjir uzatamiz, shunda o'zgartirgichni taktlar ketma-ketligi quyidagicha bo'ladi:

1. (q, a*(a+b), $, ε)|— 8
2. (q, *(a+b), $a , ε)|— 6
3. (q, *(a+b), $E , 6)|— 4
4. (q, *(a+b), $T , 64)|— 11
5. (q, (a+b), $T* , 64)|— 12
6. (q, a+b), $T*(, 64)|— 8
7. (q, +b), $T*(a , 64)|— 6
8. (q, +b), $T*(E , 646)|— 4

9. (q, +b), ST*(T , 6464)|— 2
10. (q, +b), ST*(S , 64642)|— 10
11. (q, b), ST*(S+ , 64642)|— 9
12. (q,), ST*(S+b , 64642)|— 7
13. (q,), ST*(S+E , 646427)|— 4
14. (q,), ST*(S+T , 6464274)|— 1
15. (q,), ST*(S , 64642741)|— 13
16. (q, ε, ST*(S) , 64642741)|— 5
17. (q, ε, ST*E , 646427415)|— 3
18. (q, ε, ST , 6464274153)|— 2
19. (q, ε, SS , 64642741532)|— 14
20. (q, ε, ε , 64642741532)

Chiquvchi zanjir ***64642741532*** kiruvchi *a*(a+b)* zanjirni o'ng tahliliga teng, uning daraxti 24.1 b)-rasmda keltirilgan.

Qaytish bilan ishlovchi aniqlovchilar

Qaytish bilan ishlovchi aniqlovchilar nodeterminantli magazin hotirali qayta o'zgartirgichlarni modellashtirish bilan amalga oshiriladi. Qayta o'zgartirgich nodeterminantli bo'lgani uchun, uning biror bir taktida bir nechta o'tish konfiguratsiyasi bo'lishi mumkin. Bu vaziyatda aniqlovchi bir konfiguratsiyani tanlaydi, boshqalarini saqlab qoladi va ishini davom etadi. Agar aniqlovchi chekli konfiguratsiyaga o'tsa, zanjir qabul qilingan deb hisoblanadi va ish tugatiladi. Aks holda, aniqlovchi bir nechta qadam orqaga qaytib, boshqa konfiguratsiyani tanlaydi va shu jarayon davom etadi. Agar barcha variantlarni ko'rib chiqgandan keyin aniqlovchi chekli konfiguratsiyaga yetib bormasa, zanjir qabul qilinmaydi. Umumiy holda bunday aniqlovchi n uzunlikdagi zanjirni tahliliga e^n vaqt ketkazadi. Shu sababli, bu usullar amalda kamdan-kam ishlatiladi. Chap tahlil uchun pasayuvchi qaytish bilan ishlovchi aniqlovchi qo'llanadi va o'ng tahlilda ko'tariluvchi qaytish bilan ishlovchi aniqlovchi qo'llanadi.

Pasayuvchi qaytish bilan ishlovchi aniqlovchining ish algoritmi

Agar biror bir noterminal uchun bir nechta qoida mavjud bo'lsa ular muqobil qoidalar deb nomlanadi. Pasayuvchi qaytish bilan ishlovchi algoritm muqobil qoidalarni tanlash asosida ishlaydi. Shuning uchun

bunday aniqlovchilarning boshqa nomi – muqobil tanlash aniqlovchilardir.

Muqobil tanlash algoritmi quyidagicha.

Berilgan *G(VT,VN,P,S)* KEGda qoidalar *1,2,...,r* nomerlab chiqiladi, undan tashqari har bir noterminalni muqobil qoidalari tartiblab chiqiladi. Agar $A \to \gamma_1|\gamma_2| \ldots|\gamma_k$ qoida bo'lsa, bunda $A \in VN$, $\gamma_i \in (VT \cup VN)^*$ va A_j bu A noterminalga tegishli j-chi muqobil qoidani indeksi bo'lsin.

Algoritmda ikkita magazin va sanagich ishlatiladi:

- L_1 magazin (magazinni usti o'ngda). Bu magazinda tanlangan muqobil qoidalar tartib raqami tarixi va kiruvchi zanjirdan o'qilgan belgilari turadi.
- L_2 magazin (magazinni usti chapda). Bu magazinda chap keltirib chiqarish zanjiri turadi.
- Sanagichda kiruvchi zanjirni joriy o'rni saqlanadi (n uzunlikdagi zanjir *"#"* belgisi bilan yakunlanishi kerak bo'ladi).

Algoritm konfiguratsiyasi *(s, i, α, β)* to'rtta elementga bog'liq, bunda :

- s – algoritm holati;
- i– sanagichni qiymati $(1 \le i \le n+1)$;
- α – L_1 magazin ichidagi zanjir, $\alpha \in (VT \cup J)^*$ (J-indekslar to'plami)
- β – L_2 magazin ichidagi zanjir, $\beta \in (VT \cup VN)^*$.

Algoritm uchta holatdan birida bo'lishi mumkin:

- q–normal holat;
- b–qaytish holati;
- r– yakuniy holat.

Algoritmni boshlang'ich qonfiguratsiyasi – $(q,1,\varepsilon,S)$, bunda S boshlang'ich noterminal belgi.

O'tish munosabati (|—) konfiguratsiyalar to'plami ustidan bajariladi.

Agar *(s, i, α, β)*|—*(s', i', α', β')* bo'lsa, algoritm *(s, i, α, β)* joriy konfiguratsiyadan keyingi *(s', i', α', β')* konfiguratsiyaga o'tadi.

Algoritm ishini boshlang'ich holatdan boshlaydi va yakuniy holatgacha, yoki hato sodir bo'lgunicha quyidagi 6-ta qadamni takroriy bajaradi. Har bir qadamda joriy konfiguratsiya, shu qadamda ko'rsatilgan konfiguratsiyaga mos kelishini tekshiradi. Agar mos kelsa, algoritm shu qadamda ko'rsatilgan konfiguratsiyaga o'tadi, aks holda keyingi qadamga

o'tadi. Algoritmda quyidagi turdagi qadamlar mavjud:

1-qadam. *(Kengayish).* $(q,i,\alpha,A\beta) \vdash (q,i,\alpha A_1\gamma_1\beta)$, bunda $A\to\gamma_1$ va γ_1 - A noterminal uchun birinchi muqobillik.

Agar L_2 magazinni ustida turgan belgi noterminal bo'lsa, uning birinchi muqobil qoidasi L_2 ustiga yoziladi va L_1 magazin ustiga A_1 yoziladi.

2-qadam. *(Muvaffaqiyatli solishtirish).* $(q,i,\alpha,t\beta)\vdash (q,i+1,\alpha t,\beta)$, bunda $t\in VT$-terminal belgisi.

Agar i-chi kiruvchi belgi L_2 magazinni ustida turgan belgi bilan bir hil bo'lsa, u L_2 magazindan L_1 magazinga o'tkaziladi va sanagichning qiymati bittaga oshiriladi, ya'ni $i=i+1$.

3-qadam. *(Muvaffaqiyatli yakunlash).* $(q,n+1,\alpha,\varepsilon)\vdash (r,n+1,\alpha,\varepsilon)$.

Kiruvchi zanjir ohirigacha o'qildi, algoritm yakuniy holatga o'tdi va L_1 magazinda chap keltirib chiqarish zanjiri turadi.

4-qadam. *(Muvaffaqiyatsiz solishtirish).* $(q,i,\alpha,t\beta)\vdash (b,i,\alpha,\beta)$.

Agar i-chi kiruvchi belgi L_2 magazinni ustida turgan belgi bilan bir hil bo'lmasa, algoritm qaytish holatiga o'tadi.

5-qadam. *(Kirim bo'yicha qaytish).* $(b,i,\alpha t,\beta)\vdash (b,i-1,\alpha,t\beta)$.

Qaytish holatda L_1 magazin ustida turgan terminal belgi L_2 magazinga qaytariladi.

6-qadam. *(Boshqa muqobilga o'tish).* Dastlabki konfiguratsiya- $(b,i,\alpha A_j,\gamma_j\beta)$ bo'lsa, quyidagi vaziyatlar bo'lishi mumkin:

6.1. Agar A noterminal belgi uchun $A\to\gamma_{j+1}$ boshqa muqobil mavjud bo'lsa, unda $(b,i,\alpha A_j,\gamma_j\beta)\vdash (q,i,\alpha A_{j+1},\gamma_{j+1}\beta)$.

6.2.Agar A uchun boshqa muqobillar bo'lmasa, $(b,i,\alpha A_j,\gamma_j\beta)\vdash (b,i,\alpha,A\beta)$.

6.3.Agar $i=1$, $A=S$ va S uchun boshqa muqobila bo'lmasa, zanjir qabul qilinmaydi va algoritm to'hxtatiladi.

Algoritm muvoffaqiyatli bajarish natijasida, $(r,n+1,\alpha,\varepsilon)$ chekli konfiguratsiyaga o'tadi,. Chap tahlil ro'yhatini hosil qilish uchun α zanjirdan terminallar olib tashlanadi va indekslar mos qoidaning tartib raqamiga almashtiriladi.

Bu algoritm to'g'ri ishlashi uchun, grammatika chap rekursiyasiz bo'lishi lozim. Agar dastlabki grammatika chap rekursiyaga ega bo'lsa, uni qayta o'zgartirib chap rekursiyasiz grammatikaga olib kelish kerak.

Algoritmni ishlashini misolda ko'ramiz.

Arifmetik ifodalarni grammatikasini olamiz va uni chap rekursiyasiz shaklini ko'ramiz.

Arifmetik ifodalarni ta'riflaydigan grammatika berilgan bo'lsin: $G(\{a,+,*,(,)\},\{S,T,E\},P,S)$,unda qoidalarni chap rekursiyasiz shaklini ishlatamiz:

P:T/r	Qoida	Indeksi
1.	$S\to T+S$	S_1
2.	$S\to T$	S_2
3.	$T\to E*T$	T_1
4.	$T\to E$	T_2
5.	$E\to a$	E_1

Berilgan $a+a*a\#$ zanjir uchun algoritmning ishchi qadamlari quyidagicha bo'ladi (o'tish belgini ustida algoritmning qadam raqami yozilgan):

1. $(q, 1, \varepsilon , S)\vdash^{1}$
2. $(q, 1, S_1 , T+S)\vdash^{1}$
3. $(q, 1, S_1T_1 , E*T+S)\vdash^{1}$
4. $(q, 1, S_1T_1E_1 , a*T+S)\vdash^{2}$
5. $(q, 2, S_1T_1E_1a , *T+S)\vdash^{4}$
6. $(b, 2, S_1T_1E_1a , *T+S)\vdash^{5}$
7. $(b, 1, S_1T_1E_1 , a*T+S)\vdash^{6.2}$
8. $(b, 1, S_1T_1 , E*T+S)\vdash^{6.1}$
9. $(q, 1, S_1T_2 , E+S)\vdash^{1}$
10. $(q, 1, S_1T_2E_1 , a+S)\vdash^{2}$
11. $(q, 2, S_1T_2E_1a , +S)\vdash^{2}$
12. $(q, 3, S_1T_2E_1a+ , S)\vdash^{1}$
13. $(q, 3, S_1T_2E_1a+S_1 , T+S)\vdash^{1}$
14. $(q, 3, S_1T_2E_1a+S_1T_1 , E*T+S)\vdash^{1}$
15. $(q, 3, S_1T_2E_1a+S_1T_1E_1 , a*T+S)\vdash^{2}$
16. $(q, 4, S_1T_2E_1a+S_1T_1E_1a , *T+S)\vdash^{2}$
17. $(q, 5, S_1T_2E_1a+S_1T_1E_1a* , T+S)\vdash^{1}$
18. $(q, 5, S_1T_2E_1a+S_1T_1E_1a*T_1 , E*T+S)\vdash^{1}$
19. $(q, 5, S_1T_2E_1a+S_1T_1E_1a*T_1E_1 , a*T+S)\vdash^{2}$
20. $(q, 6, S_1T_2E_1a+S_1T_1E_1a*T_1E_1a, *T+S)\vdash^{4}$

21. $(b, 6, S_1T_2E_1a+S_1T_1E_1a*T_1E_1a, \quad *T+S)|-^{5}$
22. $(b, 5, S_1T_2E_1a+S_1T_1E_1a*T_1E_1, \quad a*T+S)|-^{6.2}$
23. $(b, 5, S_1T_2E_1a+S_1T_1E_1a*T_1, \quad E*T+S)|-^{6.1}$
24. $(q, 5, S_1T_2E_1a+S_1T_1E_1a*T_2, \quad E+S)|-^{1}$
25. $(q, 5, S_1T_2E_1a+S_1T_1E_1a*T_2E_1, \quad a+S)|-^{2}$
26. $(q, 6, S_1T_2E_1a+S_1T_1E_1a*T_2E_1a, \quad +S)|-^{4}$
27. $(b, 6, S_1T_2E_1a+S_1T_1E_1a*T_2E_1a, \quad +S)|-^{5}$
28. $(b, 5, S_1T_2E_1a+S_1T_1E_1a*T_2E_1, \quad a+S)|-^{6.2}$
29. $(b, 5, S_1T_2E_1a+S_1T_1E_1a*T_2, \quad E+S)|-^{6.1}$
30. $(b, 5, S_1T_2E_1a+S_1T_1E_1a*, \quad T+S)|-^{5}$
31. $(b, 4, S_1T_2E_1a+S_1T_1E_1a, \quad *T+S)|-^{5}$
32. $(b, 3, S_1T_2E_1a+S_1T_1E_1, \quad a*T+S)|-^{6.2}$
33. $(b, 3, S_1T_2E_1a+S_1T_1, \quad E*T+S)|-^{6.1}$
34. $(q, 3, S_1T_2E_1a+S_1T_2, \quad E+S)|-^{1}$
35. $(q, 3, S_1T_2E_1a+S_1T_2E_1, \quad a+S)|-^{2}$
36. $(q, 4, S_1T_2E_1a+S_1T_2E_1a, \quad +S)|-^{4}$
37. $(b, 4, S_1T_2E_1a+S_1T_2E_1a, \quad +S)|-^{5}$
38. $(b, 3, S_1T_2E_1a+S_1T_2E_1, \quad a+S)|-^{6.2}$
39. $(b, 3, S_1T_2E_1a+S_1T_2, \quad E+S)|-^{6.2}$
40. $(b, 3, S_1T_2E_1a+S_1, \quad T+S)|-^{6.1}$
41. $(q, 3, S_1T_2E_1a+S_2, \quad T)|-^{1}$
42. $(q, 3, S_1T_2E_1a+S_2T_1, \quad E*T)|-^{2}$
43. $(q, 3, S_1T_2E_1a+S_2T_1E_1, \quad a*T)|-^{2}$
44. $(q, 4, S_1T_2E_1a+S_2T_1E_1a, \quad *T)|-^{2}$
45. $(q, 5, S_1T_2E_1a+S_2T_1E_1a*, \quad T)|-^{1}$
46. $(q, 5, S_1T_2E_1a+S_2T_1E_1aT_1, \quad E*T)|-^{1}$
47. $(q, 5, S_1T_2E_1a+S_2T_1E_1aT_1E_1, \quad a*T)|-^{2}$
48. $(q, 6, S_1T_2E_1a+S_2T_1E_1aT_1E_1a, \quad *T)|-^{4}$
49. $(b, 6, S_1T_2E_1a+S_2T_1E_1aT_1E_1a, \quad *T)|-^{5}$
50. $(b, 5, S_1T_2E_1a+S_2T_1E_1aT_1E_1, \quad a*T)|-^{6.2}$
51. $(b, 5, S_1T_2E_1a+S_2T_1E_1aT_1, \quad E*T)|-^{6.1}$
52. $(q, 5, S_1T_2E_1a+S_2T_1E_1aT_2, \quad E)|-^{1}$
53. $(q, 5, S_1T_2E_1a+S_2T_1E_1aT_2E_1, \quad a)|-^{2}$
54. $(q, 6, S_1T_2E_1a+S_2T_1E_1aT_2E_1a, \quad \varepsilon)|-^{3}$
55. $(r, 6, S_1T_2E_1a+S_2T_1E_1aT_2E_1a, \quad \varepsilon)|-$

Natijada $\alpha = S_1T_2E_1a+S_2T_1E_1aT_2E_1a$, undan terminallarni olib tashlasak, $\alpha = S_1T_2E_1S_2T_1E_1T_2E_1$ bo'ladi. *Yen*di indekslarni mos ravishda qoida raqamlariga almashtiramiz va chap tahlilga ega bo'lamiz ***14523545***, haqiqatdan ham chap keltirib chiqarish usuli shuni beradi:

$S_1 \Rightarrow T+S_4 \Rightarrow E+S_5 \Rightarrow a+S_2 \Rightarrow a+T_3 \Rightarrow a+E*T_5 \Rightarrow a+a*T_4 \Rightarrow a+a*E_5 \Rightarrow a+a*a$

Misolda ishlatilgan algoritmdan ko'rinib turibdiki, qadamlar soni kichkina zanjir uchun alaqancha bo'lib ketdi.

Ko'tariluvchi qaytish bilan ishlovchi aniqlovchining ish algoritmi

Bu aniqlovchi, kengaytirilgan magazin hotirali qayta o'zgartirgich asosida ishlaydi. Lekin u o'tish funksiyasi emas, balki qoidalar orqali bajariladi.

Berilgan $G(VT,VN,P,S)$ KEGda qoidlarni $1,2,...,r$ nomerlab chiqamiz va kiruvchi zanjirni quyidagicha $w=t_1t_2...t_n$, $|w|=n$, $t_i \in VT$ qabul qilamiz.

Algoritmda ikkita magazin va sanagich ishlatiladi:

- L_1 magazin (magazinni usti o'ngda). Bu magazinda, kiruvchi zanjirning, sanagich ko'rsatgan o'rnidan, chap qismida joylashgan zanjir qismini keltirib chiqarishda, ishlatilgan terminal va noterminal belgilari yoziladi.
- L_2 magazin (magazinni usti chapda). Bu magazinda o'tkazish va olmashtirish amallarini ketma-ketligi yoziladi.
- Sanagichda kiruvchi zanjirni joriy o'rni saqlanadi (n uzunlikdagi zanjir "#" belgisi bilan yakunlanishi kerak bo'ladi)

Algoritm konfiguratsiyasi to'rta elementga bog'liq (s, i, α, β), bunda :

- s – algoritm holati;
- i– sanagichni qiymati $(1 \le i \le n+1)$;
- α – L_1 magazin ichidagi zanjir, $\alpha \in (VT \cup VN)^*$
- β– L_2 magazin ichidagi zanjir.

Algoritm uchta holatdan birida bo'lishi mumkin:

- q-normal holat;
- b-qaytish holati;
- r- yakuniy holat;

Algoritmni boshlang'ich qonfiguratsiyasi – $(q,1,\varepsilon,\varepsilon)$.

Algoritm ishini, kiruvchi zanjirdan belgilarni L_1 magazinga o'tkazishdan boshlaydi va yakuniy holatga yoki hato sodir bo'lguncha ketma- ket qadamlarni bajaradi. Har bir qadamda joriy konfiguratsiyani algorimni qadamida ko'rsatilgan konfiguratsiyaga mos kelishi tekshiriladi. Agar mos kelsa, algoritm shu qadamda ko'rsatilgan konfiguratsiyaga o'tadi, va aks holda keyingi qadamga o'tadi. Algoritmda quyidagi turdagi qadamlar mavjud:

1-qadam. *(Almashtirish-o'rash).* $(q,i,\alpha\beta,\gamma) \vdash (q,i,\alpha A,j\gamma)$, bunda $A\to\beta$ j-chi qoida, ya'ni L_1 magazinni ustida biror bir qoidaning o'ng tarafi hosil bo'lsa, u qoidani chap tarafida turgan noterminalga almashtiriladi va birdaniga L_2 magazinga qoidaning tartib raqami yoziladi. Agar yana almashtirish imkoniyati mavjud bo'lsa, 1-qadam qaytariladi,va aks holda 2- qadamga o'tiladi.

2-qadam. *(O'tkazish).*Agar $i<n+1$ bo'lsa, $(q,i,\alpha,\gamma)\vdash (q,i+1,\alpha a_i,0\gamma)$, bunda "0" o'tishni bildiruvchi belgi, u L_2 magazinga yoziladigan raqam, so'ng 1-qadamga o'tiladi. Aks holda 3-qadamga o'tiladi.

3-qadam. *(Yakunlash).*Agar konfiguratsiya - $(q,n+1,S,\gamma)$ bo'lsa, unda $(q,n+1,S,\gamma) \vdash (r,n+1,S,\gamma)$, ya'ni kiruvchi zanjir ohirigacha o'qildi, algoritm yakuniy holatga o'tdi. Algoritm yakunlanadi. Aks holda 4-qadam bajariladi.

4-qadam. *(Qaytish holatiga o'tish).* $(q,n+1,\alpha,\gamma) \vdash (b,n+1,\alpha,\gamma)$.

Agar ohirgi belgi o'qilib, L_1 magazinda fakat S belgisi qolgan bo'lmasa, algoritm qaytish holatga o'tadi.

5-qadam. *(Qaytish).* Dastlabki konfiguratsiya- $(b,i,\alpha A,j\gamma)$ bo'lsa, quyidagi vaziyatlar bo'lishi mumkin:

5.1. Agar $j>0$ va $A\to\beta$ - j-raqamli qoida bo'lsa, va $B\to\beta'$ - $k>j$ raqamli qoida mavjud bo'lib, $\alpha\beta=\alpha'\beta'$ bo'lsa, unda $(b,i,\alpha A_j,j\gamma)\vdash (q,i,\alpha'B,k\gamma)$ va 1-qadamga o'tiladi.

5.2. Agar $i=n+1$, $j>0$ va $A\to\beta$ - j-raqamli qoida bo'lsa, va $\alpha\beta$ zanjir uchun boshqa almashtirishlar yo'q bo'lsa, unda $(b,n+1,\alpha A,j\gamma)\vdash (b,n+1,\alpha\beta,\gamma)$. 5-qadamga qaytiladi

5.3. Agar $i\neq n+1$, $j>0$ va $A\to\beta$- j-raqamli qoida bo'lsa, (va) $\alpha\beta$ zanjir uchun boshqa almashtirishlar yo'q bo'lsa, va unda $(b,i,\alpha A,j\gamma)\vdash (q,i+1,\alpha\beta a_i,0\gamma)$. 1-qadamga o'tiladi.

5.4. Agar L_2 magazin ustida *"0"* raqami va L_1 magazin ustida a_i terminal belgisi bo'lsa, $(b,i,\alpha a_i,0\gamma)\vdash (q,i-1,\alpha,\gamma)$, ya'ni sanagich bittaga kamayadi, L_1 magazindan belgi olib tashlanadi va L_2 magazindan *"0"* olib tashlanadi. 1-qadamga o'tiladi.

5.5. Aks holda hato hisoblanadi, zanjir qabul qilinmaydi, va algoritm to'xtatiladi.

Algoritmni muvoffaqiyatli bajarish natijasida $(r,n+1,S,\gamma)$ chekli konfiguratsiyaga o'tadi, shunda γ zanjirdan barcha "0" raqamlari olib tashlanadi va hosil bo'lgan raqamlar o'ng keltirib chiqarish qoidalarining ketma-ketligi bo'ladi. Uning teskari tartibi o'ng tahlilni beradi.

Bu algoritm to'g'ri ishlash uchun grammatikada bo'sh qoidalar va sikllar bo'lmasligi kerak. Agar bu shartlar bajarilmasa, uni qayta o'zgartirib keltirilgan shaklga olib kelish kerak.

Algoritmni ishlashini misolda ko'ramiz.

Arifmetik ifodalarni grammatikasini olamiz.

Arifmetik ifodalarni ta'riflaydigan grammatika berilgan bo'lsin: $G(\{i,+,*,(,)\},\{S,T,E\},P,S)$, qoidalarga tartib raqamlarni qo'yib chiqamiz:

1. $S\to S+T$
2. $S\to T$
3. $T\to T*E$
4. $T\to E$
5. $E\to i$

Berilgan $i+i*i\#$ zanjir uchun algoritmning ishlash qadamlari quyidagicha bo'ladi:

1. $(q, 1, \varepsilon$, $\varepsilon)\vdash^{2}$
2. $(q, 2, i$, $0)\vdash^{1}$
3. $(q, 2, E$, $50)\vdash^{1}$
4. $(q, 2, T$, $450)\vdash^{1}$
5. $(q, 2, S$, $2450)\vdash^{2}$
6. $(q, 3, S+$, $02450)\vdash^{2}$
7. $(q, 4, S+I$, $002450)\vdash^{1}$
8. $(q, 4, S+E$, $5002450)\vdash^{1}$
9. $(q, 4, S+T$, $45002450)\vdash^{1}$
10. $(q, 4, S$, $145002450)\vdash^{2}$
11. $(q, 5, S*$, $0145002450)\vdash^{2}$
12. $(q, 6, S*i$, $00145002450)\vdash^{2}$

2. Berilgan grammatika uchun o'ng tahlilchini tuzing va berilgan zanjir uchun ishlash taktlarini ko'rsating

a) $E \to E+T \mid T$
$T \to (E) \mid i$

b) $S \to aAb \mid c$
$A \to bS \mid Bb$
$B \to aA$

3. 1-topshiriqda ko'rsatilgan grammatikalar uchun pasayuvchi chap sintaktik tahlilchini tuzing va berilgan zanjir uchun algoritm qadamlarini ko'rsating.
4. 2-topshiriqda ko'rsatilgan grammatikalar uchun ko'tariluvchi o'ng sitaktik tahlilchini tuzing va berilgan zanjir uchun algoritm qadamlarini ko'rsating
5. Berilgan grammatikalar uchun pasayuvchi qaytish bilan ishlovchi aniqlovchini tuzing va berilgan zanjir uchun algoritmni ishlash qadamlarini keltiring

a) $S \to S\hat{}T \mid T$
$T \to T\&E \mid E$
$E \to a$
"a^a&a"

b) $S \to S\text{-}T \mid T$
$T \to T*E \mid E$
$E \to a$
"a-a*a"

6. 5-topshiriqda ko'rsatilgan grammatikalar uchun ko'tariluvchi qaytish bilan ishlovchi aniqlovchini tuzing va berilgan zanjir uchun algoritmni ishlash qadamlarini keltiring.

25-BOB. QAYTMASDAN ISHLOVCHI PASAYUVCHI ANIQLOVCHILAR

Tayanch iboralar: *LL(k) grammatikalar, rekursiv tushish grammatika, rekursiv tushish usuli, FIRST funksiya, FOLLOW funksiya, to'ldirilgan grammatika, s-grammatika, q-grammatika, LL(1) grammatika, 1-bashorat qilish algoritmi .*

LL(k) grammatikalar.

Qaytmasdan ishlovchi pasayuvchi aniqlovchilar KEGni sinf qismlari bilan ishlaydi. Ularni ishlash vaqti chiziqlidir, ya'ni cn-ga teng,va bunda n – kiruvchi zanjirni uzunligi. Bunday grammatikalarga LL(k) grammatikalar kiradi. LL(k) grammatikaning aniqlovchisi kiruvchi zanjirni chapdan o'ngga ko'rib chiqadi (left) va chap keltirib chiqarishni bajaradi(left),. Muqobil qoidalarni tanlashda, joriy belgini o'ng tarafida turgan k belgilar ketma-ketligini hisobga oladi, shuning uchun uning nomi - LL(k). LL(k) grammatikada k>1 bo'lganda tahlil algoritmi murakkablashib ketadi. Shu sababli, amalda asosan LL(1) grammatikalar ishlatiladi va ular uchun aniqlovchilar quriladi. Biz ham shu grammatikani ko'rib chiqamiz.

LL(1) grammatikani sinf qismiga rekursiv tushish grammatikasi kiradi. Uning aniqlovchisi rekursiv tushish usuli (RTU) bilan quriladi. Biz LL(1) grammatikani ko'rib chiqishdan oldin ikkita funksiyani kiritamiz.

FIRST va *FOLLOW* funksiyalar

Kontekstdan erkin tillarni sintaktik tahlil algoritmlarini yaratishda, *FIRST* va *FOLLOW* funksiyalar ko'p ishlatiladi.

FIRST funksiyani argumenti ihtiyoriy zanjir bo'lishi mumkin.

Ta'rif: FIRST(α)=\{t∈VT\| α⇒* tβ\}∪\{ ε\| α⇒*ε\}, ya'ni α zanjirdan keltirib chiqarilgan zanjirlarni bosh terminal belgilarning to'plami FIRST(α) funksiyani qiymati bo'ladi, agar α⇒*ε, unda ε∈FIRST(α).

Har qanday α zanjir uchun FIRST(α) quyidagi qoidalar bilan quriladi:

1. ***Agar α zanjir "a" belgisidan boshlansa, unda FIRST(α)={a}.***
2. ***Agar α⇒*ε, unda ε FIRST(α) to'plamiga qo'shiladi.***
3. ***Agar α zanjir A noterminal belgidan boshlansa, unda FIRST(α) to'plamiga FIRST(A)-{ε} to'plami kiradi.***

Uchinchi qoida rekursivdir. Faraz qilaylik $\alpha=AB\delta$. Agar $A\Rightarrow^*\varepsilon$ bo'lsa, unda biz B noterminal belgidan chiqariladigan barcha boshlang'ich terminal belgilarni $FIRST(\alpha)$ to'plamga qo'shamiz. Agar $B\Rightarrow^*\varepsilon$ bo'lsa, unda biz δdan chiqariladigan barcha boshlang'ich terminal belgilarni $FIRST(\alpha)$ to'plamga qo'shamiz.

Misol, bizga quyidagi qoidalar bilan tuzilgan grammatika berilgan bo'lsin:

$S\rightarrow ABCd$

$A\rightarrow e|f|\varepsilon$

$B\rightarrow g|h|\varepsilon$

$C\rightarrow p|q.$

FIRST($ABCd$) to'plamni aniqlaymiz. Buning uchun $ABCd$dan barcha chap keltirib chiqarishlarni aniqlaymiz:

$ABCd\Rightarrow eBCd;$ $ABCd\Rightarrow fBCd;$

$ABCd\Rightarrow BCd\Rightarrow gCd;$ $ABCd\Rightarrow BCd\Rightarrow hCd;$

$ABCd\Rightarrow BCd\Rightarrow Cd\Rightarrow pd;$ $ABCd\Rightarrow BCd\Rightarrow Cd\Rightarrow qd.$

Bundan ko'rinib turibdiki, FIRST($ABCd$)={e,f,g,h,p,q}.

FOLLOW funksiyaning argumenti ko'pincha noterminal bo'ladi.

Ta'rif: FOLLOW(A)={t∈VT| $S\Rightarrow^*\alpha A\beta$, t∈FIRST(β)}, ya'ni biror bir sentensial shaklda uchragan A noterminal belgidan keyin turgan terminal belgilarning to'plami FOLLOW(A) funksiyaning qiymati bo'ladi.

Ko'pincha dastlabki grammatikada zanjirni tugashini nishonlovchi qo'shimcha terminal belgisi kiritiladi (⊥) va grammatikaga S'→S⊥ yangi boshlang'ich noterminal belgisi qo'shiladi. Shunda dastlabki grammatikani hech qaysi noterminal belgi sentensial shakllarda ohirgi o'ng belgi bo'lmaydi, bunday grammatika ***to'ldirilgan grammatika deb nomlanadi.***

FOLLOW(A) funksiya to'plamini aniqlashda rekursiv algoritmni ishlatish mumkin. Grammatikada A noterminal belgi uchragan qoidalari ketma-ket ko'rib chiqiladi va har bir $B\rightarrow\alpha A\beta$ qoidada FIRST(β)ni ε dan tashqari barcha elementlari *FOLLOW(A)* to'plamiga qo'shiladi. Agar $\beta=\varepsilon$

yoki $\varepsilon\in FIRST(\beta)$(ya'ni $\beta\Rightarrow^*\varepsilon$) kirsa, *FOLLOW(B)*ni barcha elementlari *FOLLOW(A)*ga qo'shiladi.

Misol: Bizga arifmetik ifodalar uchun chap rekursiyasiz grammatika berilsin:

S'→S⊥

S→ TH

$H\rightarrow +TH\mid -TH|\varepsilon$

$T\rightarrow EF$

$F\rightarrow *EF\mid /EF|\varepsilon$

$E\rightarrow (S)\mid a\mid b$

Yechimi

$FIRST(S)=FIRST(T)=FIRST(E)=\{(,a,b\}$

$FIRST(H)=\{+,-,\varepsilon\}$

$FIRST(F)=\{*,/,\varepsilon\}$

$FOLLOW(S)=FOLLOW(H)=\{\perp,)\}$

$FOLLOW(T)=FOLLOW(F)=\{+,-,\perp,)\}$

$FOLLOW(E)=\{+,-,*,/,\perp,)\}$

Rekursiv tushish grammatikasi va uni tahlil qilish algoritmi

Rekursiv tushish grammatikasi bu KEG sinfi bo'lib, unda sintaktik tahlil rekursiv protseduralar orqali bajariladi. Rekursiv tushush usuli *cn* chiziqli vaqtda ishlaydi. Bu algoritmni rekursiv tushish grammatikalar sinfiga qo'llash mumkin. Rekursiv tushish usuli pasayuvchi chap tahlilni qaytmas yo'li bilan amalga oshiradi.

Rekursiv tushish usul aniqlovchisining algoritmi quyidagicha bo'ladi:

1. RTGni har bir noterminal belgisi uchun alohida protsedura tuziladi. Protseduralar nomiga noterminal nomi beriladi. Birinchi bo'lib boshlang'ich belgining protsedurasi chaqiriladi.
2. Ma'lum noterminal belgi uchun tuzilgan protsedura joriy kiruvchi belgini tekshiradi. Faraz qilaylik, bu A noterminal belgi va a joriy belgi hamda A uchun qoidalar $A\rightarrow\alpha_1|\ \alpha_2|\ldots\alpha_n$ bo'lsin. Agar a faqat biror bir FIRST(α_i)ga kirsa, unda tahlil uchun $A\rightarrow\alpha_i$ qoida tanlanadi. Agar a hech qaysi FIRST(α_i)ga kirmasa va qoidalar ichida $A\rightarrow\varepsilon$

mavjud bo'lsa, unda protsedura A ishini tamomlaydi. Agar qoidalar ichida $A \to \varepsilon$ bo'lmasa, protsedura hatolik to'g'risida habar beradi va zanjir qabul qilinmaydi va aniqlovchi ishini to'htatadi.

3. Protsedura tanasida har bir $A \to \alpha_i$ qoida uchun o'zining amallar ketma-ketligi quriladi. Faraz qilaylik, $\alpha_i = X_1 X_2 \ldots X_k$ bo'lsin. Agar X_r noterminal bo'lsa, shu nomli protsedura chaqiriladi. Agar X_r terminal belgi bo'lib, joriy belgiga teng bo'lsa, unda kiruvchi zanjirni keyingi belgisi o'qiladi va boshqarish X_{r+1} o'tiladi, aks holda protsedura hatolik to'g'risida habar beradi.

Misol sifatida quyidagi grammatika berilsin:

$G(\{a,b,c,d\},\{A,B,S\},P,S)$

0. $S' \to S\$$ // qo'shimcha qoida
1. $S \to ABd$
2. $A \to a$
3. $A \to cA$
4. $B \to bA$.

Ushbu grammatika uchun RTU bilan tuzilgan algoritmni dasturi quyidagicha bo'ladi:

```
#include <fstream.h>
#include <iostream.h>
using namespace std;
ifstream fin; // oqimnomi
char c;
void gc(){fin>>c;} // joriybelgi
void A();
void B();
void S()
{
  cout <<"1."; //qoida raqamini chop etish
  A();
  B();
  if (c!='d') throw c; /*xato to'g'risidagi
istisno yaratiladi */
  gc();
}
void A()
{
  if (c=='a'){cout<<"2.";gc();}
  else if (c=='c'){cout<<"3.";gc();A();}
  else throw c;
}
void B()
{
  if (c=='b'){cout<<"4.";gc();A();}
  else throw c;
}
int main(int argc, char* argv[])
{
     fin.open ("data.txt",ios::in);
/* zanjir fayli ochiladi */

  try
    {
gc();S();
     if (c!='$') throw c;
     cout << " zanjir to'g'ri"<<endl;
system("pause");
     return 0;
}
catch (char c)
{
cout <<"zanjir xato:"<<c<<endl;
      system("pause");
return 1;
}
}
```

Dasturga *abcad$* zanjirni uzatsak, natijada *1.2.4.3.2.* chiqadi, va bu chap tahlilga to'g'ri keladi.

Rekursiv tushish grammatikaga qo'yilgan shartlar

Rekursiv tushish usulini qo'llash uchun grammatika qoidalari quyidagicha bo'lishi yetarlikdir:

a) ***$A\to\alpha$, bunda $\alpha\in(VT\cup VN)^*$ va A noterminal uchun bu yagona qoida;***

b) ***$A\to t_1\gamma_1|\ t_2\gamma_2|\dots|\ t_n\gamma_n$, $t_i\in VT$, $\gamma_i\in(VT\cup VN)^*$ va barcha $i=1,2\dots n$ uchun; $t_i\neq t_j$, agar $i\neq j$, ya'ni A noterminal bir nechta qoidalardan (muqobillardan) iborat bo'lib, har bir muqobil har hil terminal belgidan boshlanishi kerak.***

Bunday shartni qonaotlantiruvchi grammatika s-grammatika deb nomlanadi.

Yuqorida ko'rsatilganlar zaruriy shart emas.

RTUni qo'llash uchun grammatikani kanonik shaklga olib kelish ma'quldir.

Agar har bir noterminal uchun qoidalar guruhi quyidagi turlarni biriga kirsa va ko'rsatilgan shartlar bajarilsa, bunday grammatika kanonik shaklda bo'ladi va unga rekursiv tushish usulni qo'llash mumkin:

a) $A\to\alpha$, bunda $\alpha\in(VT\cup VN)^*$ va A noterminal uchun bu yagona qoida.

b) $A\to t_1\gamma_1|\ t_2\gamma_2|\dots|\ t_n\gamma_n$, $t_i\in VT$, $\gamma_i\in(VT\cup VN)^*$ va barcha $i,j=1,2\dots n$ uchun; $t_i\neq t_j$, agar $i\neq j$;

v) $A\to t_1\gamma_1|\ t_2\gamma_2|\dots|\ t_n\gamma_n|\varepsilon$, $t_i\neq t_j$, agar $i\neq j$, $t_i\in VT$, $\gamma_i\in(VT\cup VN)^*$ barcha $i,j=1,2\dots n$ uchun va $FIRST(A)\cap FOLLOW(A)=\varnothing$, bunday shartni qanoatlantiruvchi grammatikalar q-grammatika deb nomlanadi.

Rekursiv tushish usulni kamchiligi shundaki u juda chegaralangan grammatikalar uchun qo'llanadi, undan tashqari har bir grammatika uchun alohida tahlil qilish va alohida protseduralar tuzish kerak bo'ladi. Bu ishni inson bajarib chiqishi kerak bo'ladi, bu esa uning asosiy kamchiligi.

Grammatikani qayta o'zgartirish

Agar berilgan grammatika kanonik shaklda bo'lmasa, uni qayta o'zgartirish bilan RTUni qo'llash mumkin bo'lgan ekvivalent grammatikaga keltirishni harakat qilib ko'rish kerak.

1. Agar grammatikada chap rekursiv qoidalar bo'lsa, ya'ni $A\to A\ \alpha_1|A\ \alpha_2|\dots\ |A\ \alpha_n|\beta_1|\dots|\beta_m$, bunda $\alpha_i\in(VT\cup VN)^*$ barcha $i=1,2\dots n$ uchun; $\beta_j\in(VT\cup VN)^*$ va β_j noterminal Adan boshlanmaydi barcha $j=1,2\dots m$ uchun bo'lsa, RTUni qo'llab bo'lmaydi. Chap rekursiyani o'ng rekursiyaga o'tkazish mumkin:

$A\to\beta_1A'|\dots|\beta_mA'$

$A'\to\alpha_1A'|\alpha_2\ A'|\dots|\ \alpha_n\ A'|\varepsilon$

Hosil bo'lgan grammatikaga dastlabki grammatikaga ekvivalent bo'ladi va unga RTUni qo'llasa bo'ladi.

2. Agar grammatikada biror bir noterminal A uchun bazi bir qoidalarini o'ng tarafi bir hil terminaldan boshlangan bo'lsa, ya'ni $A\to t\alpha_1|t\alpha_2|\dots\ |t\alpha_n\ |\beta_1|\dots|\beta_m$, bunda $t\in VT$; $\alpha_i,\beta_j\in(VT\cup VN)^*$, va β_j tdan boshlanmagan, RTUni bevosita qo'llab bo'lmaydi. Bu qoidalarni chap faktorizatsiya algoritm yordamida qayta o'zgartirib ekvivalent qoidalar olib kelish mumkin:

$A\to tA'\ |\beta_1|\dots|\beta_m$,

$A'\to\ \alpha_1|\ \alpha_2|\dots\ |\alpha_n$

3. Agar grammatikada biror bir noterminal A qodalarni o'ng tarafi noterminaldan boshlangan bo'lsa, ya'ni

$A\to B_1\alpha_1|\dots\ |B_n\alpha_n|t_1\beta_1|\dots|t_m\beta_m$, va

$B_1\to\gamma_{11}|\dots|\ \gamma_{1k},\ \dots\ ,B_n\to\gamma_{n1}|\dots|\ \gamma_{np}$ qoidalar mavjud bo'lsa, bunda $B_i\in VN$, $t_j\in VT$, $\alpha_i,\beta_j,\gamma_{il}\in(VT\cup VN)^*$, A noterminal qoidalarida B_i noterminallar o'rniga ularni qoidalarini qo'yib chiqish mumkin:

$A\to\gamma_{11}\ \alpha_1|\dots|\ \gamma_{1k}\ \alpha_1|\dots\ |\ \gamma_{n1}\ \alpha_n\ |\dots|\ \gamma_{np}\ \alpha_n|\ t_1\beta_1|\dots|t_m\beta_m$.

natijada ekvivalent grammatika hosil bo'ladi.

4. Agar biror bir A noterminal uchun bo'sh qoidali muqobil mavjud bo'lsa, ya'ni $A\to\alpha_1A|\dots|\ \alpha_n\ A|\beta_1|\dots|\beta_m\ |\varepsilon$, $\alpha_i,\beta_j\in(VT\cup VN)^*$, va β_j A belgi bilan tugallanmagan va $FIRST(A)\cap FOLLOW(A)\neq\varnothing$ bo'lsa RTUni qo'llab bo'lmaydi. Bu qoidani qayta o'zgartirish kerak: A boshqa noterminalni qoidasini o'ng tarafida uchrasa, ya'ni $B\to\alpha A\beta$ bo'lsa, uni quyidagicha qayta o'zgartirish mumkin:

$B\to\alpha A$

$A\to\alpha_1A|\dots|\ \alpha_n\ A|\beta_1\beta\ |\dots|\beta_m\beta|\beta$

Hosil bo'lgan grammatika, dastlabki grammatikaga ekvivalent, lekin unga RTUni qo'llash mumkin.

Misol, bizga G grammatika quidagi qoidalar bilan berilgan bo'lsin:

$S \to fASd|\varepsilon$

$A \to Aa|Ab|dB|f$

$B \to bcB|\varepsilon$

Bugrammatikada

$FIRST(B)=\{b\}, FOLLOW(B)=\{a,b,d,f\}$, shu sabali

$FIRST(B) \cap FOLLOW(B) \neq \varnothing$, undan tashqari A noterminal qoidalari chap rekursiyalidir, bundan chiqdi RTU to'g'ridan-to'g'ri qo'llab bo'lmaydi. Grammatikani qayta o'zgartirishimiz kerak.

Birinchi qayta o'zgartirishni A noterminal qoidalariga qo'llaymiz:

$A \to Aa|Ab|dB|f$ qoidalar o'rniga $A \to dBA'|fA'$, $A' \to aA'|bA'|\varepsilon$ qoidalar qo'shamiz.

To'rtinchi qayta o'zgartirishni B noterminal qoidalariga qo'llaymiz:

$B \to bcB|\varepsilon$, $A \to dBA'$ qoidalar o'rniga $A \to dB$, $B \to bcB|A'$ qoidalar qo'shamiz.

Uchinchi qayta o'zgartirishni B noterminal qoidalariga qo'llaymiz:

$B \to bcB|A'$ qoidalarda A' o'rniga uni $A' \to aA'|bA'|\varepsilon$ qoidalarini qo'yib chiqamiz, natijada dastlabki qoidalar o'rniga $B \to bcB|aA'|bA'|\varepsilon$ qoidalar bo'ladi.

Ikkinchi qayta o'zgartirishni B noterminal qoidalariga qo'llaymiz:

$B \to bcB|aA'|bA'|\varepsilon$ qoidalar o'rniga

$B \to bB'|aA'|\varepsilon$, $B' \to cB|A'$ qoidalar qo'shamiz.

Uchinchi qayta o'zgartirishni B' noterminal qoidalariga qo'llaymiz:

$B' \to cB|A'$ qoidalarda A' o'rniga uni $A' \to aA'|bA'|\varepsilon$ qoidalarini qo'yib chiqamiz, natijada dastlabki qoidalar o'rniga $B' \to cB|aA'|bA'|\varepsilon$ qoidalar bo'ladi.

Natijada qayta o'zgartirilgan grammatikamiz quyidagicha bo'ladi:

$S \to fASd|\varepsilon$

$A \to dB|fA'$

$B \to bB'|aA'|\varepsilon$

$B' \to cB|aA'|bA'|\varepsilon$

$A' \to aA'|bA'|\varepsilon$

Bu grammatikada S, B, B', A' noterminallar qoidalari ichida bo'sh qoida bor, shu sababli RTGni kanonik shaklini uchinchi shartini tekshirib

chiqamiz:

1. $FIRST(S)=\{f,\varepsilon\}$, $FOLLOW(S)=\{d\}$;
   $FIRST(S) \cap FOLLOW(S) = \varnothing$
2. $FIRST(B)=\{a,b,\varepsilon\}$, $FOLLOW(B)=\{f,d\}$;
   $FIRST(B) \cap FOLLOW(B) = \varnothing$
3. $FIRST(B')=\{a,b,c,\varepsilon\}$, $FOLLOW(B')=\{f,d\}$;
   $FIRST(B') \cap FOLLOW(B') = \varnothing$
4. $FIRST(A')=\{a,b,\varepsilon\}$, $FOLLOW(A')=\{f,d\}$;
   $FIRST(A') \cap FOLLOW(A') = \varnothing$

Barcha noterminallar uchun shartlar bajarildi, bundan kelib chiqadiki qayta o'zgartirishlar natijasida ekvivalent kanonik shakl hosil bo'ldi va bu grammatikaga RTUni qo'llash mumkin. Kanonik grammatikaga $M \to SS$ qoidani qo'shamiz, A' noterminal belgini nomini Dga, B'ni Cga o'zgartiramiz va qoidalarga tartib raqamini qo'yib chiqamiz, natijada qoidalar quyidagicha bo'ladi:

0. $M \to SS$

1. $S \to fASd$ 2. $S \to \varepsilon$
3. $A \to dB$ 4. $A \to fD$
5. $B \to bS$ 6. $B \to aD$
7. $B \to \varepsilon$ 8. $S \to cB$
9. $S \to aD$ 10. $S \to bD$
11. $S \to \varepsilon$ 12. $D \to aD$
13. $D \to bD$ 14. $D \to \varepsilon$

Bu grammatika uchun rekursiv tushish protseduralar programammasi quyidagicha bo'ladi:

```
#include <fstream.h>
#include <iostream.h>
using namespace std;
ifstream fin; // potoknomi
char c;
void gc(){fin>>c;} // joriybelginikiritish
void A();void B();void C();void D();
void S()
{
```

```
    If (c=='f')
    {cout <<"1."; //qoida raqamini chiqarish
    gc();  A(); S();  if (c=='d') gc(); else throw
c;
        }
     else cout <<"2.";
    }
    void A()
    {
      if (c=='d'){cout<<"3.";gc();B();}
      else if (c=='f'){cout<<"4.";gc();D();}
      else throw c;
    }
    void B()
    {
      if (c=='b'){cout<<"5.";gc();C();}
    else if (c=='a'){cout<<"6.";gc();D();}
      else cout <<"7.";
    }
    void C()
    {
    switch (c)
      {
         case 'c': cout<<"8.";gc();B(); break;
         case 'a': cout<<"9.";gc();D();break;
         case 'b': cout<<"10.";gc();D();break;
         default:  cout <<"11.";
      }
     }
   void D()
   {
     if (c=='a'){cout<<"12.";gc();D();}
     else if (c=='b'){cout<<"13.";gc();D();}
     else cout <<"14.";
   }
```



```
    int main()
    {
      fin.open ("data2.txt"); // zanjir fayli
ochiladi
     try
       {
         gc();S();
         if (c!='$') throw c;
         cout << " zanjir to'g'ri"<<endl;
         system("pause");
         return 0;
    }
    catch (char c)
    {
          cout <<"zanjir hato:"<<c<<endl;
          system("pause");
          return 1;
    }
    }
```

data2.txt faylda yozilgan *"fdbabd"* zanjir uchun dastur 1.3.5.9.13.14.2 chap tahlilni chiqaradi.

LL(1) grammatika

G(VT,VN,P,S) - KEG bo'lsin . Agar har qanday ikkita:

$S\Rightarrow^*_l wA\alpha\Rightarrow^*_l w\beta\alpha\Rightarrow^*_l wx$,

$S\Rightarrow^*_l wA\alpha\Rightarrow^*_l w\gamma\alpha\Rightarrow^*_l wy$,

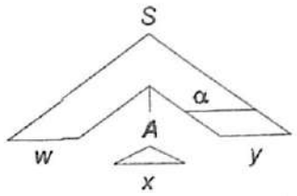
chap tomonli keltirib chiqarishlarda,

FIRST(x)=FIRST(y) teng bo'lsa, unda $\beta=\gamma$ bo'ladi va bunday grammatika LL(1) grammatika deb nomlanadi.

LL(1) grammatika quyidagi hossaga ega: agar berilgan G(VT,VN,P,S) KEGda P to'plamni ikki har hil qoidalari $A\to\beta$ va $A\to\gamma$ uchun barcha sentensial shakllar $wA\alpha$, ya'ni $S\Rightarrow^*_l wA\alpha$ uchun

FIRST(βα)∩FIRST(γα) =∅ bajarilsa, bu grammatika LL(1) grammatika bo'ladi, bunda w∈VT*,α,β, γ∈(VN∪VT)*.

Bundan kelib chiqadiki, LL(1) grammatikada noterminal belgiga tegishli muqobil qoidalarni qaysi birini o'ng tarafiga almashtirish, faqatgina joriy kiruvchi belgi bilan aniqlanadi. 25.1-rasmda *wxy* zanjirni keltirib chiqarish daraxti keltirilgan. Agar shu vaqtgacha $wA\alpha$ zanjir uchun daraxt qurilgan bo'lsa, unda A noterminal belgini almashtirish kerak bo'lgan qoidani aniqlash uchun x zanjirni birinchi belgisini bilish yetarlidir.



25.1-rasm. LL(1) grammatikani tahlili.

Har qanday G(VT,VN,P,S) KEGLL(1) grammatika bo'lish uchun, barcha A noterminal belgilarni ikki har hil qoidalari A→β va A→γ uchun

FIRST(βFOLLOW(A))∩FIRST(γFOLLOW(A))=∅ bajarilishi zaruriy va yetarliligi isbotlangan [1],.

Bunda *FIRST* funksiyani aniqlash sohasi kengaytirilgan, ya'ni argumenti zanjir emas, balki zanjirlar to'plamidir va uning aniqlash sohasi quyidagicha bo'ladi:

$$FIRST(W)=\bigcup_{s\in W} FIRST(s).$$

Bu fikrdan quyidagi ta'rif kelib chiqadi.

Bizga G(VT,VN,P,S) KEG berilsin. Agar grammatikani har qanday muqobil A→α | β juftligi uchun:

(1) ***FIRST(α) ∩FIRST(β) =∅;***

(2) ***agar α⇒*ε kelib chiqib, FIRST(α) ∩FOLLOW(A)=∅ bo'lsa;***

xamda bu shartlar faqat va faqat shu holda bajarilsa, bunday grammatika LL(1) grammatika deb nomlanadi.

Bu ta'rifdan LL(1) grammatika chap rekursiyasiz va bir ma'noligi kelib chiqadi.

Agar grammatika uchun yuqorida ko'rsatilgan shartlar bajarilmasa, unda shartlar bajariladigan ekvivalent grammatika qurish ilojisi bormi degan tabiiy savol tug'iladi.

Ihtiyoriy KEG uchun, ekvivalent LL(1) grammatika tuzish algoritmi yo'q, ya'ni bu muommani algoritmik yechimi yo'q. Lekin ba'zi bir hususiy hollarda dastlabki grammatikani qayta o'zgartish orqali LL(1) grammatikaga keltirish mumkin.

Agar berilgan G(VT,VN,P,S) KEG P to'plamning ikki har hil qoidalari A→β va A→γ uchun barcha sentensial shakllar wAα, ya'ni S⇒*l wAα bo'lsa, bunda w∈VT* va FIRSTk(βα)∩FIRSTk(γα) =∅ bajarilsa, bu grammatika LL(k) grammatika bo'lishi aniq.

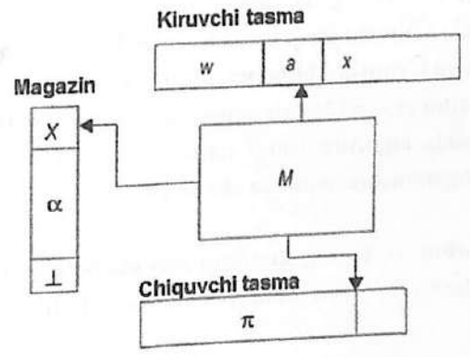
FIRSTk(α) funksiyasi quyidagicha aniqlanadi:

FIRSTk(α)={w∈VT* | agar \|w\|<k va α⇒*w, yoki \|w\|=k va α⇒*wx, bunda x∈VT*, α∈(VN∪VT)*}.

Har qanday LL(k) grammatikani LL(1) grammatikaga keltirish mumkin, shu sababli biz aynan LL(1) grammatikani ko'rib chiqamiz.

LL(1) grammatikaning aniqlovchini algoritmi

LL(1) grammatikani aniqlovchisini 1(bir belgi bo'yicha)- bashorat qilish algoritm asosida qurish mumkin. Bundagi bir – tahlil jarayonida bitta keyingi belgini ko'rish imkoniyati borligini bildiradi. Bu algoritm MHQO' o'hshaydi. Uning ko'rinishi 25.2-rasmda keltirilgan:



Rasm 21.2. LL(1) grammatika algoritmining modeli.

Magazinda $X\alpha\perp$ zanjir turadi, bunda $X\alpha$-magazin belgilardan iborat bo'lgan zanjir (X – magazin ustidagi belgi), ($\perp$) – magazin tagini aniqlovchi belgi(marker). Agar magazin ustidagi belgi tag marker bo'lsa, unda magazin bo'sh deb hisoblanadi. Chiquvchi tasmada qoidalarni π raqamlar ketma-ketligi joylashadi.

Algoritm konfiguratsiyasi (x, $X\alpha\perp$, π)-uchtalik bilan beriladi, bunda x – hali ko'rib chiqilmagan kiruvchi tasmadagi zanjir qismi, $X\alpha$-magazindagi zanjir va π - chiquvchi tasmadagi zanjir. Masalan, 25.2-rasmda (ax, $X\alpha\perp$, π) konfiguratsiya ko'rsatilgan. Magazin belgilar alifbosini V_p bilan belgilaymiz, bu terminal va noterminal belgilar.

Algoritm M-boshqaruv jadval orqali ishlaydi. Bu jadval $(V_p \cup \{\perp\}) \times (VT \cup \{\varepsilon\})$ to'plamni, quyidagi elementlardan iborat bo'lgan, to'plamga aks ettiradi:

2. (β,i), bunda $\beta \in V_p^*$ - i-raqamli qoidani o'ng tarafi.
3. Magazindan bitta belgini chiqarib tashlash – cht.
4. Zanjirni qabul qilish – qq.
5. Hatolik vaziyati - hv.

Algoritm har qadamda kiruvchi zanjirni tahlil qiladi. Faraz qilaylik, $FIRST(x)=\{a\}, a \in VT$, unda $M(X,a)$ boshqaruvchi jadvalni elementiga nisbatan amallar quyidagicha bo'ladi:

1. $(x, X\alpha\perp, \pi) \vdash (x, \beta\alpha\perp, \pi i)$, agar $M(X,a)=(\beta,i)$, ya'ni magazin ustidagi X belgisi $\beta \in V_p^*$ zanjirga almashtiriladi va chiquvchi zanjirga i – qoida raqami (o'ngdan) yoziladi. Kirituvchi surilmaydi.
2. $(ax, a\alpha\perp, \pi) \vdash (x, \alpha\perp, \pi)$, agar $M(a,a)=cht$, ya'ni magazin ustidagi belgi hamda kiruvchi joriy belgi teng bo'lsa, u magazindan chiqarib tashlanadi va kirituvchi bitta belgi o'ngga suriladi.
3. Agar algoritm $(\varepsilon,\perp,\pi)$ konfiguratsiyasiga o'tsa, bu $M(\perp,\varepsilon)=qq$ga mos tushadi, unda algoritm ishini tamomlaydi. Zanjir aniqlovchi bilan qabul qilingan hisoblanadi va chiquvchi zanjir π chap tahlilga to'g'ri keladi.
4. Agar algoritm $(x, X\alpha\perp, \pi)$ konfiguratsiyada bo'lgan holda $M(X,a)=hv$ bo'lsa, algoritm ishini tamomlaydi va hatolik to'g'risida habar beradi.

Algoritmni boshlang'ich konfiguratsiyasi bu $(w,S\perp,\varepsilon)$, bunda w – kiruvchi zanjir, S – grammatikani boshlang'ich belgisi.

LL(1) grammatika uchun M-boshqaruvchi jadvalini tuzish algoritmi. M-boshqaruvchi jadval quyidagi qoidalar bilan tuziladi:

1. Agar $A \to \beta$– grammatikani i-chi raqamli qoidasi bo'lsa, $M(A,a)=(\beta,i)$, $\forall a \in FIRST(\beta)$, $a \neq \varepsilon$. Agar $\varepsilon \in FIRST(\beta)$, unda $M(A,b)=(\beta,i)$, $\forall b \in FOLLOW(A)$
2. $M(a,a)=cht$, $\forall a \in VT$
3. $M(\perp, \varepsilon)=qq$
4. Qolgan holatlarda $M(X,a)=hv$, bunda $X \in \{VT \cup VN \cup \{\perp\}\}$, $a \in \{VT \cup \{\varepsilon\}\}$.

Misol, bizga arifmetik ifodalarni aniqlovchi chap rekursiyasiz grammatika berilsin va uni qoidalari quyidagicha bo'lsin:

1. $S \to TH$
2. $H \to +TH$
3. $H \to \varepsilon$
4. $T \to EF$
5. $F \to *EF$
6. $F \to \varepsilon$
7. $E \to i$
8. $E \to (S)$

Grammatika qoidasi	FIRST, FOLLOWto'plami	M qiymati
1. $S \to TH$	$FIRST(TH)=\{(,i\}$	$M(S,()= M(S,i)=(TH,1)$
2. $H \to +TH$	$FIRST(+TH)=\{+\}$	$M(H,+)= (+TH,2)$
3. $H \to \varepsilon$	$FOLLOW(H)=\{),\varepsilon\}$	$M(H,))= M(H, \varepsilon) =(\varepsilon,3)$
4. $T \to EF$	$FIRST(EF)=\{(,i\}$	$M(T,()= M(T,i)=(EF,4)$
5. $F \to *EF$	$FIRST(*EF)=\{*\}$	$M(F,*)= (*EF,5)$
6. $F \to \varepsilon$	$FOLLOW(F)=\{+,),\varepsilon\}$	$M(F,+)= M(F,))= M(F, \varepsilon) =(\varepsilon,6)$
7. $E \to i$	$FIRST(i)=\{i\}$	$M(E,i)= (i,7)$
8. $E \to (S)$	$FIRST((S))=\{(\}$	$M(E,()= ((S),8)$

Avval M-boshqaruvchi jadvalni tuzamiz. Bu jadvalda 11ta satr va 6ta ustun bo'ladi. Ularning elementlarini aniqlaymiz. Jadvalni satrlar bo'yicha quramiz.

1-qadam. Birinchi amal bo'yicha barcha noterminallarni va ularni qoidalarini ko'rib chiqamiz, natijasi quyidagi jadvalda keltirilgan.

2-qadam. Bir hil terminal belgi bilan nomlangan satr va ustunlar o'rnida *cht* qiymatni qo'yib chiqamiz.

3-qadam. Jadvalni *M(⊥, ε)*elementiga *qq* qiymatini beramiz.

4-qadam. Jadvalni qolgan elementlarini bo'sh qoldiramiz, va bu hato vaziyatiga(*hv*) to'g'ri keladi. Natijada quyidagi jadval hosil bo'ladi:

	I	(	)	+	*	ε
S	*(TH,1)*	*(TH,1)*				
H			*(ε,3)*	*(+TH,2)*		*(ε,3)*
T	*(EF,4)*	*(EF,4)*				
F			*(ε,6)*	*(ε,6)*	*(*EF,5)*	*(ε,6)*
E	*(i,7)*	*((S),8)*				
i	*Cht*					
(		*cht*				
)			*cht*			
+				*cht*		
*					*Cht*	
ε						*cht*
⊥						*qq*

Bizga *i+i*i* kiruvchi zanjir berilsin, unda algoritm ishlash qadamlari quyidagicha bo'ladi:

1. *(i+i*i, S⊥, ε)|—¹ M(S,i)=(TH,1)*
2. *(i+i*i, TH⊥, I)|—¹ M(T,i)=(EF,4)*
3. *(i+i*i, EFH⊥, 14)|—¹ M(E,i)=(i,7)*
4. *(i+i*i, iFH⊥, 147)|—² M(i,i)=cht*
5. *(+i*i, FH⊥, 147)|—¹ M(F,+)=(ε,6)*
6. *(+i*i, H⊥, 1476)|—¹ M(H,+)=(+TH,2)*
7. *(+i*i, +TH⊥, 14762)|—² M(+,+)=cht*
8. *(i*i, TH⊥, 14762)|—¹ M(T,i)=(EF,4)*

9. *(i*i, EFH⊥, 147624)|—¹ M(E,i)=(i,7)*
10. *(i*i, iFH⊥, 1476247)|—² M(i,i)=cht*
11. *(*i, FH⊥, 1476247)|—¹ M(F,*)=(*EF,5)*
12. *(*i, *EFH⊥, 14762475)|—² M(*,*)=cht*
13. *(i, EFH⊥, 14762475)|—¹ M(E,i)=(i,7)*
14. *(i, iFH⊥, 147624757)|—² M(i,i)=cht*
15. *(ε, FH⊥, 147624757)|—¹ M(F, ε)=(ε,6)*
16. *(ε, H⊥, 1476247576)|—¹ M(H, ε)=(ε,6)*
17. *(ε, ⊥, 14762475763)|—¹ M(⊥,ε)=qq*

Xulosa

Qaytmasdan ishlovchi pasayuvchi aniqlovchilar KEGni sinf qismlari bilan ishlaydi. Ularni ishlash vaqti chiziqlidir, ya'ni cn-ga teng,va bunda n – kiruvchi zanjirni uzunligi. Bunday grammatikalarga LL(k) grammatikalar kiradi. LL(1) grammatikani sinf qismiga rekursiv tushish grammatikasi kiradi. Uning aniqlovchisi rekursiv tushish usuli (RTU) bilan quriladi. Rekursiv tushish grammatikaga bir nechta cheklovlar qo'yilgan. Agar qo'yilgan shartlar bajarilsa unda bunday grammatika asosida qurilgan zanjirlarga Rekursiv tushish usulini qo'llash mumkin. Kerak bo'lsa grammatika qoidalarni qayta o'zgartirib rekursiv tushish grammatikaga keltirish mumkin.

LL(1) grammatikada noterminal belgiga tegishli muqobil qoidalarni qaysi birini o'ng tarafiga almashtirish, faqatgina joriy kiruvchi belgi bilan aniqlanadi. LL(1) grammatikani aniqlovchisini 1(bir belgi bo'yicha)-bashorat qilish algoritm asosida qurish mumkin. Bundagi bir – tahlil jarayonida bitta keyingi belgini ko'rish imkoniyati borligini bildiradi.

Nazorat uchun savollar

1. Qanday KEG s-grammatika deb nomlanadi?
2. Qanday KEG q-grammatika deb nomlanadi?
3. FIRST() va FOLLOW() funksiyalarni ta'rifini keltiring.
4. RTUni keltiring.
5. Noterminal uchun protsedura nima asosida va qanday yoziladi?
6. RTU yetarli va zaruriy shartlarni keltiring.

7. KEG RTGga qayta o'zgartirish usullarini keltiring.
8. Qanday KEG LL(1) grammatika deb nomlanadi.
9. LL(1) grammatikani bir-bashorat qilish algoritmini keltiring.
10. Boshqaruv jadvalni tuzish algoritmini keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan grammatikada barcha noterminallar uchun FIRST va FOLLOW funksiyalarni aniqlang, RTU qo'llash mumkinligini aniqlang:

a) $S \to cA \mid B \mid d$
$A \to abA \mid c \mid \varepsilon$
$B \to bSc \mid aAb$

b) $S \to aAbc \mid A$
$A \to bB \mid cBc$
$B \to bcB \mid a \mid \varepsilon$

c) $S \to aSB \mid bAf \mid \varepsilon$
$A \to bAc \mid cS$
$B \to cB \mid d$

d) $S \to aSB \mid bA$
$A \to aS \mid cA \mid \varepsilon$
$B \to bB \mid d$

e) $S \to bABCb \mid d$
$A \to aA \mid cB \mid \varepsilon$
$B \to Sc$
$C \to a\ \{bb\}$

f) $S \to aAb \mid cC$
$A \to a\ \{\ bab\ \}$
$B \to cAc \mid aB \mid \varepsilon$
$C \to Bb$

g) $S \to aA\{xx\}$
$A \to bA \mid cBx \mid \varepsilon$
$B \to bSc$

h) $S \to aSc \mid bA \mid \varepsilon$
$A \to cS\{da\}bA \mid d$

i) $S \to bS \mid aAB$
$A \to bcA \mid ccA \mid \varepsilon$
$B \to cbB \mid \varepsilon$

j) $S \to aASb \mid cfAd$
$A \to bA \mid c \mid \varepsilon$

k) $S \to A \mid B$
$A \to bA \mid \varepsilon$
$B \to dB \mid a \mid \varepsilon$

l) $S \to AS \mid B$
$A \to b \mid c$
$B \to cB \mid b \mid \varepsilon$

2. Berilgan grammatikani kerak bo'lsa, qayta o'zgartiring va RTU bilan aniqlovchini dastursini tuzing:

a) $S \to bS \mid aAB$
$A \to bcA \mid ccA \mid \varepsilon$
$B \to cbB \mid \varepsilon$

b) $S \to aASb \mid cfAd$
$A \to bA \mid c \mid \varepsilon$

c) $S \to Sa \mid Sbb \mid fAc$
$A \to aB \mid d$
$B \to abB \mid Sb$

d) $S \to cAd$
$A \to Aa \mid bB$
$B \to abB \mid \varepsilon$

e) $S \to E\perp$
$E \to ( ) \mid (F) \mid A$
$F \to E/E,F$
$A \to a \mid b$

f) $S \to P := E \mid$ *if E then S* | *if E then S else S*
$P \to I \mid I\ (E)$
$E \to T+E \mid T$
$T \to F*T \mid F$
$F \to P \mid (E)$
$I \to a \mid b$

g) $F \to$ *function I(I)begin S; I:=E; end*
$S \to I:=E;\ S \mid \varepsilon$
$E \to I*E \mid I+E \mid I$
$I \to a \mid b$

h) $S \to SaAb \mid Sb \mid bABa$
$A \to acAb \mid cA \mid \varepsilon$
$B \to bB \mid \varepsilon$

i) $S \to Ac \mid dBea$
$A \to Aa \mid Ab \mid daBc$
$B \to cB \mid \varepsilon$

j) $S \to fASd \mid \varepsilon$
$A \to Aa \mid Ab \mid dB \mid f$
$B \to bcB \mid \varepsilon$

3. Berilgan grammatikalar LL(1)-grammatika bo'lishini tekshiring:

a) $S \to BA$
$A \to BS \mid d$
$B \to aA \mid bS \mid c$

b) $S \to R \mid (S)$
$R \to E=E$
$E \to i \mid (E+E)$

c) $S \to aABbCD|\varepsilon$
$A \to Asd \mid \varepsilon$
$B \to Sac \mid eC \mid \varepsilon$
$C \to Cf|Cg| \varepsilon$
$D \to aBD| \varepsilon$

d) $S \to$ **begin** $D;L$ **end**
$D \to d,D \mid \varepsilon$
$L \to sQ|\varepsilon$
$Q \to :Q,|\varepsilon$

e) $S \to aAB|bBS|\varepsilon$
$A \to cBS \mid \varepsilon$
$B \to dB \mid \varepsilon$

4. Quyidagi qoidalar bilan berilgan LL(1)- grammatika uchun Bir-bashorat qilish algoritm asosida aniqlovchini quring:

a) $S \to Aa|Bd$
$A \to aA \mid \varepsilon$
$B \to cB| \varepsilon$

b) $S \to aAA|bSA|cA$
$A \to aAS|bSS|cS\backslash d$

c) $S \to aR|(S)R$
$A \to \hat{}aR \mid \varepsilon$

d) $S \to aAbBbS|\varepsilon$
$A \to aBC|bA$
$B \to aB|\varepsilon$
$C \to cC|\varepsilon$

e) $S \to ZC$
$Z \to +|-|\varepsilon$
$C \to aAD$
$D \to .dA|\varepsilon$
$A \to aA|\varepsilon$

26-BOB. QAYTMASDAN ISHLOVCHI KO'TARILUVCHI ANIQLOVCHILAR

Tayanch iboralar: *oldin keluvchi grammatika, operatorli oldin keluvchi grammatika, negiz, munosabatlar, "surish-o'rash" algoritmi, oddiy oldin keluvchi grammatika, munosabatlar matritsasi, operatorli oldin keluvchi grammatika, "qobirg'a" grammatika.*

Oldin keluvchi grammatikalar

Qaytmasdan ishlovchi ko'tariluvchi aniqlovchilar KEGni sinf qismlari bilan ishlaydi. Ularni ishlash vaqti chiziqli, ya'ni cn-ga teng, bunda n – kiruvchi zanjirni uzunligi. Bunday grammatikalarga avvalo oldin keluvchi grammatika kiradi. Bu grammatikalarni ham bir nechta turi mavjud, shu jumladan:

- oddiy oldin keluvchi;
- kengaytirilgan oldin keluvchi;
- bo'shatilgan oldin keluvchi;
- operatorli oldin keluchi grammatikalar.

Biz oddiy va operatorli oldin keluvchi grammatikalarni ko'rib chiqamiz.

Oldin keluvchi munosabatlar

Sentensial shaklni biror bir qismi biror bir qoidani o'ng tarafiga mos kelsa, bu qism negiz deb nomlanadi.

Ko'tariluvchi tahlilda asosiy muammo bu negizni aniqlash va uni qoidani chap tarafdagi noterminal belgiga almashtirishdir. Bunday masalani hal qilish uchun oldin keluvchi grammatikani ishlatish mumkin. Bunday grammatikalar oldin keluvchi munosabat tushunchasiga asoslangan.

Sentensial shaklda faqat ikkita belgi orqali negizni boshi va ohirini aniqlashga harakat qilib ko'ramiz.

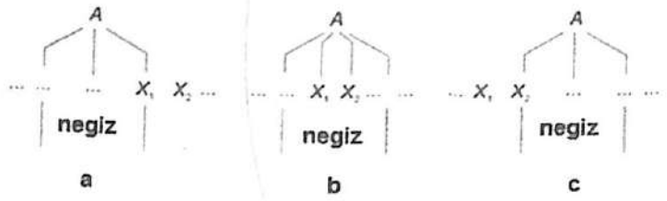
Bizda biror bir $\alpha X_1 X_2 \beta$ sentensial shakl mavjud bo'lsin, bunda $\alpha,\beta \in V^*$- ihtiyoriy zanjirlar, $X_1,X_2 \in V$ terminal yoki noterminal belgilar.

Tahlil jarayonida X_1 yoki X_2 yoki ikkalasi negizga kirishi mumkin. Shunda uchta holat bo'lishi mumkin.·

Birinchi holatda X_1 belgi negizni ohirgi belgisi (eng o'ng) bo'lib, X_2 belgi negizga kirmaydi (26.1.a-rasm), ya'ni X_1 belgi X_2 belgidan oldin o'raladi, bunday munosabat $X_1 \cdot> X_2$ ko'rinishda yoziladi. Bu holatda X_2 negizdan keyin keluchi belgi hisoblanadi va X_1 mos qoidaning o'ng tarafidagi ohirgi belgi bo'ladi.

Ikkinchi holatda X_1 va X_2 belgilar negizga kiradi (26.1.b-rasm) va ular baravariga o'raladi,. Bunday munosabat $X_1 = \cdot X_2$ ko'rinishda yoziladi. Bu holatda X_1 va X_2 negizni tashkil qiluvchi belgilar deb hisoblanadi, ya'ni ular baravariga biror bir qoidani o'ng tarafiga kiradi, bunday munosabat $X_1 = \cdot X_2$.

Uchinchi holatda X_2 belgi negizni bosh belgisi (eng chap) bo'lib, X_1 belgi negizga kirmaydi (26.1.c-rasm), ya'ni X_2 belgi X_1 belgidan oldin o'raladi,. Bunday munosabat $X_1 <\cdot X_2$ ko'rinishda yoziladi va bu holatda X_1 negizdan oldin keluchi deb hisoblanadi,. X_2 mos qoidani o'ng tarafdagi zanjirni birinchi belgisi bo'ladi.



26.1-rasm. X_1 va X_2 belgilarni $\alpha X_1 X_2 \beta$ sentensial shaklni negiziga kirish uchta holati.

Munosabatlar kommutitav va assotsiativ hossalarga ega emas, ya'ni $X_1 <\cdot X_2$ dan $X_2 \cdot> X_1$ kelib chiqmaydi.

Munosabatlar asosida oldin keluvchi grammatikada kiruvchi zanjir tahlil qilinadi. Tahlil quyidagicha bajaraladi.

Bizga $\alpha\gamma\beta$ sentensial shakl berilsin, bunda γ *negiz* bo'lsin.

$\perp_b$	α	a	c	γ	d	b	β	$\perp_o$
		$<\cdot$	$=\cdot$	$=\cdot\ =\cdot\ =\cdot$	$=\cdot$	$\cdot>$		

Bu shaklda α zanjirning ohirgi belgisi "a" bo'lib, β zanjirni birinchi belgisi "b" bo'ladi, ya'ni $a <\cdot c$, $d \cdot> b$. γ zanjirning belgilari orasida "$=\cdot$" munosabat bo'ladi, ya'ni $c = \cdot \ldots = \cdot \ldots = \cdot d$, bunda "$c$" va "$d$" negizni tashkil qilgan birinchi va ohirgi belgilari.

Tahlil jarayonida magazin hotira qo'llanadi.

«Surish-o'rash» algoritmi bo'yicha tahlil jarayonida, magazin ustidagi belgi va kiruvchi zanjirning joriy belgisi orsida "$<\cdot$" yoki "$=\cdot$" munosabatlar bo'lsa, unda surish(ko'chirish) amalini bajariladi, ya'ni joriy belgi magazin ustiga ko'chiriladi va kirituvchini keyingi belgiga suriladi. Bu amalni ular orasida "$=\cdot$" munosabat amalga oshguncha bajarish kerak, "$\cdot>$" munosabat uchraganda, o'rash amali bajariladi, ya'ni magazin ustidagi negiz (zanjir qismi) mos qoidaning chap tarafidagi noterminal belgiga o'raladi. Negizga magazin ustidagi "$=\cdot$" munosabatda bo'lgan belgilar kiradi. Agar magazin ustida faqat boshlang'ich belgi qolsa va kiruvchi zanjir to'liq o'qilsa, tahlil muvaffaqiyatli bajarilgan deb hisoblanadi, va zanjir qabul qilinadi.

Oddiy oldin keluvchi grammatika

Oddiy oldin keluvchi grammatikani rasmiy ta'rifi quyidagicha.

Agar $G(VT,VN,P,S)$, $V=VT \cup VN$ keltirilgan KEG quyidagi shartlarni qanoatlantirsa:

1. ***Har biri tartiblangan X_i va X_j belgilar orasida uchta munosabatdan fakat bittasi bajarilishi yoki umuman munosabat bo'lmasligi mumkin:***

   a) agar grammatikada $A \rightarrow \alpha X_i X_j \beta$ qoida mavjud bo'lib, $A \in VN$, $\alpha,\beta \in V^*$ bo'lsa, bu holda $X_i = \cdot X_j$ ($\forall X_i, X_j \in V$) bo'ladi, ya'ni ikkala belgi negizga kiradi;

   b) agar grammatikada $A \rightarrow \alpha X_i B \beta$ qoida mavjud bo'lib va $B \Rightarrow^+ X_j \gamma$ kelib chiqib, $A,B \in VN$, $\alpha,\beta,\gamma \in V^*$ bo'lsa, bu holda $X_i <\cdot X_j$ ($\forall X_i, X_j \in V$) bo'ladi, ya'ni X_j negizni birinchi belgisi bo'lib, X_i negizdan oldin keluvchi belgi;

   v) agar grammatikada $A \rightarrow \alpha C X_j \beta$ qoida mavjud bo'lib va $C \Rightarrow^* \gamma X_i$ kelib chiqsa yoki $A \rightarrow \alpha C D \beta$ qoida mavjud bo'lib va $C \Rightarrow^* \gamma X_i$, $D \Rightarrow^* X_j \delta$ kelib chiqsa, ya'ni X_i negizni ohirgi belgisi

bo'lib, X_j keyingi negizni birinchi belgisi bo'lsa, unda $X_i \cdot > X_j$ ($\forall X_i, X_j \in V$) bo'ladi, ya'ni X_i negizni ohirgi belgisi bo'lib, X_j negizdan keyin keluvchi birinchi belgisi., Bu yerda $A,C,D \in VN$, $\alpha,\beta,\gamma,\delta \in V^*$;

2. Grammatikaning turli xil qoidalaridagi o'ng taraflari turlicha bo'lishi kerak, ya'ni bir hildagi o'ng tarafli qoidalar bo'lmasligi kerak;

Bunday grammatika oddiy oldin keluvchi grammatika deb nomlanadi

Grammatikadagi munosabatlar asosida oddiy oldin keluvchi munosabat matritsasi quriladi. Bu matritsaning satrlari munosabatlarning chap tarafidagi, ustunlariga esa o'ng tarafida yozilgan belgilar to'g'ri keladi. Munosabatni chap tarafida turgan belgiga to'g'ri kelgan satr va o'ng tarafida turgan belgiga to'g'ri kelgan ustun kesishmasidagi katakka munosabat belgisi yoziladi. Oddiy, oldin keluvchi grammatikada ikki belgi orasida, ularga mos kelgan katagida faqat bitta munosabat bo'lishi mumkin yoki munosabat bo'lmasligi ham mumkin, bunda katak bo'sh bo'ladi.

Oddiy, oldin keluvchi munosabat matritsasini qurishda, $L(A)$ va $R(A)$ funksiyalar kiritiladi. $L(A)$ funksiyasi bu A noterminaldan keltirib chiqarilgan shakllardagi eng chap belgilar to'plami, $R(A)$ funksiyasi bu eng o'ng belgilar to'plamidan iborat. Bu funksiyalarni ta'rifi quyidagicha:

$$L(A)=\{X \mid \exists A \Rightarrow^* X\alpha\};$$

$$R(A)=\{X \mid \exists A \Rightarrow^* \alpha X\}.$$

Bu yerda $A \in VN$-noterminal, $X \in V$-biror bir belgi, $\alpha \in V^*$-biror bir zanjir.

Agar $L(A)$ va $R(A)$ funksiyalar aniqlangan bo'lsa, oldin keluvchi munosabatlarni quyidagicha ta'riflash mumkin:

a) agar $\exists A \rightarrow \alpha X_i X_j \beta \in P$ bo'lib, bu yerda $A \in VN$, $\alpha,\beta \in V^*$ bo'lsa, u holda $X_i = \cdot X_j$ ($\forall X_i, X_j \in V$) bo'ladi;

b) agar $\exists A \rightarrow \alpha X_i D \beta \in P$ *va* $X_j \in L(D)$ bo'lib, bu yerda $A,D \in VN$, $\alpha,\beta \in V^*$ bo'lsa, $X_i < \cdot X_j$ ($\forall X_i, X_j \in V$) bo'ladi;

v) agar $\exists A \rightarrow \alpha C X_j \beta \in P$ *va* $X_i \in R(C)$ yoki $\exists A \rightarrow \alpha C D \beta \in P$ va $X_i \in R(C)$, $X_j \in L(D)$ bo'lib, bu yerda $A,C,D \in VN$, $\alpha,\beta \in V^*$ bo'lsa, $X_i \cdot > X_j$ ($\forall X_i, X_j \in V$) bo'ladi.

Munosabatlarni bunday ta'riflash amalda qulayliklar yaratadi, chunki

keltirib chiqarish amallari kerak emas,. $L(A)$ va $R(A)$ funksiyalar esa faqat grammatika qoidalar orqali quyidagi algoritm bo'yicha aniqlanadi.

Har bir noterminal A uchun quyidagi qadamlar ketma-ket bajariladi:

1-qadam. $\forall A \in VN$:

$L_0(A)=\{X \mid A \rightarrow X\alpha; X \in V, \alpha \in V^*\}$;

$R_0(A)=\{X \mid A \rightarrow \alpha X; X \in V, \alpha \in V^*\}$.

ya'ni A noterminal uchun barcha qoidalarning o'ng tarafidagi birinchi belgisi $L_0(A)$ funksiyaga qo'shiladi, ohirgi belgisi $R_0(A)$ funksiyaga qo'shiladi. Agar qoidaning o'ng tarafi yagona belgidan iborat bo'lsa, u ikkala funksiyaga qo'shiladi. $i=1$;

2-qadam. $\forall A \in VN$:

$L_i(A)=L_{i-1}(A) \cup L_{i-1}(B)$, $\forall B \in (L_{i-1}(A) \cap VN)$;

$R_i(A)=R_{i-1}(A) \cup R_{i-1}(B)$, $\forall B \in (R_{i-1}(A) \cap VN)$.

Agar $L_{i-1}(A)$ga biror bir B noterminal belgi kirsa, unda $L_{i-1}(B)$ga kirgan barcha belgilar $L_{i-1}(A)$ga qo'shiladi va $L_i(A)$ hosil bo'ladi.

Huddi shunday $R_i(A)$ aniqlanadi;

3-qadam. Agar biror bir noterminal A uchun $L_i(A) \neq L_{i-1}(A)$ yoki $R_i(A) \neq R_{i-1}(A)$ bo'lsa, $i=i+1$ va 2-qadamga, aks holda keyingi qadamga o'tiladi.

4-chi qadam. $L(A)=L_i(A)$, $R(A)=R_i(A)$, $\forall A \in VN$.

$L(A)$ va $R(A)$ funksiyalar aniqlangandan so'ng oddiy oldin keluvchi munosabatlar matritsasi quriladi. Undan tashqari zanjirni boshini va ohirini belgilovchi ikkita qo'shimcha "$\perp_b$", "$\perp_o$" belgilar kiritiladi. (va) Bu belgilar uchun munosabatlar quidagicha ta'riflanadi:

agar $\exists S \Rightarrow^* X_j \alpha$ yoki $X_j \in L(S)$ kirsa, $\perp_b < \cdot X_j$, $X_j \in V$ bo'ladi;

agar $\exists S \Rightarrow^* \alpha X_i$ yoki $X_i \in R(S)$ kirsa, $X_i \cdot > \perp_o$, $X_i \in V$ bo'ladi.

Oddiy oldin keluvchi munosabatlar matritsasini qurish algoritmi

Oddiy, oldin keluvchi grammatikani barcha belgilarini (oldin terminal va keyin noterminal belgilar) tartiblab chiqamiz va ular orqali satrlar va ustunlarni belgilab chiqamiz. "$\perp_b$" belgi uchun ostiga satr qo'shamiz hamda "$\perp_o$" uchun ohiriga ustun qo'shamiz. Shunda satrlar va ustunlar soni $k+1$ga teng bo'ladi, bunda k - umumiy belgilar soni. Faraz qilamiz, i-chi satr X_i va j-chi ustun X_j belgiga tegishli bo'lsin.

Matritsani qurish algoritmi quyidagicha bo‘ladi:

1-qadam. i=1, birinchi belgi tanlanadi.

2-qadam. Har bir X_j uchun $A \to \alpha X_i X_j \beta$ kabi qoidalar qidiriladi. Agar bunday qoida mavjud bo‘lsa, i-satr va j-ustun kesishmasidagi katakka “=.” belgi qo‘yiladi.

3-qadam. $A \to \alpha X_i D \beta$ qoidalar qidiriladi. Agar bunday qoida mavjud bo‘lsa, har bir $X_j \in L(D)$ uchun i-satr va j-ustun kesishmasidagi katakka “<.” belgi qo‘yiladi.

4-qadam. $A \to \alpha S X_j \beta$ qoidalar qidiriladi. Agar bunday qoida mavjud bo‘lsa, har bir $X_j \in R(C)$ uchun j-satr va i-ustun kesishmasidagi katakka “.>” belgi qo‘yiladi.

5-qadam. $A \to \alpha S D \beta$da $X_i \in L(D)$ kirgan qoidalar qidiriladi. Agar bunday qoida mavjud bo‘lsa, har bir $X_j \in R(C)$ uchun j-satr va i-ustun kesishmasidagi katakka “.>” belgi qo‘yiladi.

6-qadam. $i=i+1$.Agar $i \le k$ bo‘lsa 2-qadamga o‘tiladi, aks holda keyingi qadamga o‘tiladi.

7-qadam. Barcha $X_j \in L(S)$ ohirgi satr va j-ustunga “<.” belgi qo‘yib chiqiladi.

8-qadam. Barcha $X_i \in R(S)$ ohirgi ustun va i-satrga “.>” belgi qo‘yib chiqiladi.

Oddiy oldin keluvchi grammatika uchun “surish-o‘rash” algoritmi

Bu algoritm uch bosqichdan iborat:

birinchi bosqichda L(A) va R(A) funksiyalarni hisoblanadi.

ikkinchi bosqichda oddiy, oldin keluvchi munosabatlar matritsasini quriladi.

uchinchi bosqichda quyidagi algoritm qo‘llanadi.

Bu algoritm bir holatli kengaytirilgan magazin hotirali qayta o‘zgartirgichga asoslangan. Oldin keluvchi munosabatlar amalni tanlash uchun ishlatiladi (surish yoki o‘rash).

Algoritm uchta elementli konfiguratsiyaga (w,α,π) ega bo‘lib, bu yerda w-hali o‘qilmagan kiruvchi zanjirning qismi, α-magazindagi zanjir, π- chiquvchi zanjir. Algoritmning boshlang‘ich konfiguratsiyasi ($aw\perp_o$, $\perp_b$, ε) bo‘ladi. Ya’ni kiruvchi zanjir to‘liqligicha bo‘ladi, magazinda zanjir boshlanish belgisi turadi, chiquvchi zanjir bo‘sh bo‘ladi. Algoritmning bir

qadami quyidagicha bo‘ladi:

1. Agar magazinning cho‘qqisidagi X belgi va kiruvchi zanjirning a joriy belgisi o‘rtasida $X=\cdot a$ yoki $X<\cdot a$ munosabat mavjud bo‘lsa, surish amali bajariladi:

$(aw\perp_o, \perp_b\alpha X, \pi) \vdash_s (w\perp_o, \perp_b\alpha Xa, \pi)$,

ya’ni kiruvchi zanjirning joriy belgisi magazinga ko‘chiriladi, va kirituvchi qurilmasini bitta o‘ngga surib, keyingi kiruvchi belgiga o‘tiladi.

2. Agar munosabat $X\cdot>a$ bo‘lsa, o‘rash amali bajariladi:

$(aw\perp_o, \perp_b\alpha\gamma X, \pi) \vdash_o (aw\perp_o, \perp_b\alpha A, \pi i)$,

ya’ni magazin ustidagi =· bilan bog‘langan belgilar (γX) negizni tashkil qilgan holda, uni magazindan olib tashlab, o‘rniga mos bo‘lgan $(A \to \gamma X)$ i-qoidani chap tarafida turgan A noterminal belgi qo‘yiladi, va qoidani raqami chiquvchi zanjirga yoziladi. Agar negiz uchun mos qoida mavjud bo‘lmasa, hato bo‘ladi va algoritm to‘htatiladi.

3. Agar joriy konfiguratsiya $(\perp_o, \perp_b S, \pi)$ bo‘lsa, tahlil muvoffaqiyatli tugadi deb hisoblanadi va π zanjir o‘ng tahlil beradi.

4. Agar X va a orasida munosabat bo‘lmasa, hato bo‘ladi va algoritm to‘htatiladi.

Misol sifatida arifmetik ifodalar chap rekursiyasiz keltirilgan grammatikani ko‘ramiz,. Bu grammatikada zanjir qoida mavjud bo‘lganiga qaramay, u siklik keltirib chiqarishga olib kelmaydi:

$G(\{+,*,(,),i\}, \{S,T,H,E,F\},P,S)$, bunda P quyidagicha:

S→TH | T

$H \to +TH \mid +T$

T→EF | E

$F \to *EF \mid *E$

$E \to (S) \mid i$

Bu chap rekursiyasiz grammatika, va unda bir hil o‘ng tarafli qoidalar yo‘q, shu sababli bu grammatika oddiy oldin keluvchi grammatikaga kiradi.

1-bosqich.

$L(A)$ va $R(A)$ quramiz:

1. $L_0(S)=\{T\}$ $R_0(S)=\{H,T\}$
   $L_0(H)=\{+\}$ $R_0(H)=\{H,T\}$
   $L_0(T)=\{E\}$ $R_0(T)=\{F,E\}$
   $L_0(F)=\{*\}$ $R_0(F)=\{F,E\}$
   $L_0(E)=\{(,i\}$ $R_0(E)=\{),i\}$, $i=1$;
2. $L_1(S)=\{T,E\}$ $R_1(S)=\{H,T,F,E\}$
   $L_1(H)=\{+\}$ $R_1(H)=\{H,T,F,E\}$
   $L_1(T)=\{E,(,i\}$ $R_1(T)=\{F,E,),i\}$
   $L_1(F)=\{*\}$ $R_1(F)=\{F,E,),i\}$
   $L_1(E)=\{(,i\}$ $R_1(E)=\{),i\}$
   $L_1(S)\neq L_0(S)$ bo'lgani uchun $i=2$;
3. $L_2(S)=\{T,E,(,i\}$ $R_2(S)=\{H,T,F,E,),i\}$
   $L_2(H)=\{+\}$ $R_2(H)=\{H,T,F,E,),i\}$
   $L_2(T)=\{E,(,i\}$ $R_2(T)=\{F,E,),i\}$
   $L_2(F)=\{*\}$ $R_2(F)=\{F,E,),i\}$
   $L_2(E)=\{(,i\}$ $R_2(E)=\{),i\}$
   $L_2(S)\neq L_1(S)$ bo'lgani uchun $i=3$
4. $L_3(S)=\{T,E,(,i\}$ $R_3(S)=\{H,T,F,E,),i\}$
   $L_3(H)=\{+\}$ $R_3(H)=\{H,T,F,E,),i\}$
   $L_3(T)=\{E,(,i\}$ $R_3(T)=\{F,E,),i\}$
   $L_3(F)=\{*\}$ $R_3(F)=\{F,E,),i\}$
   $L_3(E)=\{(,i\}$ $R_3(E)=\{),i\}$
   $L_3(S)=L_2(S)$, $R_3(S)=R_2(S)$ bo'lgani uchun, bosqichni tamomlaymiz, natijada:

$L(S)=\{T,E,(,i\}$ $R(S)=\{H,T,F,E,),i\}$
$L(H)=\{+\}$ $R(H)=\{H,T,F,E,),i\}$
$L(T)=\{E,(,i\}$ $R(T)=\{F,E,),i\}$
$L(F)=\{*\}$ $R(F)=\{F,E,),i\}$
$L(E)=\{(,i\}$ $R(E)=\{),i\}$

2-bosqich. Oddiy oldin keluvchi munosabatlar matritsasini quramiz:

	+	*	(	)	i	S	H	T	F	E	$\perp_0$
+			<·		<·			=·		<·	
*			<·		<·					=·	
(			<·		<·	=·		<·		<·	
)	·>	·>		·>			·>		·>		·>
i	·>	·>		·>			·>		·>		·>
S				=·							
H				·>							·>
T	<·			·>			=·				·>
F	·>			·>			·>				·>
E	·>	<·		·>			·>		=·		·>
$\perp_0$			<·		<·			<·		<·	

Matritsa kataklarini "+" belgi uchun to'ldirishni algoritmga asosan bajaramiz.

Birinchidan, "+" belgi $H\rightarrow +T \mid +TH$ qoidalarning T belgidan chap tarafida turibdi, shu sababli algoritmning 2-qadami bo'yicha T belgiga mos kelgan ustundagi katakka "=·" munosabatni qo'yamiz. Undan tashqari $L(T)=\{E,(,i\}$ bo'lgan uchun, algoritmning 3-qadami bo'yicha "+" belgiga mos kelgan satr va $L(T)$ funksiyasiga kirgan belgilarga mos bo'lgan ustunlar, kataklarga "<·" munosabat belgini qo'yamiz.

Ikkinchidan, algoritmni 5-qadami bo'yicha S→TH qoidani H noterminal belgisiga tegishli $L(H)$ funksiyaga "+" kiradi , shu sababli "+" belgiga mos kelgan ustun va qoidani T noterminal belgisi uchun hisoblangan R(T) funksiyaga kirgan belgilarga mos bo'lgan satrlar kataklarga "·>" munosabat belgini qo'yamiz.

Shu mulohazalarni boshqa belgilarga ham qo'llab matritsani to'ldiramiz.

3-bosqich. "Surish-o'rash" algoritmni qo'llaymiz. Buning uchun misol sifatida $i+i*i$ zanjirni tahlil qilib chiqamiz:

1. $(i+i*i\perp_a , \perp_b , \varepsilon) \vdash_s \perp_b < \cdot i$
2. $(+i*i\perp_a , \perp_b i , \varepsilon) \vdash_o i \cdot > +$
3. $(+i*i\perp_a , \perp_b E , 10.) \vdash_o E \cdot > +$
4. $(+i*i\perp_a , \perp_b T , 10.6) \vdash_s T < \cdot +$
5. $(i*i\perp_a , \perp_b T+ , 10.6) \vdash_s + < \cdot i$
6. $(*i\perp_a , \perp_b T+i , 10.6) \vdash_o i \cdot > *$
7. $(*i\perp_a , \perp_b T+E , 10.6.10) \vdash_s E < \cdot *$
8. $(i\perp_a , \perp_b T+E* , 10.6.10) \vdash_s * \cdot < i$
9. $(\perp_a , \perp_b T+E*i , 10.6.10) \vdash_o i \cdot > \perp_a$
10. $(\perp_a , \perp_b T+E*E , 10.6.10.10) \vdash_o E \cdot > \perp_a$
11. $(\perp_a , \perp_b T+EF , 10.6.10.10.8) \vdash_o F \cdot > \perp_a$
12. $(\perp_a , \perp_b T+T , 10.6.10.10.8.5) \vdash_o T \cdot > \perp_a$
13. $(\perp_a , \perp_b TH, 10.6.10.10.8.5.4) \vdash_o T \cdot > \perp_a$
14. $(\perp_a , \perp_b S , 10.6.10.10.8.5.4.1)$

Algoritm ishni tugatdi, natijada o‘ng tahlil ***10.6.10.10.8.5.4.1*** ro‘yhati hosil bo‘ldi, haqiqatdan bu zanjirni teskari o‘ng keltirib chiqarishga to‘g‘ri keladi:

$S \Leftarrow {}^{1}TH \Leftarrow {}^{4}T+T \Leftarrow {}^{5}T+EF \Leftarrow {}^{8}T+E*E \Leftarrow {}^{10}T+E*i \Leftarrow {}^{10}T+i*i \Leftarrow {}^{6}E+i*i \Leftarrow {}^{10}i+i*i$

Operatorli oldin keluvchi grammatika

Operatorli oldin keluvchi grammatikada bo‘sh qoidalar bo‘lmasligi kerak va qoidalarni o‘ng tarafida qo‘shni noterminallar bo‘lmasligi kerak. Bu grammatikada munosabatlar terminal belgilar o‘rtasida bo‘ladi.

*G(VT,VN,P,S), V=VT∪VN*KEGda quyidagi shartlar bajarilsa opreratorli oldin keluvchi grammatika deb nomlanadi.

1.Har bir tartiblangan a va b terminal belgilar orasida uchta munosabatdan faqat bittasi bajarilishi yoki umuman munosabat bo‘lmasligi mumkin:

a) agar grammatikada $A \to \alpha a b \beta$ yoki $A \to \alpha a S b \beta$ qoidalar mavjud bo‘lib, bu yerda $A,S \in VN$, $\alpha,\beta \in V^*$ bo‘lsa, bu holda $a = \cdot b$ ($\forall a,b \in VT$) bo‘ladi, ya’ni ikkala belgi negizga kiradi;

b) agar grammatikada $A \to \alpha a C \beta$ qoida mavjud bo‘lib, $C \Rightarrow^* b\gamma$ yoki $C \Rightarrow^* Db\gamma$ kelib chiqib, bu yerda $A,C,D \in VN$, $\alpha,\beta,\gamma \in V^*$ bo‘lsa, bu holda $a < \cdot b$ ($\forall a,b \in VT$);

v) agar grammatikada $A \to \alpha C b \beta$ qoida mavjud bo‘lib, $C \Rightarrow^* \gamma a$ yoki $C \Rightarrow^* \gamma a D$ kelib chiqib, bu yerda $A,C,D \in VN$, $\alpha,\beta,\gamma \in V^*$ bo‘lsa, u holda $a \cdot > b$ ($\forall a,b \in VT$).

2.Operatorli oldin keluvchi grammatikada $A \to \alpha BC \beta$ qoidalar bo‘lishi mumkin emas, bunda $A,C,B \in VN$, $\alpha,\beta \in V^*$.

3.Operatorli oldin keluvchi grammatika determinantlangan, lekin bir ma’nolik bo‘lmasligi mumkin.

Grammatikadagi munosabatlar asosida operatorli oldin keluvchi munosabat matritsasi quriladi. Bu matritsa satrlari munosabatlarning chap tarafida, ustunlari esa o‘ng tarafida yozilgan terminal belgilarga to‘g‘ri keladi. Munosabatni chap tarafida turgan terminal belgiga to‘g‘ri kelgan satr va o‘ng tarafida turgan belgiga to‘g‘ri kelgan ustun kesishmadagi katakda munosabat belgisi yoziladi. Operatorli oldin keluvchi grammatikadagi ikkita belgi orasida, ularga mos kelgan katakda faqat bitta munosabat bo‘lishi mumkin yoki munosabat bo‘lmasligi (bunda katak bo‘sh bo‘ladi) ham mumkin.

Operatorli oldin keluvchi munosabat matritsasini qurishda, $L_t(A)$ va $R_t(A)$ funksiyalarni qo‘llash qulaydir. $L_t(A)$ funksiyasi, bu A noterminalga nisbatan eng chap, $R_t(A)$ funksiyasi eng o‘ng terminal belgilar to‘plamidan iborat. Ularni ta’rifi quyidagicha:

$L_t(A)=\{t \mid \exists A \Rightarrow^* t\alpha$ yoki $\exists A \Rightarrow^* St\alpha\}$;

$R_t(A)=\{t \mid \exists A \Rightarrow^* \alpha t$ yoki $\exists A \Rightarrow^* \alpha tS\}$,

bu yerda $A,C \in VN$-noterminal, $t \in VT$- biror bir terminal belgisi,va $\alpha \in V^*$-biror bir zanjir.

Agar $L_t(A)$ va $R_t(A)$ funksiyalar aniqlangan bo‘lsa, operatorli oldin keluvchi munosabatlarni quyidagicha ta’riflash mumkin:

a) agar grammatikada $A \to \alpha a b \beta$ yoki $A \to \alpha a S b \beta$ qoidalar mavjud bo‘lib, bu yerda $A,S \in VN$, $\alpha,\beta \in V^*$ bo‘lsa, bu holda $a = \cdot b$ ($\forall a,b \in VT$) bo‘ladi, ya’ni ikkala belgi negizga kiradi;

b) agar grammatikada $A \to \alpha a C \beta$ qoida mavjud bo‘lib, $b \in L_t(C)$

kirsa, bu yerda $A,C \in VN$, $\alpha,\beta \in V^*$ bo'lsa, bu holda a<·b ($\forall$a,b$\in$VT);

v) agar grammatikada $A \rightarrow \alpha C b \beta$ qoida mavjud bo'lib, $a \in R_1(C)$kirsa, bu yerda $A,C \in VN$, $\alpha,\beta \in V^*$ bo'lsa, u holda a·>b ($\forall$a,b $\in$VT).

$L_1(A)$ va $R_1(A)$ funksiyalarni qurish algoritmi quyidagicha:

1-qadam. Oldin grammatika qoidalariga asosan yuqorida ko'rsatilgan algoritm bilan $L(A)$ va $R(A)$ funksiyalarni aniqlanadi.

2-qadam. Har bir noterminal A uchun $A \rightarrow t\alpha$ yoki $A \rightarrow Bt\alpha$ qoidalar qidiriladi va bunday qoidalar mavjud bo'lsa t terminal belgi $L_1(A)$ funksiyaga qo'shiladi; huddi shu amal $R_1(A)$ funksiyaga qo'llanadi, ya'ni:

$L_1(A)=\{t \mid A \rightarrow t\alpha$ yoki $A \rightarrow Bt\alpha$, $t \in VT$, $A,B \in VN$, $\alpha \in V^*\}$;

$R_1(A)=\{t \mid A \rightarrow \alpha t$ yoki $A \rightarrow \alpha tB$, $t \in VT$, $A,B \in VN$, $\alpha \in V^*\}$,

3-chi qadam.

$L_1(A)=L_1(A) \cup L_1(B)$, $\forall B \in (L(A) \cap VN)$;

$R_1(A)=R_1(A) \cup R_1(B)$, $\forall B \in (R(A) \cap VN)$,

Ya'ni $L(A)$ga biror bir B noterminal belgi kirsa, unda $L_1(B)$ga kirgan barcha terminal belgilar $L_1(A)$ga qo'shiladi, huddi shunday $R_1(A)$ aniqlanadi.

$L_1(A)$ va $R_1(A)$ funksiyalar aniqlangandan so'ng operatorli oldin keluvchi munosabatlar matritsasi quriladi. Undan tashqari zanjirni boshini va ohirini belgilovchi ikkita qo'shimcha "$\perp_b$", "$\perp_o$" belgilar kiritiladi,va bu belgilar qatnashgan munosabatlar ta'rifi quyidagicha bo'ladi:

agar $\exists S \Rightarrow^* a\gamma$ yoki $a \in L_1(S)$ kirgan bo'lasa, $\perp_b$<·a, $\forall a \in VT$ bo'ladi;

agar $\exists S \Rightarrow^* \gamma a$ yoki $a \in R_1(S)$ kirgan bo'lsa, a·>$\perp_o$, $\forall a \in VT$ bo'ladi.

Operatorli oldin keluvchi munosabatlar matritsasini qurish algoritmi

Operatorli oldin keluvchi grammatikani barcha terminal belgilarini tartiblab, ular orqali matritsaning satrlarini va ustunlarini belgilab chiqamiz. Matritsaga "$\perp_b$" belgini kiritish uchun, ostiga satr qo'shamiz va "$\perp_o$" uchun esa o'ng tarafiga ustun qo'shamiz. Shunda satrlar va ustunlar

soni k+1 ga teng bo'ladi, bu yerda k - umumiy terminal belgilar soni. i-satr t_i va j-ustun t_j terminal belgiga tegishli, deb faraz qilamiz.

Matritsani qurish algoritmi quyidagicha bo'ladi:

1-qadam. $i=1$, birinchi t_1 belgi tanlanadi.

2-qadam. Har bir t_j uchun $A \rightarrow \alpha t_i t_j \beta$ yoki $A \rightarrow \alpha t_i C t_j \beta$ kabi qoidalar qidiriladi,. Bu yerda $\alpha,\beta \in V^*, A,C \in VN$. Agar bunday qoida mavjud bo'lsa, i-satr va j-ustun kesishmasidagi katakka "=·" belgi qo'yiladi.

3-qadam. $A \rightarrow \alpha t_i C \beta$ qoidalar qidiriladi,. Agar bunday qoida mavjud bo'lsa, har bir $t_j \in L_1(C)$ uchun i-satr va j-ustun kesishmasidagi katakka "<·" belgi qo'yiladi.

4-qadam. $A \rightarrow \alpha S t_j \beta$ qoidalar qidiriladi. Agar bunday qoida mavjud bo'lsa, har bir $t_j \in R_1(C)$ uchun j-satr va i-ustun kesishmasidagi katakka "·>" belgi qo'yiladi.

5-qadam. $i=i+1$. Agar $i \leq k$ bo'lsa 2-qadamga o'tiladi, aks holda keyingi qadamga o'tiladi.

6-qadam. Barcha $t_j \in L_1(S)$ ohirgi satr va j-ustunga "<·" belgi qo'yib chiqiladi.

7-qadam. Barcha $t_i \in R_1(S)$ ohirgi ustun va i-satrga "·>" belgi qo'yib chiqiladi.

Operatorli oldin keluvchi grammatika uchun "surish-o'rash" algoritmi

Bu algoritm to'rta bosqichdan iborat:

Birinchi bosqichda $L(A)$ va $R(A)$ funksiyalar hisoblanadi.

Ikkinchi bosqichda $L_1(A)$ va $R_1(A)$ funksiyalar hisoblanadi.

Uchinchi bosqichda operatorli oldin keluvchi munosabatlar matritsasi quriladi.

To'rtinchi bosqichda quyidagi algoritmni qo'llanadi.

Bu algoritm oddiy oldin keluvchi algoritm kabi, bir holatli kengaytirilgan magazin, hotirali o'zgartirgichga asoslangan. Tugatish va hatolik shartlari o'zgartirgich bilan bir hil. Asosiy farqi shundaki, bu algoritmda noterminal belgilar munosabatni aniqlashda hisobga olinmaydi. Solishtirishda magazin ustiga eng yaqin bo'lgan terminal belgi va kiruvchi zanjirning joriy belgisi qatnashadi. Lekin munosabat aniqlangandan keyin, negizni tashkil qilishda noterminal belgilar qatnashadi. Operatorli oldin

keluvchi munosabatlar amalni (surish yoki o'rash) tanlash uchun ishlatiladi. Algoritm bajarishdan oldin qoidalarga tartib raqamlar qo'yib chiqiladi.

Algoritm uchta elementli konfiguratsiyaga (w, α,π) ega, bu yerda w-hali o'qilmagan kiruvchi zanjirning qismi, α-magazindagi zanjir, π-chiquvchi zanjir. Algoritmni boshlang'ich konfiguratsiyasi $(aw\perp_o, \perp_b, \varepsilon)$ bo'ladi,. Ya'ni magazinda zanjir boshlanish belgisi turadi, kiruvchi zanjir to'liqligicha, chiquvchi zanjir bo'sh turadi. Algoritmni bir qadami quyidagicha bo'ladi:

1. Agar magazinni ustiga eng yaqin bo'lgan t terminal belgisi va kiruvchi zanjirni a joriy belgisi orasida $t=\cdot a$ yoki $t<\cdot a$ munosabat mavjud bo'lsa, ko'chirish-surish amali bajariladi:
   $(aw\perp_o, \perp_b\alpha t\gamma, \pi)|\!-_k (w\perp_o, \perp_b\alpha t\gamma a, \pi)$, bunda $\alpha\in V^*$, $\gamma\in VN^*$, ya'ni kiruvchi zanjirning joriy belgisi magazinga ko'chiriladi va ko'rish qurilmasini bitta o'nga surib, keyingi kiruvchi belgiga o'tkaziladi.
2. Agar munosabat $t\cdot>a$ bo'lsa, almashtirish-o'rash amali bajariladi,. Ya'ni negizni tashkil qiluvchi magazin ustidagi =. bilan bog'langan terminal va qo'shni noterminal belgilar ($A\rightarrow\beta t\gamma$) i-chi qoidani chap tarafida turgan A noterminal belgiga almashtiriladi,va qoida raqami chiquvchi zanjirga yoziladi:
   $(aw\perp_o, \perp_b\alpha\beta t\gamma, \pi)|\!-_k (aw\perp_o, \perp_b\alpha A, \pi i)$.
   Agar negiz uchun mos qoida mavjud bo'lmasa, hato bo'ladi va algoritmni bajarishni to'xtatiladi.
3. Agar joriy konfiguratsiya $(\perp_o, \perp_b, S,)$ bo'lsa tahlil muvoffaqiyatli tugalgan, deb hisoblanadi va π zanjir o'ng tahlilni tashkil qiladi.
4. Agar t va a o'rtasida munosabat bo'lmasa, hato bo'ladi, bunda algoritmni bajarish to'htatiladi.

Misol sifatida arifmetik ifodalar grammatikasini ko'ramiz:

$G(\{+,*,(,),i\}, \{S,T,H,E,F\},P,S)$, bunda P quyidagicha:

$S\rightarrow S+T \mid T$

$T\rightarrow T*E \mid E$

$E\rightarrow (S) \mid i$

1 bosqich.

$L(A)$ va $R(A)$ funksiyalarni quramiz:

1. $L_0(S)=\{S,T\}$ $R_0(S)=\{T\}$
   $L_0(T)=\{T,E\}$ $R_0(T)=\{E\}$
   $L_0(E)=\{(,i\}$ $R_0(E)=\{),i\}$, $i=1$

2. $L_1(S)=\{S,T,E\}$ $R_1(S)=\{T,E\}$
   $L_1(T)=\{T,E,(,i\}$ $R_1(T)=\{E,),i\}$
   $L_1(E)=\{(,i\}$ $R_1(E)=\{),i\}$
   $L_1(S)\neq L_0(S)$bo'lgan uchuni=2

3. $L_2(S)=\{S,T,E,(,i\}$ $R_2(S)=\{T,E,),i\}$
   $L_2(T)=\{T,E,(,i\}$ $R_2(T)=\{E,),i\}$
   $L_2(E)=\{(,i\}$ $R_2(E)=\{),i\}$
   $L_2(S)\neq L_1(S)$bo'lgan uchuni=3

4. $L_2(S)=\{S,T,E,(,i\}$ $R_2(S)=\{T,E,),i\}$
   $L_2(T)=\{T,E,(,i\}$ $R_2(T)=\{E,),i\}$
   $L_2(E)=\{(,i\}$ $R_2(E)=\{),i\}$
   $L_3(S)=L_2(S)$, $R_3(S)=R_2(S)$ bo'lgani uchun, bosqichni tamomlaymiz, natijada:
   $L(S)=\{S,T,E,(,i\}$ $R(S)=\{T,E,),i\}$
   $L(T)=\{T,E,(,i\}$ $R(T)=\{E,),i\}$
   $L(E)=\{(,i\}$ $R(E)=\{),i\}$

2-bosqich. $L_1(A)$ va $R_1(A)$ funksiyalarni kuramiz:

1. $L_1(S)=\{+\}$ $R_1(S)=\{+\}$
   $L_1(T)=\{*\}$ $R_1(T)=\{*\}$
   $L_1(E)=\{(,i\}$ $R_1(E)=\{),i\}$,

2. $L(A)$ va $R(A)$ funksiyalar orqali, $L_1(A)$ va $R_1(A)$ funksiyalarni to'ldiramiz:

   $L_1(S)=\{+,*,),i\}$ $R_1(S)=\{+,*,(,i\}$
   $L_1(T)=\{*,),i\}$ $R_1(T)=\{*,(,i\}$
   $L_1(E)=\{(,i\}$ $R_1(E)=\{),i\}$,

3-bosqich. Operatorli oldin keluvchi munosabatlar matritsasini quramiz.

	+	*	(	)	i	$\perp_o$
+	·>	<·	<·	·>	<·	·>
*	·>	·>	<·	·>	<·	·>
(	<·	<·	<·	=·	<·	
)	·>	·>		·>		·>
i	·>	·>		·>		·>
$\perp_b$	<·	<·	<·		<·	

Matritsa kataklari "+" belgi uchun to'ldirishini algoritm bo'yicha quramiz.

Birinchidan, "+" belgi $S\to S+T$ qoidaning *"T"* belgidan chap tarafida turibdi, shu sababli matritsani qurish algoritmni 3-qadami bo'yicha "+" belgiga mos kelgan satr va $L_1(T)$ funksiyaga kirgan barcha belgilar uchun mos kelgan ustundagi katakka "<·" munosabatni qo'yamiz. Undan tashqari "+" belgi, bu qoidaning *"S"* belgining o'ng tarafida turibdi, shu sababli algoritmni 4-qadami bo'yicha "+" belgiga mos kelgan ustun va $R_1(T)$ funksiyaga kirgan barcha belgilar uchun mos kelgan satrdagi katakka "·>" munosabatni qo'yamiz. Shu mulohazalarni boshqa belgilarga ham qo'llab matritsani to'ldiramiz. Shuni aytib o'tish kerakki, "(" va ")" belgilar orasida "=·"bo'ladi, chunki ular 1-chi qadam bo'yicha $E\to(S)$ qoidada qatnashadi.

4-bosqich. "Surish-o'rash" algoritmni qo'llaymiz.

Yuqorida aytilgan bo'yicha operatorli oldin keluvchi tahlil algoritmi noterminal belgilarni hisobga olmaydi, shu sabali bitta noterminal belgisini qoldirgan holda,va grammatika qoidalarni qayta o'zgartirib chiqamiz:

P:

$S\to S+S \mid S$

$S\to S*S \mid S$

$S\to(S) \mid i$

Befoyda $S\to S$ qoidalarni olib tashlaymiz, natijada qoidalarimiz quyidagicha bo'ladi:

1. $S\to S+S$
2. $S\to S*S$

3. $S\to(S)$
4. $S\to i$.

Bu grammatika dastlabki grammatikaga ekvivalent emas, lekin u ishni soddalashtiradi. Bunday qayta o'zgartirishni munosabatlar matritsasi qurilgandan so'ng bajaramiz.

Bunday ko'rinishdagi grammatika "qobirg'a" grammatika deb nomlanadi. Unga "surish-o'rash" algoritmni qo'llaymiz. Buning uchun kiruvchi zanjir sifatida $i+i*i$ olib, tahlil qilib chiqamiz:

1. $(i+i*i\perp_o , \perp_b , \varepsilon) \vdash_s \perp_b <\cdot i$
2. $(+i*i\perp_o , \perp_b i , \varepsilon) \vdash_o i \cdot>+$
3. $(+i*i\perp_o , \perp_b S , 4) \vdash_s \perp_b \cdot>+$
4. $(i*i\perp_o , \perp_b S+ , 4) \vdash_s +<\cdot i$
5. $(*i\perp_o , \perp_b S+i , 4) \vdash_o i\cdot>*$
6. $(*i\perp_o , \perp_b S+S , 4.4) \vdash_s +<\cdot*$
7. $(i\perp_o , \perp_b S+S* , 4.4) \vdash_s *\cdot<i$
8. $(\perp_o , \perp_b S+S*i , 4.4) \vdash_o i\cdot>\perp_o$
9. $(\perp_o , \perp_b S+S*S , 4.4.4) \vdash_o *\cdot>\perp_o$
10. $(\perp_o , \perp_b S+S , 4.4.4.2) \vdash_o +\cdot>\perp_o$
11. $(\perp_o , \perp_b S , 4.4.4.2.1) \vdash_o$

Algoritm ishni tugatdi, va natijada o'ng tahlil ***4.4.4.2.1*** ro'yhati hosil bo'ldi,. Haqiqatdan bu zanjirni teskari o'ng keltirib chiqarishga to'g'ri keladi:

$S\xleftarrow{1}S+S\xleftarrow{2}S+S*S\xleftarrow{4}S+S*i\xleftarrow{4}S+i*i\xleftarrow{4}i+i*i$

Xulosa

Qaytmasdan ishlovchi ko'taruvchi aniqlovchilar KEGni sinf qismlari shu jumladan oldin keluvchi grammatikalar bilan ishlaydi. Bu grammatikani bir nechta turi mavjud shu jumladan oddiy oldin keluvchi, operatorli oldin keluvchi grammatikalar. Oddiy oldin keluvchi grammatikalar oldin keluvchi munosabatlarga asoslangan. Oldin keluvchi munosabatlar asosida munosabatlar matritsasi quriladi va u asosida "surish-o'rash" algoritmi ishlaydi. Operatorli oldin keluvchi

grammatikada munsabatlar terminallar asosida quriladi va qurilgan matritsani satrlar va ustunlarida faqat terminallar qatnashadi. Faqat bitta noterminal qatnashgan grammatika "qobirga" grammatika deb nomlanadi va unga dastlabki grammatika keltirib "surish-o'rash" algoritmi qo'llanadi.

Nazorat uchun savollar

1. Oldin keluvchi grammatikani turlarini keltiring.
2. Oldin keluvchi munosabatlar tushunchasini izohini bering.
3. Oddiy oldin keluvchi grammatikani ta'rifini bering.
4. Oddiy oldin keluvchi munosabatlar matritsasini qurish algoritmni keltiring.
5. Oddiy oldin keluvchi grammatika uchun "surish-o'rash" algoritmni keltiring.
6. Operatorli oldin keluvchi grammatikani ta'rifini bering.
7. Operatorli oldin keluvchi munosabatlar matritsasini kurish algoritmni keltiring
8. Operatorli oldin keluvchi grammatika uchun "surish-o'rash" algoritmni keltiring.

Amaliyot uchun topshiriqlar

1. Berilgan grammatika uchun oddiy oldindan keluvchi "surish-o'rash" algoritmni tuzib dastur yarating:

a) $S \to$ ***i***$:=F;$
$F \to F+T|F-T|T$
$T \to T*E|T/E|E$
$E \to (F)|-E|i$

b) $S \to$ ***i***$:=F;$
$F \to F$ ***or*** $T|F$ ***xor*** $T|T$
$T \to T$ ***and*** $E|\ E$
$E \to (F)|$***not*** $E|$ ***i***

c) $S \to F;$
$F \to$ ***if*** E ***then*** F ***else*** F ***endif*** $|$***if*** E ***then*** F ***endif*** $|$ ***i***$:=$***i***
$E \to i<i|i>i|i=i$

d) $S \to F;$
$F \to$ ***for*** (T) ***do*** $F|$ ***i***$:=$***i***
$T \to F;E;F|;E;F|F;E;|;E;|;;$
$E \to i<i\ |i>i|i=i$

e) $S \to F;$
$F \to$ ***while*** (E) ***do*** $F|$ ***i***$:=$***i***
$E \to i<i\ |i>i|i=i$

f) $S \to F;$
$F \to$ ***repeat*** F ***until*** $E|$ ***i***$:=$***i***
$E \to i<i\ |i>i|i=i$

2. 1-chi misolda berilgan grammatika uchun operatorli oldindan keluvchi "surish-o'rash" algoritmni tuzib dastur yarating.

27-BOB. LR(K) GRAMMATIKALAR

Tayanch iboralar: *kengaytirilgan grammatika, LR(k) grammatika, $f(s_i,a_j)$ amal funksiya, $g(s_i,X_k)$ o'tish funksiya, LR(0) grammatika, LR(0) tahlilchini boshqaruvchi jadvali, LR(0) vaziyatlar, closure(I) funksiya, goto(I,X) funksiyani, SLR(1)-grammatika, LR(1) grammatika, LR(1) tahlilchini boshqarish jadvali, LALR(1) grammatika, core(I) funksiyasi, tayanch vaziyatlar, notayanch vaziyatlar, o'z-o'zidan paydo bo'luvchi oldindan ko'rinuvchi belgi, tarqaluvchi oldindan ko'rinuvchi belgi, ko'chirish-o'rash to'qnashuvi, o'rash-o'rash to'qnashuvi, "osilib qolgan else" muammosi.*

Norasmiy ta'rif

LR(k) grammatikani aniqlovchisi kiruvchi zanjirni chapdan o'ngga ko'rib chiqadi (left) va o'ng tahlilni bajaradi (right).. Surish yoki o'rashni tanlashda, joriy belgini o'ng tarafida turgan k-ta belgilar ketma-ketligini hisobga oladi. Shuning uchun uning nomi-LR(k). Bu grammatika asosida qurilgan aniqlovchilar qaytmasdan ishlovchi ko'tariluvchi aniqlovchilar turkimiga kiradi.

Rasmiy ta'rif

LR(k) grammatika rasmiy tarifini beramiz:

G(VT,VN,P,S) qontekstdan erkin grammatika bo'lsin. Unda Gning kengaytirilgan grammatikasi deb, G'(VT',VN',P',S') aytiladi va unda VT'=VT; VN'=VN∪{S'}; S'∉VN; P'=P∪{S'→S} bo'ladi.

G'(VT',VN',P',S') Gning kengaytirilgan grammatikasi bo'lsin, agar quyidagi:

1. $S' \Rightarrow^*_r \alpha Aw \Rightarrow_r \alpha\beta w$,
2. $S' \Rightarrow^*_r \gamma Bx \Rightarrow_r \alpha\beta y$,
3. $FIRST_k(w)=FIRST_k(y)$

shartlardan, αAy=γBx (ya'ni α=γ, A=B, y=x) kelib chiqsa, G(VT,VN,P,S)- LR(k) grammatika bo'ladi, k≥0.

Norasmiy aytganda, agar birinchi keltirib chiqarishga ko'ra, β bu A noterminal belgiga o'raladigan negiz bo'lsa, unda ikkinchi keltirib chiqarishda ham β zanjir A noterminal belgiga o'raladigan negiz bo'lishi kerak.

Agar qoidalar tartiblab chiqilgan bo'lsa, $S' \rightarrow S$ 0-chi qoida bo'ladi.

LR(k) grammatikada *k>1* bo'lganda tahlil algoritmi murakkablashib ketadi, va shu sababli amalda asosan *LR(0)* yoki *LR(1)* grammatikalar ishlatiladi. (va) Ular uchun aniqlovchilar quriladi. Biz ham shu grammatikalar sinflarini ko'rib chiqamiz.

LR(k)-tahlilchi

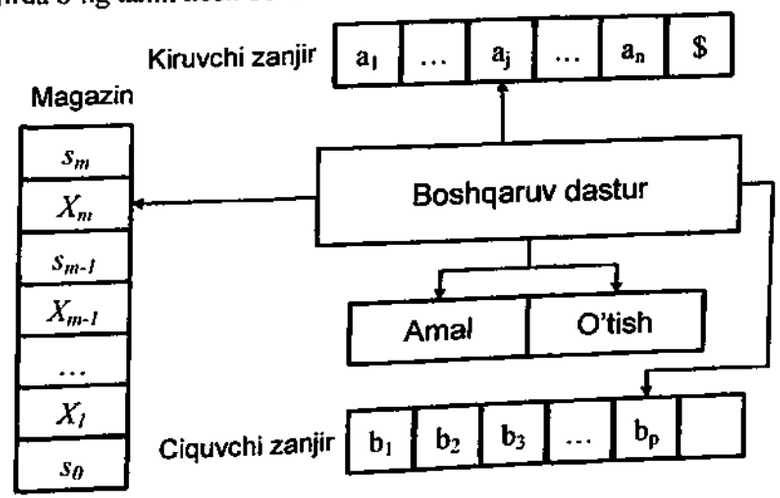
LR(k)-tahlilda "ko'chirish-o'rash" usuli qo'llanadi (shift-reduce). Bu usul determinantlangan kengaytirilgan magazin hotirali o'zgartirgichni qo'llaydi.

Bu usulni bajarish jarayoni quyidagicha:

Magazin ustida biror bir qoidani o'ng tarafi, ya'ni negiz hosil bo'lgunicha, kiruvchi zanjirning belgilari magazinga ko'chiriladi ("ko'chirish" amali).

Keyin, hosil bo'lgan negiz magazin ustidan olinadi va uni o'rniga qoidani chap tarafida turgan noterminal belgi qo'yiladi. (va) Chiquvchi zanjirga qoida raqami yoziladi ("o'rash" amali).

Agar ohirgi belgi o'zgartirgich magaziniga ko'chirilib, o'rash amali bajarilsa va magazinda faqat boshlang'ich noterminal belgi qolsa, kiruvchi zanjir o'zgartirgich bilan qabul qilindi deb hisoblanadi. (va) Chiquvchi zanjirda o'ng tahlil hosil bo'ladi.



27.1 rasm. LR(k) –tahlilchi.

Tahlilchi kiruvchi zanjir, magazin, boshqaruvchi dastur, ikki qismdan iborat bo'lgan jadval (amal va o'tish) va chiquvchi zanjirning o'zaro bog'lanishi iborat 27.1- rasmda keltirilgan.

LR(k) tahlilchilarni boshqaruvchi dasturchini algoritmi barcha tahlilchilar uchun bir hil. Ular faqat jadvallar tuzimi bilan farq qiladi.

Magazinda $s_0X_1s_1X_2s_2...X_ms_m$ ketma-ketlikdagi zanjir turadi, bunda s_m magazin ustidagi belgi, $X_i \in (VT \cup VN)$, s_i - holatlar nomidagi magazinning maxsus belgilari. Har bir holat undan oldin magazinda turgan belgi to'g'risida ma'lumotni o'ziga jamlaydi. Magazin ustidagi holat va kiruvchi zanjirning joriy belgisi bo'yicha,va boshqaruvchi dastur jadvallardagi mos ma'lumotlariga ko'ra, bajarish amalni aniqlaydi. Boshqaruvchi dastur ish jarayonida LR(k) jadvallarni ishlatadi. Bu jadvallar ikki qismdan iborat:

Amal jadvali $f(s_i,a_j)$ amal funksiya bilan beriladi,.

O'tish jadvali $g(s_i,X_k)$ o'tish funksiya orqali beriladi.

Bunda s_i- joriy holat, a_j-joriy kiruvchi belgi, X_k-grammatika belgisi.

Amal funksiyasi to'rt hil qiymatga ega bo'lishi mumkin:

1. ***ks***– ko'chirish amali, bunda "k" ko'chirish amalni belgilovchi harf, s- biror bir holat.
2. ***o'i***– o'rash amali, bunda "o'" o'rash amalni belgilovchi harf, *i*-qoida raqami. Agar $A \to \beta$ *i*-chi qoida bo'lsa, unda β zanjir *A* noterminalga o'raladi
3. ***qq***– qabul qilishni belgilaydi,.
4. ***hv***– hato vaziyatni belgilaydi.

O'tish funksiyasining qiymati holat bo'ladi.

LR(k) tahlilchi uchun "ko'chirish-o'rash" algoritmi

Algoritm uchta elementli konfiguratsiyaga, ya'ni (w,α,π) ega,. Bunda *w*-hali o'qilmagan kiruvchi zanjir qismi, α-magazindagi zanjir, π-chiquvchi zanjir.

Algoritmni boshlang'ich konfiguratsiyasi (awS, s_0, ε) bo'ladi, ya'ni magazinda boshlang'ich holat belgisi turadi,va kiruvchi zanjir to'liq bo'ladi,. Chiquvchi zanjir boshida esa bo'sh.

Faraz qilaylik, magazin cho'qqisida *s* holat belgisi va kiruvchi zanjirni joriy belgisi *a* bo'lsin.

Algoritmni bitta qadami quyidagidan iborat:

1. Agar $f(s,a)=ks'$ bo'lsa, ko'chirish amali bajariladi:

$(awS, \alpha s, \pi) \vdash_k (wS, \alpha sas', \pi)$.

ya'ni kiruvchi zanjirni joriy belgisi va yangi holat *s'* magazinga ko'chiriladi, bunda *s'*- $g(s,a)$ o'tish funksiyaning qiymati. O'qish qurilmasi bitta belgiga o'nga surilib keyingi kiruvchi belgiga o'tadi.

2. Agar $f(s,a)=o'i$ bo'lsa, o'rash amali bajariladi:

$(awS, s_0\alpha s'A\beta', \pi) \vdash_{o'} (awS, s_0\alpha s'Ag(s',A), \pi i)$,

ya'ni *i*-raqamli qoida $A \to \beta$ bo'yicha, magazin ustidagi $2|\beta|$ uzunlikdagi β' zanjir olib tashlanadi va Magazin ustiga A noterminal belgi va $g(s',A)$ qiymati qo'yiladi. Qoidani raqami chiquvchi zanjirga yoziladi, bunda *s'*o'rashdan keyin magazin ustida qolgan holat.

3. Agar $f(s,a)=qq$ bo'lsa, tahlil, muvaffaqiyatli tugadi deb hisoblanadi va π zanjir o'ng tahlilni beradi.

4. Agar $f(s,a)=hv$ bo'lsa, tahlilda hato vaziyat yuzaga kelgani ma'lum bo'ladi va algoritm to'htatiladi.

Albatta, bu algoritmni ishlatishdan oldin amal va o'tish funksiyalarning qiymatlarini aniqlash lozim. Bu funksiyalar qiymatlari boshqaruvchi jadval ko'rinishda beriladi.

LR(0) tahlilchini boshqaruvchi jadvali

LR(0)-tahlilchi LR(0) grammatika uchun ishlatiladi, bu grammatikada kiruvchi zanjirni joriy belgisi hisobga olinmaydi. Faqat magazindagi zanjirlar tahlil qilinadi, va ularni tuzilishiga qarab amallar bajariladi.

Bizga G (VT, VN, P, S) – KEG berilsin. LR(k)-vaziyat (LR(k)-item) deb $[A \to \alpha \cdot \beta, u]$ nomlanadi, bunda $A \to \alpha\beta \in P$, $u \in VT^*$, $|u| \le k$, $\alpha,\beta \in (VT \cup VN)^*$, $A \in VN$.

LR(0)-vaziyatlar $[A \to \alpha \cdot \beta]$ ko'rinishda bo'lad ($k=0$, $u=\varepsilon$).

Bu vaziyatdan, A-qoidaning α bosh qismi magazinda joylashgani kelib chiqadi va tahlilchi qoidani qolgan qismini kiruvchi zanjirni hali o'qilmagan qismidan hosil qilishni bildiradi. Negiz to'liqligini kiruvchi zanjirni $|u|$ta belgisi bo'yicha aniqlanadi. Agar negiz to'liq bo'lsa, o'rash amali bajariladi,. Aks holda joriy kiruvchi belgi magazinga ko'chiriladi.

Vaziyat ikkita son bilan berilishi mumkin. Birinchisi qoida raqamiga, ikkinchisi esa nuqtani qoidadagi tartib o'rniga teng.

Masalan, $A \to XYZ$ qoida uchun to'rtta vaziyat mavjud:

1. $[A \to \cdot XYZ]$ (1,0)
2. $[A \to X \cdot YZ]$ (1,1)
3. $[A \to XY \cdot Z]$ (1,2)
4. $[A \to XYZ \cdot]$ (1,3)

Vaziyat qoidani qaysi qismi sintaktik tahlil jarayonida aniqlanganligini bildiradi. Masalan, birinchi vaziyatda biz qoidani o'ng qismini hosil bo'lishini kutamiz. Uchinchi vaziyatda qoidaning ikkita belgisi hosil bo'ldi va uchinchi belgining qo'shilishini kutamiz. To'rtinchi vaziyatda qoidani o'ng qismi hosil bo'ldi va uni o'rash kerakligini kutamiz. Shuni aytib o'tish kerakki, 1-3 vaziyatlarda ko'chirish amali bajariladi, 4-vaziyatda o'rash amali bajariladi.

LR(0)-tahlilchi uchun boshqaruvchi jadvalini tuzishdan oldin ikkita funksiya kiritamiz: *closure(I)* va *goto(I,X)*, bunda *I*-vaziyatlar to'plami, *X*-grammatika belgisi, $X \in (VT \cup VN)$.

closure(I) funksiyani hisoblash algoritmi:

closure(I) funksiya - bu I to'plam asosida quyidagi algoritm bo'yicha hisoblangan vaziyatlar to'plami:

1. $J_0 = I$; $i = 1$;
2. $J_i = J_{i-1} \cup \{[B \to \cdot \gamma] \mid [A \to \alpha \cdot B\beta] \in J_{i-1},\ B \to \gamma \in P,\ [B \to \cdot \gamma] \notin J_{i-1}$, bunda $\alpha, \beta, \gamma \in (VT \cup VN)^*,\ A, B \in VN\}$
3. Agar $J_i \neq J_{i-1}$, $i = i+1$; 2-chi qadamga o'tamiz;
4. $closure(I) = J_i$.

goto(I,X) hisoblash algoritmi :

goto(I,X) funksiyani hisoblash uchun, yangi *J* vaziyatlar to'plami hosil qilinadi.

1. $J = \{[A \to \alpha X \cdot \beta] \mid [A \to \alpha \cdot X\beta] \in I$, bunda $\alpha, \beta \in (VT \cup VN)^*,\ A \in VN\}$
2. $goto(I,X) = closure(J)$.

Yendi LR(0)-tahlilchini chekli avtomat(CHA) ko'rinishda tasvirlaymiz.

Chekli avtomatni holatlari sifatida vaziyatlar to'plami bo'ladi,. Ularni ketma-ket butun sonlar bilan belgilaymiz. Undan tashqari o'tish funksiyasini tuzamiz.

Holatlar to'plamini qurish algoritmi:

Holatlar to'plamini qurishdan oldin grammatika kengaytiriladi so'ng quidagi algoritm qo'llanadi:

1. $I_0 = closure([S' \to \cdot S])$; $i=0$; $k=i+1$;
2. Agar I_i to'plamida $[A \to \alpha \cdot X\beta]$ vaziyat mavjud bo'lsa, 3-chi qadamga, aks holda 4 qadamga o'tiladi,. Bunda $X \in (VT \cup VN)$ grammatikani biror bir belgisi;
3. $I_k = goto(I_i, X)$; $\delta(I_i, X) = I_k$; $k = k+1$; 2-chi qadamga o'tiladi;
4. $i = i+1$; agar barcha holatlar ko'rib chiqilgan bo'lsa 5-qadamga, aks holda 2-qadamga o'tiladi.
5. Boshlang'ich holat I_0 bo'ladi, $[A \to \alpha \cdot]$ vaziyat mavjud bo'lgan holatlar avtomatni chekli holatlari deb hisoblanadi.

27.1-misol. Bizga arifmetik ifodaning $G(\{i, +, -, *, /, (,)\}, \{S,T,E\}, P, S)$ grammatikasi berilgan bo'lsin, avval uni kengaytiramiz, shunda P:

0. $S' \to S$
1. $S \to S+T$
2. $S \to T$
3. $T \to T*E$
4. $T \to E$
5. $E \to (S)$
6. $E \to i$

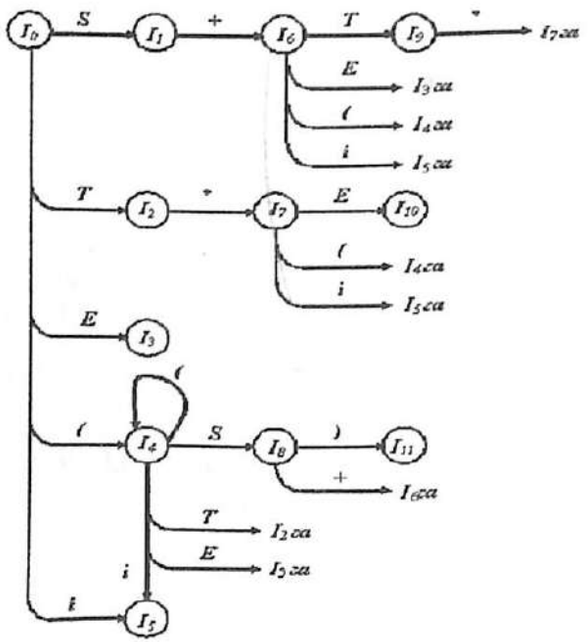
Yuqorida ko'rsatilgan funksiyalar va algoritmlar bo'yicha holatlarni aniqlaymiz (27.1-jadval) va o'tish diagrammalarni tuzamiz (27.2-rasm):

27.1-jadval

I_0:	I_1:	I_2:	I_3:
$[S' \to \cdot S]$ $[S \to \cdot S+T]$ $[S \to \cdot T]$ $[T \to \cdot T*E]$ $[T \to \cdot E]$ $[E \to \cdot (S)]$ $[E \to \cdot i]$	**$[S' \to S \cdot]$** $[S \to S \cdot +T]$	**$[S \to T \cdot]$** $[T \to T \cdot *E]$	**$[T \to E \cdot]$**

I_4: $[E\to(\cdot S)]$ $[S\to\cdot S+T]$ $[S\to\cdot T]$ $[T\to\cdot T*E]$ $[T\to\cdot E]$ $[E\to\cdot(S)]$ $[E\to\cdot i]$	I_5: $[E\to i\cdot]$	I_6: $[S\to S+\cdot T]$ $[T\to\cdot T*E]$ $[T\to\cdot E]$ $[E\to\cdot(S)]$ $[E\to\cdot i]$	I_7: $[T\to T*\cdot E]$ $[E\to\cdot(S)]$ $[E\to\cdot i]$
I_8: $[E\to(S\cdot)]$ $[S\to S\cdot+T]$	I_9: $[S\to S+T\cdot]$ $[T\to T\cdot *E]$	I_{10}: $[T\to T*E\cdot]$	I_{11}: $[E\to(S)\cdot]$

Bu jadvalda o‘rashga olib keladigan vaziyatlarning tagi chizilgan.



27.2-rasm. LR(0)-tahlilchi chekli avtomatning o‘tish diagrammasi.

LR(0)-tahlilchi uchun boshqarish jadvalini tuzish algoritmi:

Boshqarish jadvali M-ning satrlarini holatlar raqami bilan belgilaymiz. Ustunlar ikki qismdan iborat bo‘lib, chap qismi terminallar va o‘ng qismi noterminallardan tashkil topadi. Birinchi qismi amal, ikkinchisi esa o‘tish funksiyasiga to‘g‘ri keladi. Chekli avtomatni holatlar diagrammasi bo‘yicha har bir holat uchun jadvalni kataklarini to‘ldirish algoritmi:

1. Kengaytirilgan grammatika G' uchun $LR(0)$-vaziyatlardan iborat bo‘lgan $C=\{I_0, I_1, \ldots, I_m\}$ tizim tuziladi va barcha o‘tish funksiyalar hisoblanadi. Boshida I_0 to‘plam (boshlang‘ich holat) olinadi, ya’ni $i=0$;

2. Har bir $X\in(VT\cup VN)$ uchun quyidagi amallar bajariladi:

Agar $\delta(I_i, X)=I_m$ bo‘lsa va

a). X – terminal bo‘lsa, $f(I_i,X)=km$; $g(I_i, X)=I_m$, ya’ni $M[i, X]=km$, bu holda ikkala funksiyani qiymati bitta $M[i, X]$ katakka yoziladi;

b). X – noterminal bo‘lsa $g(I_i, X)=I_m$, ya’ni $M[i, X]=m$;

3. Agar $[S'\to S\cdot]\in I_i$, unda $f(I_i,S)=qq$, ya’ni $M[i,S]=qq$;

4. Agar $[A\to\alpha\cdot]\in I_i$, unda yuqoridagi shartlarga qanoatlanmagan barcha terminallar uchun, $f(I_i, X)=o'j$ va $M[i, X]=o'j$, bunda j- o‘raladigan $A\to\alpha$ qoida tartib raqami, $\alpha\in(VT\cup VN)^*$, $X\in VT$;

5. Agar yuqoridagi shartlar bajarilmasa, $f(I_i,X)=hv$ *va* $M[i,X]$ bo‘sh qoladi;

6. $i=i+1$; Agar i>n, ya’ni barcha holatlar ko‘rib chiqilgan bo‘lsa algoritm to‘htatiladi,. Aks holda 2-chi qadamga o‘tiladi.

Algoritm bo‘yicha boshqarish jadvali qurilishi 27.2-jadvalda keltirilgan:

27.2-jadval

Holat	Amal						O‘tish		
	i	+	*	(	)	*S*	*S*	*T*	*E*
							1	2	3
0	k5			k4					
1		k6				qq			
2	o‘2	o‘2	k7	o‘2	o‘2	o‘2			
3	o‘4	o‘4	o‘4	o‘4	o‘4	o‘4			

4	k5			k4			8	2	3
5	o‘6	o‘6	o‘6	o‘6	o‘6	o‘6			
6	k5			k4				9	3
7	k5			k4					10
8		k6			k11				
9	o‘1	o‘1	k7	o‘1	o‘1	o‘1			
10	o‘3	o‘3	o‘3	o‘3	o‘3	o‘3			
11	o‘5	o‘5	o‘5	o‘5	o‘5	o‘5			

LR(0)-tahlilchini ishlash algoritmi tuzilgan boshqaruv jadval bo‘yicha “*i*i+i*” kiruvchi zanjir uchun ishlash qadamlarini keltiramiz:

Tahlilchini qadamlari (konfiguratsiyalar) amallar

1. *(i*i+i$, 0 , ε)|— k5* ko‘chirish
2. *(*i+i$, 0i5 , ε)|— o‘6* o‘rash 6. $E \to i$
3. *(*i+i$, 0E3 , 6)|— o‘4* o‘rash 4.$T \to E$
4. *(*i+i$, 0T2 , 64)|— k7* ko‘chirish
5. *(i+i$, 0T2*7 , 64)|— k5* ko‘chirish
6. *(+i$, 0T2*7i5 , 64)|— o‘6* o‘rash 6. $E \to i$
7. *(+i$, 0T2*7E10 , 646)|— o‘3* o‘rash 3. $T \to T*E$
8. *(+i$, 0T2 , 6463)|— o‘2* o‘rash 2. $S \to T$
9. *(+i$, 0S1 , 64632)|— k6* ko‘chirish
10. *(i$, 0S1+6 , 64632) |— k5* ko‘chirish
11. *($, 0S1+6i5 , 64632) |— o‘6* o‘rash 6. $E \to i$
12. *($, 0S1+6E3 , 646326) |— o‘4* o‘rash 4.$T \to E$
13. *($, 0S1+6T9 , 6463264) |— o‘2* o‘rash 2.$S \to S+T$
14. *($, 0S1 , 64632641)|— qq* qabul qilindi

Tahlilchi ishlash natijasida o‘ng tahlil zanjiri hosil bo‘ldi,. Bu o‘ng keltirib chiqarishni teskarisi bo‘ladi, haqiqatdan ham :

$S \Leftarrow^{2} S+T \Leftarrow^{4} S+E \Leftarrow^{6} S+i \Leftarrow^{3} T+i \Leftarrow^{5} T* E+i \Leftarrow^{6} T*i+i \Leftarrow^{4} E*i+i \Leftarrow^{6} i*i+i$

Agar grammatika uchun LR(0)-tahlilchi qurilib, uning boshqarish jadvaldagi har bir katagidagi qiymat bittadan ortiq bo‘lmasa, bunday grammatika LR(0)-grammatika bo‘ladi.

SLR(1)-tahlilchi

LR(0)-tahlilchida, biror bir holatlar to‘plamiga *[A→α.]* vaziyat kirsa, shu holatdagi barcha aniqlanmagan kataklarga *f(I,X)* amal funksiyani o‘rash qiymati beriladi. Lekin, ba`zi bir hollarda negiz tayyor bo‘lganiga qaramasdan, ayrim terminallar uchun o‘rash amalini bajarishi kerak emas. SLR(1)-tahlilchida (simple LR), shunday holatlar hisobga olingan. SLR(1)-tahlilchida ham LR(0)-vaziyatlar qo‘llanadi va holatlar to‘plami tuziladi,. Asosiy farqi boshqarish jadvalini tuzish algoritmda,va bu algoritmning 3-qadami quyidagicha bo‘ladi:

3. Agar $[A \to \alpha.] \in I_i$ bo‘lsa, unda har bir $X \in FOLLOW(A)$ terminal uchun $f(I_i,X)=o‘j$ va $M[i,X]=o‘j$ bo‘ladi,. Bunda *j*-o‘raladigan $A \to \alpha$ qoidaning tartib raqami, $\alpha \in (VT \cup VN)^*$.

Yuqorida ko‘rsatilgan misolda har bir noterminal belgisi uchun *FOLLOW(A)* funksiya hisoblanadi:

FOLLOW(S)={+,),S};
FOLLOW(T)={+,),S,};*
FOLLOW(E)={+,),S,}.*

Shunda boshqaruv jadvali quyidagicha bo‘ladi:

27.3-jadval

Holat	Amal						O‘tish		
	i	+	*	(	)	S	*S*	*T*	*E*
0	k5			k4			1	2	3
1		k6				qq			
2		o‘2	k7		o‘2	o‘2			
3		o‘4	o‘4		o‘4	o‘4			
4	k5			k4			8	2	3
5		o‘6	o‘6		o‘6	o‘6			
6	k5			k4				9	3
7	k5			k4					10
8		k6			k11				
9		o‘1	k7		o‘1	o‘1			
10		o‘3	o‘3		o‘3	o‘3			
11		o‘5	o‘5		o‘5	o‘5			

SLR(1)-tahlilchini ishlash qadamlari va tahlil natijasi ko'rsatilgan zanjir uchun bir hil bo'ladi.

Agar grammatika uchun SLR(1)-tahlilchi qurilib, uning boshqarish jadvaldagi har bir katagidagi qiymat bittadan ortiq bo'lmasa, bunday grammatika SLR(1)-grammatika bo'ladi.

LR(1)-tahlilchi

LR(1)-tahlilchi amalni aniqlashda kiruvchi zanjirning bitta joriy belgisidan foydalanadi. LR(1)-tahlilchini boshqaruv jadvalini tuzish LR(0)-tahlilchiga o'hshaydi, lekin vaziyat tushunchasi murakkabroq, u qoidadan tashqari bitta kiruvi belgidan ham iborat.

LR(1)-vaziyat $[A\to\alpha\cdot\beta, a]$ ***nomlanadi, bunda*** $A\to\alpha\beta\in P$, $a\in VT$***-terminal belgisi,*** $\alpha,\beta\in(VT\cup VN)^*$, $A\in VN$.

Bu vaziyatdan kelib chiqadiki A-qoidaning α bosh qismi magazinda joylashgan va tahlilchi qoidani qolgan qismini kiruvchi zanjir hali o'qilmagan qismidan hosil qiladi. Agar negiz to'liq bo'lsa kiruvchi zanjirni bitta joriy belgisi oldindan ko'riladi va shunga qarab o'rash amali bajariladi,. Aks holda joriy kiruvchi belgi magazinga ko'chiriladi.

LR(1)-tahlilchi ham chekli avtomat orqali amalga oshiriladi. Uning boshqaruv jadvalini tuzishdan oldin *closure(I)* va *goto(I,X)* funksiyalar qayta ta'riflanadi.

closure(I) funksiyani algoritmi:

1. $J_0=I;\ i=1;$
2. $J_i=J_{i-1}\cup\{[B\to\cdot\gamma,t]\mid[A\to\alpha\cdot B\beta,a]\in J_{i-1},\ B\to\gamma\in P,[B\to\cdot\gamma,t]\notin J_{i-1},\ t\in FIRST(\beta a)$, bunda $\alpha,\beta,\gamma\in(VT\cup VN)^*,\ a,t\in VT,\ A,B\in VN\}$;
3. Agar $J_i\neq J_{i-1},\ i=i+1;$ 2-chi qadamga o'tiladi;
4. $closure(I)=J_i$.

goto(I,X) funksiyani algoritmi:

1. $J=\{[A\to\alpha X\cdot\beta,a]\mid[A\to\alpha\cdot X\beta,a]\in I$, bunda$\alpha,\beta\in(VT\cup VN)^*,\ A\in VN\}$
2. $goto(I,X)=closure(J)$.

Yendi LR(1)-tahlilchini chekli avtomat ko'rinishda tasvirlaymiz.

Holatlar tizimini qurish algoritmi:

Holatlar to'plamini qurishdan oldin grammatika kengaytiriladi keyin

quyidagi qadamlar bajariladi:

1. $I_0=closure([S'\to\cdot S,\$]);\ i=0;k=1;$
2. Agar I_i to'plamida $[A\to\alpha\cdot X\beta,a]$ vaziyat mavjud bo'lsa 3-chi qadamga, aks holda 4-qadamga o'tiladi,. Bunda $X\in(VT\cup VN)$ grammatikani biror bir belgisi;
3. $I_k=goto(I_i,X);\ \delta(I_i,X)=I_k\ ;\ k=k+1;$ 2-chi qadamga o'tiladi;
4. $i=i+1$; agar barcha holatlar ko'rib chiqilgan bo'lsa 5-qadamga, aks holda 2-qadamga o'tiladi.
6. Boshlang'ich holat I_0 bo'ladi, $[A\to\alpha\cdot, a]$ vaziyat mavjud bo'lgan holatlar, chekli holatlar bo'ladi.

LR(1)-tahlilchi uchun boshqarish jadvalini tuzish algoritmi:

1. Kengaytirilgan grammatika G' uchun $LR(1)$-vaziyatlardan iborat bo'lgan $C=\{I_0,I_1,\dots,I_n\}$ tizimni tuzamiz. $i=0;$
2. Har bir $X\in(VT\cup VN)$ uchun quyidagi amallarni bajaramiz:
   Agar $\delta(I_i,X)=I_j$ bo'lsa va
   a) X – terminal bo'lsa, $f(I_i,X)=km;\ g(I_i,X)=m,$ ya'ni $M[i,X]=km$ bo'ladi,. Bu holda ikkala funksiyani qiymati bitta $M[i,X]$ katakka yoziladi.
   b) X – noterminal bo'lsa, $g(I_i,X)=m,$ ya'ni $M[i,X]=m;$
3. $i=i+1;$ Agar $i\le n$ bo'lsa, 2-chi qadamga, aks holda 4-chi qadamga o'tiladi;
4. Agar $[S'\to S\cdot,S]\in I_i$, unda $f(I_i,S)=qq,$ ya'ni $M[i,S]=qq$.
5. Agar $[A\to\alpha\cdot,X]\in I_i$, unda $f(I_i,X)=o'j$ va $M[i,X]=o'j,$ bunda j-o'raladigan $A\to\alpha$ qoidaning tartib raqami, $X\in(VT\cup\{\$\}),\ \alpha\in(VT\cup VN)^*$.

LR(1)-tahlilchida ishlatilgan vaziyatlar va holatlar soni LR(1)-vaziyatdagi qo'shimcha komponent hisobiga ancha ko'payib ketadi.

27.1-misol uchun algoritmlarni qo'llaymiz.

Yuqorida ko'rsatilgan funksiyalar va algoritmlar bo'yicha holatlar tizimini aniqlaymiz :

27.4-jadval

I_0: $[S' \rightarrow \cdot S,\$]$ $[S \rightarrow \cdot S+T,\$/+][S \rightarrow \cdot T,\$/+]$ $[T \rightarrow \cdot T*E,\$/+/*]$ $[T \rightarrow \cdot E,\$/+/*]$ $[E \rightarrow \cdot (S),\$/+/*]$ $[E \rightarrow \cdot i,\$/+/*]$	I_1: **$[S' \rightarrow S \cdot,\$]$** $[S \rightarrow S \cdot +T,\$/+]$	I_2: **$[S \rightarrow T \cdot,\$/+]$** $[T \rightarrow T \cdot *E,\$/+/*]$	I_3: **$[T \rightarrow E \cdot,\$/+/*]$**
I_4: $[E \rightarrow (\cdot S),\$/+/*]$ $[S \rightarrow \cdot S+T,)/+]$ $[S \rightarrow \cdot T,)/+]$ $[T \rightarrow \cdot T*E,)/+/*]$ $[T \rightarrow \cdot E,)/+/*]$ $[E \rightarrow \cdot (S),)/+/*]$ $[E \rightarrow \cdot i,)/+/*]$	I_5: **$[E \rightarrow i \cdot,\$/+/*]$**	I_6: $[S \rightarrow S+ \cdot T,\$/+]$ $[T \rightarrow \cdot T*E,\$/+/*]$ $[T \rightarrow \cdot E,\$/+/*]$ $[E \rightarrow \cdot (S),\$/+/*]$ $[E \rightarrow \cdot i,\$/+/*]$	I_7: $[T \rightarrow T* \cdot E,\$/+/*]$ $[E \rightarrow \cdot (S),\$/+/*]$ $[E \rightarrow \cdot i,\$/+/*]$
I_8: $[E \rightarrow (S \cdot),\$/+/*]$ $[S \rightarrow S \cdot +T,)/+]$	I_9: **$[S \rightarrow T \cdot,)/+]$** $[T \rightarrow T \cdot *E,)/+/*]$	I_{10}: **$[T \rightarrow E \cdot,)/+/*]$**	I_{11}: $[E \rightarrow (\cdot S),)/+/*][S \rightarrow \cdot S+T,)/+]$ $[S \rightarrow \cdot T,)/+]$ $[T \rightarrow \cdot T*E,)/+/*][T \rightarrow \cdot E,)/+/*]$ $[E \rightarrow \cdot (S),)/+/*]$ $[E \rightarrow \cdot i,)/+/*]$
I_{12}: **$[E \rightarrow i \cdot,)/+/*]$**	I_{13}: **$[S \rightarrow S+T \cdot,\$/+]$** $[T \rightarrow T \cdot *E,\$/+/*]$	I_{14}: **$[T \rightarrow T*E \cdot,\$/+/*]$**	I_{15}: **$[E \rightarrow (S) \cdot,\$/+/*]$**

I_{16}: $[S \rightarrow S+ \cdot T,)/+]$ $[T \rightarrow \cdot T*E,)/+/*][T \rightarrow \cdot E,)/+/*]$ $[E \rightarrow \cdot (S),)/+/*]$ $[E \rightarrow \cdot i,)/+/*]$	I_{17}: $[T \rightarrow T* \cdot E,)/+/*]$ $[E \rightarrow \cdot (S),)/+/*]$ $[E \rightarrow \cdot i,)/+/*]$	I_{18}: $[E \rightarrow (S \cdot),)/+/*]$ $[S \rightarrow S \cdot +T,)/+]$	I_{19}: **$[S \rightarrow S+T \cdot,)/+/*]$** $[T \rightarrow T \cdot *E,)/+/*]$
I_{20}: **$[T \rightarrow T*E \cdot,)/+/*]$**	I_{21}: **$[E \rightarrow (S) \cdot,)/+/*]$**		

Bu jadvalda masalan ikkita vaziyat $[S \rightarrow \cdot T,\$],[S \rightarrow \cdot T,+]$ o'rniga birlashgan varianti yozilmoqda $[S \rightarrow \cdot T,\$/+]$

O'tish funksiyalari quyidagi qiymatlarga ega bo'ladi:

27.5-jadval

$\delta(I_0,S)=I_1$	$\delta(I_0,T)=I_2$	$\delta(I_0,E)=I_3$	$\delta(I_0,()=I_4$	$\delta(I_0,i)=I_5$
$\delta(I_1,+)=I_6$	$\delta(I_2,*)=I_7$	$\delta(I_4,S)=I_8$	$\delta(I_4,T)=I_9$	$\delta(I_4,E)=I_{10}$
$\delta(I_4,()=I_{11}$	$\delta(I_4,i)=I_{12}$	$\delta(I_6,T)=I_{13}$	$\delta(I_6,E)=I_3$	$\delta(I_6,()=I_4$
$\delta(I_6,i)=I_5$	$\delta(I_7,E)=I_{14}$	$\delta(I_7,()=I_4$	$\delta(I_7,i)=I_5$	$\delta(I_8,))=I_{15}$
$\delta(I_8,+)=I_{16}$	$\delta(I_9,*)=I_{17}$	$\delta(I_{11},S)=I_{18}$	$\delta(I_{11},T)=I_9$	$\delta(I_{11},E)=I_{10}$
$\delta(I_{11},()=I_{11}$	$\delta(I_{11},i)=I_{12}$	$\delta(I_{13},*)=I_7$	$\delta(I_{16},T)=I_{19}$	$\delta(I_{16},E)=I_{10}$
$\delta(I_{16},()=I_{11}$	$\delta(I_{16},i)=I_{12}$	$\delta(I_{17},E)=I_{20}$	$\delta(I_{17},()=I_{11}$	$\delta(I_{17},i)=I_{12}$
$\delta(I_{18},))=I_{21}$	$\delta(I_{18},+)=I_{16}$	$\delta(I_{19},*)=I_{17}$		

Yendi boshqaruv jadvalini tuzamiz:

27.6-jadval

Holat	Amal						O'tish		
	i	+	*	(	)	$	*S*	*T*	*E*
0	k5			k4			1	2	3
1		k6				qq			
2		o'2	k7			o'2			
3		o'4	o'4			o'4			
4	k12			k11			8	9	10
5		o'6	o'6			o'6			

27.4-jadval

I_0: $[S' \to \cdot S, \$]$ $[S \to \cdot S+T, \$/+][S \to \cdot T, \$/+]$ $[T \to \cdot T*E, \$/+/*]$ $[T \to \cdot E, \$/+/*]$ $[E \to \cdot (S), \$/+/*]$ $[E \to \cdot i, \$/+/*]$	I_1: **$[S' \to S \cdot, \$]$** $[S \to S \cdot +T, \$/+]$	I_2: **$[S \to T \cdot, \$/+]$** $[T \to T \cdot *E, \$/+/*]$	I_3: **$[T \to E \cdot, \$/+/*]$**
I_4: $[E \to (\cdot S), \$/+/*]$ $[S \to \cdot S+T,)/+]$ $[S \to \cdot T,)/+]$ $[T \to \cdot T*E,)/+/*]$ $[T \to \cdot E,)/+/*]$ $[E \to \cdot (S),)/+/*]$ $[E \to \cdot i,)/+/*]$	I_5: **$[E \to i \cdot, \$/+/*]$**	I_6: $[S \to S+ \cdot T, \$/+]$ $[T \to \cdot T*E, \$/+/*]$ $[T \to \cdot E, \$/+/*]$ $[E \to \cdot (S), \$/+/*]$ $[E \to \cdot i, \$/+/*]$	I_7: $[T \to T* \cdot E, \$/+/*]$ $[E \to \cdot (S), \$/+/*]$ $[E \to \cdot i, \$/+/*]$
I_8: $[E \to (S \cdot), \$/+/*]$ $[S \to S \cdot +T,)/+]$	I_9: **$[S \to T \cdot,)/+]$** $[T \to T \cdot *E,)/+/*]$	I_{10}: **$[T \to E \cdot,)/+/*]$**	I_{11}: $[E \to (\cdot S),)/+/*]$ $[S \to \cdot S+T,)/+]$ $[S \to \cdot T,)/+]$ $[T \to \cdot T*E,)/+/*]$ $[T \to \cdot E,)/+/*]$ $[E \to \cdot (S),)/+/*]$ $[E \to \cdot i,)/+/*]$
I_{12}: **$[E \to i \cdot,)/+/*]$**	I_{13}: **$[S \to S+T \cdot, \$/+]$** $[T \to T \cdot *E, \$/+/*]$	I_{14}: **$[T \to T*E \cdot, \$/+/*]$**	I_{15}: **$[E \to (S) \cdot, \$/+/*]$**

I_{16}: $[S \to S+ \cdot T,)/+]$ $[T \to \cdot T*E,)/+/*][T \to \cdot E,)/+/*]$ $[E \to \cdot (S),)/+/*]$ $[E \to \cdot i,)/+/*]$	I_{17}: $[T \to T* \cdot E,)/+/*]$ $[E \to \cdot (S),)/+/*]$ $[E \to \cdot i,)/+/*]$	I_{18}: $[E \to (S \cdot),)/+/*]$ $[S \to S \cdot +T,)/+]$	I_{19}: **$[S \to S+T \cdot,)/+/*]$** $[T \to T \cdot *E,)/+/*]$
I_{20}: **$[T \to T*E \cdot,)/+/*]$**	I_{21}: **$[E \to (S) \cdot,)/+/*]$**		

Bu jadvalda masalan ikkita vaziyat $[S \to \cdot T, \$], [S \to \cdot T, +]$ o'rniga birlashgan varianti yozilmoqda $[S \to \cdot T, \$/+]$

O'tish funksiyalari quyidagi qiymatlarga ega bo'ladi:

27.5-jadval

$\delta(I_0, S)=I_1$	$\delta(I_0, T)=I_2$	$\delta(I_0, E)=I_3$	$\delta(I_0, ()=I_4$	$\delta(I_0, i)=I_5$
$\delta(I_1, +)=I_6$	$\delta(I_2, *)=I_7$	$\delta(I_4, S)=I_8$	$\delta(I_4, T)=I_9$	$\delta(I_4, E)=I_{10}$
$\delta(I_4, ()=I_{11}$	$\delta(I_4, i)=I_{12}$	$\delta(I_6, T)=I_{13}$	$\delta(I_6, E)=I_3$	$\delta(I_6, ()=I_4$
$\delta(I_6, i)=I_5$	$\delta(I_7, E)=I_{14}$	$\delta(I_7, ()=I_4$	$\delta(I_7, i)=I_5$	$\delta(I_8,))=I_{15}$
$\delta(I_8, +)=I_{16}$	$\delta(I_9, *)=I_{17}$	$\delta(I_{11}, S)=I_{18}$	$\delta(I_{11}, T)=I_9$	$\delta(I_{11}, E)=I_{10}$
$\delta(I_{11}, ()=I_{11}$	$\delta(I_{11}, i)=I_{12}$	$\delta(I_{13}, *)=I_7$	$\delta(I_{16}, T)=I_{19}$	$\delta(I_{16}, E)=I_{10}$
$\delta(I_{16}, ()=I_{11}$	$\delta(I_{16}, i)=I_{12}$	$\delta(I_{17}, E)=I_{20}$	$\delta(I_{17}, ()=I_{11}$	$\delta(I_{17}, i)=I_{12}$
$\delta(I_{18},))=I_{21}$	$\delta(I_{18}, +)=I_{16}$	$\delta(I_{19}, *)=I_7$		

Yendi boshqaruv jadvalini tuzamiz:

27.6-jadval

Holat	Amal						O'tish		
	i	+	*	(	)	*S*	*S*	*T*	*E*
0	k5			k4			1	2	3
1		k6				qq			
2		o'2	k7			o'2			
3		o'4	o'4			o'4			
4	k12			k11			8	9	10
5		o'6	o'6			o'6			

6	k5			k4				13	3
7	k5			k4					14
8		k16			k15				
9		o'2	k17						
10		o'4	o'4		o'4				
11	k12			k11			18	9	10
12		o'6	o'6		o'6				
13		o'1	k7			o'1			
14		o'3	o'3			o'3			
15		o'5	o'5			o'5			
16	k12			k11				19	10
17	k12			k11					20
18		k16			k21				
19			k17						
20		o'3	o'3	o'3					
21		o'5	o'5	o'5					

LR(1)-tahlilchini ishlash algoritmi va tuzilgan jadval bo'yicha kiruvchi zanjir uchun *"i*i+i"* ishlash qadamlarini keltiramiz:

Tahlilchining qadamlari (konfiguratsiyalar) amallar

1. *(i*i+i$, 0 , ε)|— k5* ko'chirish
2. *(*i+i$, 0i5 , ε)|— o'6* o'rash 6. *E→i*
3. *(*i+i$, 0E3 , 6)|— o'4* o'rash 4.*T→E*
4. *(*i+i$, 0T2 , 64)|— k7* ko'chirish
5. *(i+i$, 0T2*7 , 64)|— k5* ko'chirish
6. *(+i$, 0T2*7i5 , 64)|— o'6* o'rash 6. *E→i*
7. *(+i$, 0T2*7E14 , 646)|— o'3* o'rash 3. *T→T*E*
8. *(+i$, 0T2 , 6463)|— o'2* o'rash 2. *S→T*
9. *(+i$, 0S1 , 64632)|— k6* ko'chirish
10. *(i$, 0S1+6 , 64632) |— k5* ko'chirish
11. *($, 0S1+6i5 , 64632) |— o'6* o'rash 6. *E→i*
12. *($, 0S1+6E3 , 646326) |— o'4* o'rash 4.*T→E*

13. *($, 0S1+6T13 , 6463264) |— o'1* o'rash 1.*S→S+T*
14. *($, 0S1 , 64632641) |— qq* qabul qilindi

Tahlilchining ishlash natijasida o'ng tahlil zanjiri hosil bo'ldi,. Bu o'ng keltirib chiqarishning teskarisi bo'ladi. haqiqatdan ham :

$S \Leftarrow^{1} S+T \Leftarrow^{4} S+E \Leftarrow^{6} S+i \Leftarrow^{2} T+i \Leftarrow^{3} T*E+i \Leftarrow^{6} T*i+i \Leftarrow^{4} E*i+i \Leftarrow^{6} i*i+i$

Agar grammatika uchun LR(1)-tahlilchi qurilib, uning boshqarish jadvaldagi har bir katagida bittadan ortiq element mavjud bo'lmasa, bunday grammatika LR(1)-grammati kabo'ladi.

LALR(1)-tahlilchi

LALR(1) – tahlilchi LR(1)-tahlilchiga qaraganda ancha ihcham, lekin unga teng kuchlikdir. Masalan, Pascal tili grammatikasi uchun LALR(1) – tahlilchida yuztaga yaqin holatlar bo'lsa, LR(1)-tahlilchida bir nechta ming holatlar mavjud.

LR(1)-vaziyat [A→α •β, a]ni yadrosi deb, A→α •β hisoblaymiz.

core(I) funksiya deb, I-vaziyatlar to'plamiga kirgan yadrolar to'plamini hisoblaymiz.

$I_0, I_1, \ldots, I_n$ - LR(1)-tahlilchini holatlar tizimi bo'lsin, unda yadro to'plami teng, ya'ni $core(I_i)=core(I_j)$ bo'lgan holatlar o'rniga ularni birlashmasini, ya'ni $I_{ij}= I_i \cup I_j$ tizimga qo'shamiz.

Yuqorida berilgan grammatika uchun qurilgan holatlar tizimini olamiz. Bu tizimda $core(I_2)=core(I_9)=\{S\rightarrow T\bullet, T\rightarrow T\bullet *E\}$, endi bu ikki holatni , ya'ni ularga kirgan barcha vaziyatlarni birlashtiramiz,va natijada yangi holat to'plami hosil bo'ladi:

$I_{29}=\{[S\rightarrow T\bullet, S/)/+], [T\rightarrow T\bullet *E, S/)/+/*]\}$,

bundan ko'rinib turibdiki, vaziyatlarni ikkinchi qomponentasi, ya'ni oldindan ko'rinuvchi belgilar yig'iladi. Huddi shunday boshqa holatlarni ham birlashtiramiz. O'tish funksiyalar ham shunga qarab o'zgariladi:

$\delta(I_0,T)=I_2$ o'rniga $\delta(I_0,T)=I_{29}$ bo'ladi.

$\delta(I_9,*)=I_{17}$ o'rniga $\delta(I_{29},*)=I_{717}$ bo'ladi

LALR(1)-tahlilchi uchun boshqarish jadvalini tuzish algoritmi:

1. Kengaytirilgan grammatika G' uchun $LR(1)$-vaziyatlardan iborat bo'lgan $C=\{I_0,I_1,\ldots,I_n\}$ tizim tuziladi.
2. C-tizimda yadro to'plami bir hil bo'lgan holatlar birlashtiriladi.
3. Yangi tizim $C'=\{J_0,J_1,\ldots,J_m\}$ belgilanadi va uning o'tish funksiyalari qayta hisoblanadi; $i=0$;
4. Agar $\delta(J_i, X)=J_l$ bo'lsa va
   a) X – terminal bo'lsa, $f(I_i,X)=kl$;$g(I_i,X)=I_l$, ya'ni $M[i,X]=kl$, bu holda ikkala funksiyani qiymati bitta $M[i,X]$ katakka yoziladi.
   b) X – noterminal bo'lsa, $g(J_i,X)=I_l$, ya'ni $M[i,X]=l$;
5. $i=i+1$; Agar $i\le n$ bo'lsa,2-chi qadamga, aks holda 4-chi qadamga o'tiladi;
6. Agar $[S'\to S\cdot,S]\in I_i$, unda $f(I_i,S)=qq$, ya'ni $M[i,S]=qq$.
7. Agar $[A\to\alpha.,X]\in I_i$, unda $f(I_i,X)=o'n$ va $M[i,X]=o'j$, bunda j-o'raladigan $A\to\alpha$ qoidani tartib raqami, $X\in(VT\cup\{\$\})$.

Yuqorida ko'rsatilgan 27.1-misol uchun algoritmni qo'llaymiz.

Birinchi qadamda holatlar S-tizimini tuzamiz. Ular 27.4-jadvalda keltirilgan.

Ikkinchi qadamda C'-tizimini tuzamiz:

27.7-jadval

J_0:I_0: [S'→ ·S,$] [S→ ·S+T,$/+] [S→ ·T,$/+] [T→ ·T*E,$/+/*] [T→ ·E,$/+/*] [E→ ·(S) ,$/+/*] [E→ ·i,$/+/*]	J_1:I_1: **[S'→S ·,$]** [S→S · +T,$/+]	J_2:$I_{2/9}$: **[S→T ·,$/)/+]** [T→T · *E,$/)/+/*]	J_3:$I_{3\ 10}$: **[T→E ·,$/)/+/*]**
J_4:$I_{4\ 11}$: [E→(·S) ,$/)/+/*] [S→ ·S+T,)/+] [S→ ·T,)/+]	J_5:$I_{5/12}$: **[E→i ·,$/)/+/*]**	J_6:$I_{6\ 16}$: [S→S+ ·T,$/)/+] [T→ ·T*E,$/)/+/*] [T→ ·E,$/)/+/*]	J_7:$I_{7\ 17}$: [T→T* ·E,$/)/+/*] [E→ ·(S) ,$/)/+/*]

[T→ ·T*E,)/+/*] [T→ ·E,) /+/*] [E→ ·(S) ,)/+/*] [E→ ·i,) /+/*]		[E→ ·(S) ,$/)/+/*] [E→ ·i,$/)/+/*]	[E→ ·i,$/)/+/*]
J_8:$I_{8\ 18}$: [E→(S ·),$/)/+/*] [S→S · +T,) /+]	J_9:$I_{13/19}$: **[S→S+T ·,$/)/+]** [T→T · *E,$/)/+/*]	J_{10}:$I_{14/20}$: **[T→T*E ·,$/)/+/*]**	J_{11}:$I_{15/21}$: **[E→(S) ·,$/)/+/*]**

O'tish funksiyalar quyidagi qiymatlarga ega:

27.8-jadval

$\delta(J_0,S)=J_1$	$\delta(J_0,T)=J_2$	$\delta(J_0,E)=J_3$	$\delta(J_0,()=J_4$	$\delta(J_0,i)=J_5$
$\delta(J_1,+)=J_6$	$\delta(J_2,*)=J_7$	$\delta(J_4,S)=J_8$	$\delta(J_4,T)=J_2$	$\delta(J_4,E)=J_3$
$\delta(J_4,()=J_4$	$\delta(J_4,i)=J_5$	$\delta(J_6,T)=J_9$	$\delta(J_6,E)=J_3$	$\delta(J_6,()=J_4$
$\delta(J_6,i)=J_5$	$\delta(J_7,E)=J_{10}$	$\delta(J_7,()=J_4$	$\delta(J_7,i)=J_5$	$\delta(J_8,))=J_{11}$
$\delta(J_8,+)=J_6$	$\delta(J_9,*)=J_7$			

Yendi boshqaruv jadvalini tuzamiz:

27.9-jadval

Holat	Amal						O'tish		
	i	+	*	(	)	$	*S*	*T*	*E*
0	k5			k4			1	2	3
1		k6				qq			
2		o'2	k7		o'2	o'2			
3		o'4	o'4		o'4	o'4			
4	k5			k4			8	2	3
5		o'6	o'6		o'6	o'6			
6	k5			k4				9	3
7	k5			k4					10
8		k6			k11				
9		o'1	k7		o'1	o'1			
10		o'3	o'3		o'3	o'3			
11		o'5	o'5		o'5	o'5			

Bu misol uchun LALR(1)-tahlilchini boshqaruv jadvali SLR(1)-tahlilchini boshqaruv jadvali bilan bir hil, lekin bu har doim ham bo'lmaydi.

LALR(1)-tahlilchini ishlash algoritmi va tuzilgan jadval bo'yicha "*i*i+i*" kiruvchi zanjir uchun ishlash qadamlari SLR(1)-tahlilchi bilan bir hil bo'ladi.

LALR(1)-tahlilchini boshqarish jadvalini samarali qurish algoritmi

LALR(1)-tahlilchini boshqarish jadvalini LR(1)-vaziyatlar orqali tuzish kup vaqt va hotirani talab qiladi. Lekin bu usuldan tashqari, boshqa usul ham bor. Boshqa usul bo'yicha oldin LR(0)-holatlar tizimi tuziladi va undagi LR(0)-vaziyatlarga biror bir algoritmlar asosida ikkinchi qomponentalar, ya'ni oldindan ko'rinuvchi belgilar qo'shiladi.

Barcha vaziyatlarni ikkita sinfga ajratamiz:

1. ***Tayanch vaziyatlarga boshlang'ich vaziyat [S'→ .S] va nuqtasi boshida bo'lmagan barcha vaziyatlar kiradi.***
2. ***Notayanch vaziyatlarga nuqtasi boshida bo'lgan vaziyatlar kiradi.***

Holatning tayanch vaziyatlar to'plami – holat tayanchi deb nomlanadi.

LALR(1)holatlar to'plamini tuzishdan oldin biz LR(0)-holatlar tizimini tuzib, unda faqat tayanch vaziyatlarni qoldiramiz,. Natijada tayanch holatlar tizimi hosil bo'ladi. Bizning misol uchun tayanch holatlar quyidagicha bo'ladi:

27.10-jadval

I_0: $[S' \to \cdot S]$	I_1: $[S' \to S \cdot]$ $[S \to S \cdot +T]$	I_2: $[S \to T \cdot]$ $[T \to T \cdot *E]$	I_3: $[T \to E \cdot]$
I_4: $[E \to (\cdot S)]$	I_5: $[E \to i \cdot]$	I_6: $[S \to S+ \cdot T]$	I_7: $[T \to T* \cdot E]$
I_8: $[E \to (S \cdot)]$ $[S \to S \cdot +T]$	I_9: $[S \to S+T \cdot]$ $[T \to T \cdot *E]$	I_{10}: $[T \to T*E \cdot]$	I_{11}: $[E \to (S) \cdot]$

Yendi har bir tayanch holatdagi LR(0)-vaziyatlardan LR(1)-vaziyatlarni hosil qilamiz, buning uchun unga ikkinchi komponentasini qo'shamiz,. Yani oldindan ko'rinuvchi belgilarni qo'shamiz.

Agar $[B \to \alpha \cdot C\delta, b]$ – I to'plamdagi biror bir tayanch vaziyat bo'lsa, undan kelib chiqqan vaziyatlarga $a \in FIRST(\delta b)$ yangi oldindan ko'rinuvchi belgilar ko'shilsa, bu belgilar shu vaziyatlardan qurilgan goto(I,X) holatdagi tayanch vaziyat uchun, o'z-o'zidan paydo bo'lgan oldindan ko'rinuvchi belgilar bo'ladi.

Ta'rif bo'icha $S' \to S \cdot$ yadro uchun S belgisi o'z-o'zidan paydo bo'lgan belgi bo'ladi.

Agar $[B \to \alpha \cdot C\delta, b]$ – I to'plamdagi biror bir tayanch vaziyat bo'lsa, va unda $b \in FIRST(\delta b)$, undan kelib chiqqan vaziyatlarga b belgi qo'shiladi,. Bu belgi shu vaziyatlardan qurilgan goto(I,X) holatdagi tayanch vaziyat uchun, tarqaluvchi oldindan ko'rinuvchi belgi bo'ladi.

Oldindan ko'rinuvchi belgilarni turini aniqlash algoritmi:

Kirish:Biror bir tayanch holat I va grammatika belgisi X bo'lsin.

Chiqish:*goto(I,X)* holatdagi tayanch vaziyatlar uchun o'z-o'zidan va tarqaluvchi oldindan ko'rinuvchi belgilar.

Algoritmda tayanch vaziyatlarga, oldindan ko'rinuvchi belgi sifatida "#" belgi ishlatiladi. Bu belgi ikkinchi qomponent sifatida barcha belgilar bo'lishi mumkunligini bildiradi.

I holat to'plamidagi har bir tayanch vaziyat uchun quyidagi qadamlar bajariladi.

1-qadam. $J'=clouser([B \to \gamma \cdot \delta, \#])$, bunda $[B \to \gamma \cdot \delta]$ - *LR(0)* tayanch vaziyati, J' bu *LR(1)*-vaziyatlar to'plami,.

2-qadam. Agar $[A \to \alpha \cdot X\beta, a] \in J'$ va a belgi "#" o'rnida bo'lmasa, unda a belgi, *goto(I,X)* holatdagi $[A \to \alpha X \cdot \beta]$ vaziyat uchun, o'z-o'zidan paydo bo'lgan oldindan ko'rinuvchi belgi sifatida qo'shiladi.

3-qadam. Agar $[A \to \alpha \cdot X\beta, \#] \in J'$ bo'lsa, unda I holatdagi $[B \to \gamma \cdot \delta, \#]$ tayanch vaziyatga kirgan oldindan ko'rinuvchi belgi, *goto(I,X)* holatlardagi $[A \to \alpha X \cdot \beta]$ tayanch vaziyat uchun tarqaluvchi belgi bo'ladi,. Ya'ni tayanch vaziyatlarga tarqaladi.

LALR(1)-tahlilchini holatlar tizimini tuzish algoritmi

Kirish: Kengaytirilgan grammatika G' bo'lsin.

Chiqish: LALR(1)-tahlilchining holatlar tizimi.

Algoritm:

1. Yuqorida ko'rsatilgan bo'yicha LR(0)-vaziyatlarni tayanch holatlar tizimi tuziladi.

2. LR(0) tizimdagi *I* holatdagi har bir tayanch vaziyat va har bir X grammatika belgici uchun, yuqorida ko'rsatilgan algoritm bo'yicha *goto(I,X)* holatni tayanch vaziyatlariga, qaysi belgilar o'z-o'zidan paydo bo'luvchi oldindan ko'rinuvchi va qaysi belgilar tarqaluvchi oldindan ko'rinuvchi belgiligi aniqlanadi.

3. Har bir holatdagi har bir tayanch vaziyatga bog'langan oldindan ko'rinuvchi belgilarni ko'rsatuvchi jadval quriladi. Boshida har bir tayanch vaziyat bilan 2-qadamda aniqlangan o'z-o'zidan paydo bo'lgan belgilar bog'lanadi.

4. Yuqorida ko'rsatilgan qadamlar, har bir vaziyat uchun, qayta o'tiladi, *i*-chi vaziyatda hosil bo'lgan belgilar 2-qadam bo'yicha unga bog'langan tayanch vaziyatlarga tarqatiladi.. Ya'ni *i*-chi vaziyatdagi oldindan ko'rinuvchi belgilar to'plami unga bog'langan tayanch vaziyat to'plamiga qo'shiladi. Bunday qayta o'tishlarni yangi tarqaladigan belgilar tugaguncha bajariladi.

5. 2-qadamda hosil bo'lgan jadval bo'yicha o'tish funksiyalar hisoblanadi.

6. 3-chi va 4-chi qadamlarda qurilgan jadval asosida LALR(1)-tahlilchining tayanch holatlar tizimi quriladi.

7. 5-chi va 6-chi qadamlarda qurilgan jadvallar asosida LALR(1)-tahlilchining boshqaruv jadvalini qurish algoritmi bo'yicha boshqaruv jadvali quriladi.

Yuqorida ko'rsatilgan 27.1-misolda I_0 boshlang'ich holatdagi tayanch vaziyat uchun, oldindan ko'rinuvchi belgilarning turini aniqlash algoritmi bo'yicha *J'* to'plamni tuzamiz:

J':

[S'→ ·S, #]

[S→ ·S+T, #/+]

[S→ ·T, #/+]

*[T→ ·T*E, #/+/*]*

[T→ ·E, #/+/]*

[E→ ·(S) , #/+/]*

[E→ ·i, #/+/]*

Bundagi barcha tayanch bo'lmagan vaziyatlarda o'z-o'zidan paydo bo'luvchi belgilar bor {+,*}. Ular vaziyatga mos bo'lgan to'plamdagi tayanch vaziyatlar belgilari to'plamiga qo'shiladi.

LALR(1)-tahlilchini holatlar tizimini tuzish algoritmi 2-chi qadami bo'yicha oldindan ko'rinuvchi belgilarni tayanch vaziyatlarga tarqatilishi 27.11 jadvalda berilgan:

27.11-jadval

Vaziyatdan	Vaziyatga	Vaziyatdan	Vaziyatga
I_0:*[S'→ ·S]*	I_1:*[S'→S ·]* I_1:*[S→S ·+T]* I_2: *[S→T ·]* I_2: *[T→T ·*E]* I_3:*[T→E ·]* I_4: *[E→(·S)]* I_5: *[E→i ·]*	I_7:*[T→T* ·E]*	I_4: *[E→(·S)]* I_5: *[E→i ·]* I_{10}:*[T→T*E ·]*
I_1:*[S→S ·+T]*	I_6:*[S→S+ ·T]*	I_8:*[E→(S ·)]*	I_{11}: *[E→(S) ·]*
I_2: *[T→T ·*E]*	I_7:*[T→T* ·E]*	I_8: *[S→S ·+T]*	I_6:*[S→S+ ·T]*
I_4: *[E→(·S)]*	I_2: *[S→T ·]* I_2: *[T→T ·*E]* I_3:*[T→E ·]* I_4: *[E→(·S)]* I_5: *[E→i ·]* I_8: *[E→(S ·)]* I_8: *[S→S ·+T]*	I_9:*[T→T ·*E]*	I_7:*[T→T* ·E]*

I_6:[S→S+ • T]	I_3:[T→E •] I_4: [E→(• S)] I_5: [E→i •] I_9:[S→S+T •] I_9:[T→T • *E]		

LALR(1)-tahlilchini holatlar tizimini tuzish algoritmi 3-chi qadami va 4-chi qadami 27.12- jadvalda keltirilgan:

27.12-jadval

Holat	Tayanch vaziyat	Oldindan ko'rinuvchi belgilar			
		Boshlang'ich	1-chi o'tish	2-chi o'tish	3-o'tish
I_0	[S'→ • S]	$	$	$	$
I_1	[S'→S •]		$	$	$
I_1	[S→S • +T]	+	+/$	+/$	+/$
I_2	[S→T •]	+/)	+/)/$	+/)/$	+/)/$
I_2	[T→T • *E]	+/*/)	+/*/)/$	+/*/)/$	+/*/)/$
I_3	[T→E •]	+/*/)	+/*/)/$	+/*/)/$	+/*/)/$
I_4	[E→(• S)]	+/*/)	+/*/)/$	+/*/)/$	+/*/)/$
I_5	[E→i •]	+/*/)	+/*/)/$	+/*/)/$	+/*/)/$
I_6	[S→S+ • T]		+/)	+/)/$	+/)/$
I_7	[T→T* • E]		+/*/)	+/*/)/$	+/*/)/$
I_8	[E→(S •)]		+/*/)	+/*/)/$	+/*/)/$
I_8	[S→S • +T]	)	)/+	)/+	)/+
I_9	[S→S+T •]	)	)	)/+	)/+/$
I_9	[T→T • *E]	)	)	)/+	)/+/*/$
I_{10}	[T→T*E •]	)	)	+/*/)	+/*/)/$
I_{11}	[E→(S) •]		)	+/*/)	+/*/)/$

Bu jadvalda "Boshlang'ich" ustunida har bir tayanch vaziyat uchun hisoblangan o'z-o'zidan paydo bo'lgan oldindan ko'rinuvchi belgilar keltirilgan. Birinchi o'tishda $ belgisi I_0dagi [S'→ • S]dan 27.11 jadvalda

ko'rsatilgan bo'yicha yettita vaziyatga tarqaladi. Oldindan ko'rinuvchi + belgisi I_1dagi [S→S • +T]dan I_6dagi [S→S+ • T] vaziyatga tarqaladi. Huddi shu usul bilan qolgan belgilar ham tarqaladi.

27.12 jadvalga ko'ra LALR(1)-tahlilchini holatlar tizimini quramiz. Bu tizim 27.13- jadvalda keltirilgan.

27.13-jadval.

I_0: [S'→ • S,$]	I_1: **[S'→S • ,$]** [S→S • +T,$/+]	I_2: **[S→T • ,$/)/+]** [T→T • *E,$/)/+/*]	I_3: **[T→E • ,$/)/+/*]**
I_4: [E→(• S),$/)/+/*]	I_5: **[E→i • ,$/)/+/*]**	I_6: [S→S+ • T,$/)/+]	I_7: [T→T* • E,$/)/+/*]
I_8: [E→(S •),$/)/+/*] [S→S • +T,)/+]	I_9: **[S→S+T • ,$/)/+]**[T→T • *E,$/)/+/*]	I_{10}: **[T→T*E • ,$/)/+/*]**	I_{11}: **[E→(S) • ,$/)/+/*]**

27.11. jadval asosida o'tish funksiyalarni hisoblaymiz. Natijasi 27.14- jadvalda keltirilgan.

Jadval 27.14

$\delta(I_0,S)=I_1$	$\delta(I_0,T)=I_2$	$\delta(I_0,E)=I_3$	$\delta(I_0,()=I_4$	$\delta(I_0,i)=I_5$
$\delta(I_1,+)=I_6$	$\delta(I_2,*)=I_7$	$\delta(I_4,S)=I_8$	$\delta(I_4,T)=I_2$	$\delta(I_4,E)=I_3$
$\delta(I_4,()=I_4$	$\delta(I_4,i)=I_5$	$\delta(I_6,T)=I_9$	$\delta(I_6,E)=I_3$	$\delta(I_6,()=I_4$
$\delta(I_6,i)=I_5$	$\delta(I_7,E)=I_{10}$	$\delta(I_7,()=I_4$	$\delta(I_7,i)=I_5$	$\delta(I_8,))=I_{11}$
$\delta(I_8,+)=I_6$	$\delta(I_9,*)=I_7$			

Natijada 27.13 va 27.14-jadvallar asosida boshqaruv jadvalni tuzamiz. Bu jadval oldingi usul bilan hisoblangan 27.9- jadvali bilan bir hil bo'ladi.

LR-tahlilchilardagi to‘qnashuvlar

Shunday kontekstdan erkin grammatikalar mavjudki, ularga to‘g‘ridan-to‘g‘ri yuqorida ko‘rsatilgan algoritmlarni qo‘llab bo‘lmaydi. Bu ko‘p ma’nolik grammatikalar.

Har qanday LR-tahlilchi bunday grammatika uchun shunday konfiguratsiyada bo‘lib qolishi mumkinki, magazindagi zanjir va kiruvchi belgi bo‘yicha, nima qilish kerakligini bilmaydi, na ko‘chirishni, na o‘rashni (ko‘chirish-o‘rash to‘qnashuvi, “shift-reduce conflict”)? Undan tashqari bir nechta o‘rashdan qaysi biri qo‘llanilishi kerak (o‘rash-o‘rash to‘qnashuvi, “reduce-reduce conflict”)? Bunday grammatikalar LR(k)-grammatikalar sinfiga kirmaydi. Lekin bazi bir ko‘p ma’nolik grammatikalar dasturlash tillarni sintaktik konstruksiyalarini ta’riflashga qulay bo‘ladi. Bu holda qoidalar ihcham va tushunarli bo‘ladi. Albatta bunday grammatikalarni qo‘llaganda, biz to‘qnashuv muammosini hal qiladigan maxsus amallarni kiritishimiz kerak bo‘ladi.

Amallar ustunligi va bajarish yo‘nalishi

Arifmetik ifodani ta’riflaydigan quyidagi grammatika

$S \rightarrow S+S|S*S|(S)|i$

ko‘p manolik grammatika bo‘ladi, bu grammatika bo‘yicha amallarni (+,*) ustunligi va bajarish yo‘nalishi aniqlanmagan.

Aksincha quyidagi bir man’nolik grammatikada:

$S \rightarrow S+T|T$

$T \rightarrow T*E|E$

$E \rightarrow (S)|i$

“+” amaliga quyi ustunlik berildi va amallarning chapdan o‘nga bajarishligi ko‘rsatildi.

Birinchi grammatikani ikkinchiga nisbatan qulayligiga ikkita sabab bor.

Birinchidan, grammatika qoyidalarini va hosil bo‘lgan holatlarni o‘zgartirmasdan amallarni ustunligini va bajarish yo‘nalishini yengil o‘zgartirish mumkin.

Ikkinchidan, sintaktik tahlilchi ko‘p vaqtini ortiqcha qoidalarni o‘rashiga ketkazadi.

Birinchi grammatika uchun masalan SLR(1)-tahlilchini quraylik, shunda uning boshqaruv jadvali quyidagicha bo‘ladi:

27.15-jadval

Holat	Amal						O‘tish
	i	+	*	(	)	S	S
0	k3			k2			1
1		k4	k5			qq	
2	k3			k2			6
3		o‘4	o‘4		o‘4	o‘4	
4	k3			k2			7
5	k3			k2			8
6		k4	k5				
7		o‘1/ k4	o‘1/k 5		o‘1	o‘1	
8		o‘2/ k4	o‘2/ k5		o‘2	o‘2	
9		o‘3	o‘3		o‘3	o‘3	

27.15-jadvaldan ko‘rinib turibdi 7-chi va 8-chi holatlarda oldindan ko‘rinuvchi “+” va “*” belgilarga tegishli kataklarda to‘qnashuv sodir bo‘ldi.

Agar amallarning ustunligi va bajarish yo‘nalishi ma’lum bo‘lsa, to‘qnashuv muammosini yechish mumkin.

Bizga $i+i*i$ zanjir berilsin, bu zanjirni tahlil qilish jarayonida $i+i$ qayta ishlangandan so‘ng, SLR(1)-tahlilchini magazinida 0S1+4E7 zanjir turadi,va $*i$- kiruvchi zanjir qismi bo‘ladi. “*” amalni ustunligi yuqori bo‘lgani sababli, biz bu belgi uchun 7-chi holatda ko‘chirish amalini bajarishimiz kerak, ya’ni M[7,*]=k5.

Agar kiruvchi zanjir $i+i+i$ bo‘lsa, unda + amal chapdan o‘nga qarab bajarishini hisobga olsak, bu belgi uchun o‘rash amalini bajarishimiz kerak, ya’ni M[7,+]=o‘1.

Huddi shundak “*” amali chapdan o‘ngga qarab bajarilishi va uni “+” amaldan ustunroq bo‘lgani sababli, 8-chi holat uchun (magazin ustida S*S bo‘lgan holat), M[8,*]=o‘2 va M[8,+]=o‘2.

Natijada SLR(1)-tahlilchini boshqaruv jadvali quyidagicha bo‘ladi:

27.16-jadval

Holat	Amal						O'tish
	i	+	*	(	)	S	S
0	k3			k2			1
1		k4	k5			qq	
2	k3			k2			6
3		o'4	o'4		o'4	o'4	
4	k3			k2			7
5	k3			k2			8
6		k4	k5		k9		
7		**o'1**	**k5**		o'1	o'1	
8		**o'2**	**o'2**		o'2	o'2	
9		o'3	o'3		o'3	o'3	

Bu mulohazalarni boshqa LR-tahlilchilarga ham tarqatish mumkin.

"Osilib qolgan else" muommasi

Ko'pincha dasturlash tillarda shartli operatorni grammatikasi quyidagicha bo'ladi:

stmt→ ***if*** *expr* ***then*** *stmt* |***if*** *expr* ***then*** *stmt* ***else*** *stmt* | *a*

Bunda *stmt* – operator, *expr* – shartli ifoda, *a* – boshqa operatorlar.

Faraz qilaylik, bizga ***if*** *expr* ***then if*** *expr* ***then*** *a* ***else*** *a* berilsin, unda grammatikaga ko'ra bu operatorni ikki hil tushinish mumkin:

1. ***if*** *expr* ***then*** ***(if*** *expr* ***then*** *a* ***else*** *a)*, ya'ni ***else*** eng yaqin ***if*** ga tegishli;
2. ***if*** *expr* ***then*** ***(if*** *expr* ***then*** *a)****else*** *a*, ya'ni ***else*** eng uzoq ***if*** ga tegishli.

Shu sababli, bu grammatika ko'p ma'nolikdir, ya'ni "osilib qolgan *else*" muommasi bor.

Grammatikani soddalashtiramiz va unda "*i*" bilan ***if*** *expr* ***then*** belgilaymiz, "*e*" bilan ***else*** belgilaymiz shunda kengaytirilgan grammatikamiz quyidagicha bo'ladi:

0. S'→S

1. S→iSes

2. S→iS

3. S→a

Bu grammatika uchun SLR(1)-tahlilchini boshqaruv jadvali quyidagicha bo'ladi:

27.17-jadval

Holat	Amal				O'tish
	i	*e*	*a*	S	S
0	k2		k3		1
1				qq	
2	k2		k3		4
3		o'3		o'3	
4		k5/ o'2		o'2	
5	k2		k3		6
6		o'1		o'1	

27.17-jadvaldan ko'rinib turibdi 4-chi holatda oldindan ko'rinuvchi "e" belgiga tegishli katakda to'qnoashuv sodir bo'ldi.

Ko'pincha dasturlash tillarda ***else*** eng yaqin ***if***ga tegishli deb hisoblanadi,. Buni etiborga olganda biz ham ko'chirish amalini ma'qul deb hisoblaymiz, shunda natijaviy jadvalimiz quyidagicha bo'ladi:

27.18-jadval

Holat	Amal				O'tish
	i	*e*	*a*	S	S
0	k2		k3		1
1				qq	
2	k2		k3		4
3		o'3		o'3	
4		**k5**		o'2	
5	k2		k3		6
6		o'1		o'1	

Albatta bu grammatika uchun bir ekvivalent bo'lgan bir ma'nolik grammatika tuzib, unga tahlilchi qurish mumkin,. Lekin bunday tahlilchida holatlar va vaziyatlar soni oshib ketadi va boshqaruv jadvali murakkablashadi.

LR-tahlilchilarda hatolikdan keyin tiklash

LR-tahlilchi hatolarni amal jadvaliga murojaat qilganda aniqlaydi.

Bunday hodisa, boshqaruv jadvalining katagida hatolik vaziyati (bo'sh) bo'lsa, sodir bo'ladi.

Bu holda LR-tahlilchi tekshirishni to'g'ri davom ettirish uchun to'g'ri bo'lgan holatgacha magazinni tiklash kerak bo'ladi. Buning uchun magazindagi belgilar cho'qqisidan boshlab ketma-ket ko'rib chiqiladi va shunday s holatni topadiki , o'tish jadvalning biror bir noterminal A uchun qiymat mavjud bo'lgunucha,va shu holatgacha bo'lgan belgilar magazindan olib tashlanadi. Toki A dan keyin bo'lishi mumkin bo'lgan belgi uchramaguncha kiruvchi zanjirdan bir nechta belgilar o'tkazib yuboriladi. Magazinga M[s,A] holat raqami yoziladi va tahlil davom ettiradi. A noterminalni tanlash varianti bir nechta bo'lishi mumkin, ko'pincha A sifatida katta operatorlar, yoki bloklar tanlanadi yoki ***end*** olish mumkin.

Har bir bo'sh katak uchun, hatoliklarni qayta ishlaydigan protsedurada alohida ko'rsatmalar yozish mumkin.

Misol. Bizga ifodalar grammatikasi berilsin:

$S \rightarrow S+S|S*S|(S)|i$

27.19-jadvalda shu grammatika uchun SLR(1) tahlilchining boshqaruv jadvali keltirilgan. Bu jadvalni 27.16-jadvaldan farqi shundaki, unda hatolarni qayta ishlash dasturlar raqami bilan to'ldirilgan.

27.19-jadval

Holat	Amal						O'tish
	I	+	*	(	)	*S*	*S*
0	k3	h1	h1	k2	h2	h1	1
1	h3	k4	k5	h3	h2	qq	
2	k3	h1	h1	k2	h2	h1	6
3	o'4	o'4	o'4	o'4	o'4	o'4	
4	k3	h1	h1	k2	h2	h1	7
5	k3	h1	h1	k2	h2	h1	8
6	h3	k4	k5	h3	k9	h4	
7	o'1	o'1	k5	o'1	o'1	o'1	
8	o'2	o'2	o'2	o'2	o'2	o'2	
9	o'3	o'3	o'3	o'3	o'3	o'3	

Holatdagi o'rashlar barcha belgilarga kengaytirilgan. Bunday almashtirish hatoliklarni o'rashdan keyin aniqlashiga olib keladi. Hatolik kataklarda maxsus dasturlarga murojaatlar yozilgan. Hatoliklarni qayta ishlash qism-dasturlari tafsiflari quyidagicha:

h1: Vaziyat:Bu qism-dasturi 0,2,4 va 5-holatlardan chaqiriladi,. Ularda *i* operand yoki chap qavs belgilar o'rniga "+" va "*" belgilar yoki zanjir tugash belgisi kelganda quyidagi amal bajarilada.

Amal: Magazinga "*i*" belgisi va 3 holatni joylashtiriladi. "Operand yo'q" habar beriladi.

h2: Vaziyat: Bu qism-dasturi 0,1,2,4 va 5 holatlardan o'ng qavs kelganda chaqiriladi.

Amal: O'ng qavsni kiruvchi zanjirdan olib tashlanadi. "Noto'g'ri o'ng qavs" habar beriladi.

h3: Vaziyat: Bu qism-dasturi 1 va 6 holatlarda amal kelishi, o'rniga *i* operandi yoki chapqavsni kelganda chaqiriladi.

Amal: Magazinga + belgi va 6 holat joylashtiriladi. "Amal belgisi yo'q" habar beriladi.

h4: Vaziyat: Bu qism-dasturi 6 holatda amal yoki o'ng qavs kelishi, o'rniga tugash belgisi kelganda chaqiriladi.

Amal: Magazinga) belgi va 9 holat joylashtiriladi. "O'ng qavchs yo'q" habar beriladi.

Kiruvchi "*i+)*" zanjir uchun, tahlilchini qadamlari quyidagicha bo'ladi:

1. *(i+)S, 0 , ε)|— k3*
2. *(+)S, 0i3 , ε)|— o'4*
3. *(+)S, 0S1 , 4)|— k4*
4. *( )S, 0S1+4 , 4)|— e2*
5. *(S, 0S1+4 , 4)|— e1* "Noto'g'ri o'ng qavs") olindi
6. *(S, 0S1+4i3 , 4)|— o'4* "Operand yo'q"*i3* yozildi
7. *(S, 0S1+4S7 , 44)|— o'1*
8. *(S, 0S1 , 441)*

Xulosa

LR(k) grammatikaga asoslangan aniqlovchi kiruvchi zanjirni chapdan o'ngga ko'rib chiqadi (left) va o'ng tahlilni bajaradi (right),.

Surish yoki o'rashni tanlashda, joriy belgini o'ng tarafida turgan k-ta belgilar ketma-ketligini hisobga oladi. Amalda asosan LR(0) yoki LR(1) grammatikalar ishlatiladi. LR(k)-tahlilda "ko'chirish-o'rash" usuli qo'llanadi (shift-reduce). Bu usul determinantlangan kengaytirilgan magazin hotirali o'zgartirgichni qo'llaydi. Bu grammatikalarni aniqlovchilari bir xil, ular faqat boshqaruvchi jadvalni qurishi bilan farq qiladi. Boshqaruvchini qurish jarayonida LR(k)-vaziyatlar ishlatiladi va ular asosida chekli avtomatning holatlar diagrammasi quriladi. SLR(1)-tahlilchida tahlilchida bazi bir vaziyatlar xisobga olingan va soddalashtirilgan. LR(1)-tahlilchi amalni aniqlashda kiruvchi zanjirning bitta joriy belgisidan foydalanadi. LALR(1) – tahlilchi LR(1)-tahlilchiga qaraganda ancha ihcham, lekin unga teng kuchlikdir. Masalan, Pascal tili grammatikasi uchun LALR(1) – tahlilchida yuztaga yaqin holatlar bo'lsa, LR(1)-tahlilchida bir nechta ming holatlar mavjud.

Ko'p ma'nolik kontekstdan erkin grammatikalarga to'g'ridan to'g'ri yuqoridagi algoritmlarni qo'llab bo'lmaydi. Bunday grammatikalarda kiruvchi belgi uchun nimani qo'llashi qo'chirishni yoki o'rashni aniq bo'lmaydi bu esa to'qnashuvga olib keladi, uni yechish uchun amallar ustuvorligi va bajarish yo'nalishi hisobga olinadi. Undan tashqari "osilib qolgan else" muammosini yechish kerak bo'ladi.

Nazorat uchun savollar

1. LR(k)- grammatikani ta'rifini keltiring.
2. LR(k)- tahlilchining ishlash tamoyilini keltiring.
3. LR(k)- tahlilchini ishlash algoritmini nimadan iborat?
4. Amal va o'tish funksiyalarni ma'nosi nimada?
5. Vaziyat tushunchasini izohlang.
6. clouser() va goto() funksiyalarni tuzish algoritmlarini keltiring.
7. LR(0)-tahlilchini holatlar to'plamini qurish algoritmini keltiring.
8. LR(0)-tahlilchini boshqaruv jadvalini tuzish algoritmini keltiring.
9. SLR(1)-tahlilchini holatlar to'plamini qurish algorimini keltiring.
10. SLR(1)-tahlilchini boshqaruv jadvalini tuzish algoritmini keltiring.
11. LR(1)-tahlilchini holatlar tizimini qurish algoritmini keltiring.
12. LR(1)-tahlilchini boshqaruv jadvalini tuzish algoritmini keltiring.

13. LALR(1)-tahlilchini holatlar tizimini qurish algorimini keltiring.
14. LALR(1)-tahlilchini boshqaruv jadvalini tuzish algoritmini keltiring.
15. LALR(1)-tahlilchini holatlar tizimini samarali qurish algorimini keltiring.
16. LALR(1)-tahlilchini boshqaruv jadvalini samarali tuzish algoritmini keltiring.
17. To'qnashuvlar turlarini keltiring.
18. Ko'p ma'nolik grammatikalarni ishlatish afzalligi nimada?
19. Amallarni ustunligi va bajarish yo'nalishini hisobga olish qanday bo'ladi?
20. "Osilib qolgan else" muammosi qanday yechilishi mumkin?
21. LR-tahlilchilarda, hatoliklardan keyingi tiklash qanday bajariladi?

Amaliyet uchun topshiriqlar

1. Berilgan G(VT,VN,P,S) uchun LR(0)-tahlilchini boshqaruv jadvalini quring va ishlash jarayonini ko'rsating:

a) $S \to aSb \mid aSc \mid ab$ b) $S \to aSSb \mid aSSS \mid c$ c) $S \to a \mid (SR \mid \wedge SR$
$R \to)$

d) $S \to cA \mid ccB \mid$
$A \to cA \mid a$
$B \to ccB \mid b$

e) $S \to aAd \mid bAc$
$A \to e \mid \varepsilon$

2. Berilgan G(VT,VN,P,S) uchun SLR(1)-tahlilchini boshqaruv jadvalini quring va ishlash jarayonini ko'rsating:

a) $S \to bASB \mid bA$
$A \to bA \mid dSca \mid c$
$B \to cAa \mid c$

b) $S \to Bv \mid vCc$ $S \to B \mid A$
$A \to vBS \mid u$ $B \to C;D$
$B \to yw \mid u$ $C \to C;d \mid bd$
$C \to yAw \mid cBy$

d) $S \to cA \mid ccB \mid o$
$A \to cA \mid a$
$B \to ccB \mid b$

e) $S \to aAd \mid bAc$
$A \to e \mid \varepsilon$

IV-BO‘LIM. TRANSLYATSIYA NAZARIYASI ELEMENTLARI

28-BOB. LEKSIK TAHLIL

Tayanch iboralar: *leksema, leksemalar jadvali, identifikatorlar jadvali, ketma-ket leksik tahlil, parallel leksik tahlil, model til, leksik tahlil holatlar diagrammasi.*

Leksik tahlil vazifalari

Tilning elementar belgilaridan tashkil qilingan bo‘linmas konstruksiyasi - leksema(token) deb nomlanadi. Leksema til birligi bo‘lib, uning asosiy tushunchasi hisoblanadi. Agar leksema bo‘linsa, uning til nuqtayi nazariyasidan ma’nosi yo‘qoladi. Tabiiy tillarda leksema bu so‘zni bildiradi. Dasturlash tillarda leksema deb identifikator, o‘zgarmas, kalit so‘zi, amal belgilari, ajratuvchilar va nishonlar xisoblanadi. Leksik tahlilchi (skaner) kompilyator qismi bo‘lib, dastlabki dastur matnni o‘qiydi va undagi leksemalarni ketma-ket ajratib beradi. Leksik tahlilchi vazifasini sintaktik tahlilchi ham bajarishi mumkin, lekin leksik tahlilchi alohida bo‘lishini sababi ham bor:

1)leksik tahlilchini ishlatish natijasida boshlang‘ich matndan leksemalar ajratib beriladi va ortiqcha narsalar olib tashlanadi (probellar, kommentariyalar va h.q.), natijada dastur qisqartiriladi;

2)leksik tahlil jarayonida sodda va samarali algoritmlar qo‘llanadi, aksincha sintaktik tahlilchining algoritmlari murakkabdir;

3)leksik tahlilchi murakkab sintaktik tahlilchini boshlang‘ich matnning ustidan ishlashni erkin qiladi, va unga tayyor kerakli ko‘rinishda leksemalarni uzatib beradi;

4)agar dasturlash tilining kodirovkasi o‘zgarilsa, u faqat leksik tahlilchiga ta’sir qiladi.

Leksik va sintaktik tahlil bir-biriga bog‘langan holda, ikkita usul bilan o‘zaro ishlashi mumkin. Bu ketma-ket usuli va parallel usuli.

Birinchi usulda leksik tahlilchi matnni to‘liq ko‘rib chiqib, uni biror bir tuzim ko‘rinishda sintaktik tahlilchiga tayyorlab beradi. Bu tuzim leksemalar jadvali deb nomlanadi. Leksem jadvali tashqi hotiraga yozib qo‘yilishi mumkin. Jadvalda leksemani ko‘rinishdan tashqari uning turi va

maxsus kodi saqlanishi mumkin. Bu jadvaldan tashqari yana boshqa jadvallar hosil bo‘lishi mumkin,. Masalan identifikatorlar jadvali, o‘zgarmaslar jadvali va hokazo.

Ikkinchi usulda, ishni sintaktik tahlilchi boshlaydi va konstruksiyalarni tekshirish jarayonida leksik tahlilchini chaqirib, joriy leksemani talab qiladi, va kutilgan leksema qoidalarga rioya qilsa, uni jadvallarga yozadi va keyingi leksemaga o‘tadi.

Birinchi usul ikkinchiga qaraganda ma’quldir.

Leksemalar turlarga bo‘linadi,va ko‘pincha quyidagi turlar ishlatiladi:

- Identifikatorlar
- Kalit so‘zlar
- O‘zgarmaslar
- Ajratuvchilar(amallar)

Bazi bir turlar juftlik bilan berilishi mumkin:

(leksema turi, leksema manzili)

Yuqorida ko‘rsatilgan leksemalar muntazam grammatika yoki muntazam ifodalar bilan berilishi mumkin va uni chekli avtomat bilan amalga oshirish mumkin.

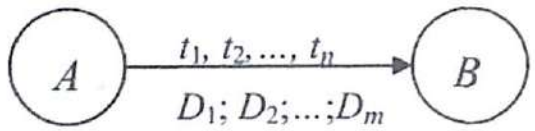
Leksik tahlilchi zanjirni tilga kirishini aniqlashdan tashqari yana bir nechta masalalarni yechish kerak:

- U dastlabki dasturdan leksemani ajratib olishi va to‘g‘ri yozilganligini aniqlash kerak.
- Leksemani kerakli jadvallarga joylashtirish kerak, yoki jadvalda borligini aniqlash kerak.
- Berilgan zanjirni juftlikka qayta ishlash kerak.
- Probel, izoh va boshqaruv belgilarni olib tashlashi kerak.

Leksik tahlilchi bu translyator, unda kiruvchi zaǹjir - dastur matni, chiquvchi zanjir bu leksemalar ketma-ketligi. Bu ketma-ketlik leksemalar jadvali deb nomlanadi.

Leksik tahlilchini chekli o‘zgartirgich orqali amalga oshirish mumkin.

Leksik tahlilni to‘liq bajarish uchun holat diagrammalarga amallar qo‘shamiz(28.1-rasm).



28.1 rasm. Holat diagrammasida amallarni ko'rsatish.

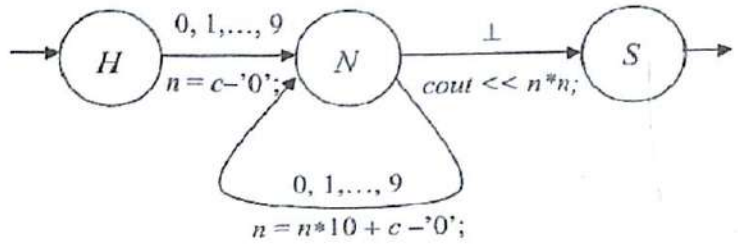
Bunda t_i terminal belgilar, D_i bajarish kerak bo'lgan amallar.

Misol: Bizga ishorasiz o'nli butun son berilsin va uni qiymatining kvadrati chop etilsin. Uning grammatikasi quyidagicha bo'ladi:

S→N⊥

N→0|1|…|9|N0|N1|…|N9

Sonni har bir raqamini o'qiganda (28.2-rasm) Gorner chizmasi bo'yicha uning qiymatini hisoblaymiz,. Ohirgi belgi o'qilganda kvadratini chop etamiz:



28.2-rasm. Amallar qo'shilgan butun sonlar diagrammasi.

Model tili

Leksik tahlilni amalda ko'rishdan oldin paskal tiliga o'xshagan model tilini kiritamiz, uni **M-til** deb nomlaymiz va bu til grammatika qoidalarini Kengaytirilgan Bekus-Naur shakli orqali quyidagicha beramiz:

P |— **program** D1; B⊥
D1 |— **var** D {;D}
D |— I {,I}: **(int ,bool)**
B |— **begin** S {;S} **end**
S |— I := E |**if** E **then** S **else** S |**while** E **do** S | B |**read** (I) | **write** (E)

E |— E1 | E1 (= ,<>, !=) E1
E1 |— T {(+ , - ,**or**) T}
T |— F {(* , / ,**and**) F}
F |— I | N | L | (+,-,**not**) F | (E)
L |— **true** | **false**
I |— C | IC | IR
N |— R | NR
C |— a | b | ... | z | A | B | ... | Z
R |— 0 | 1 | 2 | ... | 9

Bunda R grammatika boshlang'ich belgisi, "⊥" belgisi dastur tugashini bildiradi:

Amallarni bajarish ustunligi sintaksis bilan berilgan.

Dasturni ixtiyoriy joyida probellar va izohlar turishi mumkin. Izoh "{" va "}" belgilar bilan ajratiladi, izoh ichida "}" va "⊥" belgilar bo'lishi mumkin emas. **true, false, read** va **write** – bu kalit so'zlar deb xisoblanadi.

Leksik tahlilchini tuzimi.

Yendi bevosita leksik tahlilchiga o'tamiz.

Unda quyidagi jadvallar ishlatiladi:

TW — M-tilni kalit so'zlar jadvali;

TD — ajratuvchilar jadvali;

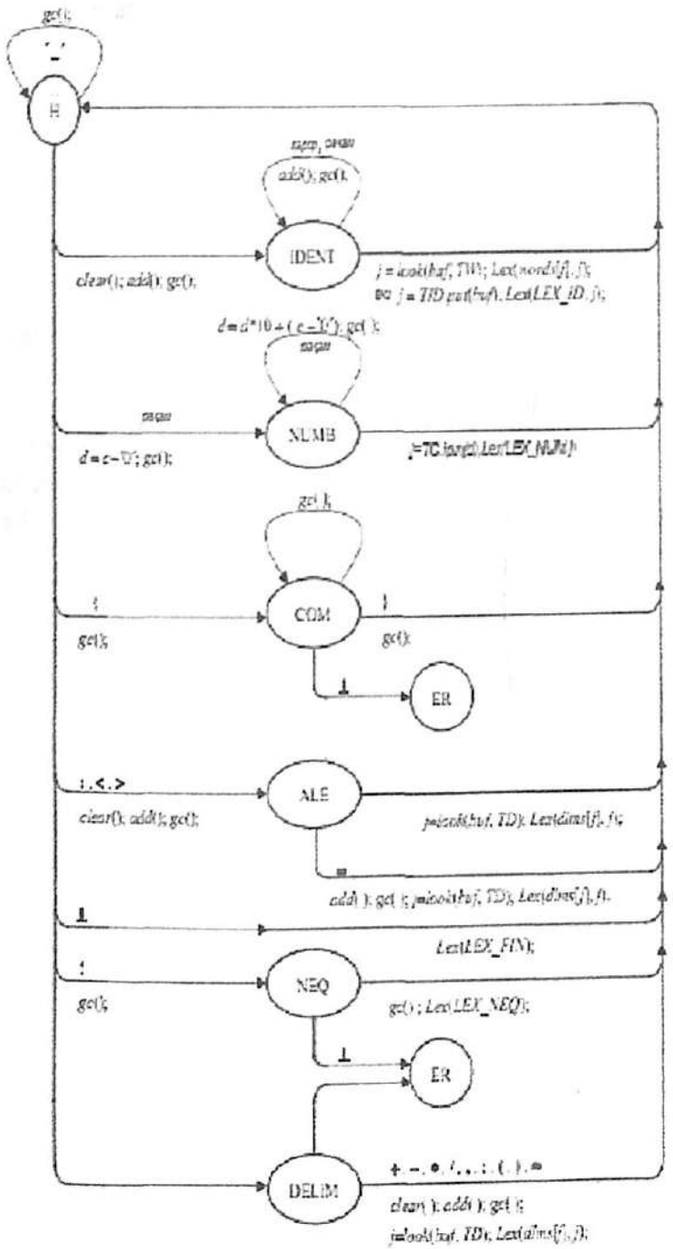
TID — identifikatorlar jadvali;

TC- o'zgarmaslar jadvali;

TW va TD jadvallar oldindan to'ldiriladi, chunki ular dastlabki dasturga bog'liq emas va bu jadvallar o'zgarilmaydi.

TID va TC jadvallar tahlil jarayonida tashkil qilinadi va to'ldiriladi. Bu jadvallarni sinf obyekti ko'rinishda, yoki massiv ko'rinishda tuzish mumkin.

Leksik tahlilchining holat diagrammasi 28.3-rasmda keltirilgan.



28.3-rasm Leksik tahlilchini holatlar diagrammasi.

XHar bir leksema turini kodlab chiqamiz, uning uchun sanagich turini ishlatamiz:

```
#include <iostream.h>
enum type_of_lex{ LEX_NULL, LEX_PROGRAM,
LEX_VAR, LEX_BOOL, LEX_INT, LEX_FALSE, LEX_TRUE,
LEX_BEGIN, LEX_END, LEX_ASSIGN /*:=*/, LEX_IF,
LEX_THEN, LEX_ELSE, LEX_WHILE, LEX_DO, LEX_READ,
LEX_WRITE, LEX_AND, LEX_NOT, LEX_OR, LEX_LSS,
LEX_LEQ, LEX_EQ, LEX_NEQ, LEX_GEQ, LEX_GTR,
LEX_DIV, LEX_PLUS, LEX_MINUS, LEX_MULT,
LEX_LPAREN/*(*/, LEX_RPAREN, LEX_COMMA/*:*/,
LEX_COLON/*,*/, LEX_SEMICOLON/*;*/,
LEX_ID,LEX_NUM,LEX_FIN}
```

Leksema sinfini e'lon qilamiz:

```
class  lex{
type_of_lex t_lex;int v_lex;
public:
  lex(type_of_lex
t=LEX_NULL,intv=0){t_lex=t;v_lex=v}
  type_of_lex get_type(){return t_lex;}
int get_value(){return v_lex;}
void set_value(type_of_lex t_l=LEX_NULL,int
v_l=0)
{t_lex=t_l;v_lex=v_l;}
friend ostream& operator << (ostream &, lex);
};
friend ostream& operator << (ostream &s, lex
l){
s<<"(turi="; s.width(2); s<< l.t_lex<<
",qiymati="<<l.v_lex<<")";
if (l.t_lex==LEX_NUM){ s<<"  ";
s.whidth(2);s<<
TC[l.v_lex].get_value();}
if (l.t_lex==LEX_ID)
```

```
    { s <<"  "<< TID[l.v_lex].get_name();}
    s << endl; return s;}
```

Identifikatorlar uchun alohida sinf e'lon qilamiz:

```
    class ident
    {
    char    * name; /* identifikator ismi */
    booldeclare; /* identifikator ta'riflanganligi
*/
     type_of_lex type; /* identifikator turi */
    boolassign; /* identifikatorga qiymat
berilganligi*/
    intvalue; /* identifikator qiymati */
    public:
    ident (){declare = false;assign  = false;}

      char *get_name (){return name;}
      void put_name (const char *n)
     { name = new char [ strlen(n) + 1 ];
      strcpy ( name, n ); }
      bool get_declare(){return declare; }
     void put_declare (){declare = true;}
     type_of_lex get_type (){return type; }
     void put_type (tipe_of_lex t){type = t;}
     bool get_assign (){return assign;}
      void put_assign (){assign = true;}
      int get_value (){return value; }
      void put_value (int v){value = v; }
    };
```

O'zgarmas o'zgarmaslarni (butun sonlar uchun) sinfini e'lon qilamiz:

```
class literal {
    int liter;
    public: literal(){liter=0;} // konstruktor
    int get_value(){return liter;}
```



```
    void set_value(int n){liter=n;}
    type_of_lex get_type(){return LEX_NUM;}
};
```

Leksik tahlil jarayonida bir nechta jadvallar hosil qilinadi, shu jumladan identifikatorlar jadvali,va bu jadvalni sinf orqali tasvirlaymiz:

```
class tabl_ident
{
 ident * p;
  int   size;
  int   top;
public:
tabl_ident(int max_size){
p = newident[size=max_size];
top = 1;}
~tabl_ident (){delete []p;}
ident& operator[](int k){return p[k];}
int put(const char *buf );
};
int tabl_ident::put(const char *buf )
{ for (intj=1; j<top; ++j )
   if ( !strcmp(buf, p[j].get_name()) )
    return j;
 p[top].put_name(buf);
 ++top;
  return top-1;
}
```

O'zgarmaslar jadvali sinfi quyidagicha:

```
class  tabl_liter {
    literal *liter; // literal jadvalismi
     int size; // jadvalo'lchami
     int top; // joriy bo'sh elementi
    public:
tabl_liter (int max_size){liter=new literal
```

```
    [size=max_size];          top=1;} //
    konstruktor
    ~tabl_liter (){delete [] liter;} // destruktor
    literal& operator[](int k){return liter[k];}
    int cput(literal n){
    for  (int j=1; j<top;j++)
    if (n.get_value()==liter[j].get_value())return
j;
    liter[top].set_value(n.get_value());
    return top ++;}
    };
```

Agar birinchi usul qo'llansa, leksemalar jadvali ham tuzilishi kerak. Uning sinfi quyidagicha bo'ladi:

```
    class tabl_lexem
    {
     lex *lexem;  // jadval ismi
      int   size; // uzunligi
    int   top;  // joriy bo'sh elementi
    public:
    tabl_lexem(int max_size){lexem =
newident[size=max_size];  top = 1;}
    ~tabl_lexem (){delete [] lexem;}
    lex& operator[](int k){return lexem[k];}
    int lput(lex l)
{lexem[top].set_value(l.get_type(),l.get_value());
    returntop ++;}
    };
```

Bevosita leksik tahlilchining dastursi quyidagi sinfda berilgan:

```
    class Scanner{
     enum state{H, IDENT, NUMB, COM, ALE, DELIM,
NEQ };
    static char  * TW[];/* kalitso'zlar jadvali */
```



```
    static type_of_lex  words[];
    static char  * TD[];/* ajratuvchilar jadvali
*/
    static type_of_lex  dlms[];
     state    CS; /* joriyholat */
     FILE     * fp;
    char      c; /* joriybelgi*/
    char      buf[80];
    int     buf_top;
    /* joriyleksemabelgilarniyig'ish uchun buferni
tozalash funksiyasi*/
    void clear (){buf_top=0;for(int j =
0;j<80;++j)
       buf[j] = '\0';}
    /* buferga joriy belgini qo'shish funksiyasi
*/
    void add(){buf[buf_top++] = c;}
    /* so'zni massivdan(jadvaldan) qidirish
funksiyasi */
    int look ( const char *buf, char **list )
     {int i = 0;
      while (list[i])
      {if (!strcmp(buf,list[i]))return i;
       ++i;
      }
       return 0;
     }
     /* joriy belgini fayldan o'qish funksiyasi */
    void gc(){c =fgetc(fp);}
    public:
    /* konstruktor */
    Scanner ( const char * program )
     { /* boshlang'ich tayyorlash amallari */
      fp = fopen ( program, "r" );
      CS = H;
```

```
  clear();
  gc();
 }
lex get_lex();
};
/* leksik tahlilni bajaradigan funksiya,
natijada joriy  leksema turini va manzilini
beradi(jadvallar uchun)*/
lex Scanner::get_lex ()
{ int d, j;Literal n;lex l;
  CS = H;
  do
 {
   switch ( CS )
  {
   case H:
    if (c ==' '||c
=='\n'||c=='\r'||c=='\t')gc();
    elseif(isalpha(c))
  {clear();add();gc();CS=IDENT;}
  elseif (isdigit(c)){d=c-'0';gc();CS = NUMB;}
    elseif(c=='{'){gc();CS = COM;}
    elseif(c==':'||c=='<'||c=='>')
  {clear();add();gc();CS=ALE;}
    else if(c=='@')return lex(LEX_FIN);
   else if(c=='!'{clear();add();gc();CS=NEQ;}
    else CS = DELIM;
    break;
   case IDENT:
    if (isalpha(c)||isdigit(c)){add();gc();}
    else if (j=look(buf,TW))return
lex(words[j],j);
    else{j=TID.put(buf); return
lex(LEX_ID,j);}
    break;
```



```
   case NUMB:
 if (isdigit(c)){d=d*10+(c-'0');gc();}
  else {n.set_value(d);j=TC.lput(n);
    return lex(LEX_NUM,j)};
break;
   case COM:
if (c=='}'){gc();CS = H;}
else if(c=='@'||c=='{')throw c;
else gc();
break;
   case ALE:
if ( c=='='){add();gc();j=look(buf,TD);
    return Lex(dlms[j],j);}
else{j=look(buf,TD);return lex(dlms[j],j);}
break;
   case NEQ:
if(c=='='){add();gc();j=look(buf,TD);
    return lex(LEX_NEQ,j);}
else throw '!';
break;
   case DELIM:clear();add();
if(j=look(buf,TD)){gc();
return lex(dlms[j],j);}
else throw c;
break;
  }    // end switch
 }
  while (true);
}
```

M-tili asosida qurilgan jadvallarini quyidagicha tasvirlash mumkin:

```
/* kalit so'zlar nomi jadvali*/
char * Scanner::TW[] =
{
 "","and", "begin", "bool", "do", "else",
"end",   "if",  "false", "int","not", "or",
```

```
    "program","read",  "then",  "true",   "var",
    "while",  "write",  NULL
    };
    //Bunda 0 element ishlatilmaydi
    /* ajratuvchilar jadvali*/
    char * Scanner:: TD[] =
    {
     "",";", "@", ",", ":", ":=", "(", ")", "=",
"<",
     ">", "+", "-", "*",  "/", "<=", "!=",  ">=",
NULL
    };
    /*kalit so'zlarni turi nomi */
    type_of_lex Scanner::words[] =
    {
     LEX_NULL, LEX_AND, LEX_BEGIN, LEX_BOOL,
LEX_DO,
     LEX_ELSE, LEX_END, LEX_IF, LEX_FALSE,
LEX_INT,
     LEX_NOT, LEX_OR, LEX_PROGRAM, LEX_READ,
LEX_THEN,
     LEX_TRUE, LEX_VAR, LEX_WHILE, LEX_WRITE,
LEX_NULL
    };
    /* ajratuvchilar nomi*/
    type_of_lex Scanner::dlms[] =
    {
     LEX_NULL, LEX_SEMICOLON, LEX_FIN, LEX_COMMA,
     LEX_COLON, LEX_ASSIGN, LEX_LPAREN,
LEX_RPAREN,
     LEX_EQ, LEX_LSS, LEX_GTR, LEX_PLUS,
LEX_MINUS,
     LEX_TIMES,  LEX_SLASH, LEX_LEQ, LEX_NEQ,
LEX_GEQ,
     LEX_NULL
    };
```



```
    /*Global jadvallar */
    tabl_ident TID(100); /* identifikator jadvali
*/
    tabl_liter TC(100); /* o'zgarmaslar jadvali */
    tabl_lexem TLEX(1000); /* leksemalar jadvali,
birinchi usul uchun */
```

Birinchi usul uchun asosiy dastur quyidagicha bo'ladi:

```
#include <iostream.h>
int main ()
{ lex l;
  try
 {      Scanner S("program.txt" );
    do
     {l=S.get_lex();
      cout <<l;}
    while (l.get_type()==LEX_FIN)
   return 0;
 }
  catch ( char c )
 {
cout<< "noma'lum belgi " <<c<<endl;
return 1;
 }
  catch ( lex l )
 {
  cout << "noma'lum leqsema";
  cout << l;
   return 1;
 }
  catch ( const char *source )
{      cout << source << endl;
   return 1;
 }
}
```

Xulosa

Leksik tahlilchi dastlabki dastur matnni o'qiydi va undagi leksemalarni ketma-ket ajratib beradi. Leksik tahlilchi vazifasini sintaktik tahlilchi ham bajarishi mumkin, lekin leksik tahlilchi alohida bo'lishini sababi ham bor bu leksik tahlilchini ishlatish natijasida ortiqcha narsalar olib tashlanadi (probellar, kommentariyalar va h.q.), natijada dastur qisqartiriladi, leksik tahlil jarayonida sodda va samarali algoritmlar qo'llanadi, aksincha sintaktik tahlilchining algoritmlari murakkabdir, leksik tahlilchi sintaktik tahlilchiga tayyor kerakli ko'rinishda leksemalarni uzatib beradi,agar dasturlash tilining kodirovkasi o'zgarilsa, u faqat leksik tahlilchiga ta'sir qiladi.

Leksik va sintaktik tahlil ikkita usul bilan o'zaro ishlashi mumkin. Bu ketma-ket usuli va parallel usuli.

Birinchi usulda leksik tahlilchi matnni to'liq ko'rib chiqib, uni biror bir tuzim ko'rinishda sintaktik tahlilchiga tayyorlab beradi. Ikkinchi usulda, ishni sintaktik tahlilchi boshlaydi va konstruksiyalarni tekshirish jarayonida leksik tahlilchini chaqirib, joriy leksemani talab qiladi, va kutilgan leksema qoidalarga rioya qilsa, uni jadvallarga yozadi va keyingi leksemaga o'tadi.

Leksik tahlilchi bu translyator, unda kiruvchi zanjir - dastur matni, chiquvchi zanjir bu leksemalar ketma-ketligi. Bu ketma-ketlik leksemalar jadvali deb nomlanadi.

Leksik tahlilchini chekli o'zgartirgich orqali amalga oshirish mumkin.

Nazorat uchun savollar

1. Leksemani ta'rifini keltiring.
2. Leksema tahlilni asosiy vazifalarini keltiring.
3. Sinf asosida leksemalar jadvalini tuzimini keltiring.
4. Sinf asosida identifikator jadvalini tuzimini keltiring.
5. Leksik tahlilni ikki usulini keltiring.
6. Leksik tahlilni holatlar diagrammasiga izoh bering.

Amaliyot uchun topshiriqlar

1. Yuqorida berilgan model tilning grammatikasiga quyidagi qo'shimchalar kiriting va mos ravishda leksik tahlil dastursini o'zgartiring:

a) haqiqiy son;

b) haqiqiy tur;

v) repeas untile – sikl operatori;

g) standart funksiyani chaqirish.

29-BOB. SINTAKTIK VA SEMANTIK TAHLIL

Tayanch iboralar: *sintaktik taxlil, kengaytirilgan rekursiv tushish usuli, M-tilni sintaktik taxlili, M-tilni semantik taxlili, tafsiflarni qayta ishlash,ifodalarni kontekst shartlarini nazorat qilish, operatorlarni kontekst shartlarini nazorat qilish .*

Sintaktik tahlilchini vazifalari

Sintaktik tahlil jarayonida :

1) Terminal sifatida leksik tahlilchidan chiqgan leksemalar hisoblanadi.
2) Berilgan leksemalar zanjirlari til sintaksisiga mos kelishini aniqlanadi.
3) Zanjirni tuzimi saqlanib qolinadi.

Ya'ni bizga leksemalar zanjiri berilgan, uni tilning grammatikasi bo'yicha keltirib chiqarish mumkinligini aniqlash kerak.. Agar chiqarilsa, uni ko'rsatish kerak yoki keltirib chiqarish daraxtini qurish kerak. Biz formal grammatikada ko'rsatkan edik, dasturlash tillarni sintaksisini tavsiflash uchun kontekstdan erkin grammatika imkoniyati yetarlikdir,. Bu grammatikada tahlilni bir nechta usullari mavjud, lekin amalda chiziqli hossaga ega bo'lgan grammatika sinflari va uning tahlil algoritmilari qo'llanadi,. Ulardan biri pasayuvchi rekursiv tushish usuli.

Kengaytirilgan rekursiv tushish usuli

Bizga ma'lumki rekursiv tushish usulni qo'llash uchun asosiy shartlari quyidagicha:

a) $A\to\alpha$, bunda $\alpha\in(VTUVN)^*$va A noterminal uchun bu yagona qoida.

b) $A\to a_1\gamma_1|a_2\gamma_2|\dots\ |a_n\gamma_n$, $a_i\neq a_j$, agar $i\neq j$, $a_i\in VT, \gamma_i\in(VTUVN)^*$ barcha $i=1,2\dots n$ uchun

v) $A\to a_1\gamma_1|a_2\gamma_2|\dots\ |a_n\gamma_n|\varepsilon$, $a_i\neq a_j$, agar $i\neq j$, $a_i\in VT$, $\gamma_i\in(VTUVN)^*$ barcha $i,j=1,2\dots n$ uchun va FIRST(A)∩FOLLOW(A)=∅.

Albatta bunday shartlar sintaksisga ma'lum chegaralar qo'yadi. Lekin grammatika qoidalar ustidan ketma-ket qayta o'zgartirishlar bajarish natijasida biz kerakli grammatikaga olib kelishga harakat qilsak bo'ladi.

Dasturlash tillar sintaksisini tavsiflashda, ro'yhatlarni ta'riflaydigan qoidalar tez-tez uchraydi,. Shu jumladan o'zgaruvchilarni ta'riflashdagi

identifikatorlar ro'yxati, funksiyani parametrlar ro'yxati, operatorlar ro'yxati. Bunday qoidalarni shakli quydagicha bo'ladi $A\to a|\ a,A$.

Ko'pincha dasturlash tillarni sintaksisi KBNSH bilan beriladi,. Unda$\{\alpha\}$ ko'rinishdagi shakl. α konstruksiyani umuman bo'lmasligi yoki bir necha marta qaytarilishini bildiradi. Bunday shakl iteratsiya deb nomlanadi.

$A\to a|a,A$ qoidalarni KBNShda $A\to a\{,a\}$ ko'rinishda yozish mumkin. Teskarisi ham to'g'ri, ya'ni iteratsiya bor $A\to\alpha\{\beta\}\gamma$ qoidani, iteratsiyasiz qoidalarga olib kelishi mumkin:

$A\to\alpha\qquad B\gamma$

$B\to\beta\ B|\varepsilon$

Bunda B yangi kiritilgan noterminal.

$A\to a\{,a\}$ qoidaga rekursiv tushish usulini qo'llash uchun uni quyidagi ekvivalent shakliga qayta o'zgartirish kerak:

$A\to aB$

$V\to,aB|\ \varepsilon$

Bu qoidalarga usulni qo'llash mumkin chunki, ikkinchi qoida uchun: $FIRST(B)\cap FOLLOW(B)=\varnothing$.

Bu qoidalarga protseduralar yozish mumkin, lekin qo'llash mumkinligini bilganimizdan keyin dastlabki qoidaga qaytish ma'qulroqdir. Bunday usul kengaytirilgan rekursiv usuli deb nomlanadi. Bu usulda iteratsiyani amalga oshirish uchun noterminalga tegishli protsedurani ichida sikl ishlatiladi masalan $A\to a\{,a\}$qoida uchun protsedurasi quyidagicha bo'ladi:

```
void A ()
{
  if ( c != 'a' ) throw c; // hato
 gc (); // keyingi belgini o'qish
  while ( c == ',' )
 {
  gc ();
   if ( c != 'a' ) throw c;
   else gc ();
 }
}
```

Yana bir misol sifatida arifmetik ifoda qoidalarini ko'ramiz. Bu qoidalardan biri iteratsiya orqali quyidagicha yoziladi:

E1 →T {(+ | -) T}

Ko'rinib turibdiki, bu qoidalarda iteratsiya qatnashmoqda, ularga kengaytirilgan rekursiv tushish usulini qo'llaymiz:

```
void E1 ()
{
 T();
  while ( c=='+' || c=='-')
 { gc();T();}
}
```

Ko'p ma'nolik grammatikalarga rekursiv tushush usulini to'g'ridan to'g'ri qo'llab bo'lmaydi, chunki bir nechta muqobil qoidalardan aniq bittasini tanlash ilojisi yo'q. Lekin usulni ozgina o'zgartirsa, ko'p ma'nolikni oldini olish mumkin. Buning uchun, muqobil qoidani tanlashda protsedurada faqat eng qulay muqobili olinishi kerak.

Masalan dasturlash tildagi to'liq va qisqa shartli operator qoidasini ko'ramiz:

S →if E then S S'|B

S' →else S | ε .

Bunda { ***if , then,else)*** kalit so'zlar(leksemalar).

Bu qoidalarda ko'p ma'nolik mavjud va FIRST(S') ∩ FOLLOW(S') ={else}≠ ∅. Agar biz doimo bo'shmas alternativani hisobga olsak, unda else eng yaqin if ga tegishli bo'ladi va ko'p ma'nolik yo'qoladi. Bu qoidani protsedurasi quyidagicha bo'ladi:

```
void S()
{ if (c=='i'){gc(); E();
if (c=='t'){gc(); S(); S1();}
  else B();}
void S1()
{if (c=='e'){gc();S();}}
```

M-tilni sintaktik tahlilchisi

Sintaktik va leksik tahlilchilar o'zaro ishlashi quyidagicha bo'lsin: ishni sintaktik tahlilchi boshlaydi,. Agar tahlil jarayonida navbatdagi

leksema kerak bo'lib qolsa, unda leksik tahlilchi ishga tushadi va u bitta leksemani qaytaradi, va har gal leksema kerak bo'lganda leksik tahlilchini Scanner sinfnig get_lex() funksiyasi chaqiriladi. Bu funksiya navbatdagi leksemani lex (sinf) turida qaytaradi.

Kelishuvlar:

- lex curr_lex- o'zgaruvchida joriy leksema saqlanadi;
- int c_val – o'zgaruvchida uning qiymati turadi;
- lex_of_type c_type – o'zgaruvchida leksemati turi saqlanadi.

Sintaktik tahlilchi rekursiv tushish usuli asosida C++ dasturlash tilida Parser sinf ko'rinishda yozilgan. Sintaktik tahlildan tashqari sinf protseduralarida semantik nazorat va ichki ko'rinishga o'tkazish amallar qo'shilgan. Sintaktik tahlilchini to'liq dastursi quyidagicha:

```
class Parser
{
    Lex curr_lex; // joriy leksema sinf
    type_of_lex c_type; // joriy leksema turi
    int c_val; // joriy leksema qiymati
    Scanner scan;      // leksik tahlil sinf
obyekti
    // butun sonlar steki
    Stack <int,100> st_int;
    /* leksema turlar steki */
    Stack  <type_of_lex,100>  st_lex;  /*  RT
usulni protseduralarni prototiplari*/
    void P();
    void D1();
    void D();
    void B();
    void S();
    void E();
    void E1();
    void T();
    void F();
```

```
    // semantik nazorat qiluvchi funksiyalar
prototiplari
        void dec ( type_of_lex type);
        void check_id ();
        void check_op ();
        void check_not ();
        void eq_type ();
        void eq_bool ();
        void  check_id_in_read  ();/*  navbatdagi
    leksemani olish fuksiyasi */
        void gl () // navbatdagi leksemani qabul
    qilish
        {
            curr_lex = scan.get_lex();
            c_type = curr_lex.get_type();
            c_val = curr_lex.get_value();
        }
    public :
        Poliz   prog;  // dasturni  ichki  tasviri
    (POLIZda)
        Parser  ( const  char  *program)  :  scan
    (program), prog (1000) {}
    /* sintaktik tahlilchi funksiyasi
    void  analyze(); // taxlilchi
    };  /* funksiyalarni ta'riflari */

    void Parser::analyze ()
    {
    gl ();P();prog.print();
    cout << endl << "Yes!!!" << endl;
    }
    void Parser::P ()
    {
        if  (  c_type  ==  LEX_PROGRAM  )gl  ();  //
    programm
```



```
    else throw curr_lex; // istisno hosil kilish
        D1 ();
        if ( c_type == LEX_SEMICOLON )gl ();// ;
        elsethrow curr_lex;
        B ();
        if ( c_type != LEX_FIN )throw curr_lex;
    }
    void Parser::D1 ()
    {
        if  (  c_type  ==  LEX_VAR  )  //  var  kalit
    so'zi
        {gl ();D ();
            while ( c_type == LEX_COMMA ) // ,
            { gl();D();}
        }
        elsethrow curr_lex;
    }
    void Parser::D ()
    {
        st_int.reset(); // stekni tozalash
        if ( c_type == LEX_ID )
        {/*  o'zgaruvchini  tartib  raqamini  stekka
    yozish
        st_int.push ( c_val );gl ();
        while ( c_type == LEX_COMMA )
        {gl();
        if(c_type==LEX_ID){st_int.push(c_val); gl();}
        else throw curr_lex;
        }
        if ( c_type == LEX_COLON )
    {gl();
            if  (  c_type  ==  LEX_INT  ){dec(LEX_INT
        );gl();}
        else
        if ( c_type == LEX_BOOL ){dec(LEX_BOOL);gl();}
```

```
    elsethrow curr_lex;
    }
    elsethrow curr_lex;
    }
    elsethrow curr_lex;
}
    void Parser::B ()
    {
    if ( c_type == LEX_BEGIN )
    {gl();S();
    while (c_type == LEX_SEMICOLON )
    {gl();S();}
    if ( c_type == LEX_END )
    gl();
    else
    throw curr_lex;
    }
    else
    throw curr_lex;
    }
    void Parser::S ()
    {
    int pl0, pl1, pl2, pl3;
    if ( c_type == LEX_IF )
    {gl();E();eq_bool(); // mantiq ifoda ?
    pl2 = prog.get_free ();prog.blank();
    prog.put_lex (Lex(POLIZ_FGO));
    if ( c_type == LEX_THEN )
    {gl();S();pl3 = prog.get_free();prog.blank();
    prog.put_lex(Lex(POLIZ_GO));
    prog.put_lex(Lex(POLIZ_LABEL,prog.get_free()),
pl2);
    if (c_type == LEX_ELSE){gl();S();
prog.put_lex(Lex(POLIZ_LABEL,prog.get_free()),pl3)
;
```



```
    }
    Yelsethrow curr_lex;
    }
    elsethrow curr_lex;
    } //end if
    else
    if (c_type==LEX_WHILE)
    {pl0=prog.get_free();gl();E();eq_bool();
    pl1=prog.get_free();
    prog.blank();prog.put_lex (Lex(POLIZ_FGO));
    if ( c_type == LEX_DO ){gl();S();
    prog.put_lex (Lex (POLIZ_LABEL, pl0));
    prog.put_lex (Lex ( POLIZ_GO));
    prog.put_lex(Lex(POLIZ_LABEL,prog.get_free()),
pl1);
    }
    elsethrow curr_lex;
    } //end while
    else
    if ( c_type == LEX_READ )
    {gl();
    if(c_type==LEX_LPAREN)
    {gl();
    if (c_type==LEX_ID)
    {check_id_in_read();
    prog.put_lex (Lex(POLIZ_ADDRESS,c_val));gl();
    }
    elsethrow curr_lex;
    if ( c_type == LEX_RPAREN )
    {gl();
    prog.put_lex (Lex (LEX_READ));
    }
    elsethrow curr_lex;
    }
    elsethrow curr_lex;
```

```
} //end read
elseif ( c_type == LEX_WRITE )
{
gl();
if (c_type == LEX_LPAREN )
{gl();E();
if (c_type == LEX_RPAREN )
{gl();prog.put_lex (Lex(LEX_WRITE));
}
elsethrow curr_lex;
}
elsethrow curr_lex;
} //end write
elseif ( c_type == LEX_ID )
{
check_id ();
prog.put_lex (Lex(POLIZ_ADDRESS,c_val));
gl();
if ( c_type == LEX_ASSIGN )
{
gl();
E(); eq_type();
prog.put_lex (Lex (LEX_ASSIGN) );
}
elsethrow curr_lex;
} //assign-end
else B();
}
void Parser::E ()
{
E1();
if(c_type==LEX_EQ||c_type==LEX_LSS||
c_type==LEX_GTR||c_type==LEX_LEQ||
c_type==LEX_GEQ||c_type==LEX_NEQ)
{st_lex.push (c_type);gl();E1();check_op();}
```



```
}
void Parser::E1 ()
{T();
while(c_type==LEX_PLUS||c_type==LEX_MINUS||
c_type==LEX_OR )
{st_lex.push (c_type);gl();T();check_op();}
}
void Parser::T ()
{F();
While(c_type==LEX_TIMES||
c_type==LEX_SLASH||
 c_type==LEX_AND )
{st_lex.push (c_type);gl();F();check_op();}
}void Parser::F ()
{if ( c_type == LEX_ID ){check_id();
prog.put_lex (Lex (LEX_ID, c_val));gl();
}
elseif ( c_type == LEX_NUM )
{
st_lex.push ( LEX_INT );
prog.put_lex ( curr_lex );
gl();
}
elseif ( c_type == LEX_TRUE )
{
st_lex.push ( LEX_BOOL );
prog.put_lex (Lex (LEX_TRUE, 1) );
gl();
}
elseif ( c_type == LEX_FALSE )
{
st_lex.push ( LEX_BOOL );
prog.put_lex (Lex (LEX_FALSE, 0) );
gl();
}
```

```
elseif ( c_type == LEX_NOT )
{gl();F();check_not();}
elseif ( c_type == LEX_LPAREN )
{gl();E();
if ( c_type == LEX_RPAREN)gl();
elsethrow curr_lex;
}
elsethrow curr_lex;
}
```

M-tilni semantik tahlili

Ko'rsatilgan M-tilni sintaksisi tilni to'liq ta'riflamaydi. Chunki, ba'zi bir shartlarni sintaktik qoidalar bilan berib bo'lmaydi. Bunday shartlar semantikaga kiradi va bizning til uchun ular quyidagicha:

1. Dasturda ishlatilgan har qanday ism, albatta ta'riflanishi lozim va bu ta'rif yagona bo'lishi shart.

2. Qiymat berish operatorining chap va o'ng qismlari turlari bir xil bo'lishi shart.

3. Shartli operatorda va sikl operatorda shart sifatida faqat mantiq ifoda bo'lishi mumkin.

Kontekst shartlarni tekshirish protseduralarni sintaktik tahlil protseduralar ichida chaqiramiz.

Semantik tahlil uchun, bizga ikkita stek kerak bo'ladi: butun sonlar va leksemalar turlari. Steklarni amalga oshirish uchun qolipli shablon sinfini kiritamiz:

```
template <class T, int max_size >  class Stack
{
T s[max_size];
int top;
public :
Stack(){top = 0;}
void reset() { top = 0; }
void push(T i){if (!is_full()){s[top] =
i;++top;}
```



```
else throw "Stack_is_full";}
T pop(){if (!is_empty()){--top;return s[top];}
else throw "Stack_is_empty";}
bool is_empty(){  return top == 0; }
bool is_full() {  return top == max_size; }
};
```

Tavsiflarni qayta ishlash.

Ifodalarga kirgan o'zgaruvchi va o'zgarmaslar turlari bir biriga mos kelishi uchun, turlarni identifikator jadvalida saqlash kerak. Undan tashqari, har bir identifikator faqat bir marta e'lon qilinishi kerak. Bu ma'lumot, sintaktik tahlilchi tavsiflarni qayta ishlash jarayonida aniqlanadi. Shu sababli, o'zgaruvchilarning turlarini saqlash va noyobligini nazorat qilish amallar tavsiflarni sintaktik qoidalariga qo'shimcha qilib kiritish kerak.

Leksik tahlilchi barcha identifikatorlarni identifikatorlar jadvalida saqlab qo'ydi,. Yendi shu jadvalda (TID) har bir identifikator uchun declare va type maydonlarni semantik tahlil jarayonida to'lg'aziladi.

Tavsiflash bo'limi sintaktik qoidasi quyidagicha:

D →I {, I}: [int | bool],

ya'ni tur nomidan oldin identifikatorlar ro'yhati keladi. Bu identifikatorlarni (to'g'rirog'i TID jadvaldagi satrlar tartib raqami) ***Stack<int,100>st_int*** butun sonlar stekida joylashtiriladi. Va tur nomi tahlil qilingandan so'ng, bu satrlarda declare va type maydonlar to'ldiriladi. TID jadvaldagi satrlar uchun maydonlarni to'ldirish va nazorat qilish Parser sinfning dec funksiyasi bajaradi. Uning ta'rifi quyidagicha:

```
void Parser::dec(type_of_lex type)
{ int i;
  while ( !st_int.is_empty())
{ i = st_int.pop();
    if (TID[i].get_declare())throw "twice";
    else
{TID[i].put_declare(); TID[i].put_type(type);}
}
}
```

Sintaktik qoida quyidagi amallar bilan kengaytiriladi:

D →{ st_int.reset( ) } I { st_int.push (c_val) }

{, I { st_int.push (c_val)) }:

[int (dec (LEX_INT)) | bool (dec (LEX_BOOL))]

Tavsiflashga tegishli protsedura quyidagicha:

```
void Parser::D ()
{
st_int.reset(); // stekni tozalash
if ( c_type == LEX_ID )
{/*  o'zgaruvchini  tartib  raqamini  stekka
yozish*/
st_int.push ( c_val );gl ();
while ( c_type == LEX_COMMA )
{gl();
if(c_type==LEX_ID);{st_int.push(c_val); gl();}
else throw curr_lex;
}
if ( c_type == LEX_COLON )
{gl();
    if  ( c_type  ==  LEX_INT  ){dec(LEX_INT
);gl();}
else
if ( c_type == LEX_BOOL ){dec(LEX_BOOL);gl();}
elsethrow curr_lex;
}
elsethrow curr_lex;
}
elsethrow curr_lex;
}
```

Ifodalarni kontekst shartlarini nazorat qilish.

Ifodalarga kirgan operandlarning turlari natijaviy turga ta'sir qiladi. Operandlar turini va amallar belgilarni **Stack (type_of_lex,100) st_lex** stekda saqlaymiz.

Agar ifodada butun son yoki true va false mantiq o'zgarmaslar uchrasa, ularga mos bo'lgan tur stekka shu zahoti yoziladi.

Agar operand o'zgaruvchi bo'lsa, oldin u tavsiflanganligi tekshirilishi kerak va keyin turi stekka yozilishi lozim. Buni check_id( ) funksiyasi bajaradi:

```
void parser::check_id()
{
  if ( TID[c_val].get_declare() )
  st_lex.push(TID[c_val].get_type());
  else throw "not declared";
}
```

Ifodaga kirgan amallarni operandlari bir biriga mos kelishligini nazorat qilish uchun check_op( ) funksiyadan foydalanamiz.. Uning ta'rifi quyidagicha

```
void Parser::check_op ()
{
 type_of_lex  t1,  t2,  op,  t  =  LEX_INT,  r  =
LEX_BOOL;
 t2=st_lex.pop();op=st_lex.pop();t1          =
st_lex.pop();
 if(op==LEX_PLUS||op==LEX_MINUS||
op==LEX_TIMES||op==LEX_SLASH )r = LEX_INT;
 if(op==LEX_OR||op == LEX_AND)  t = LEX_BOOL;
 if (t1==t2 && t1 == t )st_lex.push(r);
 else throw "wrong types are in operation";
}
```

Bir o'rinli amalni turini nazorat qilish uchun quyidagi funksiya ishlatiladi:

```
void Parser::check_not()
{

if(st_lex.pop()==LEX_BOOL)st_lex.push(LEX_BOOL)
  else throw "wrong type is in not";
 }
```

Yendi asosiy savol tug'iladi, qachon bu funksiyalarni qo'llash kerak?

Bu funksiyalar ifodaga tegishli qoidalarni ta'rifiga qo'shimcha amallar sifatida kiritiladi,. Bunday usul, kontekst shartlarni sintaktik boshqaruv nazorat qilish deb nomlanadi.va shunda M-tilni qoidalar ko'rinishi quyidagicha bo'ladi:

E →E1 | E1 [= | < | >] {st_lex.push(c_type) } E1 {check_op( ) }
E1 →T {[+ | −| or] {st_lex.push (c_type) } T {check_op( ) }}
*T →F {[* | / | and] {st_lex.push (c_type) } F {check_op( ) }}*
F →I {check_id( ) } | N {st_lex (LEX_INT) } | [true | false] {st_lex.push (LEX_BOOL) } | not F {check_not( ) } | (E)

Ifodalarga tegishli protseduralar quyidagicha:

```
void Parser::E ()
{
E1();
if(c_type==LEX_EQ||c_type==LEX_LSS||
c_type==LEX_GTR||c_type==LEX_LEQ||
c_type==LEX_GEQ||c_type==LEX_NEQ)
{st_lex.push (c_type);gl();E1();check_op();}
}
void Parser::E1 ()
{T();
while(c_type==LEX_PLUS||c_type==LEX_MINUS||
c_type==LEX_OR )
{st_lex.push (c_type);gl();T();check_op();}
}
void Parser::T ()
{F();
While(c_type==LEX_TIMES||c_type==LEX_SLASH||
 c_type==LEX_AND )
{st_lex.push (c_type);gl();F();check_op();}
}
void Parser::F ()
{if ( c_type == LEX_ID ){check_id();gl();}
```



```
elseif ( c_type == LEX_NUM )
{st_lex.push ( LEX_INT );gl();}
elseif ( c_type == LEX_TRUE )
{st_lex.push ( LEX_BOOL );gl();}
elseif ( c_type == LEX_FALSE )
{st_lex.push ( LEX_BOOL );gl();}
elseif ( c_type == LEX_NOT )
{gl();F();check_not();}
elseif ( c_type == LEX_LPAREN )
{gl();E();
if ( c_type == LEX_RPAREN)gl();
elsethrow curr_lex;
}
elsethrow curr_lex;
}
```

Operatorlarda kontekst shartlarni nazorat qilish.
Operatorlarga tegishli qoida quyidagicha:

S →I := E | if E then S else S | while E do S | B | read (I) | write (E)

1. Qiymat berish operator sintaksisi quyidagicha:
I := E.

Qiymat berish operatordagi o'zgaruvchi va ifodaning turlari bir hil bo'lishi kerak. Ifoda turini nazorat qilish natijasida, stekda shu ifodaning turi qoladi. Agar I identifikatorni tahlil qilish natijasida uning ta'riflanganini tekshirib check_id( ) funksiya orqalit urini stekka yozsak, stekdan ikki elementni o'qib, ularni turlarini bir xilligini tekshirish yetarlidir,. Buning uchun quyidagi funksiyani ishlatamiz:

```
void Parser::eq_type()
{
  if ( st_lex.pop()!= st_lex.pop() )
 throw "wrong types are in :=";
}
```

Bundan chiqdi, qiymat berish operator qoidasini quyidagicha yozish mumkin:

S →I ⟨check_id( )⟩ := E ⟨eq_type( )⟩

Bu qoidaga tegishli S protsedurani qism-dasturi quyidagicha bo'ladi:

```
...
if ( c_type == LEX_ID )
{check_id ();gl();
if ( c_type == LEX_ASSIGN )
{gl();E();eq_type();}
elsethrow curr_lex;
}
...
```

2. Qiymat berish va sikl operatorlari.

if E then S else S | while E do S

Bu operatorlarda shart sifatida faqat mantiq ifoda bo'lishi mumkin. Ifodani kontekst shartining nazorati natijasida stekda yagona tur qoladi.. Shuning uchun, u mantiqliligini tekshirish yetarlidir. Buni eq_bool () funksiya bajaradi:

```
void Parser::eq_bool ()
{
  if ( st_lex.pop() != LEX_BOOL )
   throw "expression is not boolean";
}
```

Shartli va sikl operatorga tegishli qoidalar quyidagicha bo'ladi:

S →if E ⟨eq_bool( )⟩ then S else S | while E ⟨eq_bool( )⟩

va qism-dasturi:

```
if ( c_type == LEX_IF )
{gl();E();eq_bool(); // mantiq ifoda ?
if ( c_type == LEX_THEN )
{gl();S();
```



```
    if (c_type == LEX_ELSE){gl();S();}
    elsethrow curr_lex;
}
elsethrow curr_lex;
} //end if
else
if (c_type==LEX_WHILE)
{gl();E();eq_bool();
if ( c_type == LEX_DO ){gl();S();}
 elsethrow curr_lex;
} //end while
```

3. read(I) kiritish operatordagi o'zgaruvchi ta'riflangan bo'lishi kerak. Buni check_id_in_read () funksiya tekshiradi:

```
void Parser::check_id_in_read ()
{
  if (!TID [c_val].get_declare())
   throw "not declared";
}
```

Qoidasi quyidagicha:

S →read (I ⟨check_id_in_read( )⟩) .

Qism-dasturi:

```
if ( c_type == LEX_READ )
{gl();if(c_type==LEX_LPAREN)
{gl();if (c_type==LEX_ID)
{check_id_in_read();gl();}
elsethrow curr_lex;
if ( c_type == LEX_RPAREN ){gl();}
elsethrow curr_lex;
}
elsethrow curr_lex;}
```

Xulosa

Sintaktik taxlilchi kontekstdan erkin grammatikaga asoslangan va unda kiruvchi sifatida leksemalar ketma-ketligi bo'ladi. Sintaktik tahlilchi leksemalar zanjiri til sintaksiga mos kelishini aniqlaydi. Taxlil natijasida dasturning ichki tasviri hosil bo'ladi. Tahlilchilardan biri bu kengaytirilgan rekursiv usul tahlilchisi. Kengaytirilgan rekursiv tushish usulda iteratsiya amali siql bilan bajariladi. Misol sifatida M-tilni sintaktik tahlilchisini dasturi keltirilgan. Semantik tahlilda qoydalar bilan berib bo'lmaydigan bazi bir shartlar tekshiriladi, shu jumladan har qanday ism ta'riflanishi, qiymat berish operatorda chap va o'ng ta'raflar turi bir hil bo'lishi, shartli va siql operatorlarda shart mantiqiy bo'lishi. Semantik tahlil sintaktik tahlil jarayonida baravariga bajariladi.

Nazorat uchun savollar

1. Sintaktik tahlilni asosiy vazifalarini keltiring.
2. Kengaytirilgan rekursiv tushish usul kqanday grammatikalarga qo'llash mumkin?
3. Rekursiv tushish usulni ro'yxat ko'rinishdaga sintaktik konstruksiyani qanday qo'llash mumkin?
4. Rekursiv tushish usulni shartli operatorga qanday qo'llash mumkin?
5. Semantik sharlarni keltiring.
6. Semantik shartlarni dasturda qanday tekshishrish mumkin?

Amaliyot uchun topshiriqlar

1. Yuqorida berilgan model tilning grammatikasiga quyidagi qo'shimchalar kiriting va mos sintaktik va semantik tahlil dastursini o'zgartiring:
   a) haqiqiy son
   b) haqiqiy tur
   v) repeas until e – sikl operatori.
   g) standart funksiyani chaqirish.

30-BOB. DASTURNI ICHKI TASVIRINI YARATISH

Tayanch iboralar: *POLIZ ichki ko'rinish tili, ifodani POLIZga o'tkazish, POLIZda ifodani hisoblash, dastlabki tilni operatorlarini POLIZga o'tkazish.*

POLIZ ichki tasvir tili

Biz dasturni ichki tasvir tili sifatida postfiks yozuvni olamiz.

POLIZ ichki tasviri tili interpretator tillar uchun eng qulaydir, chunki unga o'tkazish va interpretatsiya qilish ancha yengil.

Bitta o'zgarmasdan yoki o'zgaruvchidan iborat bo'lgan ifoda sodda ifoda deb nomlanadi. Bunday ifodalar POLIZda o'zgarmasdan yoziladi.

Ifodalarni POLIZda yozish rasmiy qoidalari quyidagicha:

1) agar YE sodda ifoda bo'lsa, unda YE ifodani POLIZda yozuvi uni o'zi bo'ladi;

2) YE1 θ YE2 ifodani, bunda θ — binar amal belgisi, E1 va E2 uning operandlari,. POLIZda E1′ E2′ θ yozuv bo'ladi, bunda E1′ va E2′ mos ravishda YE1 va YE2 operandlarni POLIZdagi yozuvi;

3) θ E ifodani, θ — unar amal belgisi, YE — uning operandi POLIZda,va bunda E′ - YE operandni POLIZdagi yozuvi;

4) (E) ifoda POLIZda E′ yozuv bo'ladi,va bunda E′ - YE operandni POLIZdagi yozuvi.

Misol. $a + b + c$ ifodani POLIZdagi yozuvi $a\ b + c +$bo'ladi.

Ihtiyoriy arifmetik ifodani POLIZga o'tkazish algoritmi.

Faraz qilaylik, POLIZ yozuvlari massivda leksemalar ketma-ketligi ko'rinishda saqlanadi va POLIZga o'tkazishda qo'shimcha stek ishlatiladi va stekga leksemalar yoziladi,. Shu jumladan amallar, o'zgaruvchilar, funksiyalar va qavslar.

1. Ifoda chapdan o'nga qarab bir marotaba ko'rib chiqiladi.
2. Ifoda leksemalari ohirigacha o'qiguncha quyidagi amallarni bajaramiz:
   a) Joriy leksemani o'qiymiz.
   b) Agar leksema son yoki o'zgaruvchi bo'lsa, uni POLIZ massiviga qo'shamiz.

v) Agar leksema funksiya nomi bo'lsa, uni stekga joylashtiramiz.

g) Agar leksema argumentlarni ajratuvchi bo'lsa, unda ochiluvchi qavs kelguncha stekdan massivga elementlarni ko'chiramiz. Agar ochiluvchi qavs uchramasa, ifodada hatolik bor.

d) Agar leksema θ amal bo'lsa:

ustuvorligi stek cho'qqisida turgan amalni ustuvorligidan katta bo'lguncha (chap-assotsiativ amallar uchun) yoki stek cho'qqisidagi amaldan kichik bo'lguncha (o'ng-assotsiativ amallar uchun), stekdan elementlarni. θ amalni stekka yozamiz.

e) Agar leksema ochuvchi qavs bo'lsa, uni stekka yozamiz.

j) Agar joriy leksema yopuvchi qavs bo'lsa, stek cho'qqisida ochuvchi qavs bo'lgunicha, stekdan elementlarni massivga yozmaymiz,. Ochuvchi qavsni stekdan olib tashlaymiz, agar bundan keyin stek cho'qqisida funksiya belgisi tursa, uni massivga ko'chiramiz,. Agar o'tkazish jarayonida ochuvchi qavs uchramasa, unda qavslar soni to'g'ri kelmaydi, ya'ni ifodada xatolik mavjuddir.

3. Ifodani barcha leksemalarni ko'rib chiqilgandan so'ng stekda qolgan barcha elementlarni POLIZ-massivga o'tkazamiz (stekda faqat amallar qolgan bo'ladi,. Aks holda ifoda hato yozilgan bo'ladi).

Misol: a * (b + c) − (d − e) / f POLIZga algoritm bo'yicha o'tkazamiz:

Massiv nomini R deb belgilaymiz, stekni S bilan belgilaymiz.

	Joriy massiv: belgi		stek:
1)	*a*	*P:a*	*C:*
2)	*	*P:a*	*C:**
3)	(	*P:a*	*C:*(*
4)	+	*P:ab*	*C:*(+*
5)	c	*P:abc*	*C:*(+*
6)	)	*P:abc*	*C:*(+*
7)	)	*P:abc+*	*C:*(*
8)	)	*P:abc+*	*C:**
9)	-	*P:abc+*	*C:**
10)	-	*P:abc+**	*C:-*

11)	(	*P:abc+**	*C:-(*
12)	d	*P:abc+*d*	*C:-(*
13)	-	*P:abc+*d*	*C:-(-*
14)	e	*P:abc+*de*	*C:-(-*
15)	)	*P:abc+*de-*	*C:-(*
16)	)	*P:abc+*de-*	*C:-*
17)	/	*P:abc+*de-*	*C:-/*
18)	f	*P:abc+*de-f*	*C:-/*
19)	f	*P:abc+*de-f/*	*C:-*
20)	f	*P:abc+*de-f/-*	*C:*

Natijada POLIZ yozuvi quyidagicha bo'ladi:

*a b c + * d e − f / −*

POLIZ ifodani hisoblash

POLIZda yozilgan ifodani stek yordamida hisoblash algoritmi juda sodda.

POLIZda yozilgan ifodani hisoblash algoritmi.

1) Ifoda chapdan o'nga qarab bir marotaba ko'rib chiqiladi va har bir element uchun (2) va (3) qadamlar bajariladi;

2) Agar POLIZni joriy elementi operand bo'lsa, u stekga yoziladi;

3) Agar joriy element amal bo'lsa, unda stekdan operandlar o'qiladi va amal bajariladi. Va natija qayta stekga yoziladi;

4) POLIZda yozilgan ifoda to'liq o'qilgandan so'ng, stekda faqat bitta element qoladi u ham bo'lsa ifodani qiymati bo'ladi,. Ya'ni hisob natijasi.

Ifodani hisoblash uchun jadvallarda joylashgan operandlar to'g'risida qo'shimcha ma'lumotlar kerak bo'ladi.

Binar amalni yozuvi unar amal bilan bir hil bo'lib qolishi mumkin, masalan "-" amali. Bu holda nechta operandni stekdan olish noma'lum bo'lib qoladi. Buni oldini olish uchun ikkita usulni ishlatish mumkin:

a) unar amalni binarga almashtirish, ya'ni –a o'rniga 0-a qo'llash.

b) maxsus belgi kiritish masalan **POLIZ_MINUS**, shunda **–a** POLIZda **aPOLIZ_MINUS** ko'rinishda yoziladi. Bunday o'zgartirish faqat ichki tasvirida bo'ladi,. Kiruvchi tilga o'zgartirishlar kiritilmaydi.

POLIZda dastur konstruksiyalarini aks ettirish

POLIZda faqat ifodalarni emas boshqa konstruksiyalarni ham yozish mumkin,. Shu jumladan massiv elementlari, funksiyalar, operatorlar.

Massiv elementi.

POLIZda massiv elementlarini yozish uchun qo'shimcha belgi kiritamiz **POLIZ_MAS.** Bu belgi orqali massiv elementini manzili hisoblanadi. Bu amalni operandlari massiv nomi, massiv indekslari bo'ladi,va undan tashqari kqo'shimcha elementda operandlar soni saqlanadi. Bu amalning operandlar soni o'zgaruvchan va quyidagi formula bilan hisoblanadi k=n+1, bunda n-massiv o'lchovi, ya'ni operandlar soni massiv o'lchoviga bog'liqdir. Masalan:

(a+b[i,j+1])*c+d ifoda, POLIZda quyidagicha yoziladi

abij 1+ 3 POLIZ_MAS +c*d+

Funksiya ko'rsatkichi

Funksiya ko'rsatkichini POLIZda yozish massiv elementini yozishga o'xshab ketadi. Funksiya uchun ko'shimcha belgi kiritamiz **POLIZ_FUN** va bu amalning operandlar soni, argumentlar soniga bog'liq.

Masalan,

a+f(i,j+1)*c+d ifoda, POLIZda quyidagicha yoziladi:

afij 1+ 3 POLIZ_FUNc*+d+

Qiymat berish operatori.

I := E operator POLIZda quyidagicha yoziladi:

IE' := ,

bunda ":=" — ikki operandlik operator, Iva YE' — uning operandlari; I tag chizig'i o'zgaruvchini manzilini bildiradi,va manzilni olish uchun dasturda maxsus **POLIZ_ADDRESS** belgi kiritamiz,. Shunda Idasturda I POLIZ_ADDRESS ko'rinishda yoziladi,. Xisoblashda bu belgi uchrasa, stekga o'zgaruvchini manzili yoziladi. Masalan:

A:=b+x*z operator POLIZda A b xz*+:= bo'ladi.

O'tish operatori.

O'tish operatori POLIZDA ko'rsatligan joydan hisobotni davom ettirishni bildiradi. Ko'rsatilgan joyga murojaat qilish uchun POLIZ elementlarini ketma-ket birdan boshlab tartib raqam berib (massiv indekslari) chiqamiz.

L nishon bilan belgilangan operator p raqam bilan boshlansin, unda gotoL o'tish operatorini POLIZda

p!

yozsak bo'ladi.

Dasturda "!" belgiga **POLIZ_GO** belgi to'g'ri keladi va raqamli elementga o'tishni bildiradi,. p- **POLIZ_LABEL** belgisi bilan beriladi.

Shartli operator.

Agar shartli operator if B then S1 else S2 to'g'ridan to'g'ri POLIZda B′ S1′ S2′ if ko'rinishda yozsak unda shartdan qat'iy nazar S1′ va S2′ xisoblanadi,. Bu esa shartli operatorni semantikasiga to'g'ri kelmaydi.

if konstruksiyasini to'g'ri amalga oshirish uchun, oldin uni o'tish operatorlar orqali ekvivalent shakliga olib kelinadi.

if (!E) goto L operator uchun !F qo'shimcha amal kiritamiz,va uning dasturda **POLIZ_FGO** belgilaymiz. Bu binar amalda ikkita operand ishlatiladi,. Bu E′-mantiq ifodani POLIZi va p – L nishon bilan belgilanga POLIZ elementi. Bizni operatorimiz

E′!F

ko'rinishda yoziladi, ya'ni birinchi mantiq operandni qiymati yolg'on bo'lsa p raqamli elementga o'tishni bildiradi.

Shu amal orqali o'tish

if YethenS1 elseS2

shartli operatorni semantikasi quyidagicha yoziladi:

if (! E) goto L2; S1; goto L3; L2: S2; L3: ...

Bu operatorni POLIZda yozilishi quyidagicha:

E′ p2 !FS1′ p3 ! ^{p2}S2′ ^{p3}...,

bunda pi — Li nishon bilan belgilangan operatorga mos kelgan POLIZ elementning tartib raqami,i = 2,3, E′ — mantiq ifodani POLIZdagi yozuvi. Shuni aytib o'tish kerakki, shartli operatorni ichki konstruksiyalar ketma-ketligi o'zgarmadi. Bu operatorlarni POLIZda yozishning majburiy sharti.

POLIZda dastur konstruksiyalarini aks ettirish

POLIZda faqat ifodalarni emas boshqa konstruksiyalarni ham yozish mumkin,. Shu jumladan massiv elementlari, funksiyalar, operatorlar.

Massiv elementi.

POLIZda massiv elementlarini yozish uchun qo'shimcha belgi kiritamiz **POLIZ_MAS.** Bu belgi orqali massiv elementini manzili hisoblanadi. Bu amalni operandlari massiv nomi, massiv indekslari bo'ladi,va undan tashqari kqo'shimcha elementda operandlar soni saqlanadi. Bu amalning operandlar soni o'zgaruvchan va quyidagi formula bilan hisoblanadi k=n+1, bunda n-massiv o'lchovi, ya'ni operandlar soni massiv o'lchoviga bog'liqdir. Masalan:

(a+b[i,j+1])*c+d ifoda, POLIZda quyidagicha yoziladi

abij 1+ 3 POLIZ_MAS +c*d+

Funksiya ko'rsatkichi

Funksiya ko'rsatkichini POLIZda yozish massiv elementini yozishga o'xshab ketadi. Funksiya uchun ko'shimcha belgi kiritamiz **POLIZ_FUN** va bu amalning operandlar soni, argumentlar soniga bog'liq.

Masalan,

a+f(i,j+1)*c+d ifoda, POLIZda quyidagicha yoziladi:

afij 1+ 3 POLIZ_FUNc*+d+

Qiymat berish operatori.

I := E operator POLIZda quyidagicha yoziladi:

IE' := ,

bunda ":=" — ikki operandlik operator, Iva YE' — uning operandlari; I tag chizig'i o'zgaruvchini manzilini bildiradi,va manzilni olish uchun dasturda maxsus **POLIZ_ADDRESS** belgi kiritamiz,. Shunda Idasturda I POLIZ_ADDRESS ko'rinishda yoziladi,. Xisoblashda bu belgi uchrasa, stekga o'zgaruvchini manzili yoziladi. Masalan:

A:=b+x*z operator POLIZda A b xz*+:= bo'ladi.

O'tish operatori.

O'tish operatori POLIZDA ko'rsatligan joydan hisobotni davom ettirishni bildiradi. Ko'rsatilgan joyga murojaat qilish uchun POLIZ elementlarini ketma-ket birdan boshlab tartib raqam berib (massiv indekslari) chiqamiz.

L nishon bilan belgilangan operator p raqam bilan boshlansin, unda gotoL o'tish operatorini POLIZda

p!

yozsak bo'ladi.

Dasturda "!" belgiga **POLIZ_GO** belgi to'g'ri keladi va raqamli elementga o'tishni bildiradi,. p- **POLIZ_LABEL** belgisi bilan beriladi.

Shartli operator.

Agar shartli operator if B then S1 else S2 to'g'ridan to'g'ri POLIZda B′ S1′ S2′ if ko'rinishda yozsak unda shartdan qat'iy nazar S1′ va S2′ xisoblanadi,. Bu esa shartli operatorni semantikasiga to'g'ri kelmaydi.

if konstruksiyasini to'g'ri amalga oshirish uchun, oldin uni o'tish operatorlar orqali ekvivalent shakliga olib kelinadi.

if (!E) goto L operator uchun !F qo'shimcha amal kiritamiz,va uning dasturda **POLIZ_FGO** belgilaymiz. Bu binar amalda ikkita operand ishlatiladi,. Bu E′-mantiq ifodani POLIZi va p – L nishon bilan belgilanga POLIZ elementi. Bizni operatorimiz

E′!F

ko'rinishda yoziladi, ya'ni birinchi mantiq operandni qiymati yolg'on bo'lsa p raqamli elementga o'tishni bildiradi.

Shu amal orqali o'tish

if YethenS1 elseS2

shartli operatorni semantikasi quyidagicha yoziladi:

if (! E) goto L2; S1; goto L3; L2: S2; L3: ...

Bu operatorni POLIZda yozilishi quyidagicha:

E′ p2 !FS1′ p3 ! ^{p2}S2′ ^{p3}...,

bunda pi — Li nishon bilan belgilangan operatorga mos kelgan POLIZ elementning tartib raqami,i = 2,3, E′ — mantiq ifodani POLIZdagi yozuvi. Shuni aytib o'tish kerakki, shartli operatorni ichki konstruksiyalar ketma-ketligi o'zgarmadi. Bu operatorlarni POLIZda yozishning majburiy sharti.

Sikl operatori

Sikl operatorni while YE do S semantikasi quyidagicha yozilishi mumkin:

L0: if (! E) goto L1; S; goto L0; L1: ... .

Unda while sikl operatorni POLIZda yozilishi quyidagicha bo'ladi:

^{p0}E' pI !F p0 !^{p1} ...,

bunda YE' — mantiq ifodani POLIZdagi yozuvi, S'- S operatorni POLIZdagi yozuvi.

Kiritish va chiqarish operatorlar

M-tilning kiritish va chiqarish operatorlari bir operandlik amallarga kiradi. Ular uchun POLIZda maxsus amallar mos ravishda kiritamiz **POLIZ_READ** va **POLIZ_WRITE.**

Kiritish operatori read (I) POLIZda I POLIZ_READ shaklida yoziladi.

Chiqarish operatori write (E) POLIZda YE'POLIZ_WRITE ko'rinishda yoziladi, bunda YE' — E ifodani POLIZdagi yozuvi.

Operatorlarning POLIZdagi yozuvini, ifodaning yozuviga o'hxshab interpretatsiya qilish mumkin. Dasturni POLIZ yozuvi faqat interpretatsiya uchun emas, balki obyekt kodni generatsiya qilish uchun ham qo'llasa bo'ladi,. Buning uchun hisoblash o'rniga mos buyruqlar generatsiya qilinadi.

Xulosa

Dasturni ichki tasvir tili sifatida postfiks yozuvi keltirilgan..

POLIZ ichki tasviri tili interpretator tillar uchun eng qulaydir, chunki unga o'tkazish va interpretatsiya qilish ancha yengil. Ihtiyoriy arifmetik ifodani POLIZga o'tkazish uchun algoritm mavjud. Bu algoritm leksemalar massividan va stekdan foydalaniladi. Amallar ustuvorligiga ko'ra chiquvchi Polizda chiquvchi zanjir hosil bo'ladi. POLIZda yozilgan ifodani stek yordamida hisoblash algoritmi juda sodda. Buning uchun ifoda chapdan o'nga qarab qo'rib chiqiladi va joriy element operand bo'lsa u stekga yoziladi, aksincha amal bo'lsa stekdan operandlar olinib amal bajariladi va natija yana stekga yozilad. Barcha elementlar ko'rib chiqilgandan keyin stekda natija qoladi. POLIZda faqat ifodalarni emas

balki boshqa konstruksiyalarni ham yozish mumkin. Shu jumladan massiv elementlari, funksiyalar,qiymat berish operatori, siql va shart operatorlari xamda kiritish va chiqarish operatorlarni yozish mumkin.

Nazorat uchun savollar

1. POLIZ ichki tasvirini ta'rifini keltiring.
2. Ixtiyoriy ifodani POLIZga o'tkazish algoritmini keltiring.
3. POLIZda yozilgan ifodani qiymatini hisoblash algoritmini keltiring.
4. Massiv elementi qanday qilib POLIZga o'tkaziladi?
5. Fuksiya chaqirishi qanday qilib POLIZga o'tkaziladi?
6. Qiymat berish operatorni POLIZDA yozish shaklini keltiring
7. Shartli operatorni
8. Siklik operatorni POLIZDA yozish shaklini keltiring

Amaliyot uchun topshiriqlar

1. Quyidagi ifodalar POLIZda yozilsin:

a) a+b−c b) a∗b+c/a

c) a/(b+c)∗a d) (a+b)/(c+a∗b)

e) a and b or c f) not a or b and a

g) x+y=x/y h) (x∗x+y∗y<1) and (x>0)

2. POLIZda yozilgan ifodalar uchun infiks yozuvi keltirilsin:

a) ab∗c+

b) abc∗/

c) ab+c∗

d) ab+bc−/a+

e) a not b and not

f) abca and or and

g) 2x+2x∗<

3. Stekdan foydalangan holda POLIZda yozilgan quyidagi ifodalarni qiymati hisoblansin:

a) xy∗xy / + pri x = 8, y = 2;

b) a 2+b / b 4∗+ pri a = 4, b = 3;

c) *a b not and a or not pri a = b = true;*

d) *x y*0 > y 2 x − < and pri x = y = 1.*

4. C tilidagi qism-dasturi POLIZga o'tkazilsin:

```
a) S = 0; i =1; while ( i < 10) { S = S*(i+i); i + +; }
b) if ( (x+1) > (2*y) ) x = y ; else y = (x+y)*2;
c) i = 1; S = 0; while ( i < 10 && S < 40 ) { S = S + f(i); + + i; }
d) if (z<x*y+5) a = x<y, z = (x+6)/(a−y); else z = y << 2;
e) a = x + y < z*(t + x) ? − (a + b)/(c−d)*2 : ++x+5;
f) S = x + y; i = 1; for (j = 0; j < n; j++) { S = S + i*j*S; i = i*x;
}
g) i = 1; S = 0; while ( i < 10 && S < 40 ) { S = S + f(i); ++i; }
h) if (z<x*y+5) a = x<y, z = (x+6)/(a−y); else z = y << 2;
i) do {x = y; y = 2*y;} while (x < k);
j) S = 0; for (i = 1; i <= k; i = i + 1) S + = i*i;
k) switch (k) {
  case 1:  a = not a; break;
  case 2:  b = a or not b;
  case 3:  a = b ;
}
```

Izox: C tilning operatorlari POLIZi M-tilning POLIZIga o'hxshaydi. Murakkab konstruksiyalarni POLIZga o'tkazishda uning ichki konstruktorlar tartibi saqlanib qolishi kerak. Masalan for (expr1; expr2; expr3) {body} sikl operatorni POLIZda expr3 body dan oldin kelishi kerak.

Si tilida qiymat berish operator emas, amaldir. Shu sababli uni xisoblashda, natijasi stekda saqlanadi. Kerakmas qiymatni stek cho'qqisidan olib tashlash uchun POLIZda ";" amali kiritiladi. Prefiks va suffiks, inkrement va dekrement amallarni ajratish uchun POLIZda +# va -# belgilar prefiks amallar uchun, #+ va #- belgilar postfiks amallar uchun ishlatiladi.

5. POLIZda yozilgan operatorlar ketma-ketligi C tilida yozilsin.

*a) x y z a x 5 y / + *z 6 + 8 *− = = = ;*

*b) xaxzy / + *z 6 a − *+ =*

6. POLIZdagi yozuv :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
y	15	=	;	x	x	a	b	c	2	/	1	+	*	-	*	a	-

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
=	;	y	y	2	-	=	;	y	10	<=	5	!F			

quyidagi C tilida yozilgan dasturga to'g'ri keladimi: y = 15; do {x = x*(a−b*(c/2+1))−a; y = y−2;} while (y>10), agar to'g'ri kelmasa o'z varianti taklif qilinsin.

7. POLIZdagi yozuv

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
i	1	=	;	I	n	<	33	!F	a	a	b	+	1	-	x	y	y

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
2	+	/	-	*	5	=	;	i	i	2	+	=	;	5	!

quyidagi C tilida yozilgan dasturga to'g'ri keladimi:

for (i = 1; i<n; i = i +2) a = (a+b−1)*(x−y /(y+2))+5, agar to'g'ri kelmasa o'z varianti taklif qilinsin.

8. a) C tilidagi qism-dasturini POLIZga o'tkazilsin:

i = 1; s = 0; while (i < 10 && s < 40) { s = s + f(i); ++i; };

b) POLIZdagi ifodani infiks shakliga o'tkazilsin (SI tilida)

x y z a x 5 y / + * z 6 + 8 * − = = =

9. a) C dastur POLIZga o'tkazilsin:

if (z < x*y+5) a = x<y, z = (x+6)/(a−y); else z = y << 2;

b) POLIZda yozilgan dastur C tiliga o'tkazilsin:

x a x z y / + * z 6 a − * + =

10. Quyidagi operatorlarga POLIZga o'tkazish yo'li berilsin::

a) if (E) S1; S2; S3

Semantikasi: oldin E ifodani qiymati hisoblanadi,. Agar qiymati 0 dan kichik bo'lsa S1, 0ga teng bo'lsa S2, aks holda S3 operator bajariladi.

b) choice (S1; S2; S3), E

Semantikasi: oldin E ifodani qiymati hisoblanadi,. Agar uni qiymati iga teng bo'lsa C operator bajariladi, bunda i=1,2,3; aks holda hech narsa bajarilmaydi. Qiymati 0dan kichik bo'lsa S1, 0ga teng bo'lsa S2, aks holda S3 operator bajariladi.

c) cycle (E1; E2; E3), S

d) Semantikasi: oldin E1 ifodani qiymati hisoblanadi,. Undan keyin S operator bajariladi, so'ng E3 ifoda hisoblanadi,. Undan keyin E2 ifoda hisoblanadi va uni qiymati siklni tugatishni nazorat qilish uchun ishlatiladi.

31-BOB. SINTAKTIK BOSHQARUV TARJIMA

Tayanch iboralar: *sintaktik boshqaruv tarjima, amallar bilan kentgaytirilgan grammatika, M-tilni ichki tasvir generatori, M-tilni interpretatori.*

Sintaktik boshqaruv tarjima ta'rifi

Amalda sintaktik, semantik tahlil va dasturni ichki tasvirini generatsiya qilish baravar bajariladi.

Dasturni ichki tasvirini tuzish uchun bir nechta usullar mavjud. Ulardan biri, sintaktik boshqaruv tarjima, juda sodda va samaralidir.

Sintaktik boshqaruv tarjimani asosida amallar bilan kengaytirilgan grammatika turibdi (kontekst shartlarni tekshirishda qo'llangan).

Yendi dastlabki leksemalar zanjirini tahlil qilish jarayonida parallel ravishda dasturni ichki tasvirini generatsiya qilish amallari bajariladi.

Bizga sodda arifmetik ifodalarni tafsiflaydigan grammatika berilsin.

$E \rightarrow T\{+T\}$

$T \rightarrow F\{*F\}$

$F \rightarrow a \mid b \mid (E)$

Bu grammatika asosida yozilgan ifodani POLIZga o'tkazish, amallar bilan kengaytirilgan grammatika quyidagicha bo'ladi.

$E \rightarrow T\{+T\,\{cout << '+';\}\}$

$T \rightarrow F\{*F\,\{cout << '*';\}\}$

$F \rightarrow a\,\{cout << 'a';\} \mid b\,\{cout << 'b';\} \mid (E)$

Bu grammatika bo'yicha kiruvchi $a + b * s$ zanjirni rekursiv tushish usuli bilan tahlil qilganda natijada $a\ b\ s\ * +$ zanjir chop etiladi.

Xullas, amallar bilan kengaytirilgan grammatika arifmetik ifodalar tilida yozilgan. Har bir zanjir uchun POLIZda yozilgan arifmetik ifodani zanjirini chiqaradi. Ya'ni, bu grammatika bir formal tildan boshqa tilga tarjima qilishni beradi.

Ta'rif: Bizga T1 va T2 — alifbolar berilsin. Unda formal tarjima τ — bu T1 va T2 alifboda tuzilgan juft zanjirlar to'plamini qismi bo'ladi: $\tau \subseteq (T1^* \times T2^*)$.

Tarjima τ da kiruvchi tilini $Lk(\tau) = \{\alpha \mid \exists\, \beta: (\alpha, \beta) \in \tau\}$ belgilaymiz.

Tarjima τda chiquvchi(natijaviy) tilini $Ln(\tau) = \{\beta \mid \exists\ \alpha: (\alpha, \beta) \in \tau$ belgilaymiz.

Agar $(\alpha, \beta) \in \tau$ va $(\alpha, \gamma) \in \tau$, bunda $\alpha \in T1^*$, $\beta, \gamma \in T2^*$ va $\beta \neq \gamma$,bo'lsa τ tarjima ko'p ma'nolik bo'ladi. Ya'ni kiruvchi tildagi bitta zanjir uchun har hil chiquvchi tildagi zanjirlar to'g'ri keladi. Dasturlash tilini translyatsiyasida faqat birma'nolik tarjimachilar ishlatiladi.

Tarjimani berish uchun L1 va L2 tillarni zanjirlari orasida aniq qonuniyatlar berilishi lozim.

Misol.

Bizga $L1 = \{0^n1^m \mid n \geq 0, m > 0\}$ kiruvchi til berilsin.

$L2 = \{a^mb^n \mid n \geq 0, m > 0\}$chiquvchi til.

Tarjima τ quyidagicha ta'riflanadi: har qanday $n \geq 0$, $m > 0$ uchun $0^n1^m \in$ L1 zanjirga $a^mb^n \in$ L2 chiquvchi zanjir to'g'ri keladi, ya'ni

$\tau = \{ (0^n1^m, a^mb^n) \mid n \geq 0, m > 0\}$.

Bu tarjima, amallar bilan kengaytirilgan grammatika orqali yechilsin.

Yechim.

L1 tilni quyidagi grammatika bilan berish mumkin:

$S \rightarrow 0S \mid 1A$

$A \rightarrow 1A \mid \varepsilon$

Bu grammatikaga qo'shimcha amallar kiritamiz.

$S \rightarrow 0S$ {cout << 'b';} $\mid 1$ {cout << 'a';} A

$A \rightarrow 1$ {cout << 'a';} $A \mid \varepsilon$

Yendi L1 tildagi kiruvchi zanjirga rekursiv tushish algoritmini qo'llansa, L2 tilni chiquvchi zanjirlari hosil bo'ladi.

M-tildagi dasturni ichki tasvir generatori

POLIZdagi har bir elementi — bu juftlik (leksema turi, leksema qiymati) bilan belirilgan leksema. POLIZGA o'tkazishda biz qo'shimcha leksemalar kiritgan edik, bu :

- POLIZ_GO — «!» amal uchun;
- POLIZ_FGO — «!F» amal uchun;
- POLIZ_LABEL — POLIZ elementlariga murojaat qilish uchun;

POLIZ_ADDRESS—operandni manzilini olish uchun.

POLIZdagi dastur Poliz sinfni prog obyektida joylashadi.

Poliz sinf quyidagicha beriladi:

```
class Poliz
{
 lex *p;
  int size;
  int free;
public:
 Poliz ( int max_size )
 {
  p = new Lex [size = max_size];
  free = 0;
 };
 ~Poliz() { delete []p; };
  // bo'sh joyga joriy elementni yozish
  void put_lex(Lex l) { p[free]=l; ++free; };
  //ko'rsatilgan joyga elementni yozish
  void put_lex(Lex l, int place) { p[place]=l;
};
 // bo'sh joy tashab ketish
 void blank() { ++free; };
 // bo'o' joy manjilini qaytarish
  int get_free() { return free; };
 lex& operator[] ( int index )
 {
   if (index > size)
    throw "POLIZ:out of array";
   else
    if ( index > free )
     throw    "POLIZ:indefinite    element    of
array";
    else
     return p[index];
 };
 //POLIZ massivni chop etish
  void print()
```

```
    {
      for ( int I = 0; i < free; ++i )
      cout << p[i];
    };
};
```

Dasturni ichki tasvirini saqlaydigan prog obyekti Parser sinfning ochiq qismida e'lon qilinadi:

```
class Parser
{
      Lex curr_lex;
      ...
public:
      Poliz  prog;
      Parser (const char *program ) : scan
(program),prog (1000) {}
      void          analyze();
};
```

Dasturni ichki tasviri generatsiyasi sintaktik tahlil qilish vaqtida o'tadi, shu vaqtda kontekst shartlar ham nazorat qilinadi.. Shu sababli ular ishlatgan steklarni generatsiyada qo'llash mumkin, masalan ifodani tahlil qilishda. Undan tashqari sematik tahlil qilish,va funksiyalar ichiga generatsiya qilish operatorlarni qo'shish mumkin.

check_op( ) funksiya tanasini ostiga, amalni POLIZga yozadigan operator prog.put_lex (Lex (op)) qo'shib qo'yamiz,va huddi shunday check_not( ) funksiyaga prog.put_lex (Lex (LEX_NOT)) operatorni qo'shamiz.

Shunda ifodani kontekst shartlarini tekshirish hamda tarjima qilish amallar bilan kengaytirilgan grammatikani qoidalari quyidagicha bo'ladi:

E → E1 | E1 [= | < | >] { st_lex.push (c_type) } E1 { check_op ( ) }

E1 → T { [+ | − | or] { st_lex.push (c_type) } T { check_op ( ) } }
*T → F { [* | / | and] { st_lex.push (c_type) } F { check_op ( ) } }*

F → I { check_id ( ); prog.put_lex (curr_lex); } |
N { st_lex.push (LEX_INT); prog.put_lex (curr_lex); } |
[true | false] { st_lex.push (LEX_BOOL); prog.put_lex (curr_lex); } |
not F { check_not( ); } | (E)

Qiymat berish operatorni quyidagicha bo'ladi:

S → I { check_id ( ); prog.put_lex (Lex (POLIZ_ADDRESS, c_val)); } :=
E { eqtype ( ); prog.put_lex (Lex (LEX_ASSIGN)); }

Ifoda va qiymatni berish operatorini generatsiya qilishda prog obyektimiz ketma-ket to'ldirilardi. Shartli operatorni semantikasi shundayki, o'tish amallarni operandlari generatsiya vaqtida hali noma'lum bo'ladi. Shu sababli bu operanlarni joyini bo'sh qoldirish kerak bo'ladi, va ularni manzilini saqlab qolishga to'g'ri keladi. Qiymatlar ma'lum bo'lgandan so'ng ular saqlangan manzilga yoziladi.

Shartli operatorni kontekst shartlarini tekshirish, hamda tarjima qilish amallar bilan kengaytirilgan grammatikani qoidalari quyidagicha bo'ladi:

S → if E { eqbool( ); pl2 = prog.get_free( ); prog.blank( );
prog.put_lex (Lex (POLIZ_FGO)); }
then S1 { pl3 = prog.get_free( ); prog.blank( );
prog.put_lex (Lex (POLIZ_GO));
prog.put_lex (Lex (POLIZ_LABEL, prog.get_free( )), pl2); }
else S2 { prog.put_lex (Lex (POLIZ_LABEL, prog.get_free( )), pl3); }

Sikl operatorning kontekst shartlarini tekshirish, hamda tarjima qilish amallar bilan kengaytirilgan grammatikani qoidalari quyidagicha bo'ladi:

S → while { pl0 = prog.get_free ( ); } E { eqbool ( );
pl1 = prog.get_free ( ); prog.blank ( );
prog.put_lex (Lex (POLIZ_FGO)); }
do S { prog.put_lex (Lex (POLIZ_LABEL, pl0);

prog.put_lex (Lex (POLIZ_GO));
prog.put_lex (Lex (POLIZ_LABEL, prog.get_free( )), p11); }

Kiritish va chiqarish operatorlarning kengaytirilgan grammatikasi quyidagicha bo'ladi:

S → read (I { check_id_in_read( );
prog.put_lex (Lex (POLIZ_ADDRESS, c_val)); })
{prog.put_lex (Lex (LEX_READ)); }

S → write (E) {prog.put_lex (Lex (LEX_WRITE)); }

POLIZdagi model til interpretatori

M-tilni interpretatorni algoritmi ifodalarni hisoblash algoritmiga o'xshaydi.

POLIZni chapdan o'nga qarab ko'rib chiqiladi.. Agar operand bo'lsa stekga yoziladi, agar amal bo'lsa stekdan yetarli operandlar olinadi va kerak bo'lsa natija qayta stekga yoziladi va hokazo.

Interpretator dastursi quyidagicha Interpretator sinfda joylashgan.

```
// hisobotni bajarish sinfi
class Yexecuter
{
 Lex pc_el;
public:
  void execute ( Poliz& prog );
};
void Yexecuter::execute ( Poliz& prog )
{
 Stack < int, 100 > args;
 int i, j, index = 0, size = prog.get_free();
 while ( index < size )
 {
```



```
pc_el = prog [ index ];
 switch ( pc_el.get_type () )
{
  case LEX_TRUE:
  case LEX_FALSE:
  case LEX_NUM:
  case POLIZ_ADDRESS:
  case POLIZ_LABEL:
  args.push ( pc_el.get_value () );
   break;
  case LEX_ID:
  i = pc_el.get_value ();
   if ( TID[i].get_assign () )
  {
   args.push ( TID[i].get_value () );
    break;
  }
 else
  throw "POLIZ: indefinite identifier";
case LEX_NOT:
args.push( !args.pop() );
 break;
case LEX_OR:
i = args.pop();
args.push ( args.pop() || i );
 break;
case LEX_AND:
i = args.pop();
args.push ( args.pop() && i );
  break;
case POLIZ_GO:
index = args.pop() - 1;
 break;
case POLIZ_FGO:
i = args.pop();
```

```
 if ( !args.pop() )
 index = i - 1;
 break;
case LEX_WRITE:
cout << args.pop () << endl;
 break;
case LEX_READ:
{
  int k;
 i = args.pop ();
  if ( TID[i].get_type () == LEX_INT )
 {
  cout << "Input int value for";
  cout << TID[i].get_name () << endl;
  cin >> k;
 }
  else
 {
  char j[20];
 rep:
  cout << "Input boolean value;
  cout << (true or false) for";
  cout << TID[i].get_name() << endl;
  cin >> j;
   if ( !strcmp(j, "true") )
   k = 1;
   else if ( !strcmp(j, "false") )
   k = 0;
   else
  {
   cout << "Yerror in input:true/false";
   cout << endl;
    goto rep;
  }
}
```



```
     TID[i].put_value (k);
     TID[i].put_assign ();
      break;}
     case LEX_PLUS:
     args.push ( args.pop() + args.pop() );
      break;
     case LEX_TIMES:
     args.push ( args.pop() * args.pop() );
      break;
     case LEX_MINUS:
     i = args.pop();
     args.push ( args.pop() - i );
      break;
     case LEX_SLASH:
     i = args.pop();
      if ( !i )
     {
      args.push ( args.pop() / i );
       break;
   }
      else throw "POLIZ:divide by zero";
     caseLEX_EQ:
     args.push ( args.pop() == args.pop() );
    break;
     case LEX_LSS:
 i = args.pop();
args.push ( args.pop() < i);
break;
     case LEX_GTR:
i = args.pop();
args.push ( args.pop() > i );
break;
     case LEX_LEQ:
i = args.pop();
args.push ( args.pop() <= i );
```

```
break;
    case LEX_GEQ:
i = args.pop();
args.push ( args.pop() >= i );
break;
    case LEX_NEQ:
i = args.pop();
args.push ( args.pop() != i );
 break;
    case LEX_ASSIGN:
i = args.pop();
j = args.pop();
TID[j].put_value(i);
TID[j].put_assign();
 break;
   default:
     throw "POLIZ: unexpected elem";
  }       // end of switch
++index;
};       //end of while
 cout << "Finish of executing!!!" << endl;
}

class Interpretator
{Parser pars;
 Yexecuter E;
public:
 Interpretator(char* program ): pars (program)
{};
  void interpretation ();
};

void Interpretator::interpretation ()
{
 pars.analyze ();
```



```
 E.execute (pars.prog );
}
// interpretatorni ishlatish dastursi
int main ()
{
  try
 {InterpretatorI("program.txt");

   I.interpretation ();
   return 0;
 }
  catch ( char c )
 {
  cout << "unexpected symbol " << c << endl;
   return 1;
 }
  catch ( Lex l )
 {
  cout << "unexpected lexeme";
  cout << l;
   return 1;
 }
  catch ( const char *source )
 {     cout << source << endl;
  return 1;
 }
}
```

Xulosa

Amalda sintaktik, semantik tahlil va dasturni ichki tasvirini generatsiya qilish baravar bajariladi.

Dasturni ichki tasvirini tuzish uchun sintaktik boshqaruv tarjima qo'llash mumkin. Sintaktik boshqaruv tarjimani asosida amallar bilan kengaytirilgan grammatika turibdi. Dastlabki leksemalar zanjirini tahlil qilish jarayonida parallel ravishda dasturni ichki tasvirini generatsiya

qilish amallari bajariladi.

Dasturni ichki tasviri generatsiyasi sintaktik tahlil qilish vaqtida o'tadi, shu vaqtda kontekst shartlar ham nazorat qilinadi,. Shu sababli ular ishlatgan steklarni generatsiyada qo'llash mumkin, masalan ifodani tahlil qilishda. Undan tashqari sematik tahlil qilish, va funksiyalar ichiga generatsiya qilish operatorlarni qo'shish mumkin.

Nazorat uchun savollar

1. Sintaktik boshqaruv tarjimasini ta`rifini keltiring.
2. Sintaktik boshqaruv nimaga asoslangan va qanday amalga oshiriladi.
3. Model tilni arifmetik ifodasini ichki tasviriga o'tkazishni ko'rsating.
4. Model tilni shartli operatorini ichki tasviriga o'tkazish kengaytirilgan grammatikasini tushintirib bering.
5. Model tilni sikl operatorini ichki tasviriga o'tkazish kengaytirilgan grammatikasini tushintirib bering.
6. Yuqorida berilgan dastur bo'yicha interpretator ishini izohlab bering.

Amaliyot uchun topshiriqlar

1. O'zgaruvchi a,b amallar +, −, *, / va qavslar () dan iborat bo'lgan ifodani POLIZga o'tkazadigan amallar bilan kengaytirilgan grammatika tuzilsin.
2. Berilgan grammatika uchun unar – amalni hisobga oluvchi amallar bilan kengaytirilgan grammatika tuzilsin va unga rekursiv tushish protseduralar yozilsin:

   $E \to T \{+T\}$

   $T \to F \{*F\}$

   $F \to (E) \mid i$
3. Bizga amallar bilan kengaytirilgan grammatika berilgan, kiruvchi va chiquvchi tillar aniqlansin:

a) $S \to a\,(a = 1;\ b = 0;\ )A\ \perp$

$A \to a\,(if\,(a)\,\{\,cout << 'a';\ a = 0;\ \}\ else\ a{+}{+};\ )A \mid$

$bA\,(if\,(b)\,\{\,cout << 'b';\ b = 0;\ \}\ else\ b{+}{+};\ ) \mid \varepsilon$

b) $S \to (a = 0;\ )E\perp(cout << '\perp';\ )$

$E \to a\,(a = 1;)\,E\,(cout << 'a';\ ) \mid$

$b\,(if\,(a == 0)\ cout << 'b';)\,E\,(cout << 'b';\ ) \mid \varepsilon$

3. L1 tilni grammatikasi tuzilsin. L2 tilni zanjirlarini generatsiya qilish amallar grammatikaga qo'shilsin. Amallar sifatida faqat cout << 'belgi' operatori ishlatilsin.

a) $L1 = \{a^n c^m b^n \mid n \geq 0,\ m \geq 1\}$, $L2 = \{0^i 1^k \mid i \geq 0,\ k > i\}$,
$\tau = \{(a^n c^m b^n,\ 0^n 1^{n+m}) \mid n \geq 0,\ m \geq 1\}$;

b) $L1 = \{\alpha c^n \mid \alpha \in \{a, b\}^*,\ n \geq 1\}$, $L2 = \{a^n c^m \mid n \geq 1,\ m \geq 0\}$, $\tau = \{(\alpha c^n, a^n c^m) \mid$
$\alpha \in \{a, b\}^*, n \geq 1,\ m = |\alpha|_a$ (ya'ni m — α zanjirdagi a harf soni $)\}$;

c) $L1 = \{a, b\}^+$, $L2 = \{1^i 0^j \mid i \geq 1, j \geq 0;\ i \geq j\}$,
$\tau = \{(\omega,\ 1^{n-m}0^n) \mid \omega \in \{a,b\}^+,\ n = |\omega|_a,\ m = |\omega|b\}$;

d) $L1 = \{a, b\}^+$, $L2 = \{2^n\alpha \mid n \geq 0,\ \alpha \in \{a, b\}^+\}$,
$\tau = \{(\omega,\ 2^n\omega^R) \mid \omega \in \{a,b\}^+,\ n = |\omega|_a\}$;

e) $L1 = \{1^n 0^m 1^m 0^n \mid m, n > 0\}$, $L2 = \{1^i 0^k \mid k > i > 0\}$,
$\tau = \{(1^n 0^m 1^m 0^n,\ 1^m 0^{n-m}) \mid m, n > 0\}$;

f) $L1 = \{\alpha_1\alpha_2 \ldots \alpha_n\perp \mid n \geq 1,\ \alpha_i \in \{ab, ba\},\ 1 \leq i \leq n\}$, $L2 = \{\omega\perp \mid \omega \in \{a, b\}^+\}$,
$\tau = \{(\alpha_1\alpha_2 \ldots \alpha_n\perp,\ \beta_1\beta_2 \ldots \beta_n\perp \mid n \geq 1;\ 1 \leq i \leq n\ \ \alpha_i \in \{ab, ba\},$
$\beta_i = b,\ agar\ \alpha_i = ab,\ \beta_i = a,\ agar\ \alpha_i = ba\}$;

4. L1 tilni grammatikasi tuzilsin. L2 tilni zanjirlarini generatsiya qilish amallar grammatikaga qo'shilsin. Mos zanjirlar τ tarjima bilan beriladi. Amallar sifatida har qanday operatorlarni ishlatish mumkin.

a) $L1 = \{1^m 0^n \mid n, m > 0\}$, $L2 = \{1^k \mid k > 0\} \cup \{0^i \mid i > 0\} \cup \{\varepsilon\}$,
$\tau = \{(1^m 0^n, 1^{m-n}) \mid m > n > 0\} \cup \{(1^m 0^n, 0^{n-m}) \mid n > m > 0\}$
$\cup$
$\{(1^m 0^n, \varepsilon) \mid m = n\}$;

b) $L1 = \{\omega_i \mid i \geq 0,\ \omega_i = (i)_2$ (ya'ni ω_i — bu i ∈ N ikkilik tizimdagi ko'rinishi) $\}$, $L2 = \{(\omega_i)^R \mid i \geq 1,\ \omega_i = (i)_2\}$,
$\tau = \{(\omega_i,\ (\omega_{i+1})^R) \mid i \geq 0,\ \omega_i = (i)_2,\ \omega_{i+1} = (i+1)_2\}$;

c) $L1 = \{\alpha\perp \mid \alpha \in \{a, b\}^*\}$, $L2 = \{b^n\beta\perp \mid n \geq 0,\ \beta \in \{a, b\}^*\}$,
$\tau = \{(\alpha\perp,\ b^n\alpha^R\perp) \mid \alpha \in \{a, b\}^*,\ n = |\alpha|a\}$;

d) $L1=\{\ \omega\perp \mid \omega \in \{a, b\}^*\},\ L2=\{\ a^i b^k \mid i, k \geq 0,\ i+k > 0\ \},$
$\tau=\{\ (\omega\perp,\ a^{[n/2]}b^{[m/2]}) \mid \omega \in \{a, b\}^*,\ n = |\omega|_a,\ m = |\omega|_b\ \};$

e) $L1=\{\ \omega\perp \mid \omega \in \{a, b\}^*\},\ L2=\{\ a^i b^k \mid i, k \geq 0,\ i+k > 0\ \},$
$\tau=\{\ (\omega\perp,\ a^{[(n+1)/3]}b^{[m-n/2]}) \mid \omega \in \{a, b\}+,\ n = |\omega|_a,\ m = |\omega|_b\ \};$

f) $L1=\{\omega\perp \mid \omega\in\{0,1\}^*, \omega = (i)_2^R,\ i \geq 0\ \}, L2=\{\ |^n| \ n \geq 0\ \},$
$\tau=\{\ (\omega\perp,\ |^i\ ) \mid \omega\in\{0,1\}^*, \omega = (i)_2^R,\ i \geq 0\ \}.$

5. Butun ikkilik sonlarni (0 va 1 raqamlar soni juft) to'rtlik tizimga tarjima qiladigan amallar bilan kengaytirilgan grammatikasi tuzilsin.
6. O'zgaruvchi a,b amallar +, −, *, / , ** va qavslar () dan iborat bo'lgan ifodani POLIZga o'tkazadigan amallar bilan kengaytirilgan grammatika tuzilsin. Bunday ifodalar uchun interpretator yaratilsin.

32-BOB. AVTOMATIK GRAMMATIK TAXLIL VOSITASI

Tayanch iboralar: *Lex dasturi, Lex tili , Lex tuzimi, Lex dasturdagi to'qnashuvchlar.*

Leksik taxlilchilarni tuzish dasturi - *LEX*

Dasturlash tillarni leksikasi ta'riflash uchun muntazam tillarni kuchi yetarlikdir. Muntazam tillarni berish turlaridan biri - muntazam ifodalar. ***Lex*** dasturi leksik tahlilchilarni tuzish dasturlardan eng yaxshilardan biri hisoblanadi. Bu dastur regulyar ifodalar asosida berilgan grammatika uchun tahlilchi dastur tuzadi.

Lex dasturi chekli avtomat va holatlar diagrammasi asosida ishlaydi. Dastlab u faqat UNIX operatsionn tizimida mavjud edi. Hozir u barcha operatsionn tizimlarda xam bor, shu jumladan Microsoft Windows tizimda.

Kiruvchi matn ***Lex*** tilida yoziladi bu tilda tahlil qilinuvchi tilni tafsiflovchi muntazam ifodalar faylga yoziladi. ***Lex*** tili kengaytirilgan muntazam ifodalarni ishlatadi. Har bir ifoda qolip deb nomlanadi. Qoliplarda ishlatiladigan belgilar quyidagi jadvalda keltirilgan:

Ifoda	Ma'nosi	Qolip	Qolipga tushuvchi satlar
s	bitta nooperatorli belgi s	a	"a"
\s	bevosita belgini o'zi	*	"*"
"s"	bevosita satrni o'zi(s-belgilar ketma ketligi)	"a*1."	"a*1."
.	bu belgini o'rnida yangi satrga o'tish belgisidan tashqari har qanday belgi bo'lishi mumkin	a.b	"adb"; "acb"; "a1b"; "a+b"
^	^ belgi satrni boshini bildiradi	^abc	"abc ds nm, "
$	$ belgi satrni ohirini bildiradi	abc$	"ddhgfgabc"
[s]	bu joyda satrni ihtiyoriy belgisi bo'lishi mumkin	[abc]	"a"; "b";"c"
$[c_1\text{-}c_2]$	$c_1$dan $c_2$gacha xar qanday belgi bo'lishi mumkin	[a-z]	"a"; "r";"u"

[^s]	satrga kirmagan ixtiyoriy belgi turishi mumkin	[^abc]	"d"; "1";"Z"
r*	r muntazam ifoda nol yoki undan ko'p qaytarilishi mumkin	ab*c	"ac"; "abc"; "abbc";"abbbc"
r+	r muntazam ifoda bir yoki undan ko'p qaytarilishi mumkin	ab+c	"abc"; "abbc"; "abbbc"
r?	r muntazam ifoda nol yoki bir marta qaytarilishi mumkin	ab?c	"ac"; "abc"
r{m,n}	r muntazam ifoda m dan n gacha qaytarilishi mumkin (m<n)	ab{2,4}c	"abbc"; "abbbc"; "abbbbc"
r_1r_2	r_1 dan keyin r_2 keladi (konkantenatsiya)	a+bc+d	"abcd"; "aabccd"; "abccd"
$r_1\|r_2$	r_1 yoki r_2 turadi	a+b|c+d	"ab"; "ccd"; "aaab";"cccd"
(r)	r ni o'zi turadi	(a+b)|(c+d)	"ab"; "aab"; "cd";"cccd"
r_1/r_2	r_1 agar undan keyin r_2 kelsa	"abc"/"123"	"abc123";

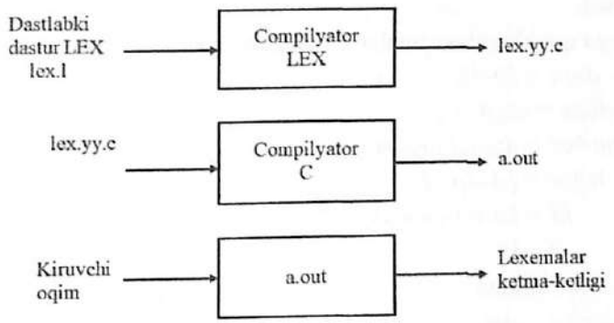
Bu faylni qayta ishlovchi ***Lex*** kompilyator holatlar diagrammasini yasaydi va C dasturlash tilida matn hosil qiladi. Hosil qilingan dastur muntazam ifodalar asosida tuzilgan holatlar diagrammani o'tish jadvalini o'z ichiga oladi. Undan tashqari dasturga, o'tish jadvali asosida leksemalarni anglovchi, standart dastur kiritilgan.

Lex dasturni qo'llash

Lex dasturni qo'llanishi 32.1-rasmda keltirilgan.

Kiruvchi fayl lex.1da ***Lex*** tilida yozilgan grammatika beriladi . ***Lex*** kompilyatori lex.1ni qayta ishlab lex.yy.c faylga C dasturlash tilidagi dasturni yozadi. So'ng bu fayl C kompilyator bilan qayta ishlab ijrovchi fayl a.out hosil qiladi. Bu fayl leksik tahlilchi bo'lib, kiruvchi belgilarni qabul qilib ularni leksemalarga ajratadi. Ko'pincha ijrovchi dastur sintaktik

tahlilchida qism-dasturi sifatida kiradi va ishlatiladi. Bu ko'rinishdagi funksiya joriy leksemani kodini qaytaradi. Undan tashqari global o'zgaruvchi yylval identifikatorlar jadvaldagi yozuvni tartib raqamini ko'rsatadi ya'ni leksemani qiymatini bildiradi.



32.1-rasm. Lex yordamida leksik taxlilchini yaratish chizmasi

Lex dasturni tuzimi

Lex tilida yozilgan dastur quyidagicha bo'ladi:

Yelonlar
%%
Translyatsiya qoidalari
%%
Qo'shimcha funksiyalar

Yelonlar bo'limida o'zgaruvchilar, nomlangan o'zgarmaslar va muntazam ta'riflar tafsif qilinadi.

Translyatsiya qoidalari quyidagicha yoziladi:

Qolip {Amal}

Har bir qolip muntazam ifodadir va unda muntazam tariflar qo'llanishi mumkin. Amallar bu C tilida yozilgan kod parchalari.

Uchinchi bo'lim amallarda ishlatiladigan hilma-hil qo'shimcha funksiyalardan iborat.

Lex dasturi bilan tuzilgan leksik tahlilchi sintaktik tahlilchi bilan birgalikda ishlaydi. Sintaktik tahlilchi joriy leksemani olish uchun leksik taxlilchini chaqiradi. Leksik tahlilchi biror bir P_i qolibga to'g'ri kelguncha qolgan belgilarni kiruvchi oqimdan ketma-ket o'qiydi, so'ng qolib bilan

bog'liq bo'lgan A_i amalni bajaradi. Ko'pincha A_i sintaktik tahlilchiga qaytish amalidan iborat. Leksik tahlilchi sintaktik tahlilchiga leksema kodini butun son ko'rinishda qaytaradi. Shuning bilan birga, leksemaga tegishli qo'shimcha ma'lumotni yylval global o'zgaruvchiga yozib qo'yadi.

Misol.

Bizga quyidagi leksemalar muntazam ifodasi berilsin:

digit = *[0-9]*
digis = *digit*
number = *digits(.digits)?(E[+-]?digits)?*
letter = *[A-Za-z]*
id = *letter(letter|digit)**
if = if
then = then
else = else
relop = <|>|<=|>=|=|<>

Lex tilida yozilgan dastur quyidagicha bo'ladi:

```
%{
 /* Nomlangan o'zgarmaslarni ta'rifi:
 LT, LE, EQ, NE, GT, GE,
 IF, THEN, ELSE, ID, NUMBER, RELOP
 Masalan
 #define LT 200 */
%}
/* Muntazam ta'riflar */
delim [ \t\n]
ws {delim}+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
number {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
%%
{ws} {/* Xech qanday amal yo'q  */}
if {return(IF);}
then {return(THEN);}
```



```
    else {return(ELSE);}
    {id} {yylval = (int) installlDO; return(ID);}
    {number} {yylval = (int) installNum();
return(NUMBER);}
    "<" {yylval = LT; return(RELOP);}
    "<=" {yylval = LE; return(RELOP);}
    {yylval = EQ; return(RELOP);}
    "<>" {yylval = NE; return(RELOP);}
    »>" {yylval = GT; return(RELOP);}
    ">=" {yylval = GE; return(RELOP);}
    %%
    int installID()
    {
    /* identifikator jadvaliga leksemani
    yozish uchun ishlatiladi va tartib raqamini
     qaytaradi. */
    ...
    }
    int installnum()
    {
    /* sonlar jadvaliga leksemani
    yozish uchun ishlatiladi va tartib raqamini
     qaytaradi. */
    ...
    }
```

E'lonlar bo'limida maxsus qavislar - %{ va %} uchraldi, bu qavislar ichidagi belgilar bevosita lex.uu.c faylga nusxalanadi va ular muntazam ta'riflar bo'lmaydi. Ko'pincha bu qavislar ichiga nomlangan o'zgarmaslarni ta'riflarni kiradi. Bu ta'riflar C tilning #define konstruksiyasi bilan beriladi. Bizni dasturda biz bu ta'riflarni keltirmadik.

Undan tashqari bu bo'limda muntazam ta'riflar ketma-ketligi keltirilgan. Agar muntazam ta'riflar boshqa ta'riflarda yoki keyingi bo'limda ishlatilsa ular figurali qavsga olinadi. Masalan delim probel sinfini ta'riflaydi va unga probel, tabulyatsiya va yangi satrga o'tilgan belgilari kiradi, so'ng ws bunday belgilarni ketma-ketligini bildiradi va u {delim}+ bilan ta'riflanadi.

Qo'shimcha funksiyalar bo'limida ikkita funksiya keltirilgan - installID() va installNum(). Qo'shicha bo'limdagi barcha narsalar lex.uu.c faylga nusxhalanadi, lekin ular ammallarda ham ishlatilishi mumkin.

Nixhoyat, qoidalar translyatsiya bo'limdagi qoliplarni ko'rib chiqamiz. Birinchi satrda ws identifikatori ko'rsatilgan va unda hech qanday amallar berilmagan., Bu degani satrda probel belgilar uchrasa ularni tashlab keyingi leksema qidiriladi. Ikkinchi satrda if muntazam ifodani oddiy qolipi keltirilgan. Agar kiruvchi oqimda if uchrasa va undan keyin boshqa harf va raqam uchramasa leksik tahlilchi uni ajratadi va unga mos bo'lgan IF leksema kodini qaytaradi. Qolgan kalit so'zlar ham huddi shunday ajratiladi.

Beshinchi qolip id muntazam ta'rifga to'g'ri keladi. Shuni aytib o'tish kerakki if kalit so'zi o'zini kqolipiga ham to'g'ri keladi va identifikator qolipiga ham to'g'ri keladi. Bunday vaziyatda Lex birinchi keltirilgan kqolipni tanlaydi. Agar oqimdagi belgilar ketma-ketligi id ketma- ketligiga mos kelsa unda quyidagi amallar bajariladi:

1. installID() funksiya chaqiriladi va u ajratilgan leksemani indentifikatorlar jadvaliga joylaydi.
2. Funksiya qaytarish qiymat sifatida identifikator jadvalidagi leksemani indeksini qaytaradi va bu qiymat yylval global o'zgaruvchida saqlanib qo'yiladi.
3. Sintaktik tahlilchiga ID kodni qaytaradi, ya'ni joriy leksema identifikatorligi ta'qiqlaydi.

Oltinchi qolip sonni ajratadi va yuqorida keltirilgan amallarni bajaradi faqat installNum() funksiya bilan.

Yettinchi qolipdan boshlab solishtirish amallar uchun leksemalar aniqlanadi. Ular uchun RELOP leksemani kodi qaytariladi va yylval o'zgaruvchiga aniq amal kodi yoziladi.

Lex dasturda to'qnashuvlarni hal qilish

Agar kiruvchi oqimdagi bir nechta belgilar ketma-ketligi bir nechta qolibga mos kelsa, Lex dasturi quyidagi qoidalarga rioya qiladi:

1. Doimo eng uzun ketma-ketlik ajratiladi.

2. Agar eng uzun ketma-ketlik bir nechta kqoliplarga mos kelsa, Lex dasturi birinchi turgan qolibga murojaat qilinadi.

Birinchi qoidaga asosan yuqoridagi misolda <= belgilarni bitta amal deb xisoblanadi.

Ikkinchi qoidaga asosan barcha kalit so'zlar oldindan aniqlab qo'yiladi.

Xulosa

Dasturlash tillarni leksikasi ta'riflash uchun muntazam tillarni kuchi yetarlikdir. Muntazam tillar muntazam ifodalar orqali berilishi mumkin. Lex dasturi regulyar ifodalar asosida berilgan grammatika uchun tahlilchi dastur tuzadi.

Lex dasturi chekli avtomat va holatlar diagrammasi asosida ishlaydi. Kiruvchi matn Lex tilida faylga muntazam ifodalar ko'rinishdagi fayliga yoziladi. Bu faylni qayta ishlovchi Lex kompilyator holatlar diagrammasini yasaydi va SI dasturlash tilida matn hosil qiladi. Hosil qilingan dastur muntazam ifodalar asosida tuzilgan holatlar diagrammani o'tish jadvalini o'z ichiga oladi. Undan tashqari dasturga, o'tish jadvali asosida leksemalarni anglovchi, standart dastur kiritilgan. SI tilidagi dastur qayta ishlanib ijrovchi fayl hosil qiladi ana shu fayl leksik tahlilchi bo'lib, kiruvchi belgilarni qabul qilib ularni leksemalarga ajratadi. Ko'pincha ijrovchi dastur sintaktik tahlilchida qism-dasturi sifatida kiradi va ishlatiladi.

Nazorat uchun savollar

1. Lex dasturning vazifasi nimadan iborat?
2. Lex dasturi nima asosida ishlaydi?
3. Lex dasturning tuzimini keltiring.
4. Kiruvchi tilning grammatikasi nima orqali beriladi?

Amaliyot uchun topshiriqlar

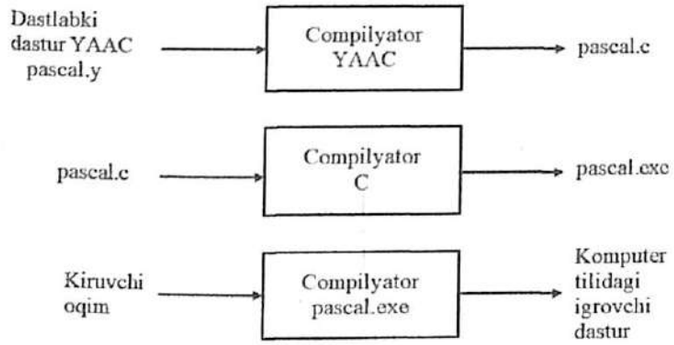
1. M-tilining leksik taxlilchini Lex dastur orqali yarating.
2. Paskal tilni leksik leksik taxlilchini Lex dastur orqali yarating.

33-BOB. AVTOMATIK SINTAKTIK TAXLIL VOSITASI

Tayanch iboralar: *Yacc dasturi, Yacc tili , Yacc tuzimi, Yacc dasturdagi to'qnashuvlar, Yacc dasturida hatoliklarni qayta ishlash.*

Sintaktik tahlilchini tuzish dasturi – YACC

Bu bo'limda sintaktik tahlilchini generatori LALR-tahlilchi YAAC (Yet another compiler-compiler)ni ko'rib chiqamiz. Bu dastur yuqorida aytilgan barcha imkoniyatlarni o'ziga oladi. Undan tashqari bu dastur eng keng tarqalgan. Bu dastur ko'p kompilatorlarni yaratishda qo'llangan.



YACC dasturni qo'llash 4.2- rasmda keltirilgan.

4.2-rasm Yacc dasturida kompilyator yaratish.

Boshida Yacc tilida masalan paskal tili uchun spetsifikatsiyalar pascale.u dastlabki faylga yoziladi.

So'ng bu faylni Yacc generator

yacc -dltv -b pascal pascal.u

buyruq bilan qayta ishlab pascal.c C tilida sintaktik tahlilchini dasturini yaratib beradi va pascal.c faylga yozadi.

Bu faylga foydalanuvchi yozgan funksiyalar ta'riflari qo'shilib mukammal kompilyator tuzilishi mumkin.

Shuning bilan birga qo'shimcha fayl yaratadi – pascal.out. Bu faylda grammatik qoidalar va holatlar (vaziyatlar) ro'yxati chiqadi.

Yacc-dasturni tuzilishi.

Dastur uch qismdan iborat:

Yelonlar

%%

Translyatsiya qoidalari

%%

Qo'shimcha funksiyalar

Birinchi elonlar qismda dasturga kerak bo'lgan ta'riflar joylashadi. Bu qism ikki bo'limdan bo'lishi mumkin. Birinchi bo'limda S tilida kerakli elonlari turadi, bu bo'lim %(va)% qavslar ichiga olinadi. Bu bo'limda ikkinchi va uchinchi qismga kerak bo'lgan vaqtincha o'zgaruvchilar va o'zgarmaslar e`lonlari yoziladi. Ikkinchi bo'limida leksemalarni e`lonlari va maxsus e`lonlar turadi.

Ikkinchi qoidalar transtlyatsiya qismda grammatikani qoidalarni va ularga tegishli semantik amallar turadi. Bu qismi %% belgilardan boshlanib shu belgilar bilan tuganlanadi. XHar bir qoida yangi satrdan boshlanadi va bir nechta satrga joylanishi mumkin. Qoida ";" belgisi bilan yakunlanadi. Agar qoidada muqobil qismi bo'lsa ular vertikal chiziq bilan ajratiladi. KQoydani chap tara`fi ":" belgi bilan ajratiladi.

Bitta qo'shtirnoq belgini ichiga olingan belgi va birinchi qismida ta'riflangan leksemalar nomi terminal hisoblanadi.

Yaccda leksemalardan farqli bo'lgan va qo'shtirnoq ichiga olinmagan belgilar ketma- ketligi noterminal deb hisoblanadi. Birinchi noterminal belgisi boshlang'ich belgi hisoblanadi.

Semantik amallar "{" va "}" belgilar ichiga olinadi. Semantik amallar S yoki C++ tilida beriladi. Semantik amallar ichida mahsus belgi "$$" ishlatilishi mumkin,. U qoidani chap tarafida turgan noterminal belgini qiymati hisoblanadi va joriy noterminal qiymati yylval global o'zgaruvchida turadi. Undan tashqari $i belgi bilan kqoyidani i-chi grammatik elementni qiymati belgilanadi.

Ko'pincha semantik amallar qoidada belrilgan amallarni bajarishini ko'rsatadi va ular asosida chap tarafdagi noterminal belgini qiymatini aniqlaydi. Agar amal ko'rsatilgmasa kelishuv bo'yicha noterminalni qiymati birinchi grammatik elementga teng ($$=$1). Semantik amallar qoidani o'ralishida bajariladi.

Uchinchi qismda C kompilyatorga kerakli funksiyalar yoziladi,. Shu jumladan leksik taxlilchini kodi, asosiy main () funksiyasi va boshqa funksiyalar. Leksik tahlilchini kodi yylex() funksiyada bo'ladi. Ko'pincha bu funksiyani LEX dasturi yaratadi. yylex() funksiya dastlabki oqimdan joriy leksemani ajratib uning kodini yoki ASCII kodini qaytaradi. Agar leksema qiymatga ega bo'lsa, qiymat global o'zgaruvchi yylvalga yozib qo'yiladi.

Yacc dasturdagi to'qnashuvlarni hal qilish yo'llari

Grammatikalar ko'p manolik bo'lishi mumkin bu holda Yacc dasturi to'qnashuv ro'yhatini y.out faylga chiqaradi.

Agar mahsus ko'rsatilmagan bo'lsa, barcha to'qnashuvlar quyidagi ikkita qoida bilan hal qilinadi.

1. Agar to'qnashuv "o'rash/o'rash" bo'lsa, dastlabki Yacc tilidagi dasturda oldin turgan qoidani tanlash bilan hal qilinadi.

2. Agar to'qnashuv "ko'chirish/o'rash" bo'lsa, Yacc ko'chirishni tanlash bilan hal qiladi. Bu qoida "osilib qolgan" else muammosini korrekt hal qiladi. Lekin bu kelishuv hamma vaqt to'g'ri bo'lmagan uchun Yacc yana bir umumiy mexanizm mavjud:

Terminallarni e'lon qilganda ularga ustunligini va bajarish tartibini ko'rsatish mumkin,. Bir hil ustunlikka ega bo'lgan terminallar bitta satrga yoziladi,va ro'yxatda oldin turgan terminallar ustunligi pastroq bo'ladi. Undan tashqari terminallar tanlash tartibini berish mumkin, chapdan o'nga (left) yoki o'ngdan chapga (right) masalan

```
%left '+' '-' /* Chapdan o'nga bajarish, past
ustunlik */
%left '*' '/' /* Chapdan o'nga bajarish, yuqori
ustunlik */
%right UNARYMINUS
```

Undan tashkqari terminallarni bajarish yo'li yo'qligini berish mumkin ya'ni ikkita amal bir-biri bilan kombinatsiya bo'lmasligi:

```
%nonassoc '<' '>' '='
```


Grammatikada uchragan "ko'chirish/o'rash" to'qnashuvlarni xhal qilish uchun Yacc dasturi to'qnashuvda qatnagan qoidalarga ham ustunlik (va) hamda bajarish yo'nalishini beradi.

Kiruvchi belgini stekga ko'chirish bilan $A \rightarrow \alpha$ joriy qoidani o'rashni tanlashda,va qoidani ustunligi yuqori bo'lsa yoki teng bo'lib qoidani bajarish yo'nalishi chap bo'lsa, Yacc o'rashni tanlaydi,. Aks holda ko'chirishni tanlaydi.

Masalan stek ustida $E \rightarrow E+E$ qoida bo'lib kiruvchi belgi "+" bo'lsa qoida o'raladi, chunki qoidani ustunligi kiruvchi belgini ustunligi bilan teng,. Lekin bajarish yo'nalishi chap. Agar kiruvchi belgi "*" bo'lsa belgi stekga ko'chiriladi, chunki belgini ustunligi qoidani ustunligidan yuqori.

Ko'pincha qoidani ustunligi va eng o'ng terminal belgini ustunligi va bajarish yo'nalishiga teng.

Agar eng o'ng terminal qoidaga to'g'ri ustunlik bermasa, unda ustunlikni quyidagicha mahsus buyruq bilan o'zgartirish mumkin:

%rges (terminal).

Shunda qoidani ustunligi va bajarish yo'nalishi ko'rsatilgan terminalnikiga teng.

Ustunlik va bajarish yo'nalishi mehanizm orqali hal qilingan "ko'chirish/o'rash" to'qnashuvlar to'g'risida Yacc xabar bermaydi.

Bunday terminal sunniy kiritilishi mumkin va u faqat ustunligni aniqlash uchun kiritiladi. Masalan quyidagi misolda

```
%right UNARYMINUS
```

UNARYMINUS leksemaga eng yuqori ustunlik beradi va

```
expr: '-' expr %prec UNARYMINUS
```

qoida unar minus amal uchun eng yuqori ustunlik qoida deb hisoblanadi.

Yacc dasturida hatoliklarni qayta ishlash

Dastlabgi til dasturida hatoliklar uchrasa Yacc dasturi hatolik bo'lmagan joygacha yoki mahsus hatolik qoidasigacha stekni tozalab, davom etishga harakat qiladi. Oldin qaysi "yirik" noterminallar uchun

hatolik qoidalarni kiritishni hal qilish kerak. Ko'pincha bu ifodalar, bloklar, protseduralarni ta'riflaydigan noterminallar bo'ladi. So'ng grammatikaga

A → *error* α qoida qo'shiladi, bunda A "yirik" noterminallardan biri, α belgilar ketma-ketligi. Hatolik uchrasa sintaktik tahlilchi hatolik qoidani punkti bor holatni qidirib stekda barcha belgilarni olib tashlaydi, toki stek ustida shu holat qolmaguncha. So'ng sintaktik tahlilchi error leksemani stek ustiga qo'yadi. Agar α bo'sh bo'lsa hatolik mahsus qoida o'raladi va unga tegishli semantik amallar bajariladi, so'ng tahlil davom etadi. Agar α bo'sh bo'lmasa sintaktik tahlilchi αgacha bo'lgan belgilarni kiruvchi oqimdan olib tashlaydi va αni stek ustiga yozadi,. Shundan keyin stek ustigi *error* αni A ga o'rab tahlilni davom etadi.

Masalan hatolik kqoida

smtp --> error ;

bo'lsa sintaktik tahlilchi ; gacha bo'lgan barcha belgilarni kiruvchi oqimdan olib tashlab, hatoga tegishli amalni bajaradi va tahlilni davom ettiradi.

Sodda kalkulyatorga oyid misol

Bizga sodda arifmetik ifodalarni tafsiflaydigan grammatika berilsin.

E → *E+T|E-T\T*

T → *T*F|T/F|F*

F → *-E|(E)|number*

Bunda *number* ixtiyoriy son.

Bu grammatika uchun ko'p manolik qisqa grammatikasini yozamiz:

E → *E+E|E-E\E*E|E/E|-E|(E)|number*

Bu misol uchun sacalc.y faylda Yacc tilida dastlabki dasturni yozamiz.

U quyidagicha bo'ladi:

```
/*
 * Sodda calculator
 */
/* Yelon qismi */
/* Birinchi bo'lim */
```



```
%{
#include <iostream.h>
#define YYDEBUG 0
#define YYSTYPE double     /*  yacc stack
elementlarni turi */
#define QUIT     ( (double) 101010 )

%}
/* Ikkinchi bo'lim */
%token RAQAM
%left '+' '-'   /* Chapdan o'nga bajarish, o'rta
uroven */
%left '*' '/'   /* Chapdan o'nga bajarish, yuqori
uroven */
%right UNARYMINUS

/* Qoidalar translyatciya qismi */

%%
list: /* bo'sh qoyda */ { prompt(); }  /* amal */
     | list '\n' { prompt(); }
     | list expr '\n' { if ($2 == QUIT) {
              return(0);}
     else {cout << "        NATIJA ==========> " <<
$2);
            prompt();
          }
        }
     | list error '\n' /* Hato qiymat */
          { yyerrok(); prompt();}
     ;
expr: NUMBER     { $$ = $1; cout << $1;}
| '-' expr %prec UNARYMINUS { $$ = -$2; cout <<
"!-"; }
| expr '+' expr        { $$ = $1 + $3; cout <<
"+";}
```

```
| expr '-' expr        { $$ = $1 - $3; cout << "-
";}
| expr '*' expr        { $$ = $1 * $3; cout <<
"*";}
| expr '/' expr        { $$ = $1 / $3; cout <<
"/";}
| '(' expr ')'         { $$ = $2; }
;
%%
/* Uchinchi qism

#include <stdio.h>
#include <ctype.h>

char *progname;        /* ?ato ma'lumotlar uchun */
int  lineno = 1;

extern int yyparse( void );

void main( int, char ** );
int  yylex( void );
void prompt( void );
void yyerror( char * );
void warning( char *, char * );

void
main( int argc, char **argv )
{
   if (argc > 1) cout << "parametrlar soni
ko'p\n";
   progname = *argv;

cout<<"/n";
cout<<"*****************************************
*\n";
cout<<"* SACALC: Sodda arifmetik kalkulyator
```



```
*\n";
cout<<"*
*\n";
cout<<"*      1)READY>da ifodani kiriting
*\n";
cout<<"*        misol : 1+2*3<CR>
*\n";
cout<<"* SACALC hisoblab natijani chiqaradi
*\n";
cout<<"*      2)hisobotni tygatish uchun-Quit
*\n";
cout<<"*
*\n";
cout<<"*****************************************
*\n";
cout <<"\n\n";
 yyparse();cout<<"/n";
cout<<"\n*****************************************
*\n";
cout<<"*  SACALC: a Simple Arithmetic Calculator
*\n";
cout<<"*
*\n";
cout<<"*       normal ishladi. Hair!!!
*\n";
cout<<"*****************************************
*\n";

}
/* Lexic tahlil * /
int yylex( void )
{  int c;
   while ((c=getchar()) == ' ' || c == '\t' );
   if (c == EOF) return 0;
   if (c == '.'||isdigit(c))
     { ungetc(c); /* belgi kiruvchi oqimga qayta
```

```
qo'iladi so'ng to'liq son o'qiladi */
 cin>>yylval; return NUMBER;}
   if (c == '\n')  lineno++;
   if (c == 'Q' || c == 'q')
     if ((c=getchar()) == 'U' || c == 'u')
       if ((c=getchar())=='I' || c=='i')
         if ((c=getchar()) == 'T' || c == 't')
           {yylval = QUIT;return( EOF );}
         else return '?';
   return( c );
}
/* yacc syntaktik hato */
void yyerror( char *s ) { warning(s, (void *) 0);}
/* habarni chop etish */
void warning( char *s, char *t )
{  cout<< progname<<  s; if (t) cout<<  t;
    cout << "Nato "<< lineno << " satrda /n");
}
/* kiritishga taklif */
void prompt( void )
{  cout<<"READY> ");}
```

Bu faylni Yacc generatorga uzatamiz va quyidagi buyruqni bajaramiz

yacc -dltv -b sacalc sacalc.y

Generator ishlash natijasida uchta fayl hosil qiladi.
Bosh saxifa fayli sacalc.h:

```
#define NUMBER 257
#define UNARYMINUS 258
```

Bu faylda leksemalar tafsiflanadi ularni qkodlari beriladi.
Generatorni ishlash protokoli sacalc.out faylga yoziladi:

```
   0  $accept : list $end
   1  list :
```



```
   2        | list '\n'
   3        | list expr '\n'
   4        | list error '\n'

   5  expr : NUMBER
   6        | '-' expr
   7        | expr '+' expr
   8        | expr '-' expr
   9        | expr '*' expr
  10        | expr '/' expr
  11        | '(' expr ')'

state 0
    $accept : . list $end  (0)
    list : .  (1)
    .  reduce 1
    list  goto 1

state 1
    $accept : list . $end  (0)
    list : list . '\n'  (2)
    list : list . expr '\n'  (3)
    list : list . error '\n'  (4)
    $end  accept
    error  shift 2
    NUMBER  shift 3
    '-'  shift 4
    '\n'  shift 5
    '('  shift 6
    .  error
    expr  goto 7

state 2
    list : list error . '\n'  (4)
    '\n'  shift 8
    .  error
```

```
state 3
    expr : NUMBER .  (5)
    .  reduce 5

state 4
    expr : '-' . expr  (6)
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 9

state 5
    list : list '\n' .  (2)
    .  reduce 2

state 6
    expr : '(' . expr ')'  (11)
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 10

state 7
    list : list expr . '\n'  (3)
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
    '+'  shift 11
    '-'  shift 12
    '*'  shift 13
    '/'  shift 14
    '\n'  shift 15
    .  error
```



```
state 8
    list : list error '\n' .  (4)
    .  reduce 4

state 9
    expr : '-' expr .  (6)
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
    .  reduce 6

state 10
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
    expr : '(' expr . ')'  (11)

    '+'  shift 11
    '-'  shift 12
    '*'  shift 13
    '/'  shift 14
    ')'  shift 16
    .  error

state 11
    expr : expr '+' . expr  (7)
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 17

state 12
    expr : expr '-' . expr  (8)
```

```
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 18

state 13
    expr : expr '*' . expr  (9)
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 19

state 14
    expr : expr '/' . expr  (10)
    NUMBER  shift 3
    '-'  shift 4
    '('  shift 6
    .  error
    expr  goto 20

state 15
    list : list expr '\n' .  (3)
    .  reduce 3

state 16
    expr : '(' expr ')' .  (11)
    .  reduce 11

state 17
    expr : expr . '+' expr  (7)
    expr : expr '+' expr .  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
```



```
    '*'  shift 13
    '/'  shift 14
    '+'  reduce 7
    '-'  reduce 7
    '\n'  reduce 7
    ')'  reduce 7

state 18
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr '-' expr .  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
    '*'  shift 13
    '/'  shift 14
    '+'  reduce 8
    '-'  reduce 8
    '\n'  reduce 8
    ')'  reduce 8

state 19
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr '*' expr .  (9)
    expr : expr . '/' expr  (10)
    .  reduce 9

state 20
    expr : expr . '+' expr  (7)
    expr : expr . '-' expr  (8)
    expr : expr . '*' expr  (9)
    expr : expr . '/' expr  (10)
    expr : expr '/' expr .  (10)
    .  reduce 10
```

```
11 terminals, 3 nonterminals
12 grammar rules, 21 states
```

Bu faylda qoidalar ro'yxati va ularni tartib raqami beriladi. So'ng LALR(1) sintaktik tahlil bo'yicha barcha holatlarni ro'yxati beriladi. Har bir holatda vaziyatlar to'plami keltirilgan va kiruvchi belgiga nisbatan bajarish amali keltirilgan.

Uchinchi sacalc.cpp faylda bevosita sintaktik tahlilchi berilgan:

```
#include <iostream.h>
#define YYDEBUG 0
#define YYSTYPE double  /* yacc stack turi */
#define QUIT    ( (double) 101010 )
#define NUMBER 257
#define UNARYMINUS 258
#define YYERRCODE 256
short yylhs[] = {-1,0,0,0,0,1,1,1,1,1,1,1,};
short yylen[] = {2,0,2,3,3,1,2,3,3,3,3,3,};
short yydefred[] ={1,0,0,5,0,2,0,0,4,6,0,0,0,
                0,0,3,11,0,0,9,10,};
short yydgoto[] = {1,7,};
short yysindex[] = {0,-10,-8,0,-39,0,-39,-
6,0,0,
            -22,-39,-39,-39,-39,0,0,-33,-
33,0,0,};
short yyrindex[] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,-3,2,0,0,};
short yygindex[] = {0,4,};
#define YYTABLESIZE 247
short yytable[] = {                          5,
6,  8,  0,  15,  0,  4,  7,  9,  13, 10,
0,  8,  0,  14,  17,  18,  19,  20,  16,  13,
11,  0,  12,  0,  14,  0,  0,  0,  0,  6,
0,  0,  0,  0,  4,  13,  11,  7,  12,  7,
14,  7,  8,  0,  8,  0,  8,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
```



```
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  3,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  2,  3, };
short yycheck[] = {                         10,
40,  10,  -1,  10,  -1,  45,  10,   4,  42,  6,
-1,  10,  -1,  47,  11,  12,  13,  14,  41, 42,
43,  -1,  45,  -1,  47,  -1,  -1,  -1,  -1, 40,
-1,  -1,  -1,  -1,  45,  42,  43,  41,  45, 43,
47,  45,  41,  -1,  43,  -1,  45,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,
```

```
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1, 257,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1,  -1,  -1,  -1,  -1, -1,
-1,  -1,  -1,  -1,  -1, 256, 257, };
#define YYFINAL 1
#ifndef YYDEBUG
#define YYDEBUG 1
#endif
#define YYMAXTOKEN 258
#if YYDEBUG
char *yyname[] = {"end-of-file"
0,0,0,0,0,0,0,0,0,"'\\n'",0,0,0,0,0,0,0,0,0,0,
0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"'('","')'","
'*'"
,"'+'",0,"'-
'",0,"'/'",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0
```



```
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,"NUMB
ER",
"UNARYMINUS",};
char *yyrule[] = {
"$accept : list",
"list :",
"list : list '\\n'",
"list : list expr '\\n'",
"list : list error '\\n'",
"expr : NUMBER",
"expr : '-' expr",
"expr : expr '+' expr",
"expr : expr '-' expr",
"expr : expr '*' expr",
"expr : expr '/' expr",
"expr : '(' expr ')'",
};
#endif
#ifndef YYSTYPE
typedef int YYSTYPE;
#endif
#define yyclearin (yychar=(-1))
#define yyerrok (yyerrflag=0)
#ifndef YYSTACKSIZE
#ifdef YYMAXDEPTH
#define YYSTACKSIZE YYMAXDEPTH
#else
#define YYSTACKSIZE 300
#endif
#endif
int yydebug;
int yynerrs;
int yyerrflag;
int yychar;
short *yyssp;
YYSTYPE *yyvsp;
```

```
    YYSTYPE yyval;
    YYSTYPE yylval;
    #define yystacksize YYSTACKSIZE
    short yyss[YYSTACKSIZE];
    YYSTYPE yyvs[YYSTACKSIZE];

    #include <ctype.h>
    #include <stdio.h>
    char *progname;       /* Hato ma'lumotlar
uchun */
    int  lineno = 1;
    extern int yyparse( void );
    void main( int, char ** );
    int  yylex( void );
    void prompt( void );
    void yyerror( char * );
    void warning( char *, char * );
    void
    main( int argc, char **argv )
    {if (argc>1) cout << "parametrlar soni
ortiqcha\n";
    progname = *argv;
    cout<<"\n*************************************
*\n";
    cout<<"*  SACALC: Sodda arifmetiq calculator
*\n";
    cout<<"*
*\n";
    cout<<"*      1)READY> ifodani ciriting
*\n";
    cout<<"*        misol : 1+2*3<CR>
*\n";
    cout<<"* SACALC javobni eqranga chiqaradi
*\n";
    cout<<"*      2)Hisobni to'htatish QUIT
*\n";
```



```
    cout<<"*
*\n";
    cout<<"*************************************
*\n";
    cout<<"\n\n";
      yyparse();
    cout<<"\n*************************************
*\n";
    cout<<"* SACALC: Sodda arifmetiq calculator
*\n";
    cout<<"*
*\n";
    cout<<"*      normal ishladi Hair!!!
*\n";
    cout<<"*************************************
*\n";
    }

    int yylex( void )
    {
      int c;
      while ((c=getchar())==' '||c=='\t');
      if (c == EOF)return 0;
      if (c == '.' || isdigit(c)) {  /* number */
       ungetc(c,stdin); cin>>yylval; return
NUMBER;}
      if (c == '\n') lineno++;
      if (c == 'Q' || c == 'q')  /* Quit code */
        if ((c=getchar()) == 'U' || c == 'u')
          if ((c=getchar()) == 'I' || c == 'i')
            if ((c=getchar()) == 'T' || c == 't')
              {yylval = QUIT;return( EOF );}
            else return '?';
      return( c );
    }
```

```
/* syntax hato  */
void yyerror( char *s )
{     warning(s, (void *) 0);}

/* ogoh habar chop etish*/
void warning( char *s, char *t )
{ cout<< progname<<  s; if (t)  cout<<  t;
  cout << "Nato "<< lineno << " satrda /n";
}

void prompt( void )
{  cout<<"READY> ";}
#define YYABORT goto yyabort
#define YYACCEPT goto yyaccept
#define YYERROR goto yyerrlab
int yyparse()
{
    register int yym, yyn, yystate;
  #if YYDEBUG
    register char *yys;
    extern char *getenv();

    if (yys = getenv("YYDEBUG"))
    {
        yyn = *yys;
        if (yyn >= '0' && yyn <= '9')
            yydebug = yyn - '0';
    }
  #endif

    yynerrs = 0;
    yyerrflag = 0;
    yychar = (-1);

    yyssp = yyss;
    yyvsp = yyvs;
    *yyssp = yystate = 0;
```



```
yyloop:
    if (yyn = yydefred[yystate]) goto yyreduce;
    if (yychar < 0)
    { if ((yychar = yylex()) < 0) yychar = 0;
     #if YYDEBUG
      if (yydebug)
      {  yys = 0;
        if (yychar <= YYMAXTOKEN)
        yys = yyname[yychar];
        if (!yys) yys = "illegal-symbol";
        printf("yydebug: state %d, reading %d
        (%s)\n", yystate,yychar, yys);
      }
     #endif
    }
  if ((yyn = yysindex[yystate]) &&
   (yyn += yychar) >= 0 && yyn <= YYTABLESIZE &&
      yycheck[yyn] == yychar)
  {
    #if YYDEBUG
      if (yydebug)
      printf("yydebug: state %d, shifting
      to state %d\n",yystate, yytable[yyn]);
    #endif
    if (yyssp >= yyss + yystacksize - 1)
       { goto yyoverflow; }
    *++yyssp = yystate = yytable[yyn];
    *++yyvsp = yylval;
    yychar = (-1);
    if (yyerrflag > 0)  --yyerrflag;
    goto yyloop;
  }
   if ((yyn = yyrindex[yystate]) &&
      (yyn += yychar) >= 0 &&
      yyn <= YYTABLESIZE && yycheck[yyn] ==
yychar)
```

```
        { yyn = yytable[yyn]; goto yyreduce;}
      if (yyerrflag) goto yyinrecovery;
     #ifdef lint
         goto yynewerror;
     #endif
     yynewerror:
         fprintf(stderr, "syntax error\n");
     #ifdef lint
         goto yyerrlab;
     #endif
     yyerrlab:
         ++yynerrs;
     yyinrecovery:
         if (yyerrflag < 3)
         {
           yyerrflag = 3;
           for (;;)
           {
            if ((yyn = yysindex[*yyssp]) &&
                (yyn += YYERRCODE) >= 0 &&
                 yyn <= YYTABLESIZE &&
                 yycheck[yyn] == YYERRCODE)
           {
           #if YYDEBUG
           if (yydebug) printf("yydebug: state %d,
              error recovery shifting\
              to state %d\n", *yyssp,
yytable[yyn]);
           #endif
           if (yyssp >= yyss + yystacksize - 1)
             {  goto yyoverflow;}
           *++yyssp = yystate = yytable[yyn];
           *++yyvsp = yylval;
           goto yyloop;
           }
           else
```



```
           {
           #if YYDEBUG
             if (yydebug)printf("yydebug: error
             recovery discarding state
%d\n",*yyssp);
           #endif
           if (yyssp <= yyss) goto yyabort;
           --yyssp;
           --yyvsp;
           }
         }
       }
       else
       {
         if (yychar == 0) goto yyabort;
         #if YYDEBUG
           if (yydebug)
           {
               yys = 0;
               if (yychar <= YYMAXTOKEN)
                  yys = yyname[yychar];
               if (!yys) yys = "illegal-symbol";
               printf("yydebug: state %d, error
               recovery discards token %d
(%s)\n",
                yystate, yychar, yys);
           }
         #endif
         yychar = (-1);
         goto yyloop;
       }
   yyreduce:
   #if YYDEBUG
       if (yydebug)
         printf("yydebug: state %d,
         reducing by rule %d (%s)\n",
```

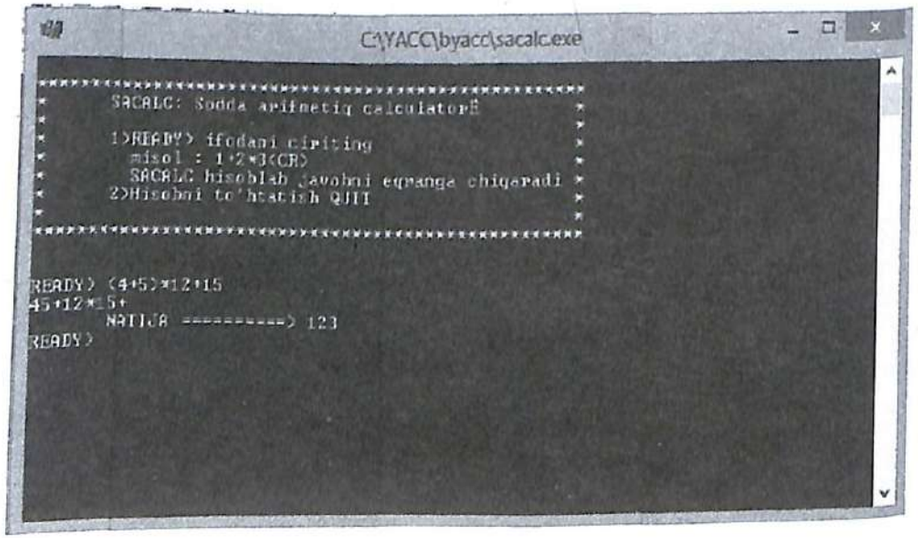
```
        yystate, yyn, yyrule[yyn]);
      #endif
      yym = yylen[yyn];
      yyval = yyvsp[1-yym];
      switch (yyn)
      {
      case 1: { prompt(); } break;
      case 2: { prompt(); } break;
      case 3:
      { if (yyvsp[-1] == QUIT) { return(0);}
        else {cout << "\n   NATIJA ==========> "
         << yyvsp[-1]<<'\n';prompt();}
      } break;
      case 4: { yyerrok;prompt();} break;
      case 5:{ yyval= yyvsp[0]; cout <<
yyvsp[0];}break;
      case 6:{ yyval = -yyvsp[0];cout << "!-
";}break;
      case 7:{ yyval = yyvsp[-2] + yyvsp[0];cout
<<"+" ;}
     break;
     case 8:{ yyval = yyvsp[-2] - yyvsp[0];cout <<
"-";}
     break;
     case 9:{ yyval = yyvsp[-2] * yyvsp[0];cout <<
"*";}
     break;
     case 10:{ yyval = yyvsp[-2] /yyvsp[0];cout <<
"/";}
    break;
    case 11:{ yyval = yyvsp[-1]; } break;
     }
     yyssp -= yym;
     yystate = *yyssp;
     yyvsp -= yym;
     yym = yylhs[yyn];
```



```
      if (yystate == 0 && yym == 0)
      {
       #ifdef YYDEBUG
         if (yydebug)
          printf("yydebug: after reduction,
shifting
          from state 0 to\ state %d\n", YYFINAL);
       #endif
       yystate = YYFINAL;
       *++yyssp = YYFINAL;
       *++yyvsp = yyval;
       if (yychar < 0)
       { if ((yychar = yylex()) < 0) yychar = 0;
         #if YYDEBUG
          if (yydebug)
           {yys = 0;
            if (yychar <= YYMAXTOKEN)
              yys =yyname[yychar];
            if (!yys) yys = "illegal-symbol";
            cout<<"yydebug: holat" >> YYFINAL >>
            ", o'qildi " << yys << yychar<< "\n";
            }
         #endif
       }
         if (yychar == 0) goto yyaccept;
         goto yyloop;
      }
      if ((yyn = yygindex[yym])&&(yyn +=
yystate)>=0 &&
          yyn<= YYTABLESIZE && yycheck[yyn] ==
yystate)
          yystate = yytable[yyn];
      else yystate = yydgoto[yym];
      #ifdef YYDEBUG
        if (yydebug)
          printf("yydebug: after reduction,
```

```
        shifting from state %d \to state %d\n",
        *yyssp, yystate);
  #endif
  if (yyssp >= yyss + yystacksize - 1)
    { goto yyoverflow; }
  *++yyssp = yystate;
  *++yyvsp = yyval;
  goto yyloop;
yyoverflow:
    cerr<< "yacc stack overflow\n";
yyabort:
     cerr<< "yacc dasturda hato\n";
     return (1);
yyaccept:
    return (0);
}
```

Bu dastur sodda qkalkulyatorni amalga oshiradi. Kiruvchi sifatida arifmetik ifodani berish mumkin. Uning ishlashi quyidagi rasmda ko‘rsatilgan :



Xulosa

YAAC - bu LALR-tahlilchi asosida ishlaydigan sintaktik tahlilchini dasturini Si tilida yaratib beradigan dastur vositasidir. Bu dastur ko‘p kompilatorlarni yaratishda qo‘llangan.

Boshida Yacc tilida masalan paskal tili uchun spetsifikatsiyalar(sintaktik qoidalar) pascale.u nomidagi dastlabki faylga yoziladi.

So‘ng bu faylni Yacc generator yacc -dltv -b pascal pascal.u buyruq bilan qayta ishlab Ci tilida sintaktik tahlilchini dasturini yaratib beradi va pascal.c faylga yozadi. Bu faylga foydalanuvchi yozgan funksiyalar ta’riflari qo‘shilib mukammal kompilyator tuziladi.

Shuning bilan birga qo‘shimcha fayl yaratadi – pascal.out. Bu faylda grammatik qoidalar va holatlar (vaziyatlar) ro‘yxati chiqadi.

pascal.c dasturni Si kompilyatordan o‘tkazib pascal.exe nomli ijrovchi dastur yaratiladi. Bu dasturga kiruvchi sifatida Paskal tilida yozilgan matn uzatiladi chiquvchi sifatida obyekt kod xosil bo‘ladi.

Nazorat uchun savollar

1. Yaac dasturning vazifasi nimadan iborat?
2. Yaac dasturi nima asosida ishlaydi?
3. Yaac dasturning tuzimini keltiring.
4. Kiruvchi tilning grammatikasi nima orqali beriladi?

Amaliyot uchun topshiriqlar

1. M-tilining sintaktik taxlilchini Yaac dastur orqali yarating.
2. Paskal tilni sintaktik taxlilchini Yaac dastur orqali yarating.
3. Mantiq ifodani hisoblaydigan kalkulyatorni Yaac dasturini yozing.
4. Arifmetik ifodani postfiks yozuviga o‘tkazadigan Yaac dasturini yozing.

GLOSSARIY

Absolyut manzildagi kod –fiksirlangan manzilga yuklanuvchi, o'zgaruvchilari esa fiksirlangan manzillarda joylashuvchi kod.

Abstrakt mashina psevdokod tili – Psevdokod yozilgan til, bu tilda xotiraga chegara qo'yilmaydi, va kompyuter arxitekturasi hisobga olmaydi. Bu til dastlabki dasturni ichki ko'rinishini shakllantirish uchun ishlatiladi.

Avtomatning boshqaruv qurilmasi – bu qurilma holatlar to'plamidan iborat bo'lib, har bir taktda joriy holatni o'zgartiradi (yoki o'zgartirmaydi) va kirituvchini bitta belgiga o'ngga suradi.

Avtomatning boshlang'ich holati – kiruvchi tasmadan hali belgilar o'qilmagan holdagi avtomat vaziyati, boshlang'ich holat H harf bilan belgilanadi.

Avtomatning kirituvchisi – kiruvchi tasmadan joriy belgini o'quvchi mantiqiy qurilma.

Avtomatning kiruvchi tasmasi – dastlabki zanjir ketma-ket joylashgan mantiqiy qurilma, shunda birinchi belgi tasmaning boshida joylashadi.

Avtomatning holatlari – joriy belgiga nisbatan har bir taktda o'tadigan avtomatning vaziyati, holatlar katta lotin xarflar bilan belgilanadi, holatlar to'plami Q bilan belgilanadi.

Avtomatni chekli holatlari – kiruvchi tasmadan dastlabki zanjir to'liq o'qilgan holdagi avtomat vaziyati, chekli holatlar bir nechta bo'lishi mumkin, chekli holat F harf bilan belgilanadi.

Amallar bilan kengaytirilgan grammatika – grammatik qoidalarga qo'shimcha amallar kiritilgan holat.

"Ahlat jamlovchi" mexanizm –vazifasi xotirada bo'shatilgan sohalarni qidirib, ularni qayta ishlatishga tayyorlash.

Bajaruvchi dastur – bajarishga tayyor bo'lib turgan mashina kodidagi dastur. Bunday dastur operatsion tizim orqali hotiraga yuklanib bajariladi.

Bevosita keltirib chiqarish – grammatikada $\omega \to \gamma$ qoida mavjud bo'lsa, $\alpha=\delta_1\omega\delta_2$ zanjirdan bevosita $\beta=\delta_1\gamma\delta_2$ zanjir keltirib chiqarish usuli. Bunda $\omega \in V^*$, $\delta_1,\gamma,\delta_2 \in V^*$, ya'ni δ_1 ,γ ,δ_2 zanjirlar bo'sh bo'lishi mumkin. Berilgan zanjirda ko'rsatilgan qoidaga binoan ω zanjir qismi o'rniga γ zanjir

qo'yiladi. Bevosita keltirib chiqarish $\alpha \Rightarrow \beta$ ko'rinishda yoziladi.

Belgilar zanjiri – ketma-ket yozilgan ihtiyoriy belgilar ketma-ketligi.

Bir ma'nolik grammatika – har qanday zanjir uchun faqat yagona shakldagi keltirib chiqarish daraxti qurilishi.

Bir marotaba o'tishli kompilyator – barcha fazalar birdaniga ishlaydigan, leksik tahlilchi matndan joriy leksemani ajratib, sintaktik tahlilchiga uzatadigan, sintaktik tahlilchi esa leksemani tahlil qilib, kerak bo'lsa, leksik tahlilchidan yana leksema so'raydigan va natijada dasturni bir fragmenti uchun ichki tasvirni taxlaydigan va semantik tahlilchiga uzatadigan vaziyat.

Boshlangich belgi – birinchi qoidani chap tarafida turgan noterminal belgi. Aynan shu belgidan tahlil boshlanadi, yoki shu belgi bilan tahlil tugatiladi.

Bo'sh zanjir – birorta belgidan iborat bo'lmagan zanjir. λ yoki ε xarflari bilan belgilanadi.

Bo'sh qoidalar – $A \to \varepsilon$ shaklidagi qoidalar, bunda $A \in VN$ noterminal belgi.

Bekus-Naur shakli (BNSH) – bir nechta qoidalarning chap tarafi bir xil, ya'ni $\alpha \to \beta_1$, $\alpha \to \beta_2$, ... ,$\alpha \to \beta_n$ bo'lsa, ularni bitta qoida $\alpha \to \beta_1|\beta_2|...|\beta_n$ shaklida yozish va quyidagi metabelgilarning ishlatish mumkin: "→" yoki "::=". Bular qoidani chap tarafini o'ng tarafidan ajratish uchun ishlatiladi. Noterminallar ixtiyoriy satr ko'rinishda yozilib "<" va ">" burchak qavslar ichiga olinadi va muqobil qoidalar " | " belgisi bilan ajratiladi.

Dastur – biror bir alohida masalani yechish uchun, yagona muallif tomonidan ma'lum operatsion muhitda, biror bir dasturlash tilida yaratilgan kod. Dastur muallifdan ajralmasdir, uning hajmi katta emas.

Dastur mahsuloti – bir nechta operatsion muhitlarda muallifsiz ishlashi mumkin o'rta hajmli dastur. Dastur mahsulotini tekshirishda, o'zgartirishda va bajarishda muallifni ishtiroki kerak emas (u muallifdan ajratilgan). Dastur mahsulotidan foydalanish va unga o'zgartirishlar kiritish uchun bir nechta xujjatlar yaratiladi. Dastur mahsuloti sozlanuvchi bo'lishi kerak (setup.exe) va uni sozlashda, biror bir sozlash parametrlar qiymati berilishi shart.

Dasturlash tizimlari – dastur maxsulotini o'zining butun hayoti davrida qo'llab turadigan dastur vositalar majmuasi.

Dastur matnni yozish ("kodlash") – matnni yozish dasturlash yoki kodlash deb nomlanadi. Kodlash natijasida biror bir dasturlash tilida dastur matni hosil bo'ladi. Dastur matini yozish uchun, qo'yilgan masalaga, talablarga va dekompozitsiya usullariga ko'ra, til tanlanadi.

Dastur majmuasini yig'ish – dasturni alohida qismlari bir-biriga ulash va bitta katta tizimli dasturiy ta'minotga keltirish.

Dastur fazalari – dastur ustidan ishlash uning butun hayot davrida davom etadi. Dastur xhayoti uch fazadan iborat: yaratish, qo'llanish va kuzatish.

Dasturni ichki tasviri – sintaktik tahlilchini ishlash natijasidagi dastlabki leksemalar ketma-ketligining biror bir ichki tuzimga o'tkazilgan matni. Ichki tasvirdagi matn sintaktik tuzimni aks ettiradi.

Dasturni optimallashtirish – dasturni samarali obyekt kodini hosil qilish uchun dasturni qayta tartiblash va ammalarni o'zgartirish kabi qayta ishlovlar.

Dinamik kutubxona – biror bir dastur bajarish vaqtida unga ulanadigan funksiyalar va protseduralarni saqlaydigan joy. Dinamik kutubxona komponentalari unga murojaat qilgan dasturlar bilan bog'liq emas va ular alohida tarqatiladi.

FIRST funksiyasi – α zanjirdan chap keltirib chiqarilgan zanjirlarni bosh terminal belgilarning to'plami FIRST(α) funksiyani qiymati bo'ladi,. Agar α zanjirdan bo'sh zanjir ε keltirib chiqarilsa, unda ε FIRST(α) funksiyani qiymatlar to'plamiga kiradi.

FOLLOW funksiyasi – biror bir sentensial shaklda uchragan A noterminal belgidan keyin turgan terminal, belgilarning to'plami FOLLOW(A) funksiyaning qiymati bo'ladi.

***f(s_i,a_j)* amal funksiyasi**– bunda s_i joriy holat, a_j joriy kiruvchi belgi. Funksiya qiymati to'rt hil bo'lishi mumkin:

ks– ko'chirish amali, bunda "k" ko'chirish amalni belgilovchi harf, s- biror bir holat;

o'i– o'rash amali, bunda "o'" o'rash amalni belgilovchi harf, *i*-qoida

raqami. Agar $A \to \beta$ *i*-chi qoida bo'lsa, unda β zanjir A noterminalga o'raladi

qq– qabul qilishni belgilaydi,

hv– hato vaziyatni belgilaydi.

Global hotira – dasturni bajarish boshlanganda, bir marta ajratiladigan va dastur ishini tugatguncha saqlanadigan hotira.

Grammatika sinflari – formal grammatika qoidalarini tuzilishiga qarab bo'linishi. Agar grammatikadagi barcha qoidalar biror kelishilgan qolipga tushsa, u holda bu grammatika qolipga mos turga kiradi. Noam Xomskiy bo'yicha grammatikalar to'rtta turga ajratilgan: erkin grammatika, kontekstga bog'liq grammatika yoki qisqarmas grammatika,. Kontekstdan erkin grammatika va muntazam grammatika.

Grammatikani Xomskiy shakli – bunda har bir qoida alohida satrda yoziladi,va noterminal belgilar indeksli A xarf bilan belgilanadi.

Greybax normal shakli – qoidalar faqat $A \to t\alpha$ yoki $S \to \varepsilon$ shaklda bo'lgan grammatika, bunda $A, S \in VN$ noterminal belgilar, $t \in VT$ terminal belgi,va $\alpha \in VN^*$ noterminal belgilardan iborat bo'lgan zanjir.

$g(s_i,X_k)$ o'tish funksiyasi – bu funksiyada s_i joriy holat belgisi, X_j grammatikani ihtiyoriy belgisi, funksiyaning qiymati esa o'tiladigan keyingi holat belgisi bo'ladi.

goto(I,X) funksiyasi – bu funksiyada I – vaziyatlar to'plami, X – grammatika belgisi bo'lib, funksiya qiymati I to'plam asosida, X belgi uchun yangi vaziyatlar to'plami.

Holatlar diagrammasi – o'tish funksiyasining graf shaklida berilishi. Holatlar diagrammasining tugunlarida holat belgilari turadi,va yoylari terminallar bilan belgilanadi. Boshlang'ich va chekli holatlar ikki marta aylanaga olinadi.

Identifikatorlar jadvali – dasturda uchragan o'zgaruvchilar, o'zgarmaslar, funksiyalar va boshqa elementlar xossalarini saqlaydigan jadval. Identifikatorlar jadvalida satrlarda identifikatorlar ro'yxati turadi va ustunlarda ularni tafsiflari yoziladi.

Integranlangan dastur mahsuloti – bu dastur vositalar majmuasidir (paket). Integrallangan dastur muhit sifatida Microsoft Office paketini

keltirish mumkin. Bu paket kelishilgan interfeysli, o'ntaga yaqin dastur maxsulotlardan iborat. Paketning har bir komponentasi o'ziga xos bo'lgan maxsus vositasiga qaramasdan, parametrlarni berish, ishlash tartibotlari va foydalanuvchi amallari barcha komponentalarda bir xil va o'xshashdir.

Interpretator – dastlabki tilda yozilgan kiruvchi dasturni o'zlashtirib uning buyruqlarini ketma–ket bajaruvchi dasturdir. Translyatordan tamoyil bo'yicha farqi shundaki, u natijaviy dastur yaratmaydi. Interpretator dastlabki dastur matnni tahlil qiladi va kodni ma'nosiga qarab uni bajaradi. Agar joriy buyruqda hato bo'lsa, xabar berib ishni to'xtatadi.

Infiks yozuv – tabiiy ko'rinishdagi yozuv, bu yozuv faqat dastlabki dasturlarda ishlatiladi, bu usul odamlar uchun qulaydir, lekin kompilyator uchun eng noqulaydir. Shu sababli bu usul ichki ko'rinishda ishlatilmaydi.

Joriy qilish (qo'llash) bosqichi – yaratilgan dastur mahsulotini qo'llash uchun, buyurtmachini taklif qilish vazifasi. Bu bosqichda ko'p tashkiliy muommolar yechiladi, shu jumladan, foydalanuvchilarni o'qitish, anikq tashkilotda va aniq berilganlar bilan dastur ishlashi hamda barqarorligini tekshirish bajariladi.

Jamlovchi – kompilyator tomondan hosil bo'lgan obyekt fayllarni va kutubxonadagi fayllarni bir-biriga bog'lab, natijaviy bajaruvchi dasturni hosil qiluvchi dastur vositasidir. Obyekt fayl yoki fayllardir, toki ularda ishlatilgan modullar unga bog'lanmaguncha alohida ishlamaydi. Jamlovchi barcha modullarni ulab, yagona bajaruvchi fayl hosil qiladi.

Keltirib chiqarish daraxti – keltirib chiqarishni quyidagicha daraxt ko'rinishdagi grafik tasviri:

1) har bir cho'qqida ($VN \cup VT \cup \varepsilon$) to'plam belgisi turishi kerak;

2) agar daraxt tugunida A noterminal belgi turgan bo'lsa va uni avlodlari $X_1X_2...X_n, X_i \in (VN \cup VT \cup \varepsilon)$ belgilardan iborat bo'lsa, u holda $A \to X_1X_2...X_n$ qoida grammatikada mavjud bo'ladi;

3) daraxtni ildizida boshlang'ich noterminal belgi turadi;

4) daraxtni barglarida terminal yoki bo'sh zanjir belgisi turadi.

Keltirilgan grammatika – erishib bo'lmas belgilar, qatnashmaydigan belgilar va bo'sh qoidalari bo'lmagan kontekstdan erkin grammatika. Bunday grammatika – kanonik grammatika deb ham nomlanadi.

Kengaytirilgan Bekus-Naur Shakli (KBNSH) – quyidagi metabelgilar bilan kengaytirilgan Bekus-Naur shakli : "(" va ")" qavs ichida turgan zanjirlar ro'yxatidan faqat bittasi shu o'rinda turishi mumkin;"|" va "|" qavs ichida turgan zanjir bo'lmasligi ham mumkin;"{" va "}" qavs ichidagi zanjir bo'lmasligi, yoki bir marta bo'lishi, yoki bir necha marta bo'lishi mumkin; "," vergul belgisi ro'yxatdagi zanjirlarni ajratish uchun ishlatiladi; "" qo'shtirnoq belgisi ichidagi metabelgini terminal sifatida yozish uchun ishlatiladi.

Kengaytirilgan grammatika – agar G(VT,VN,P,S) kontekstdan erkin grammatika bo'lsa, unda Gning kengaytirilgan grammatikasi deb, G'(VT',VN',P',S') aytiladi va bunda VT'=VT; VN'=VN∪ {S'}; S'∉VN; P'=P ∪ { S'→S} bo'ladi.

Ketma-ket leksik tahlil – ishni leksik tahlilchi boshlaydi va dastlabki dasturni leksemalarga ajratib leksema jadvaliga yozib qo'yadi,. So'ng sintaktik taxlilchi ishlaydi, u leksemalarni leksema jadvalidan oladi.

Kod generatsiyasi – obyekt kodni hosil qiluvchi dastur vositasi. Kod generatoriga kiruvchi sifatida boshlang'ich fazalarda hosil bo'lgan dastlabki dasturni ichki tasviri uzatiladi va chiquvchi sifatida ekvivalent chiquvchi obyekt dastur kodi yaratiladi.

Kolliziya – ikki yoki undan ko'p identifikatorlarni xesh- funksiyasi bir xil qiymatga ega bo'lgan holat.

Kompilyator – kiruvchi tildagi dastlabki dasturni mashina tildagi yoki assembler tildagi ekvivalent obyekt dasturga tarjima qiluvchi translyator.

Kompilyator bosqichlari – kompilyator ikki bosqichdan tashkil topadi: analiz va sintez. Analiz bosqichida, u kiruvchi tilni aniqlovchisi, ya'ni kiruvchi tilda yozilgan belgilar zanjirini qabul qilib, bu zanjir tilga mos kelishini aniqlaydi va undan tashqari tilni qaysi qoidalari asosida bu zanjir tuzilganligini topib beradi. Zanjir foydalanuvchi tomondan yaratiladi. Sintez bosqichida kompilyator natijaviy dasturni yaratish kerak, u mashina tilini ma'lum qoidalari bo'yicha chiquvchi tilda chiquvchi zanjir tuzib beradi. Undan tashqari kompilyatorda xatolarni tahlil qilish va to'g'rilash qismi ham bor, bu qism xato to'g'risida batafsil ma'lumot berishga xarakat qiladi va iloji bo'lsa to'g'ri variantlarni taklif qiladi.

Konkatenatsiya amali – α va β zanjirlarni konkatenatsiyasi αβ ko'rinishda yoziladi va natijaviy zanjirda β zanjirning belgilari αni belgilaridan keyin yoziladi.

Kontekst shartlar – ko'p tillarda kontekst shartlariga quyidagi semantik cheklovlar kiradi: dasturda ishlatilgan har bir ism ta'riflanishi lozim va bu faqat bir marta bo'lishi kerak; qiymat berish operatorda chap qismidagi o'zgaruvchini va o'ng qismidagi ifodani turlari bir hil bo'lishi kerak (yoki bir sinf turiga kirishi kerak); shartli va takrorlash operatorlarning sharti sifatida faqat mantiqiy ifoda bo'lishi mumkin; taqqoslash amallar operandlari butun bo'lishi kerak; ifoda operandlar turlari mos bo'lishi kerak; har bir nishon faqat bitta joyda bo'lishi kerak; blok yoki funksiya ichida har bir identifikator bir marta ta'riflanishi kerak; funksiyani chaqirishda faktik parametrlar soni va turi formal parametrlar soniga va turiga mos kelishi kerak.

Kontekstga bog'liq grammatika – qoidalari quyidagicha $\alpha_1 A\alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, bunda $\alpha_1,\alpha_2 \in V^*$, $A \in VN$, $\beta \in V^+$ bo'lgan grammatika. Bu turdagi grammatikani tuzilishi shundayki, noterminal belgisi jumlada uchragan kontekstga qarab, har xil zanjir bilan almashtirilishi mumkin. Bunda - α_1- chap qontekst; α_2- o'ng kontekst deb nomlanadi.

Kontekstdan erkin grammatika – qoidalari $A \rightarrow \beta$ shaklda bo'lgan grammatika, bunda $A \in VN$, $\beta \in V^*$. Bu grammatika dasturlash tillarini sintaktik konstruksiyalarini ta'riflashda keng ishlatiladi.

Ko'p ma'nolik grammatika – biror bir zanjir uchun ikkita yoki undan ortiq har xil shakldagi keltirib chiqarish daraxtlari mavjud bo'lgan grammatika.

Ko'p marotaba o'tishli kompilyator – ko'p marotaba o'tishli kompilyatorlarda har bir faza alohida ishlab, o'z natijasini tashqi xotirada saqlab qoladi. Keyingi faza oldingi faza natijasini qayta ishlab, yana faylga yozib qo'yadi va h.q., obyekt faylini yaratguncha. Ko'pincha uch marotaba o'tishli ishlatiladi. Unda birinchi o'tishda leksik tahlilchi, ikkinchi o'tishda sintaktik va semantik tahlilchilar hamda uchinchi o'tishda kod generatsiya va optimizatsiya fazalari ishlaydi. Kiruvchi til qancha murakkab bo'lsa, shuncha o'tishlar soni ko'payadi. Ko'p marotabali o'tish usulida leksik va sintaktik tahlilchilar ketma-ket ishlaydilar.

Ko'tariluvchi o'ng tahlil usuli – berilgan zanjir uchun teskari o'ng keltirib chiqarish bajarish, ya'ni ishlatilgan grammatik qoidalar pastdan yuqoriga qarab aniqlash usuli.

Ko'tariluvchi qaytish bilan ishlovchi aniqlovchining algoritmi – kengaytirilgan nodeterminantli magazin hotirali qayta o'zgartirgich asosida ishlovchi va o'ng tahlilni hosil qiluvchi algoritm.

Ko'chirish-o'rash to'qnashuvi – magazindagi zanjir va kiruvchi belgi bo'yicha, nima qilish kerakligi (ko'chirishmi yoki o'rashmi) noma'lum bo'lgan holat.

closure(I) funksiyasi – bu funksiyada I – vaziyatlar to'plami bo'lib, funksiya qiymati I to'plamdan ma'lum algoritmik usul yordamida kelib chiqgan vaziyatlardan iborat bo'lgan to'plam.

Leksema – alifbo belgilaridan tuzilgan sodda bo'linmas konstruksiya. Tabiiy tillarda bu orfografik lug'atda keltirilgan so'zlardir. Dasturlash tillarida leksema kalit so'zlar, identifikatorlar, o'zgarmaslar, amallar va ajratuvchilardan iborat.

Leksemalar jadvali – dastlabki dasturdan ajratilgan leksemalarni jadval ko'rinishda saqlash. Jadvalda leksemani ko'rinishidan tashqari uning turi va maxsus kodi saqlanadi.

Leksik tahlil – dastlabki dasturni o'qib, uni leksemalarga, ya'ni bo'linmas alohida so'zlarga ajratib berish.

Livermor "Sikllar" – bu Livermor milliy laboratoriyasida yaratilgan 24 maxsus ilmiy dasturlar. Bunga vektorlar va matritsalar ustidan bajariladigan matematik amallar, murakkab qidirish algoritmlari va boshqalar karadi.

Lex dasturi – bu dastur muntazam ifodalar asosida berilgan grammatika uchun C++ tilida taxlilchi dastur tuzadi. ***Lex*** dasturi chekli avtomat va holatlar diagrammasi asosida ishlaydi.

LL(1) grammatika – ikkita har hil qoidalari $A \rightarrow \beta$ va $A \rightarrow \gamma$ uchun barcha sentensial shakllar $wA\alpha$, ya'ni $S \Rightarrow^* wA\alpha$ uchun $FIRST(\beta\alpha) \cap FIRST(\gamma\alpha) = \varnothing$ bajarilgan kontekstdan erkin grammatika

LL(k) grammatikalar – kontekstdan erkin grammatika sinfni bir qismi bo'lib, unda muqobil qoidalarni tanlashda, joriy belgini o'ng tarafida turgan k belgilar ketma-ketligini hisobga olinadi.

LR(k) grammatika– kontekstdan erkin grammatikada kiruvchi zanjir chapdan o'ngga ko'rib chiqiladi va o'ng tahlil bajariladi. Surish yoki o'rashni tanlashda, joriy belgini o'ng tarafida turgan k-ta belgilar ketma-ketligi hisobga olinadi.

LR(0) grammatika –kiruvchi zanjirni joriy belgisi hisobga olinmaydi, faqat magazindagi zanjirlar tahlil qilinadi va ularni tuzilishiga qarab amallar bajariladi. Agar grammatika uchun LR(0)-tahlilchi qurilib,va uning boshqarish jadvalidagi har bir katagidagi qiymat bittadan ortiq bo'lmasa, bunday grammatika LR(0)-grammatika bo'ladi.

LR(k) vaziyatlar – kontekstdan erkin grammatikada $[A\to\alpha\cdot\beta,\ u]$ nomlanadi, bunda $A\to\alpha\beta\in P$, $u\in VT^*$, $|u|\le k$, $\alpha,\beta\in(VT\cup VN)^*$, $A\in VN$.

LR(0) vaziyatlar – kontekstdan erkin grammatikada $[A\to\alpha\cdot\beta]$ nomlanadi bunda $A\to\alpha\beta\in P$, $\alpha,\beta\in(VT\cup VN)^*$, $A\in VN$.

Lokal xotira – qism-dasturi (blok, funksiya, protsedura) ishni boshlaganda ajratilgan va tugatganda bo'shatilgan xotira. Lokal xotirani qism-dasturidan tashqarida ishlatib bo'lmaydi.

Leksik tahlilning baravariga bajarilishi - ishni sintaktik tahlilchi boshlaydi va konstruksiyalarni tekshirish jarayonida leksik tahlilchini chaqirib, joriy leksemani talab qiladi. Kutilgan leksema qoidalarga rioya qilsa, uni jadvallarga yozadi va keyingi leksemaga o'tadi. Sintaktik va leksik taxlilchilar parallel ravishda ishlaydi.

Magazin hotirali avtomat – berilgan zanjirning kontekstdan erkin grammatika qoidalariga qanoatlanishini aniqlovchi mantiqiy qurilma. Magazin o'ngdan chapga qarab to'ldiriladi.

Magazin hotirali determinantlangan avtomat – har qanday konfiguratsiyasidan o'zidan faqat bitta keyingi konfiguratsiyasiga o'tish mumkin bo'lgan magazin hotirali avtomat.

Magazin hotirali kengaytirilgan avtomat – magazin ustidagi chekli uzunlikdagi zanjirni, boshqa zanjir bilan almashtirish imkoniyatiga ega bo'lgan avtomat. Magazin chapdan o'ngga qarab to'ldiriladi.

Magazin hotirali nodeterminantlangan avtomat – har qanday konfiguratsiyasidan o'zidan bir nechta keyingi konfiguratsiyasiga o'tish imkoniyatga ega bo'lgan magazin hotirali avtomat.

Magazin xotirali qayta o'zgartirgich – kiruvchi zanjirdan chiquvchi zanjirga tarjima qiladigan magazin hotirali avtomat asosida tuzilgan mantiqiy qurilma.

Matn tahrirchi – yuqori darajadagi tilda dastur kodini yaratish, o'zgartirish va qayta ishlash imkoniyatini beradigan dastur.

Mashinaga bog'lanmagan optimallashtirish – bu chiquvchi tilni yoki mashinani hisobga olmagan holda, dastlabki dasturni (ichki ko'rinish shaklida ham mumkin) qayta o'zgartirish.

Mashinaga bog'liq optimallashtirish – qayta o'zgartirishlar kompilyatorni chiquvchi tilidagi dasturda qilinadi. Bu optimallashtirish obyekt dasturni generatsiya qilish jarayoni bilan birvarakayiga bajariladi.

Metabelgi – grammatika qoidalarni ta'riflashda qo'llanadigan belgi.

Metatil – grammatika qoidalarni ta'riflash tili.

Minimallashtirilgan chekli avtomat – holatlar soni eng kam bo'lgan chekli avtomat.

Muntazam grammatikaning kengayish lemmasi – Agar muntazam tilga kiruvchi biror bir uzun zanjirda, uning qandaydir qismi ixtiyoriy marta qaytarilsa, hosil bo'lgan zanjir (kengaytirilgan) muntazam tilga kiradi.

Muntazam ifodalar – muntazam ifodalar grammatikani berish bir turi, ifoda yig'im(+), konkantenatsiya, iteratsiya (*) amallardan, terminal belgilardan va boshqa qo'shimcha belgilardan iborat. Ifodani yozishda qavslarni ishlatish mumkin, shunda qavs ichidagi ifoda birinchi navbatda hisoblanadi.

Munosabatlar – oldin keluvchi munosabatlar uchta holatda bo'lishi mumkin.

Birinchi holatda munosabat $X_1 \gtrdot X_2$ ko'rinishda yoziladi. Bu holatda X_2 negizdan keyin keluchi belgi hisoblanadi va X_1 mos qoidaning o'ng tarafidagi ohirgi belgi bo'ladi. Ikkinchi holatda munosabat $X_1 \doteq X_2$ ko'rinishda yoziladi. Bu holatda X_1 va X_2 negizni tashkil qiluvchi belgilar deb hisoblanadi, ya'ni ular baravariga biror bir qoidani o'ng tarafiga kiradi.Uchinchi holatda munosabat $X_1 \lessdot X_2$ ko'rinishda yoziladi va bu holatda X_1 negizdan oldin keluchi deb hisoblanadi, X_2 mos qoidani o'ng tarafdagi zanjirni birinchi belgisi bo'ladi.

Munosabatlar matritsasi – X_i va X_j belgilar orasidagi munosabatlar matritsasi, bunda X_i va X_j grammatikani barcha belgilari.

Nisbiy manzildagi kod – bu tartibotda barcha modullar kodi noldan boshlanadi. Nisbiy manzildagi kodni afzalligi shundaki, har bir qism-dasturini (modulni) alohida kompilyatsiya qilish mumkin. Maxsus jamlovchi dastur orqali ularni birlashtirib natijaviy dasturni hosil qilishi mumkin. Bu tartibot modullarni yig'ish uchun qo'shimcha xarajatlarni talab qiladi, lekin bog'liqmas kompilyatsiyani afzalligi bu xarajatlarni qoplaydi.

Negiz – biror bir qoidani o'ng tarafiga mos kelgan, sentensial shaklni biror bir qismi.

Notayanch vaziyatlar – nuqtasi boshida bo'lgan vaziyatlar.

Noterminal belgi – tilni konstruksiyalarini ta'riflash uchun ishlatiladigan belgi. Bu belgilar tilni o'ziga kirmaydi, ular faqat qoidalarda ishlatiladi. Noterminal belgilar qoidani ham chap, ham o'ng tarafida uchrashishi mumkin, lekin ular kamida bir marta biror qoidani chap tarafida uchrashi kerak. Grammatikani boshlang'ich belgisi doim noterminal belgi bo'ladi.

Obyekt modullar – qism-dasturlarini kompilyatsiya jarayonida qayta ishlab chiqarish natijasida chiqgan obyekt dastur.

Obyekt Dastur – kod generatoridan chiqgan dastur. Uning shakli absolyut manzillardagi moshina kodi, nisbiy manzillardagi mashina kodi, assembler tilidagi yoki boshqa yuqori bosqich tildagi kod bo'lishi mumkin.

Oddiy keluvchi grammatika (OKG) – quyidagi shartlarni qonoatlantiruvchi kontekstdan erkin keltirilgan grammatika. Har biri tartiblangan X_i va X_j belgilar orasida uchta munosabatdan faqat bittasi bajarilishi yoki umuman munosabat bo'lmasligi mumkin: agar grammatikada $A\to\alpha X_i X_j\beta$ qoida mavjud bo'lib, $A\in VN$, $\alpha,\beta\in V^*$ bo'lsa, bu holda $X_i \doteq X_j$ $(\forall X_i, X_j\in V)$ bo'ladi, ya'ni ikkala belgi negizga kiradi; agar grammatikada $A\to\alpha X_i B\beta$ qoida mavjud bo'lib va $B\Rightarrow^{+} X_j\gamma$ kelib chiqib, $A,B\in VN$, $\alpha,\beta,\gamma\in V^*$ bo'lsa, bu holda $X_i \lessdot X_j$ $(\forall X_i, X_j\in V)$ bo'ladi, ya'ni X_j negizni birinchi belgisi bo'lib, X_i negizdan oldin keluvchi belgi; agar grammatikada $A\to\alpha CX_j\beta$ qoida mavjud bo'lib (va) $C\Rightarrow^{*}\gamma X_i$ kelib chiqsa yoki $A\to\alpha CD\beta$ qoida mavjud bo'lib va $C\Rightarrow^{*}\gamma X_i$. $D\Rightarrow^{*} X_j\delta$ kelib

chiqsa, ya'ni X_i negizni ohirgi belgisi bo'lib, X_j keyingi negizni birinchi belgisi bo'lsa, unda $X_i \gtrdot X_j$ $(\forall X_i, X_j\in V)$ bo'ladi,. Ya'ni X_i negizni ohirgi belgisi bo'lib, X_j negizdan keyin keluvchi birinchi belgisi,va bu yerda $A,C,D\in VN$, $\alpha,\beta,\gamma,\delta\in V^*$. Grammatikaning turli hil qoidalaridagi o'ng taraflari turlicha bo'lishi kerak, ya'ni bir hildagi o'ng tarafli qoidalar bo'lmasligi kerak.

Operatorli oldin keluvchi grammatika – quyidagi shartlarni qonaatlantiruvchi kontekstdan erkin keltirilgan grammatika.Har bir tartiblangan a va b terminal belgilar orasida uchta munosabatdan faqat bittasi bajarilishi yoki umuman munosabat bo'lmasligi mumkin: agar grammatikada $A\to\alpha ab\beta$ yoki $A\to\alpha aSb\beta$ qoidalar mavjud bo'lib, bu yerda $A,S\in VN$, $\alpha,\beta\in V^*$ bo'lsa,va bu holda $a \doteq b$ $(\forall a,b\in VT)$ bo'ladi, ya'ni ikkala belgi negizga kiradi; agar grammatikada $A\to\alpha aC\beta$ qoida mavjud bo'lib, $C\Rightarrow^{+} b\gamma$ yoki $C\Rightarrow^{+} Db\gamma$ kelib chiqib, bu yerda $A,C,D\in VN$, $\alpha,\beta,\gamma\in V^*$ bo'lsa, bu holda $a \lessdot b$ $(\forall a,b\in VT)$; agar grammatikada $A\to\alpha Cb\beta$ qoida mavjud bo'ladi. $C\Rightarrow^{+}\gamma a$ yoki $C\Rightarrow^{+}\gamma aD$ kelib chiqib, bu yerda $A,C,D\in VN$, $\alpha,\beta,\gamma\in V^*$ bo'lsa, u holda $a \gtrdot b$ $(\forall a,b\in VT)$.Operatorli oldin keluvchi grammatikada $A\to\alpha BC\beta$ qoidalar bo'lishi mumkin emas, bunda $A,C,B\in VN$, $\alpha,\beta\in V^*$. Operatorli oldin keluvchi grammatika determinantlangan, lekin bir ma'nolik bo'lmasligi mumkin.

"Oq" quti – bu testlash usulida dastur kodi ochiq bo'lib, unda nazorat nuqtalari va shu nuqtalarda kutiladigan natijalar beriladi. Bunda faqat kiruvchi qiymatlar va kutilgan natijalar beriladi.

"Osilib qolgan else" muammosi – ***if expr then if expr then a else a*** shartli operatorida ***else a*** jumlasi qaysi ***if*** ga tegishlik muammosi.

Pasayuvchi chap tahlil usuli – berilgan zanjir uchun chap keltirib chiqarish bajarish, ya'ni ishlatilgan grammatik qoidalar yuqoridan pastga qarab aniqlash usuli.

Pastdan yuqoriga qarab daraxtni qurish – daraxtni barglardan boshlab yuqoriga qarab ildizigacha qurish.

Pastdan yuqoriga ko'tariluvchi tahlilchi – ko'tariluvchi o'ng tahlil usuli asosida tuzilgan mantiqiy qurilma.

Pasayuvchi qaytish bilan ishlovchi aniqlovchining algoritmi – magazin hotirali nodeterminant qayta o'zgartirgich asosida ishlovchi va chap tahlilni hosil qiluvchi algoritm.

POLIZ ichki ko'rinish tili – polyak teskari yozuvda (POLIZ) amallar operandlardan keyin yoziladi. Operandlar yozuvi, infiks yozuvdagi tartib ketma-ketligini o'zgartirmaydi.. Amallar yozuvi bajarilish ketma-ketligida yoziladi (chapdan o'nga qarab), va ular bevosita o'z operandlaridan keyin yoziladi,. Qavslar ishlatilmaydi, amallar ustuvorligi hisobga olinmaydi.

Postfiks yozuv – postfiks, ya'ni polyak teskari yozuvda (POLIZ) amallar operandlardan keyin yoziladi. Bu usul kompilyator uchun juda qulay.

Prefiqs yozuv – prefiks yozuvni to'g'ri polyak yozuvi deb nomlashadi. Bu yozuvda amallar o'z operandlari oldida yoziladi. Bu yozuvni noqulayligi shundaki, amallar ketma-ketligi bajarilish ketma-ketligidan farq qiladi.

Qism-dasturlar kutubhona – Qism-dasturlar kutubxona bu dasturlash tizimni bir qismi bo'lib unda standart qism-dasturlar, funksiyalar va komponentalar joylashadi. Qism-dasturlar kutubxonasi ikki qismdan tashkil topgan. Birinchi qismida, obyekt fayllar joylashgan bevosita dastur kodi, ikkinchisida esa, kutubxonaga tegishli funksiyalar prototipi, o'zgaruvchilar va o'zgarmaslarning tavsiflari joylashadi. Kutubxonadagi obyekt kodi, jamlovchi orqali bajaruvchi dasturga qo'shiladi. Kutubxonaning sarlavha fayli kompilyatorga kutubxona turkumi to'g'risida ma'lumot beradi.

Qatnashmaydigan belgilar – hech qanday sentensiya keltirib chiqarila olmaydigan noterminal belgilar. Agar noterminal belgi qatnashgan barcha qoidalarning chap va o'ng tarafida birdaniga qatnashsa, bunday belgi ham qatnashmaydigan belgi bo'ladi.

Qisqarmas grammatika – qoidalari quyidagicha $\alpha \to \beta$ shaklda bo'lgan grammatika, bunda α, $\beta \in V^+$ va $|\beta| \geq |\alpha|$. Qiskqarmas grammatikada jumlaning uzunligi kamaymaydi.

Qisqarmas kontekstdan erkin grammatika – qoidalari quyidagicha $A \to \beta$, bunda $A \in VN$, $\beta \in V^+$ bo'lgan grammatika. Istisno sifatida bu grammatikada $S \to \varepsilon$ qoida bo'lishi mumkin, faqat bu holda S qolgan qoidalarni o'ng tarafida uchramasligi kerak.

"Qobirg'a" grammatika – qoidalarida faqat boshlangich noterminal belgisi qatnashgan qayta o'zgartirilgan grammatika.

"Qora" quti – bu testlash usuli bo'lib, unda faqat kiruvchi qiymatlar va kutilgan natijalar beriladi.

Formal grammatika – tilni ta'riflovchi matematik tizimdir. Grammatika to'rtta obyekt bilan aniqlanadi va $G(VT,VN,P,S)$ kabi bilan belgilanadi, bunda :

VT – terminal belgilar chekli to'plami; VN – noterminal belgilar chekli to'plami shunda $VT \cap VN = \emptyset$; $V = VN \cup VT$ belgilar to'plami. G grammatikani to'liq alifboi bo'ladi; P – $\alpha \to \beta$ shaklidagi qoidalar to'plami,. Bunda $\alpha \in V^+$; $\beta \in V^*$, ya'ni α zanjir terminal va noterminal belgilardan iboratdir (bo'sh belgisidan tashqari); β zanjir terminal va noterminal belgilardan iboratdir hamda bo'sh zanjir ham bo'lishi mumkin.

Formal til – belgilar to'plami asosida, ma'lum qoidalar bo'yicha belgilarning o'zaro ulanishidir.

Formal til anglovchisi – biror bir mantiqiy qurilma – avtomat. Unga zanjir kiruvchi sifatida uzatiladi. Bu avtomat biror bir algoritm bo'yicha ishlab, natijada kiritilgan zanjir tilga tegishli bo'lishi yoki bo'lmasligini aniqlab beradi.

RAD Texnologiya – bu dastur ta'minotini tezkor yaratish texnologiyasi bo'lib, dastur yaratishda tayyor komponentalarda foydalanishga imkoniyatlar beradi. Bunday komponentalar har xil sohalarga tegishli bo'lib, har bir foydalanuvchi o'ziga kerakligini ishlatishi mumkin bo'ladi. Ushbu usul kam kuch bilan tez vaqtda dastur yaratishga imkon beradi.

Rekursiv qoidalar – noterminal belgi o'zini o'zi orqali ta'riflaydigan qoidalar.

Rekursiv tushish grammatika (RTG) – rekursiv tushish usulni qo'llash mumkin bo'lgan $LL(1)$ grammatikani sinf qismi.

Rekursiv tushish usuli – RTGni har bir noterminal belgisi uchun alohida protsedura tuziladi. Protseduralar nomiga noterminal nomi beriladi. Birinchi bo'lib boshlang'ich belgining protsedurasi chaqiriladi. Ma'lum noterminal belgi uchun tuzilgan protsedura joriy kiruvchi belgini tekshiradi. Agar u faqat biror bir $FIRST(\alpha_i)$ga kirsa, unda tahlil uchun $A \to$

α_i qoida tanlanadi. Agar hech qaysi $FIRST(\alpha_i)$ga kirmasa va qoidalar ichida $A \to \varepsilon$ mavjud bo'lsa, unda protsedura A ishini tamomlaydi. Agar qoidalar ichida $A \to \varepsilon$ bo'lmasa, protsedura hatolik to'g'risida habar beradi va zanjir qabul qilinmaydi,. Aniqlovchi ishini to'xtatadi. Agar qoidani chap tarafida noterminal belgilar uchrasa mos protseduralar chaqiriladi.

Resurs fayllar – bu fayllarda dastur tomonidan beriladigan xabarlar, grafik elementlarni joylanishi, ranglari, yozuvlari saklanadi.

s-grammatika – bo'sh qoidasiz bo'lgan va har bir noterminalni muqobil qoidalari turli terminallardan boshlangan KEG grammatika

q-grammatika – har bir noterminalni qoidalarni o'ng qismi turli terminallardan boshlangan yoki bo'sh, qayda bo'lgan va qoidalar bir biri bilan kesishmagan KEG grammatika.

Semantik kelishuvlar – bu quyidagi shartlarni bajarilishi kiradi: agar biror bir amaldagi operandlar turi mos kelishi. Boshqarishni faqat ruhsat berilgan joyga o'tkazish mumkinligi. Ma'lum holatlarda obyekt bir marta uchrashi lozimligi. Ba'zi bir hollarda bitta nom ikki yoki undan ko'p marta ishlatilishi kerakligi.

Semantik tahlil – berilgan dastlabki dasturning semantik nuqtayi nazardan to'g'ri yozilganligini aniqlash usuli.

Semantik kontekst shartlar – bunda quyidagi shartlar bajarilishi kerak: dasturda ishlatilgan har bir nom ta'riflanishi va bu faqat bir marta bo'lishi; qiymat berish operatorida chap qismidagi o'zgaruvchini va o'ng qismidagi ifodani turlari bir hil bo'lishi (yoki bir sinf turiga kirishi); shartli va takrorlash operatorlarning sharti sifatida faqat mantiqiy ifoda bo'lishi; taqqoslash amallar operandlari butun bo'lishi; ifoda operandlar turlari mos bo'lishi; har bir nishon faqat bitta joyda bo'lishi;

blok yoki funksiya ichida har bir identifikator bir marta ta'riflanishi; funksiyani chaqirishda faktik parametrlar soni va turi formal parametrlar soniga va turiga mos kelishi.

Sentensial shakl – grammatikada S boshlang'ich belgidan keltirib chiqarilgan $\alpha \in V^*$ zanjir, ya'ni $S \Rightarrow^* \alpha$.

Sintaktik tahlil – dastlabki dastur matni tilda berilgan sintaktik konstruksiyalarga to'g'ri kelishini aniqlovchi usul.

Sentensiya – faqat terminal belgilardan iborat bo'lgan sentensial shakl, ya'ni chekli sentensial shakl.

Sintaktik boshqaruv tarjima – sintaktik tahlil jarayonida kengaytirilgan grammatika asosida qo'shimcha amallarni bajarish.

Sintaktik daraxt – kiruvchi zanjir uchun grammatika qoidalari asosida qurilgan daraxt.

Sintaktik tahlil usuli – berilgan leksemalar zanjiri til grammatikasi bo'yicha keltirib chiqarish mumkinligini aniqlab keltirib chiqarish daraxtini qurivchi usul.

Sintez bosqichi –xotira va registrlarni taqsimlash, kod generatsiyasini bajarish va mashinaga bog'liq optimallashtirish fazaralardan iborat.

Sozlovchi – tadbiqiy dasturni ishlash jarayonini tekshiradigan va kuzatishni bajaradigan dastur vositasi. Sozlovchi quyidagi funksiyalarni bajaradi: ketma-ket, qadamma-qadam va natijaviy dasturni bajaradi. Dasturni nazorat nuqtagacha (to'xtash manzili) bajaradi va to'xtab turadi,. Dastur egallagan xotiradagi berilganlarni ko'radi va kerak bo'lsa o'zgartirishlar kiritadi.

Statik kutubhona – kutubxona tomonidan bajaruvchi dastur ichiga qo'shiladigan protsedura va funksiyalar majmuasi. Barcha statik kutubxonalar bajaruvchi dasturga faqat bir marta yuklovchi tomondan qo'shiladi. Dasturga qo'shilgan komponentalar uning ajralmas qismi bo'lib, kutubxonaga bog'liq bo'lmay qoladi.

"Stek" usuli –xotiradan stek uchun ma'lum joy tashkil qilinadi. Xotira zaxirasiga ehtiyoj bo'lganda stekdan joy ajratiladi, bunda stek qoidasiga ko'ra ohirgi ajratilgan joy birinchi bo'shatiladi.

Taqroriy keltirib chiqarish – $A \Rightarrow^* A$ ko'rinishdagi keltirib chiqarishlar, bunda $A \in VN$ noterminal belgi.

Tayanch vaziyatlar– boshlang'ich vaziyat [S'→ .S] va nuqtasi boshida bo'lmagan barcha vaziyatlar.

Terminal belgi – tilni alifbosi yoki lug'atidir. Asosan bu belgilar qoidalarni o'ng tarafidagi zanjirda uchraydi. Agar ular qoidani chap tarafida uchrasa, albatta o'ng tarafdagi zanjirda ham bo'lishi shart.

Tetrada yozuv – ko'p manzilli oshkor ravishdagi kod bo'lib, unda amallar <amal>(<operand1>,<operand2>,<natija>) shaklda yoziladi.

Tilning alifbosi – tilda mumkin bo'lgan belgilarning chekli to'plami.

Tilning leksikasi – tildagi so'zlar to'plami.

Tilning semantikasi – tildagi gaplarning ma'nosini aniqlovchi qism. Ko'pincha semantika erkin va norasmiy, tushuntirish usullari bilan beriladi.

Tilning sintaksisi – tilda mumkin bo'lgan gaplarni (jumlalarni) aniqlovchi qoidalar to'plami.

To'ldirilgan grammatika – dastlabki grammatikaga zanjirni tugashini nishonlovchi qo'shimcha terminal belgisi kiritilgan (⊥). Grammatikaga S'→S⊥ yangi qoida va yangi boshlang'ich noterminal belgisi qo'shilgan grammmatika.

To'liq aniqlangan chekli avtomat – har qanday holat va barcha belgilar uchun o'tish funksiyaning qiymatlar to'plami bo'sh bo'lmagan avtomat.

Translyator –kiruvchi tilda yozilgan dastlabki dasturni chiquvchi tildagi natijaviy ekvivalent dasturga aylantiruvchi dastur.

Triada yozuv – ko'p manzilli oshkormas natijali kod bo'lib, unda amallar <amal>(<operand1>,<operand2>) shaklda yoziladi.

"Uyum" usuli – bu hotirani ihtiyoriy taqsimlash usuli bo'lib, unda hech qanday qoida yo'q. Xotirani har qanday vaqtda egallash va kerak paytda bo'shatish mumkin. Bunday amallar oshkor ravishda dasturchi tomondan yoki oshkormas ravishda, ya'ni avtomatik ravishda kompilyator tomondan bajarilishi mumkin.

Verifikatsiyalash – dasturni to'g'riligini tekshirish jarayoni.

Virt diagrammasi – har bir noterminal belgi uchun diagramma orqali berilgan qoida grafigi.

Xomskiy normal shakli – qoidalari faqat $A \to BS$ yoki $A \to a$ yoki $S \to \varepsilon$ shaklda bo'lgan grammatika, bunda $A, V, S, S \in VN$ noterminal belgilar va $a \in VT$ terminal belgi.

Xotirani taqsimlash – kompilyator til birliklariga xotira manzilini beradi. Ularni o'lchovini aniqlaydi va birlikka hususiyatlar (atributlar) beradi.

Xotirani statik taqsimlash – kompilyator xotirani avtomatik ravishda statik sohalarga taqsimlaydi. Bu sohalarda taqsimlangan obyektlarning o'lchovi hech qachon o'zgarmaydi va ularga ko'rsatilgan manzillar ham o'zgarmaydi. Faqat sohani boshlang'ich manzili o'zgarishi mumkin, bu ham bajarishdan oldin va dasturni xotiraga yuklaganda bo'ladi.

Xotirani dinamik taqsimlash – hotira dinamik ravishda taqsimlanadi . Bunda ikki usulni qo'llash mumkin: "stek" usuli va "uyum" usuli.

Xesh-manzillash – bu identifikator nomiga ko'ra xesh-funksiya qiymati orqali identifikator nomlar massiv elementining manzilini aniqlash.

Xesh-funksiya – R elementlar to'plamini Z butun musbat sonlar to'plamiga aks ettiruvchi funksiya $F(r)$.

Yuklovchi – nisbiy manzillarni haqiqiy (absolyut) manzillarga qayta o'zgartirishni bajaruvchi dastur vositasi. Yuklovchilar dasturlash tizimlar ichiga kiritilishi mumkin, lekin, ko'pincha ular operatsionn tizimga kiradi, chunki u bajaradigan funksiyalar nafaqat dastur ishlaydigan hisoblash tizimning arhitekturasiga bog'liq, balki bu tizimning aniq jisoniy konfiguratsiyasiga (hotira o'lchovi va protsessor razryadiga) ham bog'liqdir.

Yuqoridan pastga qarab daraxtni qurish – daraxtni ildizdan boshlab pastga qarab barglarigacha qurish.

Yuqoridan pastga tushuvchi tahlilchi – pasayuvchi chap tahlil usuli asosida tuzilgan mantiqiy qurilma.

Yerishib bo'lmaydigan belgilar – grammatikaning xech qaysi sentensial shaklida uchramagan terminal yoki noterminal belgi.

Erkin grammatika – qoidalari $\alpha \to \beta$ bo'lgan grammatika. Bunda $\alpha \in V^+$, $\beta \in V^*$. Erkin grammatikada qoidalar tuzilishiga xech qanday chegaralar qo'yilmaydi.

Yacc dasturi –LALR-taxlil algoritmlari asosidagi sintaktik tahlilchining generatori. YAAC (Yet another compiler-compiler) dastur C++ tilida sintaktik tahlilchi yaratadi. Bu dastur ko'p kompilyatorlarni yaratishda qo'llanilgan.

Zanjirli qoidalar – A→B shakldagi qoidalar, bunda A,B∈VN noterminal belgilar.

Zanjirni keltirib chiqarish – grammatikada shunday γ_0, γ_1, ..., γ_n zanjirlar borki $\alpha \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow ... \Rightarrow \gamma_n \Rightarrow \beta$ keltirib chiqarilsa, ya'ni γ_{i-1} zanjirdan γ_i zanjiri bevosita keltirib chiqarilsa, α zanjirdan β zanjir qadamma-qadam keltirib chiqarish jarayoni. Bu keltirib chiqarish $\alpha \Rightarrow^* \beta$ ko'rinishda yoziladi.

Zanjir qismining negizi –o'ralishda ishlatilgan qoidaning o'ng tarafi.

Zanjirni taxlil qilish – zanjirni keltirib chiqarish jarayoni, ya'ni tilning berilgan grammatikasi bo'yicha, zanjirni boshlang'ich belgi qoidasidan boshlab, qolgan qoidalarga binoan keltirib chiqarish.

Zanjirni chap o'ralishi – o'ng keltirib chiqarishdagi zanjirning biror bir eng o'ng qismini, biror bir qoidaga binoan chap tarafidagi noterminal belgiga o'rash.

O'z-o'zidan paydo bo'luvchi oldindan ko'rinuvchi belgi – agar [B→α· Cδ,b] – I to'plamdagi biror bir tayanch vaziyat bo'lsa, undan kelib chiqgan vaziyatlarga a∈FIRST(δb) yangi oldindan ko'rinuvchi qo'shilgan belgi.

O'ng tomonli keltirib chiqarish – joriy sentensial shakl oldingi sentensial shakldan eng o'ng noterminal belgi qoidasini, o'ng tarafiga almashtirish yo'li bilan chiqarish.

O'ng tahlil – berilgan zanjir uchun, S noterminal belgidan boshlab, o'ng keltirib chiqarishda ishlatilgan qoida raqamlarining teskari ketma-ketligi.

O'ng chiziqli avtomat grammatika – qoidalari quyidagicha $A \to tB$ yoki $A \to t$, bunda $A,B \in VN$, $t \in VT$ va t yagona terminal belgisi bo'lgan grammatika

O'ng chiziqli muntazam grammatika – qoidalari quyidagicha $A \to xB$ yoki $A \to x$, bunda $A,B \in VN$, $x \in VT^*$ bo'lgan grammatika.

O'rash-o'rash to'qnashuvi – bir nechta o'rashdan qaysi biri qo'llanilishi kerakliyligi bo'lgan noma'lum holat.

O'tish funksiya – QxV dekart ko'paytma to'plamini R holatlar to'plamiga aks ettiruvchi funksiya, u δ harf bilan belgilanadi va $\delta(q,t)=R$, $q \in Q$, $t \in V$, $R \subseteq Q$.

O'tish jadvali – o'tish funksiyasining jadval shaklida berilishidir. Bu jadvalda satrlar holatlarni, ustunlar esa terminallarni bildiradi. Birinchi

satrda boshlang'ich holati turadi,va ohirgi satrda chekli holat joylashadi. Satr va ustun kesishgan katakda o'tish funksiyaning qiymati (holatlar to'plami) turadi.

Chap rekursiyasiz grammatika – A noterminal uchun $A \Rightarrow A\beta$ keltirib chiqarish mavjud bo'lmagan grammatika. Bunda $\beta \in (VN \cup VT)^*$.

Chap tomonli keltirib chiqarish – joriy sentensial shaklni oldingi sentensial shakldan, eng chap noterminal belgi qoidasini o'ng tarafiga almashtirish yo'li bilan chiqarish.

Chap chiziqli avtomat grammatika – qoidalari $A \to Bt$ yoki $A \to t$ va t yagona terminal belgisi bo'lgan grammatika. Bunda $A,B \in VN$, $t \in VT$.

Chap chiziqli muntazam grammatika – qoidalari $A \to Bx$ yoki $A \to x$ bo'lgan grammatika. Bunda $A,B \in VN$, $x \in VT^*$.

Chap tahlil – berilgan zanjir uchun, S noterminal belgidan boshlab, chap keltirib chiqarishda ishlatilgan qoida raqamlarining ketma-ketligi.

Chap faktorizatsiya – noterminalga tegishli o'ng tarafi bir hil prefiksdan boshlangan qoidalarni olib tashlash amali.

Chekli avtomat – berilgan zanjirning muntazam grammatika qoidalariga qanoatlanishini anglovchi mantiqiy qurilma.

Chekli determinantlangan avtomat – har bir holati va har qanday kiruvchi belgisi uchun o'tish funksiyasining qiymatlar to'plami, faqat bitta holatdan iborat yoki umuman bo'sh to'plam bo'lgan avtomat.

Chekli nodeterminantlangan avtomat – biror bir holat va biror bir kiruvchi belgi uchun o'tish funksiyasining qiymatlar to'plami, bir nechta holatdan iborat bo'lgan avtomat.

Chekli o'zgartirgich – kiruvchi zanjirdan chiquvchi zanjirga tarjima qiladigan chekli avtomat asosida tuzilgan mantiqiy qurilma.

FOYDALANILGAN ADABIYOTLAR.

[1]. Aho A., Ulman J. The theory of Parsing, translation and compiling. Vol.1: Parsing, Prentice Hill, Inc. Engkewood Cliffs,N.J. 1972.

[2]. Aho A., Ulman J. The theory of Parsing, translation and compiling. Vol.2: Translation and compiling. Prentice Hill, Inc. Engkewood Cliffs,N.J. 1972.

[3] Aho A. Sethi R. Ullman J. Compilers. Priciples, Techniques, and Tools. Edition 1. Addison-Wesley publishig inc. 1985

[4] Aho A. Sethi R. Ullman J. Compilers. Priciples, Techniques, and Tools. Edition 2. Addison-Wesley publishig inc. 2007

[4] Gris D. Compiler construction for digital computers. Wiley New York 1971.

[5]. Ф. Льюис, Д. Розенкранц, Р. Стирнз. Теоретические основы проектирования компиляторов. — М.: Мир, 1979.

[6]. Ф. Вайнгартен. Трансляция языков программирования. — М.: Мир, 1977.

[7]. И. Л. Братчиков. Синтаксис языков программирования. — М.: Наука, 1975.

[8]. С. Гинзбург. Математическая теория контекстно-свободных языков. — М.: Мир, 1970.

[9]. Дж. Фостер. Автоматический синтаксический анализ. — М.: Мир, 1975.

[10]. В. Н. Лебедев. Введение в системы программирования. — М.: Статистика, 1975.

[11]. Б. Ф. Мельников. Подклассы класса контекстно-свободных языков. — М.: МГУ, 1995.

[12]. В. Н. Пильщиков, В. Г. Абрамов, А. А. Вылиток, И. В. Горячая. Машины Тьюринга и алгоритмы Маркова. Решение задач. — М.:МГУ, 2006.

[13] А. В. Гордеев, А.Ю. Молчанов. Системное программное обеспечение –СПб.: Питер, 2002.

[14] Волкова И.А., Руденко Т.В. Формальные языки и грамматика. Элементы теории трансляции. – И.: Диалог-МГУ, 1999.

MUNDARIJA

GAYNAZAROV S.M.

DASTURLASH TIZIMLARI

DARSLIK

Muharrir Z.N.Buranov

Bosishga ruxsat etildi 03.10.2023y. Bichimi 60X84 $^{1}/_{16}$.
Bosma tabog'i 27,0. Shartli bosma tabog'i 27,0. Adadi 21 nusxa.
Buyurtma № 102. Bahosi kelishilgan narxda.
"Ma'rifat" nashriyoti. Toshkent, Salorbo'yi kochasi, 35A.
O'zbekiston Milliy universiteti bosmaxonasida bosildi.
Toshkent, Talabalar shaharchasi, O'zMU.

ISBN 978-9910-9887-9-0