A.X.NISHANOV

DEKLARATIV DASTURLASH

DARSLIK

O'ZBEKISTON RESPUBLIKASI AXBOROT TEXNOLOGIYALARI VA KOMMUNIKATSIYALARINI RIVOJLANTIRISH VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

"DASTURIY INJINIRING" FAKULTETI

A.X.NISHANOV

DEKLARATIV DASTURLASH

DARSLIK

TOSHKENT – 2021

UO‘K 004.42(073)
KBK 32.973ya72
N71

Nishanov A.X.
Deklarativ dasturlash. Darslik./A.X.Nishanov -Toshkent: «Lesson-press» nashriyoti. 2021-y., – 196 bet.

Darslikda matematik mantiq, matematik isbotlash usullari, rezolyusiya tamoyili, deklarativ dasturlash vositalari, mantiqiy dasturlashga kirish, funksional dasturlashga kirish, turlarga ajratish masalalari, ma'lumotlarning orqaga qaytuvchi (rekursiv) tuzilmalari, izlash usullari va algoritmlari, tabiiy tilga ishlov berish usullari va algoritmlari, deklarativ dasturlash kesimida zamonaviy ob'ektga yo‘naltirilgan yondoshuvlardan foydalanish, sun’iy intellekt tuzilmalarini barpo etishga turli yondoshuvlar singari mavzular yoritilgan.

Darslik universitetning barcha yo‘nalishlari talabalari, professor-pedagogik tarkibi hamda ilmiy xodimlari uchun mo‘ljallangan.

UO‘K 004.42(073)
KBK 32.973ya72

Mazkur darslik O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta'lim vazirligining 2021 yil 18-avgustdagi 356-sonli buyrug‘i bilan Oliy ta'lim muassasalari talabalari uchun tavsiya etilgan.

ISBN 978-9943-7506-5-5

KIRISH SO‘ZI

Deklarativ dasturlash to‘g‘risida so‘z borganda va uning nima ekanligi izohlanayotganda, garchi, dasturlash tillari emasligi barchaga ma'lum bo‘lsada, darhol SQL, HTML, XAML tillari yodga olinadi.

Ushbu darslik, avtor tamonidan uzoq yillar davomida universitet magistrlari uchun o‘qilgan deklarativ dasturlash kursi asosida ishlab chiqilgan. O‘qilgan kursdagi singari deklarativ dasturlash tushunchasi mantiqiy va funksional dasturlash, mantiqiydan ko‘ra ko‘proq darajada funksional dasturlash bilan bog‘liq, chunki mantiqiy dasturlash funksional dasturlash singari keng tarqalmagan va amaliyotda qo‘llanilmaydi. Taqqoslash uchun mantiqiy dasturlash DataLog, Mercury, Prolog va RYEFAL tillari bilan, funksional dasturlash esa – Agda, Erlang, Haskell, Hope, Idris, Miranda, ML, Lisp, Ocaml, Scala, Sisal va ko‘plab soni 100 dan yuqoriroq bo‘lgan (Lisp tilida 300 dan ortiqroq lahjalar mavjud) taqdim etilgan.

O‘quvchini kursga kiritishdan oldin “Dasturiy ta'minot inqirozi” nomini olgan muammoga murojaat qilamiz. Ma'lumki, (vikipediya) inqirozning mohiyati shundan iboratki, hisoblash texnikasining ortib borayotgan murakkabligi va quvvati nafaqat kodlash miqdorining ortishi hisobiga, balki murakkabligiga ko‘ra o‘sib boruvchi vazifalarni hal qilish uchun qo‘llaniladigan kod va ma'lumotlar tuzilmalarining yanada ko‘proq darajada murakkablashib borishi hisobiga dasturiy ta'minot murakkablashuvini keltirib chiqaradi.

Inqiroz o‘zini o‘ta turlicha qiyofalarda namoyon etadi:

- Loyihalarning qiymati byudjetdan ortib ketadi.
- Loyihalardagi bajarish muddati ortadi.
- Dasturiy ta'minot o‘ta samarasiz bo‘lib chiqadi.
- Dasturiy ta'minotning sifati o‘ta past bo‘ladi.
- Dasturiy ta'minot ko‘pincha zaruriy talablarga javob bermaydi.
- Loyihalarni boshqarib bo‘lmaydi hamda kodni saqlab turishda qiyinchiliklar paydo bo‘ladi.
- Dasturiy ta'minot tarqatish uchun yaroqsiz bo‘lib chiqadi.

Deklarativ dasturlashga nisbatan quyidagi mulohazalardan yuqorida keltirilgan masalalarning ayrimlarini loaqal (kodni qisqartirish hisobiga birgacha tartibda va undan ko‘proq, ya'ni aynan shunday vazifani hal

qiluvchi dastur satrlari soni imperativ tabiat tillaridagiga nisbatan 10-15 marta kam bo'ladi) va xususan so'z Haskell tilidan foydalanish to'g'risida so'z ketganda kodning yanada aniqligi hisobiga foydalanish mumkin.

Boshqa tomondan, deklarativ dasturlash paradigmalari o'zining o'ziga xos shaklida qo'llanishiga qaramasdan, aksariyat darsliklar va qo'llanmalarda "Men uch oyda S++ da sen bir haftada Haskellda o'ta qiyinchilik bilan qiladigan ishingin osonlik bilan qilaman" turidagi taqqoslashlar baribir keltirilmoqda, imperativ tilda har qancha mahorat va ustalikda ham kodni yozishda satrlar soni aynan shu hisoblab chiqish vazifasini amalga oshiruvchi Haskell tilidagi satrlar sonidan 10 marta oshib ketadigan kod misollari ko'rsatilmoqda. Bundan tashqari qo'llashlar, shu jumladan funksional tillar uchun tijoratda qo'llashning ancha salmoqli ro'yxati keltirilmoqda, shu tarzda deklarativ dasturlash nima ekanligi bilan loaqal tanishish qiziqarli bo'lib bormoqda.

I. BOB. MANTIQIY VA FUNKSIONAL DASTURLASH

1.1. Mantiqiy dasturlash

Har qanday rivojlangan tabiiy til grammatikasida gaplarning uch mayli: istak mayli, (povelitelnoe), so‘roq mayli va buyruq mayli farqlanadi. Agar oddiy nutqda istak gaplari ustunlik qilsa, bizga ma'lum bo‘lgan deklarativgacha an'anaviy dasturlashda buyruq mayli ustundir. An'anaviy tasavvurdagi dastur – kompyuter bajaradigan buyruqlar izchilligidir. Biroq dasturlashning boshqa tarzi ham – istak maylida dasturlash mavjud, bundagi dasturlash tasdiqlarning shunchaki majmuidir. Bunday tarz (yoki hozir urf bo‘lgan tarzda aytiladigan bo‘lsa paradigma) deklarativ dasturlash nomini olgan. Bunga qarama-qarshi holda birinchi tarz imperativ dasturlash deb ataladi. Shunga mos tarzda dasturlash tillari ham imperativ va deklarativ tillarga bo‘linadi. Funksional va mantiqiy (yoki relyasion) dasturlash deklarativ dasturlashning eng muhim ko‘rinishlaridir.

Dastur odatda qandaydir muammoni hal qilish uchun yaratiladi. imperativ tarzda dasturlagan holda biz kompyuterga muammoni qanday hal qilishni aniq tekazishimiz, ya'ni hal qilish jarayonini qadam baqadam tasvirlab berishimiz lozim. Deklarativ dasturlashda biz ko‘proq kompyuterga muammo nimadan iborat ekanligini etkazamiz, dasturning o‘ziga xosligini tasvirlaymiz. Deklarativ dasturlash – bu ijroning o‘ziga xosligi deb aytish mumkin.

Robert Kovalskiy o‘zining “Algoritm = Mantiq + Boshqaruv" [1] maqolasida har qanday dasturda ikki tarkibiy qismni: muammo echimining o‘zini tasvirlovchi mantiqni va hisoblab chiqish jarayonini boshqarishni ajratib ko‘rsatish mumkinligi holatini ta'kidlaydi. Mazkur terminologiyadagi deklarativ dasturlash – boshqaruvning (majburiy tarzda emas) algoritm mantiqini tasvirlashdir.

Ehtimol, deklarativ tasvirlashning asosiy g‘oyasini quyidagi ta'rif (Jon Robinsonga tegishli [2]) barchadan yaxshiroq tasvirlaydi: dastur nazariyadir, hisoblab chiqish esa, ushbu nazariyadagi xulosalardan iborat.

Yana bir, salbiy ta'rif: deklarativ dasturlar holat tushunchasidan foydalanmaydi va o‘zgaruvchan qiymatlarga ega emas. Binobarin, bular o‘zlashtirish operatorlaridan foydalanishi mumkin emas, chunki

o‘zlashtiriladigan narsa yo‘q. Bundan tashqari buyruqlarni bajarish izchilligi tushunchasi bema'noga aylanadi, va demak hisoblab chiqish jarayonini boshqarish operatorlari befoyda bo‘lib chiqadi, ushbu ta'rifdan kelib chiqqan holda, deklarativ tillar, ayrim qulay va odatdagi vositalar olib tashlangan imperativ tillarning bor-yo‘g‘i nuqsonli ko‘rinishlari ekanligi borasidagi xulosaga kelishi mumkin. Biroq shunga e'tibor qaratish qiziqarliki, dasturlash texnikasidagi ko‘pgina takomillashtirishlar dasturlovchi uchun qulay bo‘lgan imkoniyatlarni kengaytirish bilan emas, ko‘proq cheklash bilan bog‘liqdir.

Fortrandan boshlab yuksak darajadagi oddingi tillar kod va ma'lumotlar sohasidagi global xotiraga qo‘shilgan va faqat ma'lumotlar o‘zgarishi mumkinligi cheklanishni o‘rnatgan. Modullik, tuzilmali dasturlash, qat'iy turlarga ajratish konsepsiyalari ham dasturchining erkinligini cheklaydi. Yanada qat'iyroq interfeyslar va ob'ektlarni bevosita o‘zgartirishga ta'qiqlar so‘nggi urf bo‘lgan holatlardir. Deklarativ dasturlash ushbu yo‘nalishdagi yana bir yanada keskinroq qadam sifatida ko‘rib chiqilishi mumkin. Oddiy dasturchilar standart deb hisoblaydigan vositalarda o‘zimizni cheklagan holda, biz ayrim ustunliklarga ega bo‘lishga umid qilayotganimiz tushunarlidir.

Imperativ tillar fon Neyman modeli sifatida mashhur bo‘lgan oddiy kompyuter hisoblab chiqishi modelidan abstarksiyalash yo‘li bilan rivojlangan. Shuning uchun ular kompyuter resurslaridan samarali foydalanishga imkon beradi, biroq inson kompyuter emas. Binobarin, hisoblab chiqish mashinasi iboralari va tushunchalarida mulohaza yuritishi uning uchun qiyin va notabiiydir. Anchagina murakkab vazifani hal qilgan holda, biz odatda dastlab modelni qandaydir, formal tizimda yaratamiz, so‘ngra topilgan echimni dastur ko‘rinishida qayta yozib chiqamiz. Ya'ni, bir formal tildan boshqa tilga o‘tkazamiz. Mazkur ikkinchi bosqich echim jarayonining o‘ziga tegishli emas, va ayni vaqtda katta sa'y harakatlar va maxsus ko‘nikmalarni hamda amalga oshirish uchun vaqtni talab qiladi.

Dasturchilar orasida xatto mehnat taqsimoti paydo bo‘lgan: ayrimlar (tahlilchilar) vazifani qanday hal qilsh ustida bosh qotirsa, boshqalari (kodlovchilar) esa ular topgan echimni dasturlash tiliga yozib oladi. Deklarativ tillar asosida qandaydir mashinali model emas, balki mantiq va matematika qaror topgandir, shuningdek muammo to‘g‘risidagi bilimlarimizni dasturlash tili iboralarida bevosita shakllantirsh imkoniyati

paydo bo‘ladi. bu fikr shubhali bo‘lib ko‘riishi mumkin, biroq har qanday holatda deklarativ dasturlar ularning “oddiy” o‘xshashliklariga nisbatan formal o‘ziga xosliklarga ancha yaqindir.

“Algoritm = Mantiq + Boshqaruv” tenglamasiga qaytamiz. Imperativ tilda dasturlash da ushbu ikki tarkibiy qism bir-biriga chambarchas bog‘liq va dasturchida dasturlarni bajarishning past darajali tafsilotlariga kirishdan boshqa imkoniyat qolmaydi.

Deklarativ tillarda mantiq va boshqaruv ajratilgan. Faqat (yoki asosan) mantiqiy bo‘g‘in bilan ish ko‘rish zarurati dasturlarni soddalashtiradi. Deklarativ dasturi umuman aytganda, yozib olish osonroq va shunisi ham muhimki, buni tegishli imperativ dasturdan tushunish osoronroqdir. Bundan tashqari boshqaruvni yaratganlik uchun mas'uliyatni kompyuterga uzatish imkoniyati paydo bo‘ladi.

Ko‘pincha, eng samarali algoritmning mavjudligi u qadar muhim emas va tizimning o‘zi topadigan oqilona samarali algoritmning o‘zi etarlidir. Bunday tashqari, determinasiyalashmagan, “dangasa” va parallel hisoblab chiqish singari boshqaruvning o‘ziga xos algoritmlaridan foydalanishning qiziqarli imkoniyatlari paydo bo‘ladi.

Ko‘plab amaliy muhim vazifalar hisoblab chiqishlarning ulkan sonini taqozo etadi va talab qilinayotgan hisoblab chiqish quvvatining aniq ko‘rinib turgan yo‘li – parallel kompyuterlardan foydalanishdir. Biroq izchil dasturlarni optimal tarzda paralellikdan chiqarish o‘ta kamdan kam holatda parallelizmning maqbul darajasiga erishishni ta'minlaydi Imperativ tildagi dastur “oldindan belgilab qo‘yilgan”. U ko‘pincha vazifaning echimi bilan bog‘liq bo‘lmagan hisoblab chiqishlarning muayyan tartibini belgilaydi. Vektorlashuvchi kompilyator vazifaga avval boshdan xos bo‘lgan parallelizmni retrospektiv topishga urinadi, biroq buni amalga oshirish o‘ta qiyin, ko‘pincha esa imkonsizdir. Oxir-oqibatda parallelizm uchun imkon topadigan kompilyator faqat izchil tillarga xos bo‘lgan oldindan aniqlanganlikdan qutiladi, xolos. Shunga bog‘liq holda, dasturlash parallel tillarining parallelizmni ifodalash uchun aniq vositalarga ega bo‘lgan toifasi paydo bo‘ldi. Biroq bunday tillarda dasturlash izchil tillardagiga nisbatan ancha qiyindir – insonga zamonda kechadigan jarayonni anglashdan ko‘ra turg‘un mosliklarni tushunish ancha osonroqdir hamda bir vaqtning o‘zida kechadigan jarayonlar majmuini esa aniq tasavvur qilish umuman qiyindir.

Deklarativ tillar parallel kompyuterlarni dasturlash uchun yaxshi mos keladi. Ular parallelizmni tasvirlashning maxsus usullarini talab qilmaydi. Dasturlash parallelizmi uning axborot aloqalari bilan noaniq belgilanadi. Boshqacha so‘zlar bilan aytganda, dasturchi uchun dastur izchil yoki parallel kompyuterda ishlayotganligining farqi yo‘q. Dasturlash tili qanchalik deklarativ bo‘lsa, noaniq parallelizmning mavjudligining ehtimoli shunchalik katta va dasturni parallel bajarish shunchalik osonroq bo‘ladi. Parallelizmdan bunday bilvosita foydalanish dasturchi uchun juda qulaydir. Chunki buning uchun izchil kompyuterda mavjud bo‘lgan qiyinchiliklardan tashqari qo‘shimcha qiyinchiliklar paydo bo‘lmaydi. Biroq tizimning o‘zi dasturni bajarishning eng samarali usulini topish uchun etarli darajada “ongli” bo‘lishi lozim. Bu o‘ta qiyin vazifadir, biroq ayrim ijobiy natijalarga erishilgan va yaqin kelajakda dasturchi uchun parallel kompyuterlarda deklartiv dasturlarni samarali bajarishning “shaffof” imkoniyati paydo bo‘lishiga umid qilish mumkin.

Deklarativ tillarning yana bir afzalligi – formal mulohazalar uchun yaroqliligidir. Ma'lumki, testdan o‘tkazish xatolarni topishga imkon beradi. Biroq ularning yo‘qligini kafolatlay olmaydi. Dastur to‘g‘riligiga ishonch hosil qilishning yagona usuli uning kodini tahlildan o‘tkazishdir. Afsuski, dasturlarni formal tahlil qilishning mavjud usullari o‘ta ko‘p mehnat talab qiladi, gap shundaki, Si turidagi keng foydalanilayotgan turlar o‘ta murakkab va chalkash semantikaga ega. Bu shunday tillardagi dasturlar to‘g‘risida formal (va hatto noformal) mulohaza yuritish o‘ta qiyinligini anglatadi. Bir muncha soddaroq va qat'iyroq tillarni yaratishga, shuningdek tahlil usullarini takomillashtirishga ko‘p harakatlar qilingan, biroq natijalar hozircha kutilgandek emas, formal usullardan foydalanish o‘ta qimmat turadi va dasturiy ta'minotning ishonchliligi zaif bo‘lgan sohalar bilan cheklanmoqda.

Deklarativ dastur formal nazariyadan iboratdir va binobarin mantiqiy tahlilga nisbatan osonroq tortilishi mumkin. Shu asnoda ushbu tahlil holatlardagi tushunchalar sifatidagi qo‘shimcha formalizmlarni jalb etishni talab etmagan holda dastur iboralarining o‘zida amalga oshirilishi mumkin. Buning ustiga dasturlarni tahlil qilish, o‘zgartirish, umumlashtirish, jarayonlarini avtomatlashtirshga imkon beruvchi instrumental vositalarni yaratish imkoniyati vujudga keladi. Bunday vositalarning paydo bo‘lishi dasturlashni tubdan o‘zgartirishi mumkin.

Deklarativ tillar uchun bunday turdagi vositalarni yaratish oson vazifa deb bo‘lmaydi, biroq ko‘plab jiddiy bo‘lmagan qiyinchiliklar yo‘qoladi va loaqal umuman bo‘lishi lozim bo‘lmagan muammoni hal qilishga intilgan holda behuda vaqt yo‘qotishga to‘g‘ri kelmaydi.

Deklarativ dasturlash borasida maqtov so‘zlardan so‘ng ushbu yondoshuvga xos bo‘lgan nuqsonlarni ham tilga olib o‘tish zarur.

Deklarativ dasturlashning mavjud shakllari ko‘plab vazifalarning zamon jihatlarini yaxshi uddalay olmaydi. Foydalanuvchilar yoki tashqi jarayonlari bilan faol o‘zaro harakatga kirishuvchi ilovalar uchun deklarativ vositalarning ko‘pincha etarli bo‘lmaydi va dasturchilarga dasturlarning deklarativligini izdan chiqaruvchi ancha o‘ziga xos usullardan foydalanishiga to‘g‘ri keladi.

Ko‘pincha deklarativ tillar parallel dasturlash uchun mos velmasligi ta'kidlanadi. Shu asnoda “parallel dasturlash” deyilganda hsioblab chiqishlarni parallellikdan chiqarish emas, balki muayyan ma'noda qarama-qarshi vazifa – dasturdagi tabiiy parallelizmda munosib ifodalash tushuniladi. Ya'ni, gap yana zamon jihatlari to‘g‘risida boradi.

Ushbu qiyinchiliklarning sababi shundaki, deklarativ tillar mohiyatiga ko‘ra “turg‘un” nazariyalarga asoslangandir. Deklarativ tillarga zamon vositalarini kiritishga imkon beruvchi yangi “dinamik” formalizmlar zarur. Ularni yaratishning extimoliy yo‘llarini “zamon mantiqlarida”, jarayonlarning algebraviy nazariyalarida va xatto avtomatiklar nazariyasida ko‘rish mumkin. Xozirgi vaqtda ushbu nazariyalar amalda keng qo‘llash uchun etarli darajada rivojlanmagan, yohud qo‘llashning ancha tor sohasiga ega, biroq uzluksiz jarayonlarni turg‘un tasvirlashga imkon beruvchi differensional hisoblab chiqish kashfiyoti fanda qanday inqilob yaratganligini yodga olish zarur.

Deklarativ tillarning boshqa muammosi – samaradorlikdir. Ko‘pincha bo‘lganidek, nuqsonlar afzalliklarning davomidir. Ushbu tillarning tuzilmasi hisoblab chiqish mashinalarining qurilmasidan uzoq bo‘lganligi bois dasturlash unumdorligini baholash ancha qiyin kechadi. Amalga oshirish texnikasi so‘nggi o‘n yilliklarda ancha yuksalganligi ham haqiqat, biroq qisqa, tushunarli va nozik dasturni yozib tashlash juda oson bo‘lsada, kam unumdorligi bois, ular butkul nomaqbuldir. Samarali dasturlarni yaratish esa, ayrim holatlarda katta san'atni taqozo etadi. mazkur muammoni hal qilishning taklif etilayotgan yo‘li – dasturlarning

maxsus annotasiyalaridan va avtomatik o‘zgarishlardan foydalanishdir. Bularning birinchisi ba'zan qo‘llanilayotgan bo‘lsada, ikkinchisi hanuzgacha tadqiqotlar mavzuidir.

Hozircha esa amaliyotda dasturlashning deklarativ tillariga imperativ xossalarni kiritish afzal ko‘rilmoqda. Bunday yondoshuv bema'no emas, axir buyruq gaplar bekorga mavjud emas-ku, agar biz nimadir yuz berishini xohlasak, “buni bajar” deb aytish hamma narsadan osonroq-ku, biroq imperativ operatorlarni to‘g‘ridan to‘g‘ri qo‘shish ko‘pincha deklarativ tillarning ko‘plab afzalliklarini yo‘qqa chiqaradi. Bir til doirasida turli dasturlarni birlashtirish puxta ishlov berishni taqozo etadi. Afsuski, funksional va mantiqiy tillarni yaratuvchilar uzoq vaqtga qadar deklarativlik g‘oyasiga etarli darajada jiddiy yondoshmadi va natijada amaliy dasturlarni Prolog yoki Lispdagi amaliy dasturlarni xatto Si dagi dasturdan ko‘ra tahlil qilish unchalik oson emas.

Biroq, o‘zining nuqsonlariga qaramasdan deklarativ yondoshuv o‘rganishga arziydi. Birinchidan, deklarativ tillarni qo‘llash sohasi ancha keng va kengayishda davom etmoqda. Ikkinchidan, funksional va mantiqiy dasturlash zahirasida an'anviy imperativ dasturlashda ham qo‘llanilishi mumkin bo‘lgan ko‘plab vositalar jamlangan.

Ma'lumki dasturlash – kompyuter dasturlarini yaratish jarayonidir.

Mantiqiy dasturlash – kompyuter dasturlarini yaratishga yondoshuvlardan biri bo‘lib, shu asnoda yuksak darajadagi til sifatida Xorn iboralari shaklidagi birinchi tartibli predikatlar mantiqidan foydalaniladi. Formal mantiq sohasidagi tadqiqotlar eramizga qadar IV asrda Aristotelning asarlarida boshlangan edi. Birinchi tartibdagi predikatlar mantiqi asosan XX asrda rivojlangan formal mantiq tarmog‘idir. Bu - vazifalarini tasavvur qilish uchun va vazifalarni hal qilish uchun mo‘ljallangan universal mavhum tildir. Buni munosabatlarning umumiy nazariyasi sifatida ko‘rib chiqish mumkin. mantiqiy dasturlash birinchi tartibdagi predikatlar mantiqining ko‘pligiga tayanadi, shu asnoda bu qamrab olish sohasi bo‘yicha mantiq bilan bir xil darajada kengdir.

Mantiqiy mulohaza yuritganda mantiqiy dasturlash – mantiqqa asoslangan dasturlashdir. Ko‘pincha bu hisoblab chiqish mexanizmi sifatida mantiqiy xulosadan foydalanish bilan birgalikda mantiq tilidan dasturlash tilidan sifatida foydalanish tarzida ta'riflanadi. Bu juda yaxshi,

biroq o‘ta keng ta'rifdir. Bunday tarzda ta'riflangan mantiqiy dasturlash o‘z ichiga funksional dasturlashni ham, shuningdek dasturlashning ko‘plab boshqa tarzlarini ham qamrab oladi.

Mantiqiy dasturlash dasturchiga predikatlar mantiqi formulalari yordamida vaziyatni tasvirlashga so‘ngra esa, ushbu formulalardan paydo bo‘lgan xuloalarni bajarish uchun vazifalarni avtomatik echuvchisidan, ya'ni qandaydir tartibotdan foydalanash imkonini beradi. Mantiqiy dasturlash tilidan foydalanishda kompyuterga nima qilish to‘g‘risida ko‘rsatmalarni ishlab chiqishga emas, balki amaliy vazifa tuzilmasini tasvirlashga asosiy e'tibor qaratiladi. Ma'lumotlarning relyasion bazalari, dasturi muhandislik va bilimlarni taqdim etish singari sohalardagi axborotlashning boshqa tushunchalarini ham mantiqiy dasturlash yordamida tasvirlash va (binobarin amalga oshirish) mumkin. Bu nuqtai nazardan mantiqiy dasturlash informatikaning ko‘plab jihatlariga yashirin inqilobiy ta'sir o‘tkazadi.

Tarixan shunday shakllanganki, “mantiqiy dasturlash” iborasi ostida birichni tartibdagi sof mantiqning ayrim ko‘pliklarini dasturlash tili sifatida foydalanish anglatiladi. Bu harakat konsepsiyasini ifodalash uchun munosabat yoki predikatning matematik tushunchasi qo‘llanilishini anlatadi.

Ko‘pincha til xornovcha takliflarni qo‘llash, xulosa qilish mexanizmi esa rezolyusiya usulining muayyan turi bilan cheklanganda mantiqiy dasturlash tushunchasining yanada torroq talqini uchraydi. Prolog va dasturlash ning unga o‘xshash tillari mana shu asoslarda yaratilgandir. Ushbu tillar asosida aynan bir ifodani mantiqiy tasdiq siftiada talqin qilish imkoniyati yotadi.

A, agar B1va B2 va ... Bn hamda A ni bajarish uchun muayyan tartibot belgilash sifatida

B1 bajarish;

B2; bajarish;

...

Bn. bajarish;

Biroq bunday tor tushunchani o‘ta cheklanganligni e'tirof etish lozim. “Mantiqiy” nomiga loyiq dasturlashning ko‘plab tillari bu sxemaga jo bo‘lmaydi, boshqa tomondan aynan shu prolog ikkinchi tartibdagi

mantiq unsurlariga ega. Prologning ayrim xossalari esa kazuistik yondoshuvda uni deklarativ tillarga kiritishga umuman imkon bermaydi.

Shunday bo'lsada, prolog mantiqiy dasturlashning eng tarqalgan tili ekanligicha qolmoqda va biz aynan undan foydalanamiz. Prolog misolida biz mantiqiy dasturlashni ma'lumotlarning deduktiv bazalari, sintaktik tahlil (xususan bir ma'noli bo'lmagan grammatikalar) va murakkab kombinatorlik vazifalari singari ayrim qo'llashlar uchun eng yaxshi mos keladigan mantiqiy dasturlashga aylantiradigan mantiqiy dasturlarning o'ziga xos xossalarini ko'rib chiqamiz.

Mantiqiy tillarning kelgusidagi rivoji ikki yo'nalishda bormoqda. Bularni algoritmik va ortiqcha yo'nalishlar deb atash mumkin. bularning birinchisi asosan samarali algoritmlar ma'lum bo'lgan hamda prologdagi mavjud vositalardan ko'ra boshqaruvning yanada nozik vositalarini taqozo etuvchi vazifalarni hal qilishga mo'ljallangandir. Ikkinchisi esa, hal qilish uchun samarali algoritmlar bo'lmagan va turlarining ortiqchaligi sezilarli joy egallaydigan vazifalarga mo'ljallangandir. Ushbu yo'nalish tillari prolog uchun xususiyatli bo'lgan ayrim sodda ortiqchalikni yanada nozikroq usullarga almashtiradi.

1.2. Funksional dasturlash

Dasturlashda biz mohiyatlarning ikki ko'rinishi: ma'lumotlar va amaliyotlar bilan ish ko'ramiz. Bularning birinchisini biz materialini o'zimiz boshqarayotgan hisoblab chiqish jarayonining nofaol tarkibiy qismi sifatida ko'rib chiqamiz. Ikkinchisi esa, - uning faol tarkibiy qismi, ma'lumotlar ustidan harakatlarni bajarish qoidalarini tasvirlashdir. Shuning uchun dasturlashning har qanday tili ushbu mohiyatlarni tasvirlash imkoniyatlarini berishi mumkin. Eng avvalo, til sodda ma'lumotlarni hamda ular ustidan amaliyotlarni namoyon etuvchi elementar ifodalarga ega bo'lishi lozim. So'ngra bizga sodda ob'ektlarni "ham ma'lumotlarni ham amaliyotlarni" yanada murakkabroq ob'ektlarga birlashtirish vositalari zarur bo'ladi. Vanihoyat dasturlash tili murakkab ob'ektlarning muayyan tuzilmasidan chetlashishga va ularni yagona mohiyatlar sifatida boshqarishga imkon beruvchi mavhumlashtirish vositalariga ega bo'lishi lozim.

Funksional dasturlash harakat konsepsiyasini ifodalashi uchun funksiya matematik tushunchasidan foydalaniladi. oddiy matematik funksiyalarga o'xshash tarzda funksional tillarning tartibotlari

(funksiyalari) ayrim ob'ektlarni (dalillarni) boshqa ob'ektlarda (qiymatlarda) aks ettiradi. Shu asnoda imperativ tillarning tartibotlaridan (funksiyalaridan) farqli o'laroq funksiyalarning qiymatlari ularning dalillari bilan bir ma'noda belgilanadi hamda hisoblab chiqish jarayonining tarixiga bog'liq bo'lmaydi, biroq matematik funksiyalar va tartibotlar o'rtasida muhim farq mavjud. Tartibotlar samarli belgilanishi lozim. Masalan matematikada biz a sonidan kvadrat ildizni kvadratga ko'tarilgan son sifatida a ni beradigan kvadrat ildiz sifatida belgilashimiz mumkin. Bu butkul qonuniy ta'rif. Shunday bo'lsada, mutlaq asossizdir, u ushbu ildizning o'zini topish usulini bermaydi (ammo bunday ta'rif butkul befoyda deb bo'lmaydi, chunki u o'ziga xoslik vositasi sifatida muhimdir). Keyinroq biz "tartibot" va "funksiya" iboralaridan sinonimlar sifatida foydalanamiz, funksiya matematik tushunchasi va dasturlash tilidagi uning ta'rifi o'rtasidagi farqni ta'kidlash zarur bo'lgan holatlar bundan mustasnodir.

Funksiyaning aynan shu tushunchasi ma'lumotlar konsepsiyasini ifodalash uchun ham qo'llanishini keyinroq ko'rib chiqamiz. Umuman olganda funksionali dasturlashni o'rganish, "mu'lumotlar" va "amaliyotlar" o'rtasidagi farq unchalik muhim emasligiga ishonch xosil qilish uchun yaxshi sababdir. Fuknsional tillardagi funksiyalar "birinchi toifa" ob'ektlaridir.

Dasturlash tili "birinchi toifa" unsurlari tushunchasi Kristof Streychi tomonidan kiritilgan. U dasturlash tili til unsurlaridan foydalanish usullariga turli cheklashlar qo'yishini ta'kidlagan. Birinchi toifa unsurlari – cheklanishlarni eng kam miqdoriga ega bo'lgan unsurlardir. Bunday birinchi toifali unsurlarning muhim xossalariga:

- ularga o'zgaruvchi qiymatlar vositasida tayanish mumkinligi;
- ularni ma'lumotlar tuzilmasiga kiritish mumkinligi;
- ularni parametrlar sifatida uzatish mumkinligi;
- ularni natija sifatida qaytarish mumkinligi kiradi.

Ayrim istisnolarni olmaganda (masalan, Algol-68) imperativ tillar funksiyalar va tartibotlarning "huquq va erkinliklarini" cheklaydi, Bulardan farqli o'laroq, funksional tillar funksiyalarga birinchi toifa maqomini beradi. Bu ulardan samarali foydalanish uchun qiyinchilik yaratadi, biroq ushbu tillarning ifodalovchi kuchini sezilarli darajada orttirishga olib keladi.

Kelgusidagi bayon uchun bizga dasturlash tili zarur bo'ladi va bunday til sifatida Haskell tanlangan. Bu tanlov tilni favqulodda sodda semantikasidan va uning muayyan darajadagi bir turli sintaksisidan kelib chiqadi. Bu xossalar funksional dasturlashning aossiy g'oyalarini shaffof ifodalashga imkon beradi.

Nazorat savollari

1. Deklarativ dasturlash imperativ dasturlashdan nimasi bilan farqlanadi?
2. Mantiqiy va funksional dasturlashning farqi nimada?
3. Nima uchun hozirgi vaqtda deklarativ dasturlashga qiziqish ortmoqda?
4. Deklarativ dasturlash asosida nima qaror topgan?

II. BOB. MANTIQ. MANTIQIY TAFAKKURNI NAMOYON ETISH USULLARI VA VA VOSITALARI

2.1. Mulohazalar, formulalar, so‘z o‘yinlari

Tafakkur qonunlari va shakllari to‘g‘risidagi fan mantiq deb ataladi. Formal mantiq to‘g‘ri mulohaza shakllarini o‘rganadi. Matematik mantiq formal mantiqning bir qismi bo‘lib, matematikada qabul qilingan mulohazalar shakllarini o‘rganadi.

Har qanday fan yoki ilmiy bilimning asosi mantiqiy mulohazalarga tayanadi. Tamoyillar va qoidalar qat'iy mantiqiy tuzilmalarga bo‘ysunadi.

Oldingi ma'ruzada biz har qanday dastur o‘zida mantiq va boshqaruv unsurlariga ega ekanligini eslatib o‘tgan edik.

Bizga Prolog asosida qaror topgan birinchi tartibdagi mantiq tushunchalarining tagiga etishga urinib ko‘rishga to‘g‘ri keladi; buning uchun biz matematik mantiq ish ko‘radigan asosiy tushunchalar bilan tanishib o‘tamiz.

Ta'rif. Haqiqiy yoki yolg‘on (biroq, bir vaqtda emas) bo‘lgan tasdiq mulohaza deb ataladi.

Ya'ni, qandaydir gap mulohaza ekanligini aniqlash uchun dastlab tasdiq ekanligiga ishonch hosil qilish va so‘ngra esa u haqiqat yoki yolg‘on ekanligini aniqlash zarur.

Masalan. “Toshkent – O‘zbekiston poytaxti ” – haqiqiy mulohaza.

“5 – juft son”– yolg‘on mulohaza.

“x–1=4” – mulohaza emas (x qanday qiymat qaul qilishi noma'lum).

“Ikkinchi kurs talabasi” mulohaza emas (tasdiq emas).

Mulohazalar sodda va tarkibiy mulohazalar bo‘lishi mumkin.

Elementar mulohazalar, boshqa mulohazalar orqali ifodalanishi mumkin emas, tarkibiy mulohazalar elementar mulohazalar yordamida ifodalanishi mumkin.

Misol. “ 22 soni juft sondir” – elementar mulohaza.

“22 juft sondir va 11 ga bo‘linadi” – tarkibiy mulohaza.

Mulohazalar yunon alifbosining bosh harflari: A, V, S,... bilan belgilanadi. Bu harflar mantiqiy Atomlar deb ataladi.

Harflar ko ʻpligi qayd etilganda A={A, B, C, . . .} ning ko ʻpligini haqiqiy, (mantiqiy) qiymatlar ko ʻpligida aks ettiruvchi I funksiya Talqin deb ataladi, T={haqiqat, yolg ʻon, ya'ni I: $A \to T$

Haqiaqat va yolg'onning asl qiymatlari h, yo yoki T, F, yohud 1,0 bilan qisqargan tarzda anlatiladi. Biz va 1 va 0 qiymatidan foydalanamiz. Muayyan talqinda harflar 1 yoki 0 qiymatini qabul qiladi.

Mulohazalar va harflarga diskret matematika kursidan ma'lum bo'lgan mantiqiy bog'lamalar yoki mantiqiy amaliyotlarni qo'llash mumkin. Shu asnoda Formulalar (shakllar) yuzaga keladi. Formulalar harflarning barcha qiymatlari kiritilgandagina mulohazalarga aylanadi.

Asosiy mantiqiy amaliyotlarning haqiqiyligi jadvali.

A	B	A	A∧B	A∨B	A→B	A↔B
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Formula yanada qat'iyroq quyidagicha ta'riflanadi.

Ta'rif.

1) har qanday harf formuladir.

2) Agar A, B – formulalar bo'lsa, A, $A \wedge B$, $A \vee B$, $A \to B$, $A \leftrightarrow B$ ham formulalardir.

3) Timsol (simvol) 1) va 2) dan kelib chiqqandagina formuladir

Mumtoz mantiqda formulani dumaloq qavslarga olish qabul qilingan, biroq biz bunday qilmaymiz. Har qanday formula uchun haqiqiylik jadvalini yaratish mumkin.

I ning berilgan talqinidagi F formulasi qiymati $|F|$ (yoki $|F|_I$, yoki $I(F)$) ni anglatadi.

Formulaning o'zi ham formula bo'lgan bir qismi ushbu formulaning formulachasi deb ataladi.

Ta'rif. Formula agar u harflarning har qanday qiymatida asl qiymatlarni qabul qilsa Tavtologiya deb ataladi.

Boshqacha so'zlar bilan aytganda, tavtologiya – bu o'xshash haqiqiy formuladir.

- Formulaning tavtologiya ekanligini:
- haqiqiylik jadvali bo'yicha;

mantiqiy amaliyotlar xossalaridan foydalangan holda aniqlash mumkin.

Diskret matematika kursidan tavtologiya misollari bo‘lgan asosiy mantiqiy ekvivalentliklar (mantiqiy amaliyot xossalari) ma'lumdir .

1. Kommutativlik: $A \vee B = B \vee A$, $A \wedge B = B \wedge A$

2. Birlashganlik(Assosiativlik):

$A \vee B \vee C = A \vee B \vee C$, $A \wedge B \wedge C = (A \wedge B) \wedge C$

3. Taqsimlovchilik(Distributivlik):

$A \vee B \wedge C = A \vee B \wedge A \vee C$, $A \wedge B \vee C = (A \wedge B) \vee (A \wedge C)$

4. Idempotentlik: $A \vee A = A$, $A \wedge A = A$

5. Ikkiyoqlama inkor qonuni: $A = A$

6. Uchinchini mustasno etish qonuni: $A \vee A = 1$

7. Ziddiyat qonuni: $A \wedge A = 0$

8. De Morgan qonuni:

$A \vee B = A \wedge B$, $A \wedge B = A \vee B$

9. Mantiqiy doimiy qiymatlarga ega amaliyotlar xossalari:

$A \vee 1 = 1$, $A \vee 0 = A$, $A \wedge 1 = A$, $A \wedge 0 = 0$

Bu yerda A, B va C – har qanday harflar.

1-misol. $A \wedge B \rightarrow A$ formula tavtologiya ekanligin isbotlang.

Isbot. Aytaylik, $A \wedge B \rightarrow A$ kiymat 0 ga teng, ya'ni $A \wedge B \rightarrow A = 0$ bu $A \wedge B = 1$ va $A = 0$ teng kuchli ekanligni anglatadi. Bu esa $A = 1$, $B = 1$ va $A = 0$ ga tengdir, ya'ni ziddiyatga ega bo‘lamiz.

Teorema. A va $A \rightarrow B$ formulasi tavtologiya bo ‘lsin. Unda B - formula tavtologiyadir.

Isbot. $P_1, P_2, \ldots, P_k$ bo ‘lsa - formuladagi harflar A va $A \rightarrow B$ harflardir. Bulev funksiyalari nazariyasida barcha bulev funksiyalar va binobarin formulalar ham harflarning aynan bir miqdoriga tobe deb hisoblash mumkinligi isbotlangan edi. $\sigma_1, \sigma_2, \ldots \sigma_k$, qiymatlarning ayrim to‘plamini ko‘rib chiqamiz, bu yerda $\sigma_i < -\{0,1\}, i = 1,2, \ldots k.$ Qiymatlarning ushbu to ‘plamini A va $A \rightarrow B$ formulasidagi tegishli harflar o ‘rniga kiritamiz. Formulalar teorema shartiga ko ‘ra tavtologiyalardir, binobarin $A(\sigma_1, \sigma_2, \ldots \sigma_k) = 1$ ва $A(\sigma_1, \sigma_2, \ldots \sigma_k) \rightarrow B(\sigma_1, \sigma_2, \ldots \sigma_k) = 1$. Aloqa (Implikasiya) haqiqiyligi jadvaliga ko‘ra $B(\sigma_1, \sigma_2, \ldots \sigma_k) = 1$ ga ega bo‘lamiz. $\sigma_1, \sigma_2, \ldots \sigma_k$ qiymatlar

to‘plami erkin bo‘lganligi uchun formula B – tavtologiyadir, shuni isbot qilish talab qilingan edi.

Teopema. formula A – tavtologiya bo‘lsin, P_1, P_2, . . . P_k – formuladagi harflar $A, A_1, A_2, \ldots A_k$ ham formulalardir, bunda yangi formula B=A($A_1, A_2, \ldots A_k$) – tavtologiyadir.

Bu isbot oldingi teorema isbotiga o‘xshashdir.

2.2. Isbotlarning mumtoz usullari

Yuqorida keltirilgan misollar mumtoz mantiqda qo‘llaniladigan isbotlashlar usullarini ko‘rsatadi:

1. Deduksiya to‘g‘risidagi teoremadan foydalanish.
2. Qarama qarshi qo‘yish (Kontrpozisiya) yordamida implikasiya bilan isbotlash.
3. Ziddiyat yordamida isbotlash.
4. Qarshi misol bilan isbotlash.
5. Matematik induksiya usuli.

2.2.1. Deduksiya to‘g‘risidagi teoremadan foydalanish

Deduskiya to‘g‘risidagi teorema ko‘pincha matematik isbotlarda foydalaniladigan usul asosi bo‘lib xizmat qiladi. “agar A, unda B” tasdig‘ini isbotlash uchun A ning to‘g‘riligi B ning to‘g‘riligini ham isbotlaydi. Deduksiya to‘g‘risidagi teorema mulahazalarni hisoblab chiqish uchun to‘g‘ridir, biroq u birinchi tartibdagi nazariya uchun ham to‘g‘riligicha qoladigan bo‘lib chiqadi. Aniqroq qilib aytadigan bo‘lsa, agar birinchi tartibdagi predikatlar formulasi yopiq bo‘lsa (unda har qanday talqinda ularning haqiqiyligi butkul aniqdir), unda ushbu formulalar uchun deduksiya to‘g‘risidagi teoremani tasdiqlash to‘g‘ridir.

Misol. Har bir yaxlit n uchun, agar n –juft bo‘lsa, – n^2 ham juft ekanligini isbotlang.

Isbot. n – juft bo‘lganligi uchun $n = 2 \cdot m$ ko‘rinishida tasavvur qilish mumkin, bu yerda m – yaxlit sondir. Shuning uchun $n^2 = (2 \cdot m)^2 = 4 \cdot m^2 = 2 \cdot (2 \cdot m^2)$, bu yerda $2 \cdot m^2$ – yaxlit son, t.e. n^2 – juftdir.

2.2.2. Kontrpozisiya yordamida implikasiya bilan isbotlash

A → V ko ʻrinishidagi A - yo ʻnaltirishlar bog ʻlovchi bo ʻlgan, V - xulosa/ ko ʻrinishidagi shartli mulohazani ko ʻrib chiqamiz. Ba'zan, ushbu implikasiyani isbotlash o ʻrniga boshlang ʻich mulohazaga teng kuchli bo ʻlgan ayrim boshqa mulohazaning mantiqiy haqiqiyligini aniqlash qulayroqdir. Isbotlashning bunday shakllari isbotlashning bilvosita usullari deb ataladi.

Isbotlashning bilvosita turlaridan eng ko'p qo'llaniladiganlaridan biri - bu kontrpozisiya yordamidagi isbotdir..

$A \to B$ formulasining kontrpozisiya deb, $\neg A \to \neg B$ teng kuchli formula aytiladi. Shuning uchun agar biz kontrpozisiyaning haqiqiyligini aniqlasak, bu bilan boshlang ʻich implikasiyani isbotlaymiz.

Misol. Kontrpozisiya asosida agar m va n–$m \cdot n \leq 100$ singari erkin musbat yaxlit sonlar bo'lsa, unda yo $m \leq 10$, yoki $n \leq 10$ dir.

Isbot. Quyidagi mulohaza boshlang'ich tasdiqqa kontrpozisiya bo'lib xizmat qiladi: "Agar $m > 10$, $n > 10$, bo'lsa, $m \cdot n > 100$" ekanligi yaqqol ko'rinib turibdi.

Kontrpozisiya yordamida isbotlash usulining afzalligi isbotlashning avtomatlashgan usulida, ya'ni isbotlashni teoremalarning isbotlashning maxsus dasturiy tizimlari yordamida kompyuter amalga oshirganda namoyon bo'ladi. Xulosalarni barpo etishda xulosa chiqarish qoidasini shunchaki qo'llagan holda izohlanayotgan xulosa paydo bo'lishini kutish doimo ham maqsadga muvofiq bo'lavermaydi. A→B implikasiyasini isbotlash uchun biz A ga yo'l qo'yganimizda aynan shu holat ko'proq yuz beradi. Biz oxirida B ga erishish uchun A va boshqa iqtiboslarga zanjirli qoida hamda moduc ponens qo'llaymiz. Ammo noto'g'ri yo'ldan ham borish mumkin va unda maqsadimizga aksariyati tegishli bo'lmagan bo'lmagan ko'proq gaplar isbotlanishi mumkin. Ushbu usul to'g'ri to'lqin nomini olgan va agar uni kompyuter uchun dasturlansa hamda chuqurligi cheklanmasa, ko'plab oraliq natijalarni vujudga keltirish tendensiyasiga ega.

Boshqa imkoniyat - kontrapozisiyadan foydalanish va aytaylik $A \to B$ o ʻrniga $\neg A \to \neg B$ isbotlashga urinib ko ʻrish mumkin. Bunda biz

$\neg B$ ga yo ‘l qo ‘yamiz va $\neg A$ ekanligini isbotlashga urinib ko ‘ramiz. Bu go ‘yoki oxiridan bosh tomon qarab orqaga harakat qilishga imkon beradi. Qoida bunday qo‘llanilganda eski xulosa iqtibos rolini o‘ynaydi. Izlashni bunday tashkil etish qaysi natijalar ishga tegishli ekanligini yaxshiroq ko‘rsatishi mumkin. bu maqsaddan izlash deb ataladi.

2.2.3. Qarshidan isbotlash

Ziddiyatga olib kelish isbotlash bilvosita usullarining xususiy holatidir. Isbotlash usuli quyidagi tasdiqqa asoslanadi, Agar G, |S|-F, bu yerda F – har qanday ziddiyat (o‘xshash soxta formula), unda GH-S

(Agar G,¬S dan F chiqarib olinsa, bu yerda F – har qanday ziddiyatdir (o‘xshash soxta formula), unda G dan S chiqarib olinadi).

Ushbu usulda quyidagi teng kuchliklardan fodalaniladi:

$$A \to B \equiv \neg(A \to B) \to (C\&\neg C) \equiv (A\&\neg B) \to (C\&\neg C)$$

$$A \to B \equiv (A \to \neg B) \to \neg A$$

$$A \to B \equiv (A\&\neg B) \to B$$

$A \to B$ isbotlash uchun keltirilgan teng kuchliliklarningn ikkinchisidan foydalangan holda biz bir vaqtning o ‘zida A va $\neg B$ ga yo ‘l qo ‘yamiz, ya'ni xulosa soxtaligini faraz qilamiz:

$$\neg$$
$$A \to B$$
$$= \neg$$
$$\neg A \vee B$$
$$= A\&\neg B$$

Endi biz A dan oldinga ham va $\neg B$dan orqaga ham harakatlanishimiz mumkin. Agar V A dan chiqariladigan bo‘lsa, A ga yo‘l qo‘ygan holda, biz V ni isbotlagan bo‘lar edik. Shuning uchun $\neg B$ ga yo‘l qo‘ygan holda biz ziddiyatga ega bo‘lamiz. Agar $\neg B$ dan $\neg A$ni chiqarib olsak, bu bilan A ziddiyatga ega bo‘lamiz. Umumiy holatda biz qandaydir S aniqlikni kiritgan, oldinga harakatlangan holda ikkala tomondan ham harakat qilishimiz mumkin. Uning inkori $\neg C$ bilan orqaga harakatlanish mumkin. Muvaffaqiyatli holatda bu bizning iqtiboslarimiz mos kelmasligi yoki ziddiyatli ekanligini isbotlaydi. Bundan biz qo ‘shimcha iqtibos $A\&\neg B$ soxta ekanligini va demak unga qarma-qarshi bo ‘lgan $A \to B$ haqiqiy

ekanligi borasidagi xulosaga kelamiz. Qarshidan isbotlash usuli matematikning eng yaxshi qurollaridan biridir.

1 misol. Ikkidan kvadrat ildiz irrasional son ekanligini isbotlaymiz.

Isbot. Ya'ni bizga

$$\sqrt{2} = \frac{p}{q}$$

Nisbati bajarilishi uchun p va q singari ikki yaxlit son mavjud emasligini isbotlashga to‘g‘ri keladi.

Aslida biz

$$\sqrt{2} = \frac{p}{q}$$

ekanligini faraz qilamiz, unda $p^2 = 2q^2$

Biz $\frac{p}{q}$ kasri qisqartirilmaydi deb hisoblashimiz mumkin, aks holda biz avval boshdanoq uni p va q sonlarining eng katta umumiy bo‘luvchisiga qisqartirgan bo‘lar edik. O‘ng tomondan ko‘paytiruvchi sifatidagi 2 mavjud va shuning uchun p^2 juft sondir va demak p ning o‘zi– ham juftdir, chunki toq son kvadrati toq sondir. Bunday holatda p=2$r$ni qo‘yish mumkin. Unda tenglik quyidagi ko‘rinishga keladi:

$$4r^2 = 2q^2 \text{ yoki } 2r^2 = q^2$$

Endi chap tomonda ko‘paytiruvchi sifatidagi 2 borligi uchun q^2 dir, binobarin, q – juftdir. Shunday qilib, p ham, q ham – juft sonlardir, ya'ni 2 ga bo‘linadi, bu esa p/q kasri qisqartirmasligi yo‘l qo‘yilishiga zid keladi. demak, $p^2 = 2q^2$ ning g imkoni yo‘q va 2 dan ildiz rasional son bo‘la olmaydi.

2-misol. Sodda sonlar cheksiz nihoyasizligini isbotlash.

Isbot. Sodda sonlarning cheksiz ko‘pligi mavjudligini va p ulardan eng kattasi ekanligini: 2,3,5,7,11,...,p faraz qilamiz. $N = p! + 1$ sonini belgilaymiz. N soni 2,3,5,7,11, ... ,p laridan har biriga bo‘lishda qoldiqda 1 ni beradi. Oddiy bo‘lmagan har bir son loaqal bir oddiy songa bo‘linadi. N soni biron bir sodda songa bo‘linmaydi, binobarin N ning o‘zi sodda sondir. Shu asnoda N>p. Shu tarzda biz sodda sonlar cheksiz ko‘pligini isbotlovchi ziddiyatga kelamiz.

2.2.4. Kontrmisol bilan isbotlash

Aksariyat matematik gipotezalar o‘z asosida: “A xossasiga ega barcha ob'ektlar V xossasiga ega” shakliga ega, biz buni formula ko‘rinishida

Har qanday X uchun $x(A(x) \rightarrow B(x))$,

Bu yerda A(x) “X A xossasiga ega ekanligini”, B(x) – "x B xossasiga egaligi” predikatini anglatadi. Agar x ning ehtimoliy qiymatlari soni yakuniy bo‘lsa, umuman isbot holatlarni tahlil qilish yordamida, ya'ni har bir ob'ekt uchun gipoteza bajarilishini bevosita tekshirish bilan keltirilish mumkin. Agar ob'ektlar soni yakuniy bo‘lmagan holatda esa, bunday imkoniyat hatto umuman mavjud bo‘lmaydi. Ammo gipotezaning soxtaligini isbotlash uchun loaqal gipoteza bajarilmaydigan bir misol (mazkur holatda kontrmisol deb ataluvchi) keltirish etarlidir.

2.2.5. Matematik induksiya

Matematik isbotlarda ko‘pincha quyidagi tarzda shakllantiriladigan matematik induksiya tamoyilidan foydalaniladi. Agar n natural parametriga tobe bo‘lgan P(n) mulohazasi n=0 uchun isbotlangan va n erkin bo‘lganda P(n) haqiqiyligidan P(n+1) haqiqiyligini asoslashga muvaffaq bo‘linsa, unda P(n) barcha n uchun haqiqiydir.

Induksiya bo‘yicha isbotlashga misol keltiramiz. Isbot, natural sonlarning uch izchil kublari qiymati 9 ga bo‘linishini isbotlash.

Induksiya bazasi. $1^3 + 2^3 + 3^3 = 36$ 9 ga bo‘linadi.

Induktiv yondoshuv. Induktiv o‘tishni amalga oshirish uchun, n uchun isbotlanadigan tasdiqni faraz qilish zarur so‘ngra esa, u n+1 uchun ham to‘g‘ri ekanligini isbotlash zarur. $n^3 + (n+1)^3 + (n+2)^3$ 9 ga bo‘linsin deylik. Unda

$$\begin{aligned}(n+1)^3 + (n+2)^3 + (n+3)^3 \\ &= (n+1)^3 + (n+2)^3 + n^3 + 9n^2 + 27n + 27 \\ &= n^3 + (n+1)^3 + (n+2)^3 + 9n^2 + 27n + 27 \\ &= (n^3 + (n+1)^3 + (n+2)^3) + (9n^2 + 27n + 27)\end{aligned}$$

Biroq, so‘nggi qavsdagi barcha qo‘shiluvchilar 9 ga bo‘linadi, birinchi qavs esa 9 ga taxmin bo‘yicha bo‘linadi. Demak, boshlang‘ich qiymat ham 9 ga bo‘linadi. Shu tarzda biz n dan boshlanadigan qiymatning bo‘linishi $n+1$ boshlanadigan qiymatning bo‘linishini keltirib chiqarishini aniqladik.

Binobarin, tasdiq barcha n uchun isbotlandi.

Shunday qilib, matematik induksiyada induktiv bazis – ko‘rib chiqilayotgan sonlardan eng kichigi uchun xossa bajarilganligi tasdig‘i va induktiv o‘tish –n sonidan n+1 soniga o‘tishni asoslash mavjud.

Matematik induksiya

$$A(0)$$

$$A(0)\&\forall n(A(n) \to A(n+1)) \to \forall n A(n)$$

Matematik induksiya tamoyili mumtoz mantiqda boshlang‘ich tamoyilga ekvivalent bo‘lgan bir muncha qayta shakllantirishlarga yo‘l qo‘yadi.

Bulardan biri– qaytuvchi induksiyadir. Bu yerda o‘tish bir qiymatidan navbatdagisiga emas, balki oldingi barcha qiymatlardan navbatdagisiga o‘tishda yuz beradi, induksiya qadamini $A(n)$ dan $A(n+1)$ ga emas, balki $\forall x<nA(x)$ dan $A(n)$ ga o ‘tkaziladi.

Nazorat savollari

1. Funksional dasturlash paradigmasini ishlab chiqarish uchun qaysi matematik formalizmlar asosga aylandi?
2. Birinchi tartibli mantiq nima?
3. Matematik isbotlarning qanday ko‘rinishlarini bilasiz?

III. BOB. REZOLYUSIYA USULI

3.1. Asosiy ta'riflar

Mazkur bob formula formulalarning mantiqiy natijasi ekanligini isbotlash usulini ko‘rib chiqishga bag‘ishlangan. Bu usul rezolyusiya usuli deb ataladi.

Rezolyusiya qoidasi – bu ziddiyatlarni izlash orqali teoremalarni isbotlash usuliga yuksalib boradigan chiqarib olish qoidasidir; bu mulohazalar mantiqida va birinchi tasdiq mantiqida qo‘llaniladi. YEchim ro‘yxati uchun izchil qo‘llaniladigan rezolyusiya qoidasi mantiqiy ifodalarning boshlang‘ich ko‘piligida, ziddiyat bor yoki yo‘qligi savoliga javob berishga imkon beradi. Rezolyusiya qoidasi 1930 yilda Jak Erbranning (Jacques Herbrand) doktorlik dissertasiyasida birinchi tartibdagi formal tizimlardagi teoremalarni isbatlash uchun taklif etilgan edi. Bu qoida 1965 yilda Jon Alan Robinson tomonidan ishlab chiqilgan.

Ushub usul asosida barpo etilgan $A \vdash B$ chiqarib olinishini isbotlash algoritmlari sun'iy intellektning ko‘plab tizimlarida qo‘llaniladi. Shuningdek, mantiqiy dasturlash tili “Prolog” barpo etilgan poydevor hamdir.

Mantiqiy oqibat to‘g‘risidagi vazifa bajarilish to‘g‘risidagi vazifaga borib taqalishini ta'kidlab o‘tamiz. Haqiqatan ham formula formulalar ko‘pligi bajarilmaydigan holatdagina formulalarning mantiqiy oqibatidir. Rezolyusiya usuli agar yanada aniqroq aytiladigan bo‘lsa, bajarilmaslikni belgilaydi. Bu usulning birinchi o‘ziga xosligidir.

Usulning ikkinchi o‘ziga xosligi shundan iboratki, u erkin formulalar bilan emas, balki diz'yunktlar (yoki elementar diz'yunksiyalar) bilan ish ko‘radi.

Dastlab, mulohazalar mantiqini ko‘rib chiqamiz

Ta'rif. Atomar formula yoki uning inkori Literal deb ataladi.

Ta'rif. Literallar diz'yunksiyasi diz'yunkt deb ataladi

Diz'yunkt bir literaldan iborat bo‘lishi mumkin. Ba'zan biz diz'yunkt ga literallar ko‘pligi sifatida qaraymiz, ya'ni diz'yunksiyaning kommutativligi yoki assosiativligi, va shuningdek idempotentligi yordamida bir-biridan olinadigan diz'yunktlarni farqlab o‘tirmaymiz. Idempotentlik, masalan, diz'yunkt tengligini ham anglatadi.

Ta'rif. Bo‘sh diz'yunkt bu literallarga ega bo‘lmagan diz'yunktdir. Belgilanishi: □.

Ta'rif. **L va ¬L literallar** qarama-qarshi literallar deb ataladi.

3.2. Mulohazalar mantiqi uchun rezolyusiyalar usuli

Mulohazalar mantiqidagi rezolyusiya usuli rezolyusiya qoidasi ga asoslanadi.

Qandaydir formula (birinchi bobda kiritilgan alifbo bo‘yicha barpo etilgan) mavjud bo‘lsin. Ushbu formulani KNSh ga (kon'yuktiv normal shaklga) yozib qo‘yamiz (ulardan har qandayini tanlab olamiz) va endilikda faqat ushbu yangi fomula bilan ishlaymiz.

Shunday qilib bizning formulamiz $F1 \wedge F2 \wedge \ldots \wedge Fn$ ko‘rinishidagi ifodaga ega, shu asnoda Fi ifodalaridan birontasi kon'yunksiyaga ega emas.

Har qanday kon'yunktiv normal shaklni (KNSh) diz'yunktorlarning ayrim soni kon'yunksiyasiga olib kelish mumkin. Haqiqatan ham takrorlanuvchi literallar va ularning inkorini idempotentlik qonuni $xi \vee xi = xi$ va mustasno etilgan uchinchi qonuni $xi \vee \neg xi = 1$ dan foydalangan holda Fi ning barcha ifodalaridan chiqarib tashlash mumkin.

Agar $xi \vee F$ и $\neg xi \vee G$ formulasi chiqarib olingan bo‘lsa, $F \vee G$ formulasi

rezolyusiya usuli bilan chiqarilgan hisoblanadi.

3.2.1. Rezolyusiya

Robinson kompyuter yordamidagi isbotlash jarayonini avtomatlashtirishda qo‘llaniladigan xulosa qoidalari inson foydalanadigan xulosa qoidalariga lbatta mos kelishi shart emasligi borasidagi xulosaga keladi. U xulosaning umumqabul qilingan qoidalari, masalan, modusponens qoidasi isbotlash tartibotining har bir qadamini inson hissiy kuzatib borishi uchun maxsus “zaif” yaratilganligini ko‘radi. Robinson xulosa qilishning kuchliroq qoidasini kashf qildi va buni rezolyusiya (yoki rezolyusiyalar qoidasi) deb atadi. Bu qoidani inson anglashi qiyin, biroq kompyuterda samarali qo‘llanilmoqda (rezolyusiya qoidasi sillogizmning diz'yunktiv qoidasiga o‘xshashdir).

Rezolyusiya qoidasi hisoblab chiqish maqsadlari uchun o‘ta jozibadordir, chunki uning o‘zi predikatlar mantiqi ibora shakllari uchun xulosa qilish qoidalarining to‘liq ko‘pligidir. Boshqacha so‘zlar bilan aytganda birgina rezolyusiya qoidasining o‘zidan foydalanilganda ibora shaklida keltirilgan aksiomalar ko‘pligidan har qanday natijani chiqarib olish mumkin.

3.2.2. Rezolyusiya qoidasi

Rezolyusiya qoidasi quyidagi tarzda amal qiladi. Ikki ibora agar ulardan biri ijobiy neytralga ega bo‘lsa, boshqasi esa predikatlarning aynan bir xil alomati va dalillarning bir xil miqdoriga ega bo‘lsa, hamda ikki literalning dalillari bir-biri bilan unifikasiya qilinishi (ya'ni muvofiqlashtirilishi) mumkin bo‘lsa, bir-biri bilan rezolvirovat qilinishi mumkin. Ikki iboradan tashkil topgan nazariyani ko‘rib chiqamiz:

$P(a) \vee \neg Q(b, c)$ (1)

$Q(b, c) \vee \neg R(b, c)$ (2)

(1) iborada salbiy literal $\neg Q(b, s)$, (2) iborada esa mos keluvchi ijobiy literal $Q(b, s)$ borligi hamda ikki literallarning dalillari moslashtirilishi mumkin bo‘lganligi bois (ya'ni, b b bilan, s esa s bilan moslashadi) (1) ibora (2) ibora bilan rezolyusiyalanishi mumkin. Buning natijasida rezolventa deb ataluvchi (3) ibora yuzaga keladi. U;

$$P(a) \vee \neg R(b, c)$$

Nazariyasining yangi iborasiga aylanadi.

Ushbu rezolyusiya bajarilganidan so‘ng nvbatdagi rezolyusiyalarda (1), (2) va (3) iboralarining har biridan foydalanish mumkin. (4) va (5) iboralar bir-biri bilan rezolyusiyalashmaydi. Chunki Q literallarning dalillari moslashishga bo‘ysunmaydi:

$P(a) \vee \neg Q(b, c)$ (4)

$Q(c, c) \vee \neg R(b, c)$ (5)

3.2.3. O‘zgaruvchan qiymatlarni ixchamlashtirish

O‘zgaruvchi qiymatlarning aniq kvantifikasiyasi ibora shaklida qo‘llanilmaydi., ammo barcha o‘zgaruvchan qiymatlar mavjudlik kvantorlari bilan tasniflangani aniq emas. masalan,

$$Q(x, y) \vee \neg R(x, y)$$

iborasida:

$$\forall x \forall y (Q(x,y) \forall \neg R(x,y))$$

kvantorlar mavjudligi nazarda tutiladi.

Agar dalil sifatida o‘zgaruvchan qiymat yuzaga chiqsa u har qanday konstanta bilan moslashadi. Agar aynan bir iborada o‘zgaruvchan qiymat bir martadan ortiq uchrasa, ushbu o‘zgaruvchan qiymat rezolyusiya jarayonida konstanta bilan moslashsa, rezolvent boshlang‘ich iborada ko‘rib chiqilayotgan o‘zgaruvchan qiymatlar bor joylarda mazkur konstantaga ega bo‘ladi.

Masalan,

$P(a) \vee \neg Q(a,b)$ (6)

$Q(x,y) \vee \neg R(x,y)$ (7)

iboralar rezolyusiyalashadi, chunki Q literali dalillari moslashadi. Shu asnoda o‘zgaruvchan qiymat x a konstanta bilan, o‘zgaruvchan qiymat u — b konstanta bilan moslashadi. (8) iborada

$P(a) \vee \neg R(a,b)$ (8)

ya'ni, rezolventda (7) iboradagi R dalillar bo‘lib xizmat qilgan o‘zgaruvchi qiymatlar endi konstantalarga almashtirilganligiga e'tibor qarating.

3.2.4. Bo‘sh ibora

Ikki iborani ko‘rib chiqamiz:

$P(a)$ (9)

$\neg P(a)$ (10)

(9) ibora — bu shartsiz xulosa, (10) ibora esa — bu xulosasiz shartdir. Bir nazariyada (9) va (10) iboralarining mavjudligi ziddiyatdir. Agar (9) va (10) iboralar bir- biri bilan moslashsa, qo‘lga kiritilgan rezolvent bo‘sh ibora deb ataladi. Agar nazariya tarkibiga kiruvchi ikki iborani rezolyusiyalashda bo‘sh ibora qo‘lga kiritilsa, bu nazariya izchil bo‘lmasligi lozim. Semantika nuqtai nazaridan bo‘sh iboraning kiritilish imkoniyati ushbu nazariyaning xerbrandcha modelini yaratib bo‘lmasligini anglatadi.

3.2.5. Rezolyusiyaga asoslangan algoritm

Asosiy muammo qandaydir ibora nazariya natijasi ekanligi yoki yo‘qligini isbotlashdan iborat. Isbotlashning asosiy usuli sifatidagi rad etish tartibotini qo‘llash (yuqoriga qarang) bu vazifani nazariy izchil yoki izchil emasligini aniqlash zaruratiga taqagan holda soddalashtiradi. YEchimni izlash jarayonini avtomatlashtirish uchun iboralar ko‘pligi ziddiyatliligini topishga imkon beruvchi rezolyusiya qoidasiga asoslangan samarali algoritm topish yaxshi bo‘lgan bo‘lur edi.

Nazariya ibora shaklida ifodalanganligini tasavvur qilamiz. Ushbu nazariyaning barcha iboralari bir iboradan tashqari bir-biriga mos keladi. ushbu izlanadigan iboraning mavjudligi butun nazariyanini izchil bo‘lmagan nazariyaga aylantiradi. Tasodifiy qonunga ko‘ra, nazariya iboralariga nisbatan rezolyusiya qoidasini qo‘llovchi qandaydir faol kuch (inson yoki mashina) mavjudligini tasavvur qilamiz. Har bir rezolyusiya bajarilganidan so‘ng qo‘lga kiritilgan rezolventa nazariyaga qo‘shiladi. Agar bo‘sh ibora to‘plansa, bu nazariya izchil emasligini anglatadi va unda ushbu faol kuch o‘z harakatini to‘xtatadi. Ko‘rib chiqilayotgan nazariya haqiqatan izchil bo‘lmaganligi bois faol kuch ertami-kechmi bo‘sh iborani jamlaydi va bu bilan nazariyaning izchil emasligini ko‘rsatib beradi. Ammo rezolyusiyalardan aksariyati yoki ortiqcha yoki izlanayotgan iboraga tegishli emas bo‘lib chiqadi. Nazariyaning izchil emasligininini topishning samarali algoritmini yaratish uchun mazkur jarayonga cheklash qo‘yish va uning faolligini bo‘sh iboraning paydo bo‘lishiga olib keladigan eng katta extimolliklarga ega qismlarga yo‘llash zarur.

Rezolyusiya qoidasidan foydalanishda birdan ortiq vazifalarni hal qilish strategiyasini qo‘llash mumkin. Mazkur bo‘limning qolgan qismida pastlab boruvchi (yoki teskari) strategiya ko‘rib chiqiladi. Ushbu strategiyada S yagona iborasi T iboralarning mavjud ko‘pligi natijasi ekanligini topish maqsadi qo‘yiladi. T iboralar ko‘pligi ziddiyatli emasligi talab qilinadi. Algoritm quyidagi tarzda ishlaydi. Dastlab iboralarning mavjud ko ‘pligiga tekshirilayotgan iboraning inkori ¬S qo ‘shiladi. Shu asnoda T Ko ‘piligidan plyus ¬S iboradan tashkil topgan T1 iboralarning yangi ko ‘pligi shakllanadi. Agar algoritm bo ‘sh iborani T′ dan chiqarishga imkon bersa, unda T′ ¬S iborasi mavjudligi bois

izchil bo'lmagan bo'lib chiqadi va shuning uchun S T ning oqibati bo'lishi lozim. Bu jarayonni quyidagi misolda ko'rsatish mumkin.

Pastlovchi usul bilan rezolyusiyaga misol

Iboralarning ko'pligi mavjud bo'lsin

$P(a) \vee \neg Q(a,b)$ (6)

$Q(x,y) \vee \neg R(x,y)$ (7)

$S(b)$ (11)

$R(a,b)$ (12)

$P(a)$ (13)

iborasi iboralarning mavjud ko'pligi natijasi ekanligini aniqlash zarur. Birinchi qadamda boshqa iboralarga R(a)iborasining inkorini qo'shamiz. Natijada quyidagilarga ega bo'lamiz:

$P(a) \vee \neg Q(a,b)$ (6)

$Q(x,y) \vee \neg R(x,y)$ (7)

$S(b)$ (11)

$R(a,b)$ (12)

$\neg P(a)$ (14)

Rezolyusiya ushbu algoritmining harakati iborani (14) iboralarning mavjud ko'pligiga qo'shish natijalariga jamlanadi. Bu quyidagi ikki qoida yordamida erishiladi:

1) birinchi bajariladigan rezolyusiyada iboraning faqat qo'shilgan inkoridan (ya'ni (14) iboradan) foydalanish zarur.;

2) har bir navbatdagi rezolyusiyada oldingi rezolyusiyaning rezolventi ishtirok etishi lozim (bu algoritmning maqsadsiz "adashib" yurishining oldini oladi).

Birinchi qoidaga muvofiq birinchi rezolyusiyada (14) ibora ishtirok etishi lozim. U (6) ibora bilan rezolyusiyalashadi. buning natijasida:

$\neg Q(a,b)$ (15)

qo'lga kiritiladi.

(2) qoidaga muvofiq navbatdagi rezolyusiyada (15) ishtirok etishi lozim. Bu ibora G) ibora bilan rezolyusiyalashadi.

$\neg R(a,b)$ (16)

iborasi buning natijasiga aylanadi.

(R uchun boshlang'ich bo'lgan formuladagi barcha o'zgaruvchan qiymatlar (16) iborada konstantalarga almashtirilganligiga e'tibor

qarating.) (16) ibora (12) ibora bilan rezolyusiyalashadi, bu bo‘sh iboraning shakllanishiga olib keladi. Bu ziddiyat topilganligini anglatadi. Iboralarning mavjud ko‘pligiga ¬R(a) qo‘shish ziddiyatga olib kelganligi bois R (a) iboralar ko‘pligi natijasidir.

3.2.6. Vazifalarni hal qilish strategiyasi

Endigina tasvirlangan vazifalarni hal qilish strategiyasi quyidagi xossalar bilan xususiyatlanadi. Bu strategiya pastlab boruvchi strategiyadir, chunki u hal qilish jarayonini xulosani rad etishdan (ya'ni, ¬R (a) iboradan) boshlaydi. So‘ngra esa, hal qilish jarayonida nazariyaning qolgan iboralaridan foydalaniladi, bu bo‘sh ibora chiqarib olinmaguniga qadar davom etadi. Hal qilishning mazkur strategiyasi “dastlab chuqurlikka” izlash strategiyasi deb ataladi, chunki so‘nggi rezolyusiya natijasi doimo undan keyin keluvchi rezolyusiyada qo‘llaniladi.

Nazorat savollari

1. Mulohazalar mantiqidagi rezolyusiya usulining mohiyati nimada?
2. Mulohazalar mantiqidagi va peridkatlar mantiqidagti rezolyusiya usullari nima bilan farqlanadi?
3. Qanday holatda mulohazalar mantiqining (predikatlar mantiqining) B formulasi mulohazalar mantiqining(predikatlar mantiqining) A formulasi mantiqiy natijasi bo‘ladi?
4. Formal aksiomatik nazariya sifatidagi mulohazalarni hisoblab chiqishning asossiy xossalari qanday?
5. Mulohazalarni hisoblab chiqish aksiomalarining ko‘pligi cheksizmi yoki cheklanganmi?
6. Formal aksiomatik nazariya siftiadagi birinchi tartib predikatlarini hisoblab chiqishning asosiy xossalari qanday?

IV. BOB. DEKLARATIV DASTURLASH VOSITALARI

Deklarativ tillardagi dastur ob'ektlar va ular o‘rtasidagi aloqalarni tasvirlashdan ibortadir. Funksional tillarda ushbu aloqalar funksiyalar bilan, mantiqiy tillarda esa munosabatlar (predikatlar) bilan taqdim etalidi.

Funksional dasturlash nuqtai nazaridan dastur kiritishga va chiqishni qaytarishga qo‘llanuvchi funksiyadir.

output = program(input)

mantiqiy dastur - kirish va chiqish o‘rtasidagi munosabatdir.

program(input, output)

Bir qarashda paradigmalar o‘rtasidagi ushbu farq arzimas bo‘lib ko‘rinishi mumkin, chunki bir tomondan, har qanday funksiya uning dalillari va qiymatlari o‘rtasidagi munosabatning xususiy holati, xolos. Boshqa tomondan esa, har qanday munosabatni haqiqiylik qiymatlarning ko‘pligidagi fuknsiya sifatida ko‘rib chiqish mumkin. Ammo yanada jiddiyroq farqlar mavjud. Funksional tillardagi funksiyalar “bir yo‘nalishlidir”. Ular dalillarni oladi va natijalarni qaytaradi.

Mantiqiy tillardagi kirish va chiqish o‘rtasidagi farq shartlidir. Biz kutilayotgan chiqishni ko‘rsatishimiz va buni ta'minlaydigan kirishni olishimiz mumkin. Bulardagi natija , albatta bir ma'noli belgilanmaydi.

Ushbu farqlarni soddagina misol bilan namoyish etish mumkin.

Mintaqalarning maydonini belgilovchi ikki dasturni ko‘rib chiqamiz.

Funksional dastur	Mantiqiy dastur
area(Eurasia) = 55	area(Eurasia,55)
area(Africa) = 29	area(Africa,29)
area(NorthAmerica) = 20	area(NorthAmerica,20)
area(SouthAmerica) = 18	area(SouthAmerica,18)
area(Australia) = 7	area(Australia,7)
area(Antarctica) =14	area(Antarctica,14)

Endi funksion dasturga area (Eurasia) so‘rovi bilan murojaat qilish va 55 javobini olish mumkin.

Mantiqiy dasturga so‘rovlar har xilroqdir: area (Eurasia,a) $\{a = 55\}$ javobini beradi, area эca (m,29) – $\{m = Africa\}$. area (m,a) so‘rovi ehtimoliy olti juftlikdan har birini, biroq cheklovlar qo‘shgan hoda chiqarib beradi, biz yo‘l qo‘yiladigan natijalar sohasini toraytirishimiz mumkin. Area (m,a) ga ehtimoliy javoblar, $a > 20 - \{m = Eurasia, a =$

55} ва $\{m = Africa, a = 29\}$, area (m,a) га эса, $a > 20, a < 40$ yagona ehtimoliy javobni: $\{m = Africa, a = 29\}$ olamiz. Bu Xoar ning iborasiga ko‘ra "farishtacha nodeterminizm" dir hamda mantiqiy dasturlashning asosiy kuchini tashkil etadi.

Ushbu ikki tarzni birlashtirishga intilishlar amalga oshirilmoqda. Yondoshuvlardan biri -"funksional-mantiqviy-dasturlash" bo‘lib, undagi mantiqiy munosabatlar funksiyalar bilan beriladi. Biroq so‘rovlar mantiqiy tillardagidek amalga oshriladi, ya'ni $area(m) = 20 \rightarrow \{m = NorthAmerica\}$ ko‘rinishidagi murojaatga yo‘l qo‘yiladi.

Hisoblab chiqishlar boshqaruv strategiyasida yanada nozik farqlar namoyon bo‘ladi. Masalan, funksional tillarni odatda "dangasa" (ifodalash qiymatini hisoblab chiqishni bu aniq talab qilinmagunicha chetga surib qo‘yadigan) va “chaqqon” tillarga bo‘linadi. Mantiqiy va xususan funksional-mantiqiy tillar uchun turli tumanlik kengroqdir. Parallel hisoblab chiqishlar holatida yanada ko‘proq turlar vujudga keladi.

Tilning boshqa bir xususiyati unda qo‘llaniladigan ma'lumotlar tizimidir. Umuman olganda deklarativ tillar uchun ma'lumotlarni shakllantirishga ham “deklarativ” yondoshuv xosdir. Ya'ni ma'lumotlar tuzilmasi tushunchasi kompyuter xotirasida uni taqdim etishga umuman bog‘liq emas, ko‘rsatkichi sifatidagi tushuncha butkul yo‘q. Massivlar ham kam uchraydi. Eng keng tarqalgan tuzilmalar – ro‘yxatlar va shajaralardir.

Ayrim tillar ma'lumotlar turi tushunchasini butkul nazardan qochiradi. Ulardagi o‘zgaruvchan qiymatlar nomlari qandaydir tur bilan jim turish bo‘yicha bilan bog‘lanmaydi. Bunday tillar tursiz tillar, shuningdek “dinamik turlashga ega tillar” yoki “yashirin turlarga ega tillar” deb ataladi. Bu ma'noda Lisp va Prolog eng radikal tillardir. Bularda dasturlar va ma'lumotlarni taqdim etish shakllari bir xil – ramziy ifodalardir. Bu dasturga boshqa dasturlarga ishlov berish va xatto o‘zini qayta o‘zgartirishga imkon beradi.

Qanday bo‘lsada, turlar tabiiy tarzda xatto ob'ektlar ulardan foydalanishga muvofiq tasniflanadigan tursiz tillarda ham paydo bo‘ladi, biroq qaysi biri yaxshi turlarni apriori mavjud deb hisoblashmi yoki yagona sohadan ob'ektlar ko‘pligini shakllantirishmi? Bu masala hal etilmaganicha qolmoqda, chunki tilning ishonchliligi va moslashuvchanlgi o‘rtasidagi (ehtimol hal qilinmaydigan) ziddiyatni aks ettiradi. Turlarni nazorat qilish zarurati to‘g‘risidagi munozaralar bugun ham davom

etmoqda, biroq tilning amaliy muvaffaqiyatini ko‘pincha yon berishga ozmi ko‘pmi muvaffaqiyatli intilish ko‘proq belgilaydi. Bu ma'noda Xindli-Milner – turlari tizimi eng qiziqarli misollardan biridir. Uning ko‘rinishlari ko‘plab funksional tillarda qo‘llaniladi.

Tillarning yana bir muhim hossasi modullikni saqlab turishdir. Dastlabki tillar bu ma'noda zaif rivojlangandir. Bular nomlarning Si ruhidagi “yassi” makoniga ega. SML va OCAMLdagi modullar tizimi eng rivojlangan, biroq ko‘plar buni o‘ta murakkab hisoblaydi. Ehtimol Modullar-2 tarzidagi sodda modulli mexanizmdan foydalanuvchi Haskell, Mercury va boshqalar oltin o‘rtaliqni tashkil etar. Amaliy ehtiyojlar uchun shuning o‘zi butkul etarlidir

Quyida eng ommalashgan deklarativ tillar va menga qiziqarli bo‘lib ko‘ringan ayrim dasturlash tizimlari sanab o‘tilgan.

4.1. Mantiqiy tillar

Prolog – amalda keng qo‘llaniladigan yagona mantiqiy til. Ko‘plab qo‘llanishlarga ega.

- Ikki tijorat tizimlari Quintus Prolog va SICStus Prolog hozirgi vaqtda rivojlanmoqda hamda SICS (Swedish Institute of Computer Science) bilan qo‘llab-quvvatlanmoqda. Ishlanmalar orasida vositalarning katta to‘plami va o‘ta keng kutubxonalar mavjud. Aksariyat ishlab chiqaruvchilar SICStus dan foydalanadi.
- Strawberry Prolog Prolog bilan tanishish va Windows uchun uncha katta bo‘lmagan dasturlarni yaratishga yaxshi mos keladi, biroq jiddiy vazifalar uchun o‘ta past unumdorlikka ega.
- SWI Prolog – asosan qulay muhiti va grafik interfeys yaratish uchun ko‘chma kutubxona tufayli ancha ommalashgan tizimdir.
- B-Prolog – bayt-kodga asoslangan amalga oshirishlardan eng jadalidir. Ayrim kengaytirishlarni: yakuniy domenlarda CLP(FD) cheklanishlarni va natijalarni eslab qolishni (tabling) – memoizasiya (memoiazation) sifatidagi funksional dasturlashda mashhur texnikani o‘z ichiga oladi.
- XSB – qachonlardir mashhur bo‘lgan Stony Brook Prologning rivojlantirilgani. Bu ham tablingni saqlab turadi. Buning qiziqarli

o‘ziga xosligi HiLog ikkinchi tartibdagi tildan foydalanish imkoniyatidir.

- GNU Prolog – ikkilanma koddagi kompilyator. Tezligi bo‘yicha SICStus dan salgina orqada qoladi.

Sterling va Shapironing kitobi [12] – Prolog bo‘yicha eng yaxshi darslik va mantiqiy dasturlashga a'lo darajada kirishdir. Til to‘g‘risidagi tasavvurni M.N Morozovning [16] ma'ruzalar kursidan olish mumkin. Mantiqiy dasturlash nazariyasiga [10] va [11] bag‘ishlangan.

VisualProlog – Daniya firmasi bo‘lgan PrologDevelopmentCenterning mahsulotidir. Ilgari Turbo Prolog (Borland) va PDC Prolog nomi ostida tarqalgan. O‘z nomiga qaarmasdan bu Prologning jori y etilishi emas, balki turlarning qat'iy nazoratiga ega bo‘lgan butkul o‘ziga xos tildir. Afsuski, polimorfizmning yo‘qligi turlar tizimini kam ifodaliligiga aylantirmoqda. So‘nggi versiyalarda OYD saqlab turish paydo bo‘ldi.

Mercury – katta dasturiy tizimlarni yaratishga mo‘ljallangan ancha murakkab tildir. Prolog asosida yaratilgan, biroq ayrim deklarativ o‘ziga xosliklar olib tashlangan. Muqobillik sifatida kompilyatorga yanada samarali kodni jamlashga imkon beruvchi turlar deklarasiyasi kiritilgan. Turlarning rivojlangan polimorf tizimga ega. Juda yaxshi kompilyator. Umuman dastlabki o‘rganish uchun bu til murakkabroq, biroq sanoat dasturlashi uchun o‘ta yaxshidir.

Godel – turlarning qat'iy nazoratiga ega bo‘lgan eksperimental mantiqiy tildir. Bu Prologga deklarativ muqobillik yaratishga urinishdir. Metadasturlash vositalariga sezilaril e'tibor qartilgan. Mantiqiy dasturlashni o‘rganish uchun ajoyib vosita, afsuski Windows uchun qo‘llab quvvatlash yo‘q va rejalashtirilmagan.

4.2. Funksional tillar

Lisp eng keng qo‘llaniladigan funksional tilligacha qolmoqda. O‘z rivojlanishining uzoq tarixida Lisp ko‘plab lahjalarni yaratdi. Ularning deyali barchasi "Some Lisp" ko‘rinishidagi nomga egaligiga qaramasdan, ular o‘rtasidagi farq Algol va Pask yohud C va *C++* o‘rtasidagidan kam bo‘lmagan sezilarli farqlar mavjud. Shuning uchun ularni har xil tillar deb hisoblash to‘g‘riroqdir.

Tillarni standartlashga urinish bormoqda. Ulardan eng mashhuri - ANSI Common Lisp standartidir. Afsuski, bu o'ta murakkab til bo'lib chiqdi. uning ta'rifi 1000 sahifadan ortiqroqni tashkil etadi. Ham tijorat (Allegro, Harlequin, Corman), ham bepul (GnuCL, CLISP, CMUCL) joriy etish imkoniyatlari mavjud.

ISLISP – xalqaro standart (ISO) sifatidaqabul qilingan Lispning o'ta ixcham lahjasidir. OpenLisp , TISL, OKI ISLISP tarzida joriy etish imkoni mavjud.

EuLisp Lispning bir guruh evropacha foydalanuvchilari tomonidan dasturlashning Common Lisp singari beso'naqay bo'lmagan rivojlangan tizimini yaratish umidida ishlab chiqilmoqda. Til hali ishlab chiqish bosqichida turibdi, biroq TELOS (The Eulisp Object System – CLOS o'xshash) ni o'z ichiga olgan bir necha joriy etish (EuScheme, Feel, Youtoo) turlari hozirda mavjud.

Scheme – Lispning zamonaviylashtirilgani, bir muncha sodda va xatto nozik tildir. Tadqiqot ishlarida va o'rganish uchun faol qo'llanilmoqda. Scheme tarafdorlari "til to'g'risidagi ma'lumot" hajmi Common Lisp ta'rifiga ko'rsatma hajmidan ancha kichik ekanligini (R4RS, R5RS ~50 sahifa) ekanligini ta'kidlashni yoqtirashadi.

Schemening 50 dan ortiq joriy etilishlari mavjud. Bulardan ayrimlari birlashtirilgan muhitlarga, (DrScheme, WinScheme48, 3DScheme i EdScheme) ega. Anchagina samarali kompilyatorlari (Gambit, Bigloo) bor.

O'rganish va tajribalar uchun Scheme48, WinScheme48, XLISP, MzScheme, DrScheme mos keladi.

Scheme muvaffaqiyatiga Abelson va Susmanning kitobi [29]ancha katta imkon yaratdi. Bu funksional dasturlash dasturi emas, biroq dasturlash bo'yicha eng yaxshi kitoblardan biridir.

Lispning funksional kichik ko'pliklari Xendersonning darsligida foydalanilgan. Lisp bo'yicha rus tilida bir necha kitoblar nashr qilingan.

Erlang Ericsson kompaniyasi tomonidan ishlab chiqilgan qiziqarli tildir. Lispga o'xshash tarzda turga ega emas, biroq namuna bilan taqqoslashga asoslangan va Prologni bir muncha eslatuvchi o'ta ifodali sintaksisga ega. Bu uni o'zlashtirishni bir muncha osonlashtiradi. Tilning o'ziga xosligi aniq vaqt tizimlarida paralel dasturlashdir. Bu til CSP Xoa notasiyasiga yaqin bo'lgan o'zaro haraktalanuvchi jarayonlani tasvirlash

uchun vositalarni o‘z ichiga oladi. Joriy etishda boy kutubxonaga ega va ancha faol rivojlanmoqda.

ML: StandardML va CAML(CategoricalAbstractMachineLanguage) oilasiga mansub ikki tildir. Bular hisoblab chiqishning “chaqqon” strategiyasiga ega bo‘lgan funksional tillardir. Ularda turlar va modullarning rivojlangan tizimi mavjud. Oddiy imperativ vositalar: o‘zlashtirish, davriyliklar, istisnolarga ishlov berish ham bor. Ikki lahja o‘rtasidagi farqlar asosan sintaktik farqlardir.

Standard ML

- Moscow ML - CAML Light asosida yaratilgan ommalashgan kompilyator, shu tufayli SML dan keng foydalana boshlandi.
- Poly/ML ham SML ga eksperementlar uchun yomon bo‘lmagan vositadir. Grafik interfeysli kutubxona taklif etiladi, biroq bu hozircha unchalik barqaror emas (loaqal Windows versiyasi).
- SML/NJ (Standard ML of New Jersey) - SMLning mumtoz joriy etilishidir. Yaxshi optimallashadigan, biroq ancha qo‘pol kompilyator.
- MLton - SML/NJ singari aynan bir xil nuqsonlarga ega.
- MLj - JavaVM bayt-kodidagi kompilyator. SMLning kichik ko‘pligini amalga oshiradi, biroq uncha sifatli bo‘lmagan darajada.
- SML.NET buyuk va dahshatli bo‘lib, SMLni chetlab o‘ta olmadi.
- CAML
- Objective CAML bayt-koddagi kompilyatorga va CAMLni ob'ektli-yo‘naltirilgan kengaytirish uchun optimallashtiruvchi kompilyatorga ega.
- CAML Light – bayt-koddagi kompilyatordir. OCAML paydo bo‘lishi bilan bog‘liq holda eskirgan deb hisoblanishi mumkin, biroq rivojlanishda davom etmoqda. O‘quv muassasalarida keng qo‘llanilmoqda, chunki juda tez ishlaydi va zahiralarga talabchan emas.
- F# - NET uchun maxsus CAML asosida yaratilgan ML ning lahjasidir.

Haskell – jadal ommalashib borayotgan tildir. Xalqaro qo‘mita tomonidan dasturlashning sof funksional industrial tili sifatida ishlab chiqilmoqda. Bu til garchi murakkab bo‘lib ko‘rinsada, ancha qat'iy va ifodali tildir. Asosiy o‘ziga xosliklari: qat'iy funksionallik, (imperativ operatorlar mavjud emas), dangasa hisoblab chiqishlar, turlarning va turlar

toifalarining polimorf tizimi – funksional dasturlashga OYD ayrim konsepsiyalarini kiritishga urinishdir.

- Hugs interpetator Haskell. O‘rgatish uchun aynan shuni tanlagan yaxshi.
- GHC – Glasgow Haskell Compiler (inglizcha Glazgodan Haskell tilining kompilyatori) — butun jahon bo‘ylab yig‘ilgan ko‘p sonli ishlab chiqaruvchilarning erkin ishchi guruhi tomonidan ishlab chiqarilayotgan va Glazgo universiteti laboratoriyasidan muvofiqlashtiriladigan Haskell dasturlash funksional tilining bugungi kundagi eng qudratli va rivojlangan kompilyatorlaridan biridir.
- nhc98 - ghc singari samarali bo‘lmasada kompyuterga ancha kamroq talablar qo‘yadi.
- Mondrian – ob'ektli yo‘naltirilgan kengaytirish bo‘lib, .NET uchun MSIL da Haskell kodini kompillyasiya qilishga imkon beradi.

Na **www.haskell.org** da Haskell ning ta'rifi va o‘quv materiallari mavjud. Tilga ruscha kirishni Roman Dushkin va Anton Xolmevning kitoblarida topish mumkin.

Haskell asosida ko‘plab tillar yaratilgan. Bular asosan tilni parallel (DFH, GPH, pH, Goffin, Eden) yoki imperativ (Mondrian) vositalar bilan kengaytrishdir. OYD (O‘Haskell) imkoniyatlarini qo‘shishga (PolyP,Cayenne) turlari tizimi ifodaliligini boshqa usullar bilan ortirishga urinishlar mavjud.

Haskellga juda o‘xshash bo‘lgan Clean tili Neymegen universitetida ishlab chiqilgan. Samarali kompilyator va uncha yomon bo‘lmagan kutubxona (xususan grafik interfeys) uni aniq vazifalarni hal qilish uchun butkul yaroqlilikka aylantirmoqda.

Hope – o‘ta sodda til va ayni vaqtda funksional tillarning barcha muhim o‘ziga xosliklariga ega, keng qo‘llanilmaydi. Funksional dasturlashga o‘qitish uchun foydalanilmoqda. Ros Paterson (Hope) sahifasida tilning tasvirini va dangasa interpretatorning boshlang‘ich matnini topish mumkin. DOSuchun ancha eski va noqulay bo‘lgan ichope interpretatori ham xali mavjud. Unda WEB interfeys orqali ham dasturlarni bajarish mumkin.

RYEFAL (RYEkursivnx Funksiy ALgoritmicheskiy) — ramziy hisoblab chiqishlarga: ramziy satrlarga (masalan, algebraviy

hisoblashlarga) ishlov berish bir tildan (sun'iy yoki tabiiy) boshqa tilga tarjima qilishga mo'ljallangan; sun'iy intellekt bilan bog'liq muammolarni hal qilishga mo'ljallangan dasturlarning eng eski fuknsional tillaridan biridir. U o'zida katta va murakkab dasturlarni yozishga amaliy mo'ljallangan matematik soddalikni birlashtiradi. Bu tilning farqli jihati namuna bilan taqqoslashdan foydalanish va funksiyalarni belgilashning asosiy usuli sifatidagi termlarni qayta yozib olishdir.

Ushbu tildan foydalanuvchi Fild va Xarrison [30] darsligi, funksional dasturlash bo'yicha eng yaxshi va rus tilida nashr qilingan eng yaxshi kitoblardan biridir.

4.3. "Multiparadigmatik" tillar

Curry – Haskell asosida yaratilgan funksional-mantiqiy tildir, uning aksariyat xossalarini meros qilib olgan bo'lib, biroq mantiqiy o'ziga xosliklarni ham qo'shadi. Xanuzgacha ishlab chiqarish bosqichida turibdi.

Escher – yana bir funksional-mantiqiy tildir. Dastlab funksional xossalar qo'shga holda Godel asosida yaratilgan edi, biroq keyinchalik ancha o'zgardi. Hozirgi vaqtda Curry ga juda o'xshashdir, biroq asosan hisoblab chiqish strategiyasi bilan farq qiladi.

Oz – bir til doiralarida turli paradigmalarning asosiy o'ziga xosliklarini birlashtirishga urinishdir. U o'z ichiga toifalarni meros qilib olish, faol ob'ektlar, oliy tartib funksiyalari, determinasiyalashmagan hisoblab chiqishlarni o'z ichiga oladi. Hozirgi vaqtda bir qator tadqiqotchilik guruhlari Mozart dasturlash muhitini ishlab chiqishga birlashgandir..

Poplog – dasturlashning ko'p tilli muhitini yaratish yo'li bilan turli paradigmalarni birlashtirish misolidir. O'z ichiga Pop-11, Common Lisp, Standard ML va Prolog tillarining kompilyatorlarini oladi.

Nazorat savollari

1. Dasturlashning "dangasa" funksional tillari "chaqqon" tillardan nimasi bilan farqlanadi?

2. Deklarativ dasturlashda ma'lumotlarning qanday tuzilmalari eng keng tarqalgan?

3. Dasturlashning qaysi tillari tursiz tillar deb ataladi?

4. Nima uchun funksional dasturlash sun'iy intellektda o‘z qo‘llanishining bevosita sohasini topgan?

5. Qanday oddiy vazifalar fuknsional dasturlash paradigmasi doiralarida taklif etiladigan usullar bilan eng oson hal qilinadi?

V. BOB. MANTIQIY DASTURLASH

Mantiqan mulohaza yuritganda, mantiqiy dasturlash mantiqqa asoslangan dasturlashdir. Bu ko‘pincha mantiq tilidan dasturlash tili sifatida foydalanish, hisoblash tili mexanizmi sifatida mantiqiy xulosa chiqarishdan birgalikda foydalanish sifatida ta'riflanadi. Bu juda yaxshi ta'rif, ammo o‘ta keng ta'rifdir. Bu tarzda ta'riflangan mantiqiy dasturlash o‘z ichiga funksional dasturlashni ham, dasturlashning ko‘plab boshqa usullarini ham oladi.

Tarixan shunday shakllanganki, “mantiqiy dasturlash” iborasi bilan dasturlash tili sifatida birinchi tartibdagi sof mantiqning ayrim kichik ko‘pliklaridan foydalanish anglatiladi, bu harakat konsepsiyasini ifodalash uchun munosabat yoki predikat matematik tushunchasini qo‘llashni anglatadi.

Ko‘pincha til Xornovcha takliflarni (prejlojeniya) qo‘llash, xulosa chiqarish mexanizmi esa. rezolyusiya usulining muayyan ko‘rinishi bilan cheklangandagi mantiqiy dasturlash tushunchasining yanada tor talqini uchraydi.Prolog va dasturlashning unga o‘xshagan tillari ushbu tamoyillarda yaratilgandir. Ushbu tillar asosida aynan bir xil ifodani mantiqiy talqin siftiada talqin qilish imkoni qaror topgandir.

A, agar B1 va B2 va ... Bn
hamda tartibot ta'rifi sifatida
A ni bajarish uchun
B1 bajarish;
B2 bajarish;
...
Bn bajarish.

Biroq bunday tor tushunchani o‘ta cheklangan deb e'tirof etish lozim. “Mantiqiy” nomiga ega bo‘lgan dasturlash ning ko‘plab tillari ushbu sxemaga jo bo‘lmaydi. Boshqa tomondan, aynan shu Prolog ikkinchi tartib mantiqi unsurlariga ega. Prologning ayrim xossalari esa, kazuistik yondashilganda uni deklartiv tillarga kiritishga umuman imkon bermaydi.

Shunday bo‘lsada, Prolog mantiqiy dasturlash ning eng keng tarqalgan tilidir va biz aynan undan foydalanamiz, Prolog misolida biz mantiqiy dasturlashni ma'lumotlarning deduktiv bazasi, sintaktik tahlil (xususan bir ma'noli bo‘lmagan grammatikalar uchun) va murakkab

kombinatorli vazifalar singari ayrim qo‘llashlar uchun yaxshi mos keladiganga aylantiradigan mantiqiy dasturlarning o‘ziga xos xossalrini ko‘rib chiqamiz.

Mantiqiy tillarning kelgusidagi rivojlanishi ikki asosiy yo‘nalishda davom etmoqda, bu yo‘nalishlarni “algoritmik” va “ortiqcha” yo‘nalishlar deb atash mumkin. Bularning birinchisi asosan, samarali algoritmlar ma'lum bo‘lgan vazifalarni hal qilish uchun mo‘ljallangan hamda Prologda mavjud bo‘lganga nisbatan boshqaruvning yanada nozik vositalarini talab qiladi. Ikkinchi yo‘nalish samarali algorimtlar bo‘lmagan va ko‘rinishlarning ortiqchaligi sezilarli joy egallaydigan vazifalarga mo‘ljallangandir. Ushbu yo‘nalish tillari Prolog uchun xos bo‘lgan qaytishlarga ega ayrim sodda ortiqchalikni yanada nozikroq usullarga almashtiradi.

5.1.Prolog

Prolog lahjalarining turli tumanligi Lispdagi singari u qadar katta emas va ular o‘rtasidagi farqlar ham u qadar sezilarli emas, aniqlik uchun SWI-Prolog dan foydalanamiz, u barcha joriy etishlar singari asosan, DEC-10 uchun mashhur edinburgcha Prologga ergashadi.

Ishga tushirilganidan so‘ng Prolog interpretatori so‘rovlarni kiritish taklifini beradi. Bular shuningdek savollar yoki maqsadlar deb ataladi. Odatda so‘rov tartibotni chaqirishdan iboratdir va tartibot nomi sifatida yoziladi. Buning ortidan dumaloq qavslarda vergullar bilan ajratilgan dalillar keladi. So‘rov oxirida nuqta qo‘yiladi. Prologdagi tartibotlar predikatlar deb ataladi. Bulardan ayrimlari tilga joylashtirilgan va ularni darhol chorlash mumkin.

?- true.

Yes

?- fail.

No

Kiritilgan true va fail predikatlari mos ravishda o‘xshash haqiqiy va o‘xshash soxta mulohazadir. Birinchi savolga javob doimo “ha”, ikkinchisiga “yo‘q” dir. True so‘rovi muvaffaqiyat bilan fail esa-muvaffaqiyatsiz tugaydi deyishadi. Xuddi shu tarzda har qanday so‘rov yo muvaffaqiyat yoki muvaffaqiyatsizlik bilan nihoyasiga etadi.

$$?- integer(1).$$

$$Yes$$

$$?-3+4<3*2.$$

$$No$$

$$?-write('salom').$$

salom

$$Yes$$

Predikat nomi va ochiladigan qavs o‘rtasida bo‘sh joy bo‘lmasligiga e'tibor qarating. Ikkinchi so‘rov an'anviy infiks shaklida yozilgan. Ayrim predikatlar bunday yozuvga yo‘l qo‘yadi. So‘nggi so‘rovni bajargan holda Prolog, garchi biz undan go‘yoki hech narsa haqida so‘ramagan bo‘lsakda, Yes deb javob beradi, bu shu tarzda so‘rovning muvaffaqiyatli yakunlanganligi to‘g‘risida shunchaki xabar beradi.

Yana bir imkoniyat ham mavjud – so‘rov xato bilan yakunlanishi mumkin.

?- abracadabra.

ERROR: Undefined procedure: abracadabra/0

?- 3/0 > 1.

ERROR: //2: Arithmetic: evaluation error: `zero_divisor'

Sistemaga abracadabra borasida hech narsa ma'lum emasligi bois istisno vaziyat vujudaga keladi, ikkinchi holatdagi xato arifmetik hisoblab chiqishlar jarayonida vujudga keldi, umuman olganda xato ham muvaffaqiyatsizlik, biroq o‘ziga xos muvaffaiqyatsizlikdir. Bundan so‘ng so‘rovni yana davom ettirish be'manidir.

Biz funksional til interpretatorida yoki Python interpretatorida amalga oshirilgani singari sonlar yig‘inlisini topishga urinib ko‘rishimiz mumkin:

$$?-2+3.$$

$$ERROR: Undefined\ procedure: (+)/2$$

va xato to‘g‘risidagi xabarni olamiz. So‘rovimiz gap shakliga ega emas. Boshqachasiga urinib ko‘ramiz.

$$?-write(2+3).$$

$$2+3$$

$$Yes$$

Tizim xatto nimanidir hisoblab chiqishga urinmasdan ifodani chiqarib beradi. Balki shundaydir?

$$?- X = 2 + 3.$$
$$X = 2 + 3$$
$$Yes$$

Yana muvaffaqiyatsizlik. Prolog eng avvalo, ramziy axborotga ishlov berish va "2+3" ko'rinishidagi ifoda tili sifatida ishlab chiqilgan edi – bular uning ma'lumotlar ob'ektidir (bular termlar deb ataladi). Aynan shu ifoda o'zgaruvchan X ga ham beriladi. Biroq arifmetik hisoblab chiqishni qandaydir tarzda bajarish zarur-ku. Buning uchun maxsus arifmetik predikatlar mo'ljallangan.

$$?- X\ is\ 2 + 3.$$
$$X = 5$$
$$Yes$$

Va nihoyat! Is predikati ifoda qiymatini hisoblab chiqadi va uning o'zgaruvchan qiymatni o'zlashtiradi. So'rov muvafaqiyatli yakunlanadi va o'zgaruvchan qiymat javob siftida chiqarib beriladi.

Boshqa arifmetik predikatlar (=:=, =\=, <, =<, >, >=) o'zining ikkala dalillarini hisoblab chiqadi va ularning qiymatini taqqoslaydi

$$?- 3 + 2 =:= 5.$$
$$Yes$$
$$?- 2 * 2 =:= 2 + 2.$$
$$Yes$$
$$?- exp(2) >= sin(2).$$
$$Yes$$

Sodda ob'ektlarning boshqa bir ko'rinishi – atomlardir. Ko'pincha ular kichik harflar bilan boshlanadigan harflar va raqamlar izchilligi sifatida yozib olinadi. Bu ularni katta xarflar bilan boshlanadigan o'zgaruvchan qiymatlardan farqlaydi. Atomlarni ==, \==, @ <, @ =< , @ >, @ >=. predikatlari vositasida qiyoslash mumkin. Arifmetik predikatlar bunga mos kelmaydi, chunki ular atomlarning arifmetik qiymatlarini hisoblab chiqishga intiladi. Bu xatoga olib keladi, sonlarni esa (biroq ifodalarning qiymatini emas) unisiga ham bunisiga ham qiyoslash mumkin.

$$?- a @< b.$$
$$Yes$$
$$?- 1 @< 2.$$

$$Yes$$
$$?-2 < a.$$
$$ERROR: Arithmetic: `a/0' is\ not\ a\ function$$

$$?-2 < e.$$
$$Yes$$

Nima uchun so‘nggi so‘rov muvaffaqiyatli bo‘lganligin o‘zingiz o‘ylab ko‘ring.

$$?-X\ is\ 2+3.$$
$$X = 5$$
$$Yes$$
$$?-Y\ is\ X+1.$$
$$ERROR: Arguments\ are\ not\ sufficiently\ instantiated$$

Xato to‘g‘risidagi xabarda o‘zgaruvchan qiymat (X nazarda tutilmoqda) aniqlashtirilmaganligi ko‘rsatilmagan. Biroq , biz hozirgina unga qiymat berdik-ku, gap shundaki, bu endilikda boshqa qiymatdir. O‘zgaruvchan qiymat harakati sohasi faqat bir so‘rovga tegishlidir, biroq so‘rovlar bir necha maqsadlardan iborat bo‘lgan murakkab so‘rovlar bo‘lishi mumkin.

$$?-X\ is\ 2+3, Y\ is\ X+1.$$
$$X = 5$$
$$Y = 6$$
$$Yes$$

Hozircha bizning savollar berish imkoniyatlarimiz cheklangan. Yanada qiziqroq savollar berish uchun qandaydir dasturni yuklash lozim, bu consult tartiboti bo‘lishi mumkin, buning uchun qisqartma o‘ylab chiqilgan. Masalan agar dastur “my_program.pl” faylida turgan bo‘lsa, uni

?- consult(my_program)

yoki

?- [my_program].

kiritgan holda yuklash mumkin.

?- [my_program,another_program,yet_another_program]

buyruqni esa, darhol barcha ko‘rsatilgan dasturlardek yuklash mumkin, biroq dasturni yuklashdan oldin uni yozish zarur.

5.1.1. Dasturlar

Prologdagi dasturlar ko‘pincha “ma'lumotlar bazasi” yoki ular mavzu sohasidagi ob'ektlar o‘rtasidagi munosabatlarni yohud ushbu ob'ektlarning xossalarini belgilovchi gaplar majmuini ifodalash ma'nosida hatto “bilimlar bazasi” deb ataladi. Prologdagi xossalar va munosabatlar huddi mantiqdagidek predikatlar deb ataladi.

Har bir tushuncha (predikat) tartibot deb ataluvchi gaplar izchilligi bilan belgilanadi. An'anaga ko‘ra gaplarning ikki turi: holatlar (fakt) va qoidalar ajratib ko‘rsatiladi. Dastlab birinchi turini ko‘rib chiqamiz. Gap-holat predikatning nomidan iborat. Bundan so‘ng dumaloq qavslarda dalillar (argument) keladi. Bunday gap aniq ob'ektlar to‘g‘risidagi tasdiqni tasvirlaydi. Faqat holatlardan iborat dastur mohiyatiga ko‘ra ma'lumotlarning nisbiy (relyasion) bazasidir.

Misol sifatida uncha katta bo‘lmagan Horns&Hoofs Ltd kompaniyasi ma'lumotlar bazasining bir bo‘g‘inini ko‘rib chiqamiz. Bir o‘rinli employee предикати “ходим бўлмоқ” хоссасини белгилайди.

```
employee(anthony).
employee(barbara).
employee(charles).
employee(diana).
employee(edward).
employee(frederic).
employee(gregory).
employee(herbert).
employee(isabella).
employee(john).
```

Umuman olganda atoqli otlar katta harf bilan yozilishi lozim, biroq Prologda munosabatlar va ob'ektlar nomlari kichik harf bilan boshlanishi yoki qo‘shtirnoqlarga olinishi lozim. Odatda qo‘shtirnoqlarsiz ish ko‘rish afzal hisoblanadi. Dastur bir xil nomlarga ega, biroq dalillarning turli soniga ega predikatlarni o‘z ichiga olishi mumkin. ularni farqlash uchun predikat nomi va uning miqdoridan iborat bo‘lgan nomlardan foydalaniladi. Hozirgina aniqlangan predikatning to‘liq nomi:employee/1.

```
salary/2 predikati har bir xodimning maoshini
belgilaydi
salary(anthony,500).
```

```
salary(barbara,200).
salary(charles,180).
salary(diana,150).
salary(edward,150).
salary(frederic,140).
salary(gregory,150).
salary(herbert,100).
salary(isabella,90).
salary(john,160).
```

`boss/2` predikati kompaniyaning tashkiliy tuzilmasi tasvirini belgilaydi. boss(A,B) tasdiqi B – A ning rahbari ekanligini anglatadi.

```
boss(barbara, anthony).
boss(charles, anthony).
boss(diana, barbara).
boss(edward, barbara).
boss(frederic,charles).
boss(gregory, charles).
boss(herbert, charles).
boss(isabella,gregory).
boss(john, gregory).
```

Dasturni yuklagach unga bevosita tegishli bo‘lgan savollarni berish mumkin.

```
?- [hh].
% hh compiled 0.00 sec, 0 bytes
Yes
?- employee(anthony).
Yes
?- employee(ronald).
No
?- boss(barbara, anthony).
Yes
```

Bular savollarning eng sodda turidir. Tizim dasturdagi ayrim gap yozilgan yoki yozilmaganligini shunchaki tekshirib ko‘radi va bu shunday bo‘lsa “ha” yoki aks holatda “yo‘q” javobini beradi. Shu tarzda, agar aniqning teskarisi yozilmagan bo‘lsa, gap soxta hisoblanadi.

Agar so‘rovlarga o‘zgaruvchan qiymatlar keltirilsa, yanada qiziqroq savollarni olish mumkin. Masalan, biz

?- salary(charles,S).

so‘ragan holda Charlzning maoshini bilib olishimiz mumkin:

S = 180

Yes

Ushbu savolga javob bergan holda, Prolog uni ma'lumotlar bazasidagi barcha holatlar bilan o‘zgaruvchanning mos keluvchi qiymatini tanlashga intilgan holda taqqoslab ko‘radi. Ayrim savollar javoblarning bir necha turlariga yo‘l qo‘yadi. Bunday savollar aniqlanmagan savollar deb ataladi.

?- boss(A,charles).

A = frederic

Yes

Biroq, bu yagona javob emas. O‘zgaruvchan qiymatlarga ega savollarga javob bergan holda tizim qiymatni chiqarib bergan holda Enter ga bosilishni kutadi. Aslida u bizga boshqa javoblar ham zarurligi bilan qiziqmoqda. Enter ga bosgan holda “rahmat, etarli” deymiz va tizim Yes deb javob beradi. Boshqa javoblarni olish uchun ";" kiritilishi zarur.

?- boss(A,charles).

A = frederic ;

A = gregory ;

A = herbert ;

No

So‘nggi No boshqa javoblar yo‘qligini anglatadi, javoblar dasturda qanday uchrasa, shunday tartibda beriladi. Ikki o‘zgaruvchan qiymatga ega savol mos keluvchi qiymatlarning barcha juftligini qaytaradi.

?- boss(A,B).

A = barbara

B = anthony ;

A = charles

B = anthony ;

A = diana

B = barbara

Yes

O‘zgaruvchan qiymatni ko‘rsatish zarur bo‘lgan, biroq uning qiymati biz uchun ahamiyatsiz bo‘lgan holatlarda ostiga chizishning yagona ramzdan iborat bo‘lgan yashirin (anonim) o‘zgaruvchi deb ataluvchi foydalanish mumkin. Anonim o‘zgaruvchiga ega bo‘lgan savolga javob qaytargan holda Prolog uning uchun qiymatni tanlaydi, biroq buni chiqarib bermaydi.

```
?- boss(A,_).
A = barbara ;
A = charles ;
A = diana ;
A = edward
Yes
```

Hanuzgacha biz ancha oddiy savollarni berib keldik, bularga javob berish uchun yagona munosabatni ko‘rib chiqish etarli edi. Biroq biz oddiy savollar yanada murakkabroq savollarga birlashish mumkinligini bilamiz. Bu yerda qo‘yilgan vergul gaplar kon'yunksiyasining mantiqiy ma'nosiga ega bo‘ladi. Masalan, kimdir o‘z boshlig‘idan ko‘proq maosh olishini so‘rab ko‘ramiz:

```
?- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.
A = john
B = gregory
SA = 160
SB = 150 ;
No
```

Va tartibsizlikka duch kelamiz: Jonning maoshi Gregoriynikidan katta bo‘lib chiqmoqda.

Gaplarni birlashtirish uchun bog‘lovchidan (kon'yunksiyadan) tashqari nuqtali vergul bilan belgilanadigan";" tarqoqlikdan (diz'yunksiyadan) ham foydalanish mumkin.

```
?- salary(X,S),(S < 100;S > 200).
X = anthony
S = 500 ;
X = isabella
S = 90 ;
No
```

Inkor "\+" bilan (yoki not, biroq ushbu shakl garchi odatiy bo'lsada tavsiya etilmaydi) anglatiladi.

```
?- \+ employee(X).
No
```

So'rov "bironta ham xodim mavjud emas" ekanligini anglatadi, bu, albatta, soxtadir.

```
?- \+ (employee(X),salary(X,300)).
X = _G157
Yes
```

Bu esa "300 maoshga ega bironta ham hodim majud emas" so'rovidir. Bunday xodimlar, albatta, yo'q. O'zgaruvchan X xoli qoldi (hech qanday qiymat olmadi), shuning uchun qiymat sifatida qandaydir kam tushuniladigan narsa chiqarib beradi.

Ko'rinib turganidek, inkor umumiy (mumtoz mantiq iborasidagi umum rad etuvchi) mulohazalarni shakllantirishga imkon beradi. Kim boshliq emasligini va kim boshliqlarga ega emasligini aniqlaymiz.

```
?- employee(X), \+ boss(_,X).
X = diana ;
X = frederic ;
X = herbert ;
X = isabella ;
X = john ;
No
?- employee(X), \+ boss(X,_).
X = anthony ;
No
```

Eng maoshi katta xodimni topamiz.

```
?- salary(A,Smax), \+ (salary(_,S), S>Smax).
A = anthony
Smax = 500
S = _G161
Yes
```

Q ning barcha echimi R uchun echimlar bo'lishini tekshiradigan inkorni - (\+ (Q, \+ P)), konstruksiyani qo'llash qiziqarlidir. Masalan,

```
?- \+ (employee(X), \+ salary(X,_)).
```

X = _G157

Yes

Bu so‘zma-so‘z “maoshsiz xodim mavjud emasligini” yoki “barcha xodimlar maoshga ega ekanligini”anglatadi. SWI-Prolog da aynan shu ma'noga ega bo‘lgan o‘rnatilgan forallpredikati mavjud.

?- forall(employee(X), salary(X,_)).

X = _G157

Yes

Holatlar – shartsiz, haqiqiy gaplardan tashqari dasturga shartli gaplarni – mavjud bo‘lgan iboralarda yangi predikatlarni belgilovchi qoidalarni kiritish mumkin. Har bir qoida: ":-" ramzi bilan ajratilgan ikki qismdan iborat. Qoidaning chap qismi – sarlavha predikatini tasvirlaydi, o‘ng qismi – jism u haqiqiy bo‘ladigan shartlarni belgilaydi. Sarlavhada belgilangan predikatni chaqirib olishda bajariladigan har qanday so‘rov jism bo‘lishi mumkin. Masalan biz so‘ragan savollarni qoidalar ko‘rinishida shakllantirish mumkin.

```
all_paid :- forall(employee(X), salary(X,_)).
costly(A) :- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.
noboss(A) :- employee(A), \+ boss(_,A).
boss(B) :- boss(_,B).
```

Dasturni yangi qoidalar bilan to‘ldirgan holda biz yangi savollar bera olamiz. Shu asnoda bizni qiziqtirmaydigan o‘zgaruvchan qiymatlar qoidalar ichida yashiringan bo‘lib chiqadi va chiqarib berilmaydi.

```
?- all_paid.
Yes
?- costly(A).
A = john ;
No
?- noboss(A).
A = diana ;
A = frederic ;
A = herbert ;
A = isabella ;
A = john ;
No
?- boss(charles).
```

Yes
?- boss(B).
B = anthony ;
B = anthony ;
B = barbara ;
B = barbara ;
B = charles ;
B = charles ;
B = charles ;
B = gregory ;
B = gregory ;
No

So‘nggi so‘rovda har bir javob bir necha bor, dasturda necha marta uchrasa, shuncha chiqarib berilayotganligiga e'tibor qarating, bizga bu o‘ziga xoslik bilan yana to‘qnashishimizga to‘g‘ri keladi – Prolog qancha usullar bilan javobni topsa, uni shuncha marta chiqarib beradi.

Biz aniqlagan predikatlar relyasion SUBD tasavvurlariga juda o‘xshashdir. Tasavvur yoki ko‘rinish (view) bazada so‘rov sifatida saqlanishi mumkin, biroq unga jadval tarzida murojaat qilish mumkin. Huddi shuningdek, qoida bilan belgilangan predikat holatlar sanab o‘tilishi bilan belgilangan predikatdan umuman farqlanmaydi. Masalan,

employee(anthony, none, 500).
employee(barbara, anthony, 200).
employee(charles, anthony, 180).
employee(diana, barbara, 150).
employee(frederic,charles, 140).
employee(gregory, charles, 150).
employee(herbert, charles, 100).
employee(isabella,gregory, 90).
employee(john, gregory, 160).

ва сўнгра

employee(A) :- employee(A,_,_).
salary(A,S) :- employee(A,_,S).
boss(A,B) :- employee(A,B,_),B \== none.

belgilagan holda dastlab belgilangandan farq qilmaydigan predikatlarga ega bo‘lamiz.

Predikatbirgina qoida bilan umuman belgilanishi shart emas, xizmat pillapoyasida ikki xodim bir qatorda turishini anglatuvchinear munosabatini belgilaymiz, bu birinchisi ikkinchisini boshlig'i bo'lgan yoki aksincha ikki holatda yuz berishi mumkin

near(A,B) :- boss(A,B).

near(A,B) :- boss(B,A).

albatta, buni bir gap bilan ham yozish mumkin.

near(A,B) :- boss(A,B); boss(B,A).

biroq, sal murakkab holatlarda bunday yozuv o'ta qo'pol bo'lib chiqadi.

Qoidalar va holatlar o'rtasidagi farq anchagina shartlidir, holatlarni qoidalarning xususiy xodisasi sifatida va aynanasl jismga o'xshash qoida sifatida ko'rib chiqish mumkin. bizning bazamiz

employee(anthony) :- true.

employee(barbara) :- true.

...

sifatida ham boshlanishi mumkin edi.

Hanuzgacha, bizning so'rovlarimiz relyasion algebra doiralarida edi. Endi biz ushbu chegarlardan tashqariga chiqamiz. Boss munosabati xodimning bevosita boshlig'ini belgilaydi. Biroq ushbu xodimning boshqa boshliqlari – uning boshlig'ining boshlig'i, boshliq boshlig'ining boshlig'i va hokazolar ham bo'lishi mumkin. boss munosabatini umumlashtiruvchi chief munosabatini belgilaymiz.

chief(A,C) :- boss(A,C).

chief(A,C) :- boss(A,B), chief(B,C).

Ushbu qayta (rekursiv) belgilangan munosabat barcha boshliqlarni

?- chief(john,X).

X = gregory ;

X = charles ;

X = anthony ;

No

Xuddi shuningdek, barcha bo'ysunuvchi xodimlarni ham

?- chief(A,charles).

A = frederic ;

A = gregory ;

A = herbert ;

A = isabella ;
A = john ;
No

topishga imkon beradi.

Matematik nuqtai nazardan biz munosabatlarning tranzitiv yopilishini barpo etdik, boss – vazifaning oddiy misoli bo'lib, buni relyasion algebra vositalari bilan hal qilib bo'lmaydi.

Ma'lumotlar bazasiga tug'ilish sanasini tasvirlovchi birthday munosabatini qo'shamiz, bizda sanalar uchun ma'lumotlarning maxsus turi bo'lmaganligi uchun bizga yil, oy va sanani alohida kodlashimizga to'g'ri keladi.

birthday(anthony,1970,8,12).

Biroq, Prolog bizga ushbu uch unsurni biz date deb ataydigan bir tuzilmaga birlashtirishga imkon beradi.

birthday(anthony,date(1970,8,12)).

So'rovlarda umuman tuzilmaga yoki uning alohida bo'g'inlariga murojaat qilish mumkin.

?- birthday(anthony,D).
D = date(1970, 8, 12)
Yes
?- birthday(anthony,date(Y,_,_)).
Y = 1970
Yes
?- birthday(A,date(_,12,31)).
A = frederic
Yes

`date` nomi va unsurlar tartibi erkin tanlangan, boshqa nomlardan ham foydalanish, masalan sanani 12/8/1970 yoki 1970-8-12 sifatida yozish mumkin. Biz ishonch hosil qilganimizdek, ushbu ifodalar garchi arifmetik ifodadek bo'lib ko'rinsada, biz bularga is predikatini qo'shmagunimizcha tuzilma sifatida ko'rib chiqiladi. Shuningdek, baribir yil, oy, sana tartibi juda qulay bo'lib chiqadi. Gap shundaki, @<turidagi kichik predikatlar tuzilmani taqqoslashga imkon beradi, shu asnoda dastlab birinchi bo'g'inlar taqqoslanadi, so'ngra agar ular teng bo'lsa, ikkinchisiga o'tiladi va hokazo.

?- date(1970,8,20)@ < *date*(1970,10,1).

Yes

MASHQLAR.

birthday vazifasining ikki usuli uchun A. B dan yoshroq ekanligini ajratuvchi younger (A,B) munosabatini aniqlang.

Mashq. Uch munosabat berilgan bo'lsin haunt(Drinker, Bar) - Drinker Bar ga qatnaydi;

serves(Bar, Beer) - Bar barida Beer rusumli pivo beriladi;

likes(Drinker, Beer) - Drinker Beer pivosini yoqtiradi.

Quyidagi predikatlarni aniqlang.

- happy(Drinker) - Drinker yoqtiradigan pivoni u qatnaydigan barlardan kamida bittasida beriladi.
- goodBar (Drinker, Bar) – bu yerda yaxshi pivoning loaqal bir turi beriladi.
- veryGoodBar(Bar) –ushbu barga doimo qatnovchilar bu yerda o'z ta'biga ko'ra pivoni topadi.
- veryHappy(Drinker) - Drinker qatnaydigan har bir barda mos keladigan pivoni berishadi.
- verySad(Drinker) – Drinker boradigan barlardan birontasida yaxshi pivo berilmaydi.

5.1.2.Yana bir bor trigger xususida

Ma'lumotlar bazasidan chetlanish uchun raqamli sxemalarni modellashtirish vazifasini ko'rib chiqamiz. Endilikda biz signallarning o'zgarishi xususiyati to'g'risidagi shubhali taxminlarni yaratmaymiz, balki sxemaning barqaror holatlarini ko'rib chiqamiz. Xuddi oldingidek, signallarni 0 va 1 qiymatlar bilan tasavvur qilamiz. VA-YO'Q (I-NYE) unsurini ikki kirish signallarini qiymatlarini chiqish signali qiymati bilan bog'lovchi jadval bilan tasvirlash mumkin.

nand(0,0,1).
nand(0,1,1).
nand(1,0,1).
nand(1,1,0).

O'tkazuvchilar (provodov) o'rniga o'zgaruvchilardan foydalanamiz. Unsurlarning kirishi va chiqishi birlashganligini ko'rsatganligi uchun ularni bir o'zgaruvchi qiymat bilan ko'rsatish lozim. Masalan, VA-YO'Q

unsur chiqishini uning 1 kirishi bilan birlashtiramiz, boshqasiga esa signalini beramiz.

?- nand(0,X,X).

X = 1

Yes

Эндиликда хам айнан шу, бироқ 1 гап эга.

?- nand(1,X,X).

No

Ushbu sxema barqaror holatlarga ega emas. Amalda bu sxema tebranish qilishini yoki yanada haqiqatga o'xshash tarzda signallar kuchlanishi mantiqiy 0 va mantiqiy 1 o'rtasidagi qaerdadir qaror topadigan holatga kiradi. Bunday turdagi natijalarni bizning modelimiz qamrab olmaydi.

RS-triggerni barpo etish uchun VA-YO'Q ikki unsurini olish va ularni sxemaga muvofiq birlashtirish etarlidir.

rs(R,S,P,Q):- nand(S,P,Q), nand(R,Q,P).

Kirish signallarini 1 va 0 ga o'rnatgan holda chiqish qiymatlariga ega bo'lamiz.

?- rs(1,0,P,Q).

P = 0

Q = 1

Yes

Triggerningbarcha barqaror holatlarini ham topish qiyin emas.

?- rs(R,S,P,Q).

R = 1

S = 1

P = 1

Q = 0 ;

R = 0

S = 1

P = 1

Q = 0 ;

R = 1

S = 1

P = 0

Q = 1 ;
R = 0
S = 0
P = 1
Q = 1 ;
R = 1
S = 0
P = 0
Q = 1 ;
No

Mashq. Ma'lumki, VA-YO'Q unsuri tomonidan joriy etiladigan "Sheffer shtrixi", tegishli funksiyalar tizimini to'liq asosini shakllantiradi. VA-YO'Q unsurlaridan EMAS, VA, YoKI unsurlarini yig'ing. Olingan bo'laklardan bir razryadli summatora dd (A, B, CarryIn, Sum,CarryOut) ni yig'ing. Shunday to'rtta summatorni to'rt razryadli summatorga birlashtiring.

Mashq. Bizning modelimiz kirish va chiqish signallari o'rtasidagi farqni hisobga olmaydi va bir unsurining chiqishi boshqa unsurning chiqishiga birlashtirilgan "noto'g'ri sxemalar" yig'ish imkonini beradi. Buni to'g'rilash usulini o'ylab ko'ring.

5.1.3. Sintaksis

Prologdagi ma'lumotlar ob'ektlari termlar deb ataladi. Term o'zgarmas, o'zgaruvchan yoki tarkibiy term bo'lishi mumkin. Konstantalar ikki ko'rinishda – sonlar va atomlar ko'rinishida bo'ladi, bulardan mavzu sohasidagi aniq ob'ektlarni hamda ular o'rtasidagi munosabatlarni nomlash uchun foydalaniladi.

Odatda sonlarning ikki turi – maqsadli va moddiy turlari mavjud. yaxlit sonlarning standart o'nli tasavvuridan tashqari Si ruhidagi ikkilik, sakkizlik va o'n oltilik (*0b*101, *0o*777, *0xFFFF*) konstantalardan foydalanish, shuningdek, asosni aniq ko'rsatilgan yozuvdan (2'101, 8'777, 16'*FFFF*) foydalanish mumkin. 0'*A*ko'rinishdagi ifoda A timsoli kodini namoyon etadi.

Atomlar yoki timsollar qavslarga olingan erkin alomatlar izchilligi yoki kichik harfdan boshlanuvchi harflar, raqamlar va alomatlarni tagiga chizish izchilligidan, yohud maxsus alomatlarning (odatida~@#$^& *

–+=<>/\?), yoki o‘ziga xos atomlar: '!','.',',',';','[]','{}' izchilligidan iboratdir.

Odatda dasturlarda qo‘shtirnoqlarsiz atomlardan foydalanadi. Qo‘shtirnoqlarsiz yozilishi mumkin bo‘lgan har qanday atomni qavslarga olish, shu asnoda aynan shu atomga ega bo‘lish mumkin.

O‘zgaruvchan qiymatlar – kichik harf yoki tagiga chizish alomati bilan boshlanuvchi harflar, raqamlar va tagiga chizish alomatlari izchilligidan iborat. Tagiga chizishning yagona timsolidan '_' iborat bo‘lgan o‘zgaruvchan qiymat anonim o‘zgaruvchan qiymat deb ataladi.

Tarkibiy term (yoki tuzilma)– atom turining (term1, term2,...termn) yozuvidir.

Qavslar olidida turgan atom funktor, qavslardagi term esa, dalil yoki komponentlar deb ataladi.Komponentlar soni tuzilmaning arligi (arnostyu) deb ataladi. “Funktor” va “argumentlar” nomi, shuningdek yozuv shakli funksiyalarni argumentlarga qo‘llash natijasida olingan ob'ekt nomi sifatidagi tarkibiy termning tabiiy talqini bilan bog‘liqdir. Masalan,"+(2,3)"termi “amaliyotni qo‘llash natijasi + 2 va 3 sonlariga” ekanligini anglatadi. Ammo biz ishonch hosil qilganimizdek, bu umuman "5" va hatto “2 va 3 sonlarining yig‘indisi emas”. Prolog funktorlarga muayyan qiymatini bermaydi va "+" ostida barcha narsani nazarda tutish mumkin. Tarkibiy termalarni kortejlar sifatida ham ko‘rib chiqish mumkin. Masalan "+(2,3)" termi

+,2,3

kortejiga mos keladi (Erlang sintaksida). Murakkab termlarni shajaralar ko‘rinishida tasavvur qilish qulaydir.

O‘zgaruvchan qiymatlarga ega bo‘lmagan termlar asosiy bazaviy termlar deb ataladi. Bunday har bir term butkul aniq ob'ekt nomini beradi. Agar term o‘zgaruvchan qiymatlarga ega bo‘lsa, bunday term o‘zgaruvchan qiymatlar o‘rniga turli termlarni kiritishda olingan ob'ektlarning yaxlit toifasi vakili sifatida yuzaga chiqadi.

Ayrim tuzilmalar sintaksisining muqobil shakllariga yo‘l qo‘yiladi.

- Har qanday atomni prefiksli yoki postfiksli unar operator yoki infiksli binar operator sifatida e'lon qilish, shuningdek ushbu operator ustuvorligini belgilash mumkin. Bunday operatorlarning ayrim miqdori oldindan belgilangan bo‘ladi va garchi ularni qayta belgilash mumkin bo‘lsada, o‘ta zaruratsiz buni amalga oshirish

kerak emas, masalan, $a*1+b*2+c*3$ ifodasi+(+($*(a,1)$,* $(b,2)$),* $(c,3)$)sifatida, a,b;c,d ni; $(,(a,b),,(c,d))$ sifatida o'qilishi mumkin.So'nggi ifodadagi verguldan ikki yoqlama foydalaanishga e'tibor qarating. Qavslar vositasida ifoda tuzilmasini qo'shimcha aniqlashtirish mumkin.

Ro'yxatlar ma'lumotlarining ommaviy va moslashuvchan tuzilmasi bo'lganligi uchun Prolog ro'yxatlar namoyon etadigan tuzilmalar uchun qiymatlar tizimini ta'minlaydi. Umuman, ro'yxatlar '.' funktorli tuzilmalar bilan taqdim etiladi, ya'ni ".(H,T)" termi boshi N dumi esa T bo'lgan ro'yxatni anglatadi. Bo'sh ro'yxat [] atomi bilan anglatiladi. Biroq bunga qo'shimcha sintaktik oqqand cifatida ko'rib chiqilishi mumkin bo'lgan tarzda [H|T] qiymatlarni tizimidan foydalaniladi.

- a, b va c atomlaridan iborat ro'yxat turli usullar bilan yozilishi mumkin:
- .(a, .(b, .(c, []))) – asosiy tasavvur;
- a. b. c. [] –infiksli operator sifatidagi '.' e'lonidan foydalangan holda;
- [a, b, c] – ro'yxatlar uchun maxsus ko'rsatmadan foydalangan holda;
- [a | [b | [c | []]]] – aynan shu, biroq yoyilgan ko'rinishda.

Timsollar (simvol) satrlari ham ma'lumotlarning ommaviy tuzilmasidir va bular uchun ham maxsus notasiya – ikkiyoqlama qo'shtirnoqlardagi butkul an'anaviy yozuv ko'zda tutilgan. Bunday yozuv timsollar kodlari ro'yxatini anglatadi. Masalan, "string" – [115,116,114,105,110,103] uchun sintaktik oq qand, bu ham o'z navbatida (115,.(116,.(114,.(105,.(110,.(103,[])))))) uchun sintaktik oq qanddir.

Infiksli notasiyadan foydalanish termlarni yanada ko'rinarliroq yozib olishga yordam beradi, biroq tushunmovchiliklarga olib kelishi mumkin. agar shubhalar paydo bo'lsa, termlarni standart prefiksli yozuvga chiqarib beradigan display o'rnatilgan predikatidan foydalanashi mumkin.

```
?- display(8/12/1970).
/(/(8, 12), 1970)
Yes
?- display(a+b/c:[1,2,3]).
```

:(+(a, /(b, c)), .(1, .(2, .(3, []))))

Yes

Operatorlardan ko‘pincha bog‘liq termlarni sintaktik birlashtirish uchun foydalaniladi. Masalan write predikati faqat bir termni chiqarib beradi, biroq infiksli operatorlardan foydalangan holda ushbu cheklanishni bartaraf etish mumkin.

?- write(a=3:b->1:c-18).

a=3:b->1:c-18

Yes

Holatlar sintaksisi termlar sintaksisidan deyarli farqlanmasligini osonlik bilan ko‘rish mumkin. Yagona farq son holat bo‘la olmasligidir. Qoidalar sintaksisi ham garchi bu bir qarashda yaqqol ko‘rinmasada termlar sintaksisidan farq qilmaydi. Darhaqiqat, $a :- b, c, d$ iborasi $:-\left(a, ,\left(b, ,(c, d)\right)\right)$ sifatida yozilishi mumkin hamda ushbu so‘nggi alomatning faqat sintaktik shirinlashtirilgan shakli, xolos. Aynan bir ibora mazmunga bog‘liq holda ham ob'ektni ham munosabatni anglatishi mumkin. Ushbu muhim xossa Prologni Lispga yaqinlashtiradi.

5.1.4. Semantika

Prologdagi dastur semantikasni turli nuqtai nazarlardan belgilash mumkin.

Dasturni mantiqiy tasdiqlar majmui sifatida ko‘irb chiqqan holda dasturning mantiqiy yoki deklarativ semantikasiga ega bo‘lamiz. Har qanday asosiy so‘rov qandaydir tasdiqdan iborat. Bu tasdiq dasturdan chiqariladigan bo‘lishi (bunda u haqiqiy deb faqaz qilinadi) yoki chiqarib olinmaydigan bo‘lishi (va demak, soxta bo‘lishi) mumkin. O‘zgaruvchan qiymatlarga ega bo‘lgan so‘rov “ $(x_1, \ldots x_n)$ mavjud, bundaylar $Q(x_1, \ldots x_n)$” yana haqiqiy yoki soxta bo‘lishi mumkin” ko‘rinidagi ekzsitensial tasdiqdan iborat.

Birinchi holatda o‘zgurvchan qiymatlar o‘rniga kiritilishi mumkin bo‘lgan ayrim termlar topiladi va biz haqiqiy tasdiqqa ega bo‘lamiz. Shu tarzda dasturning deklarativ ma'nosi ushbu maqsad haqiqiy emasligini belgilaydi va agar ha bo‘lsa, qanday o‘zgaruvchan qiymatlarda u asl mulohazaga ega bo‘lishini belgilaydi.

Operasion yoki tartibotli semantika javobni izlash jarayonini belgilaydi. So‘rov bajarilishi zarur bo‘lgan maqsadlar izchilligi sifatida

ko‘rib chiqaladi, har bir tartibot esa ayrim maqsad qanday bajarilishi lozimligini tasdiqlab beradi.

Bir dasturga ikki turli nuqtai nazar mavjudligi ortiqcha bo‘lib ko‘rishi mumkin. Agar bizni dasturning deklarativ ma'nosi qiziqtiradigan bo‘lsa, buni bajarish tafsilotlariga kirish nima uchun kerak? Ba'zan bunga haqiqatan zarurat yo‘q, biroq faqat deklarativ semantika bilan ish ko‘rish imkonsizligi holatlari ham uchrab turadi.

Tartibotli semantika hisobga olinmay yozilgan dasturlar o‘ta samarasiz bo‘lib chiqishi mumkin. Ba'zan dasturni sezilarli darajada tezlatish yoki xotiraga talabni kamaytirish uchun ikki qoida yohud qoidadagi ikki maqsad o‘rnini almashtirish etarlidir.

Eng yomoni, dasturlar javob topishga qodir bo‘lmagan holatda bo‘lishi mumkin. chief ta'rifini o‘zgartiramiz.

```
chief(X,Y):- chief(X,Z),boss(Z,Y).
chief(X,Y):- boss(X,Y).
```

Ushbu tartibotining mantiqiy ahamiyati xuddi dastlabkidagidek, biroq hatti-harakati farqlanadi. Ayrim tartibotlar umuman mantiqiy ma'noga ega emas va mutlaq nuqsonli samara yo‘lida qo‘llaniladi. Masalan Prologdagi

```
?- chief(barbara,Y) tartiboti.
ERROR: Out of local stack
```

Dastur osilib qoladi, chunki chief tartiboti dastlab o‘z-o‘ziga murojaat qiladi. Biroq, buni faqat tartibot nuqtai nazaridan izohlash mumkin.

```
sum :-
read(A),
read(B),
  C is A+B,
write(C).
```

Umaman Paskaldagi quyidagi tartibotga mos keladi.

```
procedure sum;
var A,B,C :real;
begin
  read(A);
  read(B);
C := A+B;
```

```
  write(C)
end;
```

sum predikatining mantiqiy semantikasini aniqlash befoydadir. Mazkur holatdagi "predikat" iborasi (xuddi shuningdek, Lispdagi print uchun "funksiya" iborasi) bir muncha masxaraomuz yangraydi.

Determinasiyalashgan dasturlarni bajarish jarayonini izohlash hammasidan osonroq. So'rovning bajarilishi chapdan o'ngga qarab uning maqsadchalarining bajarilish izchilligini har bir kichik maqsadning bajarilishi esa, tegishli tartibotni chorlashni anglatadi. Bunga dastlabki yaqinlashuvda boshqaruv konstruksiyasiga ega bo'lmagan Paskalni tasavvur qilish mumkin. Masalan,

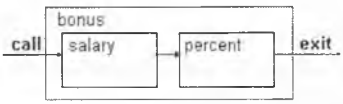
bonus(A,P) :- salary(A,S),percent(20,S,P) predikatini

quyidagi tartibot sifatida

```
procedure bonus(var A:atom; var P:number);
var S :number;
begin
  salary(A,S);
  percent(20,S,P)
end;
```

ёзиб олиш

ёки



sxemasini chizish mumkin.

Ammo bunday tasavvur tartibotlarning muvaffaqiyatsiz yakunlanishi imkoniyatini hisobga olmaydi. Misolimizga tekshiruvni qo'shamiz.

bonus(A,P) :- salary(A,S),merit(A),percent(20,S,P).

Endilikda buni chiziqli shaklda yozib bo'lmaydi. Predikatlarni bulevli funksiyalar sifatida tasavvur qilgan holda yanada aniqroq mos kelishga erishish mumkin.Paskaldagi mantiqiy ifodalarni qisqa hisoblab chiqish qoidasini hisobga olgan holda.

function bonus(var A:atom; var P:number):boolean;

```
var S :number;
begin
bonus := salary(A,S) and merit(A) and percent(20,S,P);
end;
```

yozib olish mumkin.

Yangi sxema har bir tartibotdan ikki chiqishni: muvaffaqiyatli (exit) va (fail) ko‘zda tutadi.

Bazada

```
merit(gregory).

merit(herbert)
```

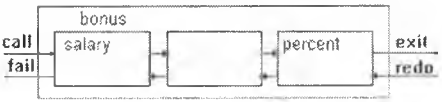
yozilgan bo‘lsin.

Bonus (john,P) so‘rovi bajarilishi ko‘rib chiqamiz. Ushbu so‘rovni olgan holda Prolog ma'lumotlar bazasidan bonus ga tegishli gaplarni izlaydi va so‘rovga sarlavhasiga mos keladigan qoidani topadi. So‘ngra u parmetrlarni uzatgan holda bonus tartibotini chorlaydi, ya'ni salary (john,S), merit(john), percent(20,S,P) maqsadiga o‘tadi. Endi salary(john,S) maqsadi bajariladi, buning natijasida S 160 qiymatiga ega bo‘ladi. merit(john) chorlovi esa muvaffaqiyatsiz, bu bilan esa butun so‘rov muvaffaqiyatsiz yakunlanadi.

Endi bonus(A,P) so‘rovini kiritamiz. Dastlab bu oldingiga o‘xshash tarzda bajariladi, biroq salary(A,S) maqsadi aniqlanmagan bo‘lib chiqadi: bazada bir necha mos keluvchi gaplar mavjud, xolos. Shuning uchun o‘zgaruvchan qiymatlarga {A=anthony,S=500} qiymatini bergan holda, Prolog boshqa imkoniyaitlar borligini ham eslab qoladi –tanlash nuqtasini yaratadi. Ikkinchi maqsad merit(anthony) ko‘rinishini qabul qiladi va bazada bunday gap yo‘qligi bois muvaffaqiyatsiz yakunlanadi. Biroq bajarish bu bilan to‘xtamaydi, tizim orqaga qaytishni amalga oshiradi – berilgan qiymatlarni bekor qiladi va takroran salary(A,S) ni chorlaydi. Tartibotning ushbu takroriy chorlovi mazkur tushunchasi Paskalda ham

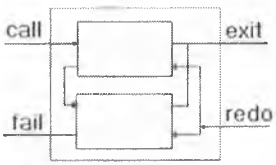
yo‘q. Takroriy chorlovda salary {A=barbara,S=200} juftligini, so‘ngra {A=charles,S=180} va hakozoni qaytaradi. Oxir-oqibatda {A=edward,S=150} qiymatlari olinadi. Shundan so‘ng ikkinchi maqsad merit(edward) muvaffaqiyatli xuddi shuningdek butun so‘rov muvaffaqiyatli yakunlanadi, biroq bu xali xotima emas. Biz Prologdan navbatdan javobni chiqarib berishni so‘raganimizda yanada orqaga qaytish mexanizmi ulanadi va bonus tartiboti takroran chorlanadi. Barcha imkondagi javoblar tugaganidan so‘nggina tanlov nuqtasi olib tashlanadi va tartibotdan uzil-kesil chiqish yuz beradi.

Ushbu jarayonlarni sxemalarda aks ettirish uchun har bir tartibotga yana bir kirish: tartibotni takroriy chorlashni anglatuvchi: redo ni qo‘shamiz. Ushbu kirish navbatdagi tartibotning fail chiqishiga qo‘shiladi.bonus quyidagi ko‘rinishga ega bo‘ladi.



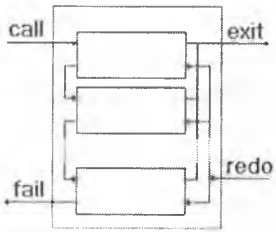
Ushbu sxemaga nazar tashlagan holda Prolog tartibot jismi bo‘ylab oldinga, orqaga qanday harakatlanayotganligini ko‘rish mumkin oldinga haraktalanganda o‘zgaruvchan qiymatlarni anglatish va boshqa foydali harakatlar bajariladi. Orqaga harakatlanganda esa o‘zgaruvchan qiymatlarning “ozod bo‘lishi” yuz beradi.

Kirish va chiqishlarning joylashuvi maqsadlar kon'yunksiyasini tasvirlash qulay bo‘lishi uchun shu tarzda tanlangandir. Diz'yunksiya uchun sxema bir muncha murakkabroq ko‘rinadi.

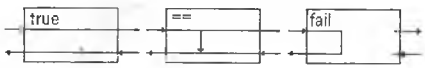


Orqaga qaytishda (redo) boshqaruv so‘nggi muvaffaqiyatli maqsadga, ya'ni so‘nggi chiqish (exit) bo‘lgan joyga uztiladi. Ushbu sxema

ikki gapdan iborat (masalan, merit) tartibotiga to‘g‘ri keladi. Gaplarning katta miqdorini diz'yunksiyani izchil qo‘llash bilan tasvirlash mumkin.



Ko‘plab predikatlar true ga yoki==ga o‘xshash tarzda yaratilgan va tanlash nuqtalarini yaratmaydi. fail predikati doimo muvaffaqiyatsiz yakunlanadi.



Rekursiv belgilangan predikatlar tanlov nuqtalarining cheksiz izchilligini yaratishi mumkin, masalan.

repeat.

repeat :- repeat.

Repeat predikati doimo muvaffaqiyatli yakunlanadi, takroran chorlanganda esa o‘zini o‘zi chorlaydi. Bunisxemada quyidagicha tasvirlash mumkin.

repeat,fail – so‘rovi dasturni cheksiz davriylikka kiritish uchun ishonchli vositadir.

Ko‘rinib turganidek, Prologning tartibot semantikasi dasturlashning ko‘plab an'anviy tillariga qaraganda murakkabroqdir. Ayni vaqtda agar

tartibot sodda deklarativ ma'noga ega bo‘lsa, ko‘pincha uni bajarish tafsilotlarini nazarga olmaslik mumkin. Shuning uchun dasturning imkon qadar katta qismini deklarativ qilish tartibot jihatlarini esa, predikatlarning katta bo‘lmagan miqdoriga joylashga intilish muhimdir. Agar bunga muyassar bo‘linmasa, demak vazifani hal qilish uchun dasturlarning mos kelmaydigan tili yoki mos kelmaydigan dasturchi tanlanganligini anglatadi.

5.2. O‘rnatilgan predikatlar

Har qanday Prolog-tizim o‘rnatilgan predikatlarning katta miqdoriga (ba'zan yuzlab va minglab) ega biz deyarli har bir joriy etishda mavjud bo‘lgan egn muhim predikatlarni ko‘rib chiqamiz

Ixchamlashtirish.

=/2 predikati sof ko‘rinishda Prologning eng muhim konsepsiyasini –termlarni ixchamlashtirish yoki taqqoslashni ifodalaydi. Asosiy termlarni ixchamlashtirishda ular shunchaki o‘xshashlikka taqqoslanadi va natijaga bog‘liq holda “ha”yoki “yo‘q” javobi qaytadi.

```
?- 1 = 1.
Yes
?- a(1) = b(1).
No
```

Agar termlar o‘zgaruvchan qiymatlarga ega bo‘lsa, ikkala term ham o‘xshash bo‘lishi uchun o‘zgaruvchanlarga qiymat berishga urinib ko‘riladi. Agar o‘zgaruvchanlarga qiymat berishning bunday imkoni bo‘lsa, ixchamlashtirish muvaffaqiyatlidir va o‘zgaruvchanlar qiymati javob sifatida chiqarib beriladi.

```
?- X = 1.
X = 1
Yes
?- 2+3 = X.
X = 2+3
Yes
```

Ko‘rinib turganidek, o‘zgaruvchan qiymatlar '=alomatidan ham chapda, ham o‘ngda joylashish mumkin.

```
?- X = 1, X = 1.
X = 1
Yes
```

?- X = 1, X = 2.

No

Birinchi misolda o‘zgaruvchan X 1 bilan taqqoslanadi va 1 qiymatiga ega bo‘ladi. So‘ngra ushbu qiymat 1 bilan muvaffaqiyatli taqqoslanadi. Ikkinchi misolda X 1 qiymatga ega bo‘ladi, so‘ngra bu 2 ga taqqoslanadi (ya'ni amalda 2 = 1 ixchamlashtirishga urinish amalga oshiriladi), bu albatta, muvaffaqiyatsiz yakunlanadi.

Bir necha o‘zgaruvchanlarga bir vaqtda qiymat berilishi mumkin.

?- f(X,Y,Z) = f(a,b,g(c)).

X = a

Y = b

Z = g(c)

Yes

?- X+2 = 3+Y.

X = 3

Y = 2

Yes

Har qanday o‘zgaruvchan qiymat o‘z-o‘zi bilan taqqolanadi va shu asnoda hech qanday qiymat olmaydi.

?- X = X.

X = _G160

Yes

Ikki har xil o‘zgaruvchan qiymatlarni taqqoslash doimo muvaffaqiyatlidir.

?- X = Y.

X = _G160

Y = _G160

Yes

Shu asnoda o‘zgaruvchanlar hech qanday qiymat qolmaydi, biroq bular o‘rtasida aloqa o‘rnatiladi, shu tarzda ular kelgusida sinonimlar sifatida yuzaga chiqadi. X va Y ning “qiymati” bir xilligiga e'tibor qarating. Bunday o‘zgaruvchan qiymatlar zanjirlangan deb ataladi. (bog‘langan deb emas), chunki “bog‘langan o‘zgaruvchan qiymat” aniqlashtirilgan yoki “qiymat berilgan” o‘zgaruvchanni, ya'ni “qiymat olgan o‘zgaruvchanni” anglatadi.

?- X = Y, Y = Z, Z = a.
X = a
Y = a
Z = a
Yes

Bu yerda X Y bilan, Y esa Z bilan bog'lanadi va nihoyat Z 'a' qiymatiga ega bo'ladi. Shundan so'ng barcha uch o'zgaruvchan aynan bir 'a' qiymatiga ega bo'ladi.

Albatta, turli termlarni ixchamlashtirgan holda tajriba o'tkazib ko'ring – bu Prologni tushunish uchun kalitdir. Ixchamlashtrishning muximligi shundaki, buning yordamida dalillarni uzatish va tartibotlardan qiymatlarni qaytarib olish yuz beradi. Aytaylik dastur bir gapdan iborat bo'lsin.

unify(X, X).

Ushbu dastur bilan belgilanadigan unify munosabati '=' dan sira farqlanmaydi.

?- unify(f(1,X), f(Y,2)).
X = 2
Y = 1
Yes

haqiqatda,= predikati odatda aynan shunday belgilanadi.

X = X.

Bu ixchamlashtirsh mexanizmi Prologning chuqur qa'riga shunchaki o'rnatilganligini anglatadi.

O'rnatilgan \= predikati ikki termni taqqoslaydi va agar ixchamlashtirish imkonsiz bo'lgan holatdagina muvaffaqiyatlidir.

?- X \= a.
No

Termlarni taqqoslash.

==, \==, @<, @=<, @>, @>=predikatlari har qanday termlarni taqqoslaydi. Faqat tenglikni taqqoslash balki barcha termlarning ko'pligini taqqoslash mumkin bo'lishi uchun stantart tartib belgilangan. Bu tartib quyidagi tarzda belgilanadi.

1. "Yoshiga qarab" tartibga solingan o'zgaruvchan qiymatlar. Birinchi bo'lib uchragan o'zgaruvchan qiymat barcha qolgan qiymatlardan kichikroq bo'ladi.

2.Atomlar leksikografik tartibda.

3.Sonlar arifmetik tartibda.

4.Ar ligiga ko‘ra tartibga solingan tarkibiy termalar so‘ngra funktor nomi bo‘yicha, so‘ngra chapdan o‘ngga dalillar bo‘yicha o‘rnatiladi.

Ushbu tartibdagi ro‘yxatlar o‘rni '.' funktorga ega binar tuzilmalar sifatidagi ularning namoyon bo‘lishi bilan belgilanadi.

```
?- a @< -a .
Yes
?- f(a) @< [a] .
Yes
?- [a,b,c] @<f(a,b,c) .
Yes
```

Barcha termlar ko‘pligiga bunday yalpi tarkibning mavjudligi bir muncha sun'iy ko‘rinsada biroq amalda juda foydalidir. Har qanday termlarga ega bo‘lgan ro‘yxatlar uchun yaroqli bo‘lgan turlarga ajratishning universal predikatlari standart tartibdan foydalanishga misoldir. sort va msort predikatlari standart tartibga mos ravishda ro‘yxatni tartibga soladi. Shu asnoda ulardan birinchisi takrorlanuvchi unsurlarni bir vaqtda bartaraf etadi.

```
?- sort([6,a,1,a,1,3,b,6],X).
X = [1,3,6,a,b]
Yes
?- msort([6,a,1,a,1,3,b,6],X).
X = [1, 1, 3, 6, 6, a, a, b]
Yes
```

Keysort predikati “kalit” qiymat ko‘rinishidagi termlarga ega ro‘yxatlarga ishlov beradi. U bularni kalitlar qiymatiga ko‘ra tartibga soladi.

```
?- keysort([3-a,1-b,2-b,1-b],X).
X = [1-b,1-b,2-b,3-a]
Yes
```

Barcha echimlarni izlash.

Savolga orqaga qaytish yo‘li bilan turli javoblarni olish imkoniyati foydalidir, biroq etarli bo‘lmaydi. Ba'zan barcha javoblarni darhol olish talab qilinadi. Buning uchun findall, bagofvasetof “ko‘plangan predikatlari” mo‘ljallangandir.

findallpredikati savolga javob bo'lgan o'zgaruvchan qiymatlar ro'yxatini yaratadi. Agar bironta ham javob bo'lmas, bo'sh ro'yxat qaytadi.

?- findall(A,boss(A,charles),L).
A = _G157
L = [frederic, gregory, herbert]
Yes
?- findall(A,boss(A,john),L).
A = _G157
L = []
Yes

Predikatning birinchi dalili – ro'yxatning umumiy unsurini tasvirlovchi term, ikkinchisi so'rovni namoyon etuchi termdir (ZFni-funksional tillardagi ifodalar bilan taqqolang).

?- findall(A=S,(boss(A,barbara),salary(A,S)),L).
A = _G157
S = _G158
L = [diana=150, edward=150]
Yes
?- findall(A,boss(A,B),L).
A = _G157
B = _G158
L = [barbara, charles, diana, edward, frederic, gregory,
herbert, isabella, john]
Yes

So'nggi so'rov "barcha A larni topish"ni anglatadi, bular qandaydir V uchun boss(A,B) bilan bajariladi. Bagof predikati fmdall dan asosan erkin qiymatlarga ega o'xshash so'rovlar qanday talqin qilinishi bilan farqlanadi.

?- bagof(A,boss(A,B),L).
A = _G157
B = anthony
L = [barbara, charles] ;
A = _G157
B = barbara
L = [diana, edward] ;

A = _G157
B = charles
L = [frederic, gregory, herbert] ;

A = _G157
B = gregory
L = [isabella, john] ;
No

Erkin o‘zgaruvchan B qiymatga ega bo‘ladi va bulardan har biri uchun A qiymatlarning tegishli ro‘yxati chiqarib beriladi, ammo bagof ni fmdall ga o‘xshash tarzda bajarilishga majbur etish mumkin. Buning uchun qaysi o‘zgaruvchilar qiymat olmasligi lozimligini aniq qo‘rsatish zarur.

?- bagof(A,B^boss(A,B),L).
A = _G157
B = _G158
L = [barbara, charles, diana, edward, frederic, gregory,
herbert, isabella, john]
Yes

X^P konstruksiyasi mavjudlik kvantoriga ("∃X, P") o ‘xshash tarzda harakat qiladi. Shu tarzda findallga ega so ‘rov barcha erkin qiymatlar mavjudlik kvantori bilan bog ‘liq bo ‘lgan bagof so ‘roviga o ‘xshash tarzda bajariladi. Bundagi yagona farq javoblar mavjud bo‘lmagan holatda bagof bo‘sh ro‘yxatini qaytarmaydi va muvaffaqiyatsizlik bilan yakunlanadi.

Setof predikati bagofga o‘xshashdir, biroq natijalar ro‘yxati turlarga ajratiladi va undan dublikatlar chiqarib tashlanadi. Buni quyidagicha belgilash mumkin.

setof(Var,Goal,Set) :- bagof(Var,Goal,Bag), sort(Bag,Set).

Turlarga ajratish ushning uchun zarurki, bu tartibga solingan ro‘yxat – Prologdagi ko‘pliklarning umumqabul qilingan tasavvuridir.

?- bagof(B,boss(B),L).
B = _G157
L = [anthony, anthony, barbara, barbara, charles, charles,
charles, gregory, gregory]

Yes

?- setof(B,boss(B),L).

B = _G157

L = [anthony, barbara, charles, gregory]

Yes

Mashq. Juftliklar ro'yxatini (boshliq-unga bo'ysunganlar ro'yxati) chiqarib beruvchi so'rov yozing.

Termlarni tahlil qilish.

Oddiy (Tipovoy)predikatlar termlar turini tekshirishga u tarkibiy (compound) yoki sodda (atomic) , so'ngi holatda esa bu (atom) yoki son (number) ekanligini, shu asnda bu son yaxlit (integer) yoki moddiy (float) ekanligini tekshirishga imkn beradi.

?- atom(1).

No

?- atomic(1).

Yes

?- compound([1]).

Yes

?- compound([]).

No

Var predikati va uning inkori bo'lgan nonvar term erkin o'zgaruvchan ekanligini ground predikati esa – asosiy yoki asosiy bo'lmagan term ekanligini tekshiradi.

?- var(X).

X = _G157

Yes

?- X = a, var(X).

No

?- ground(a+X).

No

functorvaargpredikatlari tarkibiy termning funktorini, miqdorini arnost va dalillarni olishga yoki tarkibiy termni barpo etishga imkon yaratadi.

?- functor(a+b,F,A).

F = +

A = 2

Yes

?- arg(1,a+b,X).

X = a

Yes

functor(X,+,2), arg(1,X,2*a), arg(2,X,b).

*X = 2*a+b*

Yes

=..predikati tarkibiy termni funktor va dalillardan iborat bo'lgane ro'yxat bilan taqqoslaydi. Bundan ro'yxatdagi tarkibiy termni o'zgartirish uchun va aksincha foydalanish mumkin.

?- a + b =.. X.

X = [+, a, b]

Yes

?- X =.. [f, 1, 2, 3].

X = f(1, 2, 3)

Yes

Yana bir predikat – name bo'linmasni bo'lishga imkon beradi. U atom nomini alohida timsollarga ajratadi yoki timsollar ro'yxatidan atom yaratadi.

?- name(atom,X).

X = [97,116,111,109]

Yes

?- name(A,"proton+electron").

A = 'proton+electron'

Yes

Mantiqdan tashqari predikatlar.

findall, bagof va setof predikatlari, xuddi shuningdek \+ i forall termlar va gaplarning ixchamlashtirilgan tasavvuriga asoslangan. Ular o'z dalillarini (termlar, ya'ni ma'lumotlar ob'ekti bo'lgan dalillarni) gap sifatida (ya'ni bajarilishi lozim bo'lgan tartibot sifatida)talqin qiladi. bunday turdagi eng sodda predikat call qandaydir term oladi va buni maqsad sifatida bajarishga chorlaydi.

?- call(employee(john)).

Yes

Keltirilgan misol ancha bema'ni bo'lib ko'rinsada, biroq callning vazifasi dinamik yaratiladigan maqsadlarni chorlashdir.

?- functor(G,employee,1),arg(1,G,X),call(G).
G = employee(anthony)
X = anthony ;
G = employee(barbara)
X = barbara
Yes

SWI-Prolog da dalillar qo‘shgan holda bir vaqtning o‘zida termlarni yig‘ishga va uni chorlashga imkon beruvchi call/1, call/2, call/3 va xakozo predikatlarning yaxlit oilasi mavjud.

?- call(boss(isabella,B)).
B = gregory
Yes
?- call(boss(isabella),B).
B = gregory
Yes
?- call(boss,isabella,B).
B = gregory
Yes

Мақсад сифатида учровчи ўзгарувчан (*метаўзгарувчи*) `call` чорлови ёзувининг қисқартирилган шакли сифатида талқин қилинади, албатта ўзгарувчан боғлиқ бўлиши лозим.

?- G = (salary(edward,X),X>100),G.
G = salary(edward, 150), 150>100
X = 150
Yes

Call predikat va o‘zgaruvchanlar eval va apply funksiyalariga o‘xshash tarzda Lispga oliy tartib predikatlarining imkoniyatlarini muayyan darajada amalga oshirishga imkon yaratadi. Masalan, agar “barcha uchun” dizyunksiyasi ham tilga o‘rnatilmaganida biz bularni quyidagi tarzda belgilashimiz mumkin edi.

A ; B :- A.
A ; B :- B.
forall(A, B) :- \+ (A, \+ B).

Odatda, bunday qudratli vositalar xavfli bo‘lib chiqishi mumkin. Dastur noto‘g‘ri bajarilishi yoki dasturning matnida so‘rov yo‘qligi tufayli halokatli yakunlanishi mumkin. Sozlashni yanada jozibador bo‘lishi uchun

Prologda dasturnni bajarish jarayonida dinimik o‘zgartirish ko‘zda tutilgan. Assert predikati gapni ma'lumotlar bazasiga qo‘shadi, retract esa mavjud gapni o‘chiradi.

?- assert(suicide(X) :- retract(suicide(X))).

X = _G157

Yes

?- assert(suicide(1)).

Yes

?- suicide(X).

X = 1

Yes

?- suicide(X).

No

Clause predikati ma'lumotlar bazasida qanday gaplar borligini aniqlashga imkon beradi.

?- clause(boss(A,B),Body).

A = barbara

B = anthony

Body = true ;

A = charles

B = anthony

Body = true ;

A = diana

B = barbara

Body = true

Yes

Boshqaruvchi konstruksiyalar

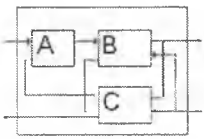
Boshqaruvchi konstruksiyaloarga mohiyatiga ko‘ra kon'yunksiya, diz'yunksiya va inkor kiradi.

(A,B) ifodasi "A ∧ B" anglatadi va ani vaqtda dastlab Ani so‘ngra esa Bni bajarish zarurligini ko‘rsatadi.

(A;B) ifodasi "A ∨ B" maqsadlar diyunksiyasini anglatadi. Tartibot nuqati nazaridan ";" tanlov nuqtasini anlatadi.

(\+ A) ni bajarish A ni chorlashdan boshlanadi. Agar bu muvaffaqiyatsiz nihoyasiga etsa, unda \+ A – muvaffaqiyatli va aksinchadir.

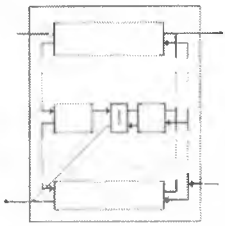
(A → B; C) konstruksiyasi mantiqan (A, B ; \+ A, C)ni anglatadi, biroq birmuncha samaraliroq bajariladi. A maqsadii faqat bir marta chorlanadi. Agar u muvaffaqiyatli yakunlansa V, aks holda S bajariladi. Boshqacha so ‘zlar bilan aytganda, bu yaxshi tanish bo‘lgan shartli operatordir.



Kesib qo‘yish '!' operatori mantiqiy ma'noga ega emas, va orqaga qaytish mexanizmini boshqarish uchun mo‘ljallangandir. U o‘zidan chapda joylashgan maqsadlar tomonidan yaratilgan barcha tanlov nuqtalarini yo‘q qiladi.

?- employee(X),!.
X = anthony
Yes

Ushbu so‘rov endilikda yagona javobni qaytaradi, chunki employee(X) maqsadchasi bilan yaratilgan tanlov nuqtasi '!' bajarilgandan so‘ng yo‘q qilinadi. Agar kesib qo‘yish gap jismida uchrasa, uning harakati ushbu gap chorlanguniga qadar tarqaladi, ya'ni ushbu predikat uchun barcha xali tekshirilmagan qoidalar agar ular mavjud bo‘lsa, kesib qo‘yiladi. Buni sxematik tarzda kesib qo‘yish fail i chiqishini, unga ega bo‘lgan tartibot fail i chiqishi bilan birlashtirib tasavvur qilish mumkin.



Keling, ikki tartibotni ko‘rib chiqamiz.
test1(X) :- atom(X), write(atom(X)),nl.

```
test1(X) :- integer(X), write(integer(X)),nl.
test1(X) :- float(X), write(float(X)),nl.
test1(X) :- number(X), write(number(X)),nl.
test1(X) :- atomic(X), write(atomic(X)),nl.
test1(X) :- compound(X), write(compound(X)),nl.

test2(X) :- atom(X),!, write(atom(X)),nl.
test2(X) :- integer(X),!, write(integer(X)),nl.
test2(X) :- float(X),!, write(float(X)),nl.
test2(X) :- number(X),!, write(number(X)),nl.
test2(X) :- atomic(X),!, write(atomic(X)),nl.
test2(X) :- compound(X),!, write(compound(X)),nl.
```

Hamda ikki so‘rovni taqqoslaymiz.

```
?- test1(1), fail.
integer(1)
number(1)
atomic(1)
No
```

test1(1) chorlovi determinasiyalashmagan, chunki bir necha qoidalarga taqqoslanadi. Bu yerda tanlov nuqtasi yaratiladi. test1 uchun birinchi qoida muvaffaqiyatsiz yakunlanadi va Prolog ikkinchi qoidaga o‘tadi. Bu qoida muvaffaqiyatli bajariladi, buning natijasida integer(10) chiqarilib beriladi. Fail predikat doimo muvaffaqiyatsizdir va Prolog hali tekshirilmagan to‘rt qoida qolgan tanlov nuqtasiga qaytadi. So‘ngra barcha qoidalar tugamagunicha jarayon takrorlanaveradi.

```
?- test2(1), fail.
integer(1)
No
```

Endi integer(1) muvaffaqiyatli yakunlangandan so‘ng ikkinchi qoidada kesib qo‘yish amalga oshiriladi. Natijada test2 ni chorlashda yaratilgan tanlov nuqtasi yo‘q qilinadi va fail dan so‘ngso‘rovni bajarish to‘xtaydi.

Kesib qo‘yishdan foydalangan holda dasturlar samaradorligini sezilarli darajada oshirish mumkin, biroq ani vaqtda ularning aniqligiga

sezilarli zarar etadi. Inkor va shartli operator uchun belgilanishi mumkin bo'lgan kesib qo'yishni ongli qo'llashga misoldir.

```
\+ A :- A,!,fail.
\+ A.
A ->B; C :- A,!,B.
A ->B; C :- C.
```

Aslida bu konstruksiyalar xuddi imperativ tillarda shartli va shartsiz o'tish operatorlariga tuzilmali muqobilliklar paydo bo'lgani singari kesib qo'yishga tuzilmali muqobilliklarni yaratishga urinish singrai vujudaga kelgan.

Kirish va chiqish predikatlari.

Prologda krish va chiqishni imperativ tarzda bajarishga to'g'ri keladi. Bu Prologning hisoblab chiqish modeliga uyg'unlashmaydi. Asosiy muammo kirish yoki chiqish natijasini orqaga qaytishda bekor qilib bo'lmasligidan iboratdir. Garchi o'qib bo'lingan barcha narsaning nusxasini saqlab qolgan holda kirishni bekor qilishni amalga oshirish mumkin bo'lsada, chiqishni bekor qilishni xatto nazariy jihatdan imkoni yo'q..

Kutubxonada kirish va chiqish uchun ko'plab turli-tuman predikatlarni topish mumkin, biroq eng ko'p qo'llaniladiganlari read va writedir. Writeq predikati write ga o'xshashdir, biroq zarurat bo'lganda termlar read vositasi bilan sanab chiqilishi mumkin bo'lishi uchun atomlar nomini qo'shtirnoqlarga oladi. Nl predikati navbatdagi satrga o'tishni yuzaga keltiradi tab(N)esa N bo'shliqlarini chiqarib beradi. Bir muncha past darajali bo'lgan get va put timsollar bo'yicha (baytlar bo'yicha) kirish/chiqishni amalga oshiradi.

```
?- get(X),get(Y),put(Y),put(X).
|: ab
ba
X = 97
Y = 98
Yes
```

end_of_streamu predikat kirish oqimining yakuniga etilganda muvaffaqiyatlidir. Ushbu holatda readend_of_file atomini, get esa I qiymatini qaytaradi. Ushbu barcha predikatlar ma'lumotlarn krishning standart oqimidan o'qiydi va chiqishning standart oqimiga yozadi. see va

tell predikatlari kirish va chiqishni ko‘rsatilgan fayllarga mos ravishda qayta yo‘llaydi, seenva told esa standart qiymatlarni qayta tiklaydi.

Bir necha fayllar yoki ma'lumotlarning boshqa manbalari bilan bir vaqtda ishlash uchun bir dalili ko‘proq ega bo‘lgan o‘xshash predikatlar mavjuddir. Masalan, write (Stream,Term) Term ni Stream oqimiga chiqarib beradi. Dastlab, open predikati yordamida oqimni ochish so‘ngra esa, close predikati bilan uni yopishni unutmaslik zarur. Fayldan nusxa ko‘chiruvchi tartibotning ikki ko‘rinishi misol uchun keltirilmoqda.

```
copy_file(FileIn,FileOut):-
  see(FileIn),
  tell(FileOut),
  repeat,
    get(X),
    put(X),
  at_end_of_stream,
seen,
  told.
```

```
copy_file(FileIn,FileOut):-
  open(FileIn,read,StreamIn),

open(FileOut,write,StreamOut),
  repeat,
    get(StreamIn,X),
    put(StreamOut,X),
  at_end_of_stream(StreamIn),
close(StreamIn),
  close(StreamOut).
```

Umuman olganda, tashqi muhit bilan faol o‘zaro harakatlanuvchi dasturlarni Prologda yozish ancha murakkabdir va agar barcha zarur ma'lumotlarni darhol kiritish imkolni bo‘lsa, bundan foydalangan yaxshiroqdir.

5.3. Ko‘rsatmalar

Gaplardan tashqari so‘rovlarga ham ega bo‘lishi mumkin. Ular qoidalar sifatida sarlavhasiz yoziladi va dasturlarni yuklash jarayonida amalga oshiriladi. Bulardan ayrimlari (yoki hatto faqat) yuklash/ kompilyasiya vaqtida qo‘llaniladi va ko‘rsatmalar deb ataladi.

dynamic direktiva ko‘rsatmasi ushbu predikat dinamik ekanligini, ya'ni unga tegishli bo‘lgan gaplarni qo‘shish yoki olib tashlash mumkinligini ko‘rsatadi. Jim qolish (Po umolchaniyu) bo‘yicha predikatlar turg‘un hisoblanadi va o‘zgartirishga yo‘l qo‘ymadi. Masalan,

:- dynamic employee/1, salary/2, boss/1.

Butkul tabiiy deb e'lon qilish mumkin bo‘lar edi.

Agar bazada bironta ham gap bo‘lmasa, biroq predikat dinamik sifatida e'lon qilingan bo‘lsa, ushbu predikatga so‘rov xato bilan emas, muvaffaqiyatsizlik bilan yakunlanadi.

Odatda predikat belgilovchi gaplar bir-biridan keyin keladi, biroq ba'zan ularni boshqa tamoyilga ko‘ra, masalan, ma'lumotlarni bir turiga mansub bo‘lgan turli predikatlar gaplarini bir joyga jamlash quyladir. Bunday predikatlar discontiguous sifatida e'lon qilinishi lozim. Multifile ko‘rsatmasi esa, tartibot ta'rifini bir necha fayllarga taqsimlashga imkon beradi.

Modullar ehtimol, Prologning eng kam standartlashgan qismi bo‘lsa kerak. An'anaga ko‘ra Prologda nomlarning bir “yassi” makonidan foydalaniladi. Zamonaviy joriy etishlar cheklangan predikatlarni yashirib qo‘yishga imkon beradi. Lekin buni har kim o‘z xolicha amalga oshiradi. SWI-Prolog da boshlang‘ich matnlar “oddiy” dasturlar yoki modullar bo‘lishi mumkin. Modullar faylning boshida turgan module ko‘rsatmasi bilan belgilanadi. Masalan.

```
:- module(module1,[p/1,q/2]).
```

p/1 i q/2. predikatlarini chetga chiqaruvchi module1, nomi bilan modulni belgilaydi.use_module/2buyrug‘i modulni yuklaydi va ko‘rsatilgan predikatlarni olib kiradi, use_module/1esa barcha umumiy kirish mumkin bo‘lgan predikatlarni olib kiradi. Masalan,

```
:- use_module(module1).
```

Aynan

```
:- use_module(module1,[p/1,q/2])
```

ni anglatadi.

Agar predikat orlib kirilmagan bo‘lsa, modul nomini aniq ko‘rsatgan holda uni chorlash mumkin.

```
:- use_module(module1,[]).
...
module1:p(X),
module1:q(X,Y)
```

Chorlashning bunday shakli nomlarning ehtimoliy to‘qnashuvidan qochish uchun qo‘llaniladi.

Sintaksisni boshqarish.

Prolog sintaksisini taqriban quyidagi qoidalar bilan tasvirlash mumkin.

```
term      ::= konstanta
            | funktor"(" term {","term}")"
            | operator term
            | term operator
            | term operator term
            | "(" term ")".
konstanta ::= atom | son.
operator  ::= funktor.
funktor   ::= atom.
```

Atomdan operator sifatida foydalanishdan oldin uni shunday deb e'lon qilish lozim. Prolog sintaktik tahlil jarayonida qo'llaniladigan operatorlar jadvalini saqlaydi. Jadvalga op predikati bilan o'zgartish kiritish predikatning joriy qiymatlarini esa current_op bilan olish mumkin. Ushbu predikatlar uch dalilga: ustuvorlik, operatorning turi va nomiga ega.

Ustuvorlik –operator kattaligini belgilovchi 0 dan 1200 gacha diapazondagi yaxlit sondir. Ustvorlik qancha kichik bo'lsa, operator shunchalik katta bog'lovchi kuchga ega. Masalan, jim turish bo'yicha quyidagi ustuvorliklar belgilangan' $*$ ',' / ' $-$ 400; " $+$ "," $-$ " $-$ 500; ; " $=$ "," $<$ " $-$ 700; "," $-$ 1000; ":$-$" $-$ 1200.

Biroq ifodani bir ma'noli tagiga etish uchun birgina ustvorlik etarli bo'lmaydi. Masalan, "*a–b–c*" "(*a–b*)–*c*sifatida,"*a,b,c*" "*a,*(*b,c*)" sifatida , a (*a=b=c*) – esa yo'l qo'yilmaydigan ifodadir. Aytishlariga ko'ra "-" chap o'xshatuvchi, "," o'ng o'xshatuvchi, a "=" esa o'xshatmaydigan operatordir. Operatorning o'zshatuvchanligi uning turi bilan birgalikda beriladi, masalan,

```
op(700,xfx,=).
op(500,yfx,+).
op(1050,xfy,->).
op(500,fx,-).
op(900,fy,\+).
```

Bu yerda xfx "infiksli o'xshatmaslik"ni anglatadi. "f" harfini operatorning o'zi ko'rsatadi, "y" bu joyda aynan shunday yoki kamroq ustvorlik turishi mumkinligini anglatadi, "x" esa faqat kamroq ustuvorlikka ega bo'lgan termga ruhsat beradi. Prologda oldindan belgilangan postfiksli operatorlar mavjud emas, biroq o'zinikini kiritishga hech narsa halal bermaydi.

?- op(200,yf,!).

Yes

?- X = 5+3!,display(X).

+(5, !(3))

X = 5+3!

Yes

Operatorlarning e'lonlari dasturga ta'sir ko‘rsatishi uchun ularni ko‘rsatmalar sifatida berish zarur.

:- op(1070,fx,if).

:- op(1050,xfx,then).

:- op(1060,xfx,else).

if X then Y :- X,!, Y.

if X then Y .

if X then Y else Z :- X,!, Y.

if X then Y else Z :- Z.

VI. BOB. FUNKSIONAL DASTURLASH

Dasturlashda biz mohiyatlarning ikki turi: ma'lumotlar va amaliyotlar bilan ish ko‘ramiz. Bularning birinchisini o‘zimiz boshqarishni istayotgan hisoblash jarayonini nofaol tashkil etuvchi jarayon sifatida, ikkinchisini ma'lumotlar ustidan harakatlarni bajarish qoidalarini tasvirlovchi faol tarkibiy qism sifatida ko‘rib chiqamiz. Shuning uchun dasturlashning har qanday tili ushbu mohiyatlarni tasvirlash imkoniyatini berishi lozim. Eng avvalo, til sodda ma'lumotlar va bular ustidan amaliyotlarni namoyon etuvchi sodda iboralarga ega bo‘lishi lozim. So‘ngra bizga sodda ob'ektlarni (ham ma'lumotlarni, ham amaliyotlarni) yanada murakkabroq ob'ektlarga birlashtirish vositalari zarur bo‘ladi va nihoyat dasturlash tili murakkab ob'ektlarning aniq tuzilishidan chetlashishga va ularni yagona mohiyatlar sifatida boshqarishga imkon beruvchi mavhumlashtirish vositalariga ega bo‘lishi lozim.

Funksional dasturlash harakat konsepsiyasini ifodalash uchun funksiyaning matematik tushunchasidan foydalaniladi. Oddiy matematik funksiyalarga o‘xshash tarzda funksional tillarning tartibotlari (“funksiyalar”) bir ob'ektlarni (dalillarni) boshqalariga (qiymatlarga) aylantiradi. Shu asnoda imperativ tillarning tartibotlaridan (funksiyalaridan) farqli o‘laroq funksiyalar qiymatlari ularning dalillari bir ma'noda belgilanadi va hisoblab chiqish jarayoni tarixiga bog‘liq bo‘lmaydi, biroq matematik funksiyalar va tartibotlar o‘rtasida muhim farq mavjud. Tartibotlar samarali belgilanishi lozim. Masalan, matematikada biz a sonidan kvadrat ildizni kvadratga ko‘tarilganda ham a ni beradigan shunday son sifatida belgilashimiz mumkin. Bu butkul qonuniy ta'rifdir, shunday bo‘lsada, mutlaqo asosli emas, u ushbu ildizni topish usulini bermaydi (ammo bunday ta'rif butkul foydasiz deb bo‘lmaydi, chunki u o‘ziga xoslashtirishi vositasi sifatida muhidir). Kelgusida biz “tartibot” va “funksiya” iboralaridan sinonimlar sifatida foydalanamiz, funksiyaning matematik tushunchasi va dasturlash tilidagi uning ta'rifi o‘rtasidagi ushbu farqni ta'kidlash zarur bo‘lgan holatlar bundan mustasnodir.

Funksiyaning aynan shu tushunchasi ma'lumotlar konsepsiyasini ifodalash uchun ham qo‘llanishini biz kelgusida ko‘rib chiqamiz. Umuman olganda, funksional dasturlashni o‘rganish – “ma'lumotlar” va

"amaliyotlar" o'rtasidagi farq u qadar keskin emasligiga ishonch hosil qilish uchun yaxshi sababdir. Funksional tillardagi funksiyalar "birinchi toifa" ob'ektlaridir.

Dasturlash tilining "birinchi toifali" unsurlari tushunchasi Kristofer Streychi tomonidan kiritilgan. U dasturlash tillari til unsurlaridan foydalanish usullariga turli cheklashlar qo'yishini aniqlagan. Birinchi toifa unsurlari – cheklanishlarning eng kam miqdoriga ega bo'lgan unsurlardir. Bunday birinchi toifali unsurlarning muhim xossalari:

- O'zgaruvchan qiymatlar vositasida ularga tayanish mumkinligi
- Ularni ma'lumotlar tuzilmasiga kiritish mumkinligi.
- Ularni parametrlar sifatida uzatish mumkinligi.
- Ular natija sifatida qaytarilishi mumkinligidir.

Ayrim istisnolarni hisobga olmaganda (masalan, Algol-68) imperativ tillar funksiyalar va tartibotlarning "huquqlari va erkinliklarini" cheklab qo'yadi. Ulardan farqli o'laroq funksional tillar funksiyalarga birinchi toifa maqomini beradi. Bu ularni samarali joriy etish uchun qiyinchiliklar tug'diradi, biroq ushbu tillarning ifodalash kuchini sezilarli darajada orttirishga olib keladi.

Kelgusidagi bayonimiz uchun bizga dasturlash tili kerak bo'ladi va bunday til sifatida Haskell tanlangan. Bu tanlov tilning o'ta sodda semantikasidan va uning muayyan darajadagi bir turli sintaksidan kelib chiqadi. Ushbu xossalar funksional dasturlashning asosiy g'oyalarini shaffof ifodalashga imkon beradi.

Dasturlashning har qanday tili uni boshqa tillardan ajratib turadigan muayyan xossalar yig'indisiga egadir. Dasturlash tillarininng guruhlari, shu jumladan funksional tillarning barcha ko'pligi to'g'risida ham aynan shuni aytish mumkin. Ularning barchasi bunday tillarni imperativ tillardan farqli aynan funksional tillarga aylantiradigan muayyan xossalarga ega.

Funksional tillarning (faqat bunday tillarga mansub bo'lishi shart emas) asosiy xossalari sifatida quyidagilarni qisqacha ko'rib chiqamiz:

1) qisqalik va soddalik;
2) qat'iy turlarga bo'linish;
3) modullik;
4) funksiyalar — bu qiymatlar va hisoblab chiqish ob'ektlaridir;
5) soflik(nuqsonli samaralarning yo'qligi va aniqlanganlik);
6) chetga surilgan hisoblab chiqishlar.

6.1.Qisqalik va soddalik

Funksional tillarning qisqaligi va soddaligi ularning mavjud xossasi emas, balki ular shunday tarzda barpo etilganligining oqibatidir. Bu demak, dasturlashning funksionaltillari o'z-o'ziga ko'ra sintaksis nuqtai nazaridan o'ta dahshatli bo'lishi mumkinligini anglatadi. Ammo dasturlashning aksariyat zamonaviy tillari qisqa ta'riflarni yozib olishga imkon beradigan butkul sodda sintaksisga ega.

Shu tarzda funksional tillardagi dasturlar aynan shu harakatlarni bajaruvchi, biroq imperativ tillarda yozligan dasturlardan qisqaroq va soddaroqdir. Misol tariqasida S va Haskell tillaridagi berilgan ro'yxatni Xoar (masalan, funksionaltillarning afzalliklarini tasvirlashda mumtozga aylanib ulgurgan misol) usuli bilan jadal turlarga ajratish funksiyasini taqqoslash mumkin.

1.1.Misol. S tilida Xoar bo'yicha jadal turlarga ajratish

```
void quicksort (int a[], int l, int r)
{
  int i = l;
  int j = r;
  int x = a[(l + r)/2];
  do
  {
    while (a[i] < x) i++;
    while (x < a[j]) j--;
    if (i <= j)
    {
      int temp = a[i];
      a[i++] = a[j];
      a[j--] = temp;
    }
  }
  while (i <= j);
  if (l < j) quicksort (a, l, j);
  if (i < r) quicksort (a, i, r);
}
```

Shunday qilib, bir-biriga kiritilgan boshlang'ich kodning jami 20 satri va ikki rekursiv chorlash, shu asnoda funksiyaning o'zida kirish parametri o'zgaradi, ya'ni mohiyatiga ko'ra boshlang'ich ro'yxatni buzish va uning o'rniga yangi turlarga ajratilgan ro'yxatni yaratish amalga oshiriladi.

Mana aynan shu funksiyaning Haskell tilidagi ta'rifi.

1.2.Misol. Haskell tilida Xoar bo'yicha turlarga ajratish.

```
quicksort [] = []
quicksort (x:xs) = quicksort [y | y<- xs, y< x] ++
[x] ++
                quicksort [y | y <- xs, y >= x]
```

bu erda, mohiyatiga ko'ra kodning ikki satri keltirilgan (ikkinchi satr o'qish qulay bo'lishi uchun uch tarkibiy qismga bo'lingan, aks holda u juda uzun bo'lgan, talqin (yoki kompilyasiya) uchun bunday ta'rif aynan ikki satrda yozilgan bo'lar edi.

1.2 misolni quyidagicha o'qish zarur.

1) Agar ro'yxat bo'sh bo'lsa, turlarga ajratish natijasi ham bo'sh ro'yxat bo'ladi.

2) (Agar ro'yxat bo'sh bo'lmasa) bosh (birinchi unsur) va dum (o'z navbatida bo'sh bo'lishi mumkin bo'lgan qolgan unsurlardan tuzilgan ro'yxat) boshqacha ajratib ko'ratiladi. Mazkur holatda boshdan kichik bo'lgan dumning barcha unsurlaridan turlarga ajratib olingan ro'yxatni va boshdan katta yoki unga teng dumning barcha unsurlaridan tuzilgan ro'yxatni konkatenasiya (zanjirlash) natijaga aylanadi.

Natijada ushbu ta'rifda dasturchining ko'ziga ko'rinmaydigan ikki davriylik (Haskell tilida ish boshlayotgan dasturchilar bularning mavjudligi haqida bilmasliklari ham mumkin) va ikki rekursiv chorlovdan foydalaniladi. Bundan tashqari yangi turlarga ajratilgan ro'yxatni yaratish yuz beradi. Turlarga ajratilishi zarur bo'lgan eski ro'yxat esa, tegilmaganligicha qoladi. Dasturchida undan qayta foydalanish zarurati yuzaga keladigan boshqa maqsadlar uchun tegilmaganligicha qoladi.

Haskell tilining go'zalligi va ichki uyg'unligini oxirigacha anglash uchun (dasturlashning boshqa zamonaviy funksionaltillari misolida ham) matematik notasiyada yozib olingan quicksort funksiyasi ta'rifini ko'rib chiqish mumkin. Bunday ta'rif taqriban quyidagicha ko'rinishda bo'ladi:

quicksortl = [y : $y \in t(l) \wedge y < h(l)$] + [$h(l)$] + [$y: y \in t(l) \wedge y > h(l)$], (1.2)

bu yerda h(l) — ro‘yxat boshini olish uchun funksiya, a t(l) — ro‘yxat dumini olish uchun funksiyadir.

Umuman olganda izohlash ortiqcha.

Ko‘rinib turganidek, hatto shunday oddiy misolda dasturlashning funksional tarzi ham yozib olingan kod soniga ko‘ra va uning nazokatiga ko‘ra ustun bo‘lib chiqmoqda.

Binobarin, empirik jihatdan funksional tildagi kod satrlarining o‘rtacha miqdori imperativ tildagi shunday miqdordan o‘n marta kamligini dasturlashda foydalaniladigan tillardan qat'iy nazar kam ekanligini faraz qilish mumkin.

Bundan tashqari xotiraga ega bo‘lgan barcha amaliyotlar dasturlashning aksariyat funksional tillarida avtomatik tarzda bajariladi. Qandaydir ob'ektni yaratishda unga avtomatik tarzda xotira chiqarib beriladi. Ob'ekt o‘z vazifasini bajarib bo‘lganidan so‘ng u har qanday funksional tildagi interpretatorning qismi bo‘lgan axlat yig‘uvchi tomonidan avtomatik tarzda yo‘q qilinadi.

Dasturlashning funksional tilidagi dasturni qisqartirishga imkon beradigan yana bir foydali xossa, namuna bilan taqqoslashning o‘rnatilgan mexanizmidir. Bu funksiyalarni induktiv ta'riflar sifatida yozib olishga imkon beradi. Masalan:

1.3 misol. FibonachchiN-ro sonini hisoblab chiqish.

fibonacci 0=1

fibonacci 1=1

fibonacci n = fibonacci (n - 2) + fibonacci (n - 1)

Mohiyatiga ko‘ra, namunaga taqqoslash muayyan shartlarga bog‘liq holda dasturlashning ko‘plab tarmoqlarini tasvirlashdir, bu o‘z navbatida funksiyalar deklarsiyasini mavhumlashtirishning yuqoriroq darajasiga ko‘tarishga va ularning hajmini kichraytirishga imkon beradi.

Namunaga taqqoslash mexanizmi keyingi qismlarda ko‘rib chiqiladi, ammo bu yerda funksional tillar an'anaviy imperativ tillarga nisbatan (ob'ektli-yo‘naltirilgan paradigmaga va uning kengaytirishlarga nisbat bermagan holda) yanada abstrakt darajaga chiqishi ko‘rinib turibdi.

Afsuski, ushbu xossaning salbiy oqibati – unumdorlik yo‘qolishidir. Dasturlash funksional tillarining ichiga o‘rnatilgan hamda boshlang‘ich

kod miqdorini qisqartirishga va uning nazokatini oshirishga yo‘naltirilgan barcha optimalliklar mexanizmlari va vositalar dasturni bajarish tezligiga tabiiy tarzda ta'sir ko‘rsatgan.

Ammo funksional tilda qo‘llaniladigan abstraksiyalash, ma'lumotlarni taqdim etish va muammoni tasvirlash darajasi, shuningdek hisoblab chiqish texnikasining kundan kunga quvvatlari ortib borishi yuqoridagi salbiy xossani amalda deyarli yo‘qqa chiqarishga imkon beradi. Masalan, o‘tkazilgan tadqiqotlar Haskell tilidagi dastur unumdorligi bo‘yicha S tilidagi aynan shunday dasturlardan o‘rtacha 7 marta orqada qolishini ko‘rsatdi (boshlang‘ich kod miqdorining xatto o‘rtacha 10 marta qisqartirib amalga oshrilishini yodga olamiz, funksional tillarning boshqa ijobiy tomonlari haqida gapirmasa ham bo‘ladi).

6.2. Qat'iy turlarga ajratish

Dasturlashning ko‘plab zamonaviy turlari qat'iy turlarga ajratilgandir (ehtimol JavaScript singari ssenariylarni amalga oshirish tili va uning lahjalari bundan mustasno bo‘lsa kerak, “tur” tushunchasiga ega bo‘lmagan imperativ tillar mavjud emas). Dasturchi dastur ustida ishlash vaqtida ish ko‘radigan ob'ektlarni qat'iy turlarga ajratish xavfsizlikni ta'minlaydi. Bunday tekshirishdan o‘tgan dastur “access violation” (xotiraga kirish xatosi) singari xabar bilan operasion tizimga kira olmaydi, bu hususan ko‘rsatkichlar tildan foydalanishning oddiy usuli sifatida qo‘laniladigan S yoki C++ va Object Pascal singari tillarga tegishlidir. Funksional tillarda xatolarning katta qismi kompilasiya bosqichida tuzatilishi mumkin. Shuning uchun sozlash bosqichi va dasturlarni ishlab chiqishning umumiy vaqti qisqaradi. Bunga qo‘shimcha tarzda qat'iy turlarga ajratish kompilayatorga yanada samaraliroq kodni jamlashga, va bu bilan dasturni bajarishni tezlatishga imkon beradi.

Shunday qilib, qat'iy turlarga ajratish xossasi faqat funksional tillarga xos emas, biroq aynan ularda ushbu xossa optimallashtirishning har xil ko‘rinishlarini o‘tkazish, polimorf funksiyalarini yaratish uchun ham qo‘llaniladi.

Haqiqiy yoki parametrik polimorfizimni Xoarning jadal turlariga ajratishni amalga oshirish tuchun funksiyaning yuqorida keltirilgan misollarida ko‘rish mumkin

Ushbu misolni nazardan o‘tkazgan holda S tilidagi ko‘rinish va Haskell tilidagi ko‘rinishlar o‘rtasida, eslatib o‘tilgan farqlardan tashqari yana bir muhim farq borligini ko‘rish mumkin: S tilidagi funksiya Int (yaxlit sonlar), turi qiymatlari ro‘yxatini turlarga ajratadi, Haskell — tilidagi funksiya esa tartibga solingan qiymatlar toifasiga mansub bo‘lgan har qanday tur qiymatlari ro‘yxatini turlarga ajratadi. Shuning uchun 1.2. misolidagi funksiya ham yaxlit sonlar ro‘yxatini, ham oquvchi nuqtaga ega bo‘lgan sonlar ro‘yxatini, ham satrlar ro‘yxatini turlarga ajratishi mumkin. Qandaydir yangi turni tasvirlash mumkin. Bunday tur uchun taqqoslash amaliyotini belgilagan holda (ushbu funksiyada foydalanish uchun “kamroq” v “ko‘proq yoki teng” taqqoslash amaliyotini tasvirlash etarlidir), perekompilyasiya qilmasdan ushbu yangi tur qiymatlari ro‘yxati bilan ham quicksort funksiyasidan foydalanish imkoniyati bor, bunday polimorfizim aksariyat funksional tillarda saqlab turiladi.

Tasvirlangan xossani aniq tushunish uchun quicksort funksiyasi turini tasvirlash uchun majburiy bo‘lmagan direktikvani yozish mumkin:quicksort :: Ord *a* => [a] –> [a]

Ushbu direktiva Haskell tili translyatori uchun funksiya turini tasvirlaydi, ammo aksariyat xollarda interpretator har qanday funksiya turini mustaqil chiqarib beradi. Funksiyalar turlari to‘g‘risida 2.3. qismda batafsil yozilgan. Bu yerda esa, keltirilgan direktivadagi a o‘zgaruvchan qiymati keltrilgan direktivada turining o‘zgaruvchan qiymatidir, ya'ni buning o‘rniga Ord toifasining ekzemplyari bo‘lgan har qanday tur nomini, ya'ni taqqoslanadigan qiymatlar nomini kiritish mumkin.

Ayni vaqtda C++ tilida dasturchiga quicksortga o‘xshash polimorf funksiyalarni belgilashga imkon beradi, C++ STL standart kutubxonasiga shunday funksiya va boshqa polimorf funksiyalarining hamda konteynerli toifalar kiradi., biroq C++, shablonlari huddi Ada tilining turdosh funksiyalari singari aslida ko‘plab qayta yuklangan funksiyalarni vujudga keltiradi, bularni, o‘z o‘rnida aytish kerakki, kompilyator har safar har bir foydalanadigan tur uchun bunday funksiyalarni alohida joriy etishni avtomatik tarzda yaratgan holda kompilyasiyalashi lozim, bu ham kompilyasiya vaqtiga, ham mashina kodi hajmiga nomaqbul ta'sir ko‘rsatadi. Funksional tillarda esa polimorf funksiya quicksort — bu bir yagona funksiyadir.

Funksiyalar nomlarini qayta yuklash — turlicha, biroq nimasi bilandir o‘xshash funksiyalarga bir xil nomlarni berishga imkon yaratuvchi polimorfizimning yana bir turidir. Qo‘shish oddiy amaliyoti qayta yuklangan funksiyaga eng oddiy misoldir. Yaxlit sonlar uchun va suzuvchi sonlar uchun qo‘shish funksiyalari turlichadir, biroq qulaylik uchun ular aynan bir xil nomga ega. C++ tilidagi shablonli ob'ektlar aynan shu mexanizmga asoslangandir.

Garchi funksiyalar nomlarini qayta yuklashbu ancha zaif echim bo‘lsada, ayrim funksional tillar parametri polimorfizimdan tashqari ushbu mexanizmni ham saqlab turadi, bu Haskell tilida ham mavjud.

Ayrim tillarda, masalan, Adatilida qta'iy turlarga ajratish dasturchini barcha qiymatlar va funksiyalar turini aniq tasvirlashga majbur etadi. Bundan qochish uchun qat'iy turlarga ajratilgan funksional tillarga konstantalar, mazmundagi iboralar va funksiyalar turlarini kompilyatorga ajratishga imkon beruvchi maxsus mexanizm o‘rnatilgan. Bu mexanizm turlarni chiqarib berish mexanizmi deb ataladi. Bunday mexanizmlarning bir necha turlarima'lum, ammo ularning aksariyati XX asrning 80 yillari boshida ishlab chiqilgan Xindli-Milner turlarga ajratish modelining har xil ko‘rinishlaridir. Shunday qilib, aksariyat hollardafunksiyalar turlarini ko‘rsatmaslik mumkin, bu quicksort funksiyasi misolida ko‘rsatib o‘tildi.

6.2.1.Modullik

Modullik mexanizmi dasturlarni bir nechta nisbatan mustaqil qismlarga (modullarga) ular o‘rtasidagi aniq aloqalar bilan ajratishga imkon beradi. Bu bilan loyihalash jarayoni va aksariyat dasturiy tizimlarni kelgusidagi saqlab turish engillashadi. Modullikni saqlab turish dasturlashning aynan funksional turlari xossasi emas, biroq bunday tillarning aksariyatida saqlab turiladi. Ko‘plab rivojlangan modulning imperativ tillari mavjud. Bunday tillarga misol qilib, Modula-2 va Ada 95 ni keltirish mumkin. C++ yoki Java singari dasturlashning boshqa zamonaviy tillari ham endilikda modullarda foydalanishsiz, fikr qilib bo‘lmaydi (Java tilida esa, modullardan foydalanish sintaksis tomonidan qat'iy begilangan).

Haskell tili modullikni o‘ta rivojlangan tizimiga ega, bu dasturlarning boshlang‘ich matnlarini alohida ozmi-ko‘pmi mustaqil qismlarga bo‘lishgagina emas, balki inkapsulyasiya singari narsani, ya'ni

ma'lumotlarga ishlov berish usullarini va ma'lumotlarning o‘zini (ularning tuzilmalarini) bunday ma'lumotlarga ishlov berish uchun faqat interfeysli usullarni tashqariga “chiqarib qo‘ygan holda” yashirish imkonini beradi.

Buning ustiga, Haskell tili modullardagi ob'ektlar ko‘rinishini nafaqat modulning ichidan (mazkur holatda ko‘rinishni modula muallifi belgilaydi) balki tashqaridan ham, ya'ni muayyan modul import qilinadigan dastur joylarida ham boshqarishga imkon beradi (mazkur holat ko‘rinishni modeldan foydalanuvchi boshqaradi).

6.2.2. Funksiyalar —bu hisoblab chiqish qiymatlari va ob'ektlaridir

Ma'lumki, ish jarayonida funksiya muayyan parametrlar (dalillar) to‘plamini kirishda qabul qiladi, uzatilgan dalillar qo‘llaydigan berilgan hisoblab chiqish jarayonini bajaradi, shundan so‘ng funksiya berilgan hisoblab chiqishni qaytaradi. U funksiyani oddiy tushunish bo‘lib, u hisoblab chiqilgan qiymatlarga ega bo‘lish zarurati bo‘lgan har qanday joyda funksiyani chorlashga imkon beradi. Shu jumladan dalillarni boshqa funksiyaga uzatishda ham dalillar sifatida turishi va boshqa funksiyalarni chorlashi mumkin.

Bunga o‘xshash xolatda interpretator o‘zi belgilagan qiymatlarni uzatish bilan barcha kirtilgan funksiyalarni chorlashni amalga oshiradi, ular hisoblab chiqqan natijalarni oladi va endilikda ularni dastlabki funksiyaning yakuniy chorloviga uzatadi.

Masalan, C++ tilidagi

n = getNofContainers (countContainerLevels(1, 10), true)

funksiyasini chorlashda dastlab, countContainerLevels funksiyasi unga amaldagi dalillar 1 va 10 ni uzatgan holda choralanadi va u hisoblab chiqan natijani olganidan so‘ng (aniqlik uchun ushbu funksiya 3 qiymatini qaytara qolsin) getNof Containers(unda dalillar sifatida 3 i trueqiymatlari uzatiladi) aniqlik uchun ushbu funksiya 3 qiymatini qaytara qolsin) trueyasini chorlash amalga oshriladi. Bu dasturlashning barcha tillarida amalga oshiriladiganfunksiyadan oddiy foydalanishdir.

Funksional tillarda funksiya o‘z-o‘zidan dalil . sifatida boshqa funksiyalarga uzatilishi yoki natija sifatida qaytarilishi mumkin.

Funksional dalillarni qabul qiluvchi funksiyalar oliy tartibdagi funksiyalar yoki funksionallar deb ataladi. Eng ma'lum funksional - bu extimol tar funksiyasi bo‘lsa kerak. Ushbu funksional ayrim funksiyani

olingan natijalardan boshqa ro'yxat shakllantirgan holda ro'yxatning barcha unsurlariga qo'llaydi.

Oliy tartibdagi ushbu funksiyadan ayrim ro'yxatning barcha unsurlarini kvadratga ko'tarish uchun foydalanish uchun mumkin:

squareList = *map*(^2) [1, 2, 3, 4]

ushbu ko'rsatmani bajarish natijasi

1, 4, 9, 16

ro'yxati bo'ladi.

Shu tarzda, funksiyani chorlashdagi paramertlar sifatida hisoblab chiqish jarayoniga uzatilishi zarur bo'lgan boshqa funksiyani ko'rsatish mumkin. Yuqorida keltirilgan misoldagi ushbu funksiya — (^2) — kvadratga ko'tarishdir. Bunday xolatlarda parametrlar sifatida uzatilgan funksiyalar chorlanmaydi. Ularning o'zi hisoblab chiqish jarayonida ishtirok etayotgan ob'ektlardir. Shu asnoda ularni jarayonning ichida ularga muayyan parametrlarni uzatgan holda chorlash, shuningdek ulardan boshqa maqsadlarda (masalan, dalillar ko'rinishida boshqa funksiyalarni uzatgan holda) foydalanish mumkin.

Boshqa tomondan funksional tillardagi funksiyalar jarayon bajarilganligi natijasi sifatida qaytarilishi mumkin bo'lgan ob'ektlardir, ya'ni bir funksiya o'z navbatida funksiya bo'ladigan natijani hisoblab chiqishi mumkin. Ushbu samaraga xususiy hisoblab chiqishlar deb ataluvchi tushunchalar yordamida erishiladi.

Masalan, bu yerda batafsil ko'rib chiqilmaydigan, biroq bayon qilingan farazni tushunish uchun etarli bo'lgan mumtoz misol. Ushbu misol soninkrementi funksiyasini ikki sonni qo'shishi funksiyasi orqali ta'rifini ko'rsatadi (misol, albatta, biroz hayolparastlikka ega, biroq ko'rib chiqilgan hodisani yaqqol tasvirlaydi):

```
add x y = x + y
inc = add 1
```

ushbu misolda ikki dalilni qabul qiladigan add funksiyasida chiqishga faqat bir dalil uzatilishi ko'rinmoqda. Funksiyalar til translyatori uchun bu uning ta'rifiga faqat majud dalillarni qo'ygan holda funksiyani qisman qo'llash zarurligini anglatadi. Bu ish natijasida kirishda boshlang'ich funksiyaning qolgan dalillarini kutib turadigan yangi funksiya qo'lga kiritiladi.

Misol tariqasida C++ tili natija sifatida funksiyani (to‘g‘rirog‘i funksiyaga iqtibosni) qaytarishga imkon berishni ta'kidlash qoldi, xolos. Ammo bu yerda funksiyaga manzilni qaytarishgina amalga oshiriladi, unga muayyan parametrlarni uzatgan holda uni chorlash mumkin. Ammo mazkur texnologiya funksional tillarning bu yerda tasvirlangan xossasi bilan hech qanday umumiylikka ega emas.

6.2.3.Soflik

Funksiyaning zararli samarasi — o‘z hisoblab chiqishini bajarish jarayonida global o‘zgaruvchan qiymatlarni o‘qish va modifikasiya qilish, kirish/chiqish amaliyotlarini bajarish, favqulodda vaziyatlarga munosabatda bo‘lish, ularga ishlov beruvchilarni chorlash imkoniyatidir. Shuning uchun agar bunday funksiyani kirish dalillari qiymatlarining aynan bir xil to‘plami bilan ikki marta chorlansa natija sifatida turli qiymatlar hisoblab chiqilishi yuz berishi mumkin. Bunday funksiyalar zarrali samaraga ega aniqlanmagan funksiyalar deb ataladi. Funksiyaning aniqlanmaganligi – bu funksiyaga kirishda kirish dalillarining bir xil qiymatlari uzatilganligiga qaramasdan funksiya tomonidan turli qiymatlarni qaytarish imkoniyatidir, ya'ni zararli samaralarning mavjudligini funksiya qiymatlarining bir ma'noli jadvalini barpo etish imkoniyati bo‘lmagan funksiyaning aniqlanmaganligi sifatida ko‘rib chiqish mumkin. zararli funksiyalarga ega bo‘lgan funksiyalar uchun ularning qiymatlar jadvali kirish parametrlarining berilgan to‘plamida u qabul qilgan extimoliy qiymatlar (cheksiz bo‘lishi ham mumkin) ro‘yxati sifatida ko‘ringan bo‘lur edi.

Funksiyaga uzatilgan parametrlarni (o‘zgaruvchan qiymatlarni) modifikasiya qilish, zararli samaralarning boshqacha ko‘rinishidir, ya'ni mohiyatiga ko‘ra buzg‘unchi o‘zlashtirish bo‘lib, bunda funksiyaning chiqishdagi kiymatini hisoblab chiqish jarayonida kirish parametrning qiymati ham o‘zgaradibuzg‘unchi xatti-harakatning bunday misoli S++ tilida yozilgan quicksort funksiyasida keltirilgan edi. (1.1 misolga qarang).

Dasturlashning deyarli har qanday tili zararli samaralarga ega bo‘lmagan aniqlangan funksiyalarni tasvirlashga imkon beradi. Ammo ayrim tillar funksiyalardagi ayrim ko‘rinishlardan zararli samaralardan foydalanishni rag‘batlantiradi yoki xatto talab qiladi. Masalan, ko‘plab

ob'ektli yo‘naltirilgan tillarda toifa a'zosi bo‘lgan funksiyaga yashirin parametr (ko‘pincha bu this — C++ ga, yoki self — Object Pascalga deb ataladi) buni ushbu funksiya mavhum modifikasiyalaydi.Mohiyatiga ko‘ra bu tegishli funksiya – toifa a'zosi nomi bilan chorlanadigan toifa ekzemplyariga ko‘rtsagichidir.

Dasturlash tillarini qo‘llashda funksiyalardagi zararli samaralar doimo mavjud bo‘ladigan eng jiddiy soha kirish/chiqishdir. Foydalanuvchidan ma'lumotlarni kiritishning har qanday amaliyoti zararli samaraga ega harakat deb faraz qilish mumkin, chunki hisoblab chiqish jarayonida qo‘llaniladigan parametrlar qiymati sifatida foydalanuvchi aynan nimani kirtishini oldindan aytib bo‘lmaydi. Garchi ayrim tadqiqotchilar va nazariyotchi olimlar kirish/chiqishni zarrali samaralar mavjudligi misoli sifatida ko‘rib chiqish mumkin emasligi, chunki mohiyatiga ko‘ra kirish/chiqish bu dastur atrofini o‘zgartirish ekanligini ta'kidlashsada, biroq har qanday xolatda kirish/chiqish uni qo‘llaydigan funksiyalarni aniqlanmagan funksiyaga aylantiradi.

Sof funksional dasturlashda o‘zlashtirish operatori mavjud emas. ob'ektlarni o‘zgartirib yoki yo‘q qilib bo‘lmaydi. Mavjud o‘ektlarni dekompozisiya va sintez yo‘li bilan yangi ob'ektlarni yaratish mumkin, xolos.Keraksiz ob'ektlarborasida funksional tilning har qanday translyatoriga o‘rnatilgan axlat yig‘uvchi bosh qotiradi.

Shu tufayli sof funksional tillarda barcha funksiyalar zararli samaralardan xolidir, ammo bu ushbu tillarga istinolarga ishlov berish va o‘zgartiriladigan (buzg‘unchi) massivlar singari ayrim foydali imperativ xossalarga taqlid qilishga xalal bermaydi. Buning uchun maxsus usullar mavjud.

Biroq, zararli samaraga ega funksional mavjudligining ayrim sabablarini dasturlashning funksional tillaridan butkul olib tashlab bo‘lmaydi, chunki bunday xolatda bunga o‘xshash tillar qo‘llash ma'nosida o‘ta tor bo‘lib qolar edi.

Birinchi navbatda bu aynan kirish/chiqishga tegishlidir. Interfaol tartibda foydalanuvchidan ma'lumotlarni kiritishni amalga oshirish, shuningdek, foydalanuvchi uchun ma'lumotlarni chiqarishni amalga oshirish imkoniyati bo‘lmagan dasturlashning to‘laqonli tilini tasavvur qilish qiyindir.

Dasturlashning ko‘plab funksional tillarida shu jumladan Haskell tilida ham, soflik xossasini pastga urmagan holda kirish/chiqish singari texnologiyalardan foydalanish imkoniyatini ta'minlash uchun “monada” deb ataluvchi maxsus mexanizmdan foydalaniladi. Monadalar go‘yoki, zaruriy imperativ xossalarni bularga funksional tilning sof sintaksisiga aralishib ketishiga yo‘l qo‘ymagan holda go‘yoki o‘rab qo‘yadi. Monadlardan foydalanish funksiyalardagi zararli samaralar mavjudligini belgilab qo‘ygan barcha tor joylarni amalga oshirishga imkon berdi.

Masalan, kirish/chiqishni ta'minlash uchun Haskell tilda 10 standart monadi qo‘llanilgan. Bundan tashqarida kirish/chiqishning biron bir amaliyotini bajarib bo‘lmaydi. Haskell tili uchun ishlatilgan barcha qolgan standart monadlar ham aynan shunday xossalarga ega.

Sof funksional tillarning afzalliklari nimada? Dasturlar tahlilini soddalashtirishdan tashqari, yana bir salmoqli ustuvorlik – parallelizm mavjud

Hisoblab chiqish uchun barcha funksiyalar hamonki o‘z parametrlaridan foydalanar ekan, mustaqil funksiyalarni hisoblab chiqishni erkin tartibda yoki paralleltashkil etish mumkin, hisoblab chiqishlar natijasiga bu ta'sir ko‘rsatmaydi. Shu asnoda paralellizm faqat til kompilyatori darajasida emas, balki texnik vositalar arxitekturasi darajasida ham shakllantirilishi mumkin.Ayrim ilmiy laboratoriyalarda shunga o‘xshash arxitekturalarga asoslangan eksperimental kompyuterlar yaratilgan va qo‘llanilmoqda. Misol tariqasida Lisp-mashinani keltirish mumkin.

6.2.4. Chetlatilgan hisoblab chiqishlar

Hisoblab chiqishlarning chetga surilish xossasi faqat dasturlashning fnuksionali tillarini hisoblaydi. Ushbu muhim xossaga barcha funksional tillar ega emasligi ham haqiqatdir. Qisqacha ko‘rib chiqilayotgan xossani hisoblab chiqishlar natijasi talab qilinmaydigan xolatlarda hisoblab chiqishlar yo‘qligi sifatida ta'riflash mumkin.

Dasturlashning an'anaviy turlarida (masalan, S++) funksiyani. chorlash u kirishga uzatgan barcha dalillarni hisoblab chiqishga olib keladi. konstantalar dalillar sifatida funksiyaga bevosita uzatiladi. O‘zgaruvchan qiymatlarga alomatlar qo‘yiladi hamda funksiyaning o‘zida ushbu o‘zgaruvchanlarning funksiyadagi parametrlar sifatida uzatish

lahzasida ular olgan qiymatlaridan foydalaniladi.Agar dalillar orasida funksiyani chorlash mavjud bo‘lsa, dastlab ushbu funksiyani hisoblab chiqish jarayonini ishga tushirish yuz beradi. Olingan natija esa, boshlang‘ich hisoblab chiqish jarayonining ichiga uzatiladi.

Shu tarzda,funksiya tomonidan tasvirlangan hisoblab chiqish jarayonining o‘ziga kirish lahzasida hisoblab chiqishlarni chetga surish xossasiga ega bo‘lmagan oddiy dasturlash tillarida barcha kirish parametrlariningg qiymati ma'lumdir. Bu hisoblab chiqish jarayoni chog‘ida jarayonning o‘zida qandaydir kirish dalillari o‘z qiymatini o‘zgartirib yuborishi mumkinligi to‘g‘risida tashvishlanmaslik mumkinligiga imkon beradi.

Funksiyani chorlashning ushbu usuli “qiymatga ko‘ra chorlash” deb ataladi, binobarin, agar qandaydir dalil funksiyada foydalanilmagan bo‘lsa, uni hisoblab chiqish natijasi yo‘qolishi va binobarin hisoblab chiqishlarning o‘zi bekorga amalga oshirilganligi butkul aniqdir. Qandaydir ma'noda “zarurat bo‘yicha chorlash” qiymatga ko‘ra chorlashning butkul teskarisidir. Mazkur holatda dalil funksiya qaytarib yuboradigan yakuniy natijani hisoblab chiqish uchun zarur bo‘lgan xoldagina hisoblab chiqiladi.

Bunday hatti-harakatning misoli sifatida agar birinchi dalil soxta qiymatga ega bo‘lsa, ikkinchi dalilni hisoblab chiqmaydigan aynan o‘sha C++ (&&)dagi barcha konyuksiyalar opartorini olish mumkin. Agar birinchi unsur HAQIQAT qiymatiga ega bo‘lsa, diz'yunksiya (||) amaliyotining ikkinchi operandi qiymati mutloqo hisoblab chiqilmaydi.Shuning uchun bunday kodning qandaydir ko‘rinishi sifatida:

```
if (false && (x / 0))
{
}
```

ni yozish mumkin.

Hamda (&&) amaliyotining ikkinchi operandini hisoblab chiqish nolga bo‘lish” xatosi paydo bo‘lishiga aniq olib kelishiga qaramasdan amalda bunday xato hech qachon yuz bermaydi.

Chetga surilgan hisoblab chiqishlarning tasvirlangan xolati imperativ tillarda bunday dasturlash tillarini yaratishning umumiy tamoyiliga nisbatan ko‘proq istisnodir. Yuqorida tasvirlangan mantiqiy amaliyotlardan tashqari bunday yondoshuvdan hech qaerda

foydalanilmaydi, ayrim tillarda esa (masalan, Object Pascal tilida) ushbu butkul bir ma'noli xolatlar uchun ham foydalinmaydi.

Dasturlashning funksional tillari— bu endi boshqa masala. Zamonaviy funksional tillarning aksariyati chetga surilgan hisobalb chiqishlar konsepsiyasiga ega. Shuning uchun ularda quyidagilarga o'xshash konstruksiyalarni aniqlash va qo'llash osondir:

```
bot = bot
constant.1 n = 1
x = constant_1 bot
```

So'nggi ta'rif constant_1 funksiyasi o'z-o'zini cheksiz rekursiv chorlash sifatida ta'rifi ko'rinadigan botfunksiyasiga uzatilganda Haskell tili translyatori uchun hech qanday oqibatlarga olib kelmaydi, chunki xfunksiyasining qiymati butkul aniqlangan va 1 ga tengdir. Agar S++ tili translyator taxminan quyidagi funksiyalar berilganda u nima qilishini tasavvur qilish qiyin emas:

```
int bot ()
{
  return bot ();
}
int constant_1 (int n)
{
  return 1;
}
int x ()
{
  return constant_1 (bot ());
}
```

x()funksiyasi chorlanganda oqim ostiga ajratib chiqarilganxotira tugaganligi bo'yicha faqat dasturning vayron bo'lishi bilan nihoyasiga etganda uzluksiz davriylikka kirish yuz bergan bo'lur edi.

Agar funksional til chetga surilgan hisoblab chiqishlarga ega bo'lmasa, u qat'iy til deb ataladi. Aslida bunday funksionla tillarda hisoblab chiqish tartibi qat'iy belgilangan, qat'iy tillarga misol sifatida Scheme, StandardML va Caml ni keltirish mumkin.

Chetga surilgan hisoblab chiqishlardan foydalanadigan tillar qat'iy bo'lmagan tillar deb ataladi (ba'zan "dangasa tillar" iborasi qo'llaniladi).

Haskell tili — huddi Gofer va Miranda singari qat'iy bo‘lmagan tildir. Qat'iy bo‘lmagan tillar ko‘pincha sof, ya'ni soflik xossasini saqlab turadigan tillardir.

Ko‘pincha qat'iy tillar qat'iy bo‘lmagan tillarga xos ayrim foydali imkoniyatlarni saqlab turish vositalarini o‘z ichiga oladi. Masalan, Standard ML etkazib berishdagi (v pastavke)cheksiz ro‘yxatlar chetga surilan hisoblab chiqishlarni saqlash uchun maxsus modulga ega. Objective Caml esa, bundan tashqari qo‘shimcha kalit so‘z lazy ga hamda zarurat bo‘yicha hisoblab chiqiladigan qiymatlarning ro‘yxati uchun maxsus konstruksiyaga ega.

6.3. Haskell ga kirish

Haskellyaratilgandan beri 30 yildan ko‘proq vaqt o‘tdi. Bu vaqt orasida deyarli barcha sohalarda ushbu tildan keng qo‘llanila boshladi va bu mazkur tilni universal til deb atashga imkon beradi. Bizning maqsadimiz tilni o‘rganish emasligi bois, biz o‘zimizga foydali bo‘lgan ayrim xossalarni qisqacha ko‘rib chiqamiz.

Albatta, Haskell tili ob'ektli olamni modellashtirish uchun “ro‘yxat” singari tushunchani chetlab o‘tmagan (an'anaga ko‘ra aynan ro‘yxatlar funksional dasturlashning barcha paradigmalari uchun tamal toshiga aylangan, chunki masalan, xattodastlabki funksional til bo‘lgan Lisp “List processing” — “ro‘yxatlarga ishlov berish” iborasining qisqartmasidir). Ro‘yxatlar bilan ishlash maqsadida Haskell tilida mexanizmlar va usullarning ulkan jamlanmasi mavjud. Ularni qo‘llashning osonligi va soddaligi biron bir dasturchi yoki matematikni unga nisbatan loqayd qoldirmaydi.

Ro‘yxatlarga va umuman ma'lumotlarning har qanday boshqa turlariga ishlov berish uchun funksional dasturlashda sodda (oddiy) va oliy tartibdagi funksiyalar bo‘lgan funksiyalardan foydalaniladi. Bunday funksiyalardan foydalanish uchun bularni tasavvuri funksional tillar ish ko‘radigan boshqa ob'ektlar tasavvuridan farqlanmasligi zarur. Aynan shu bois ma'lumotlarni, shu jumladan hisoblab chiqishlarni amalga oshirish uchun ma'lumotlar sifatidagi funksiyalarni turlarga ajratish sifatidagi oddiy-netrivialny mexanizm ko‘rib chiqiladi,

Zamonaviy ilovalarni ishlab chiqishda dasturlarni alohida modellarga ajratishsiz ish ko‘rib bo‘lmaydi, bu xal qilinayotgan vazifalarni

nafaqat taqsimlashga, balki qo‘llanilgan komponentlardan takroran foydalanishga imkon beradi. Mazkur muammo dasturlarni modullashtrishning barcha imkoniyatlaridan foydalanishga yordam beradigan Haskell tilining sintaksisida ham o‘zining bevosita aksini topgan.

6.3.1. Haskell dagi dastur tuzilmasi

Ushbu qismda biz Haskellningabstraktsintaksisi va semantik tuzilmasini, shuningdek u tasvirlashning boshqa qismlari tuzilishiga qanday nisbatda bo‘lishini bayon qilamiz.

1) Haskell dagi dasturning eng yuqori bosqichi modullar to‘plamidan iborat. Modullar nomlar makonlarini boshqarish hamda ulkan dasturlardagi dasturiy ta'minotdan takroran foydalanish imkonini beradi.

2) Modulning yuqori bosqichi bir necha turlardan ibrat bo‘lgan e'lonlar majmuidan iboratdir. E'lonlar oddiy qiymatlar, ma'lumotlar turlari, turlarning toifalari, operatorlarning o‘xshashliklari va ustuvorliklari singari mohiyatlarni belgilaydi.

3) Navbatdagi bir muncha pastroq darajada iboralar turadi. Ibora qiytmatni anglatadi va turg‘un ko‘rinishga ega; iboralar Haskell dasturlashidagi “kichik” asosida qaror topgandir.

4) Pastki darajada Haskell ning leksik tuzilmasi joylashgan. Leksik tuzilma Haskell dasturlaridagi matnli fayllardagi aniq tasavvurni qamrab oladi.

5) Мазкур баён Haskellнинг синтактик тузилмасига нисбатан пастдан юқорига йўналтирилгандир.

Mazkur matndagi Haskell ga dasturlarning bo‘g‘inlari misoli monoshirinnom shriftda (CourierNew) keltirilgan:

```
let x = 1
z = x+y
in z+1
```

Tilga oson va tez kirish uchun interpretator bilan ishlashdan tanishishdan boshlaymiz.

6.3.2. Ifodalar va ifodalarning ahamiyati

Odatda Haskellinterpretatori bilan ishlash quyidagi ssenariy bo‘yicha yuz beradi.

Foydalanuvchi ifodani kiritadi.
Interpretator ushbu iboraning qiymatini hisoblab chiqadi va natijani chop etadi.

Eng sodda misollardan boshlaymiz (interpretatorning javoblari kursiv bilan ajratib ko‘rsatilgan).

Hugs>10
10
Hugs>10.5
10.5

Sonli konstantalar sonlarni anglatadi va bu sonlar ularning qiymatlari hamdir. Sonlarning erishiladigan turlari xilma-xilligi tilni joriy etishga bog‘liq., biroq barcha joriy etishlar yaxlit va moddiy sonlarga, aksariyati esa rasional va kompleks sonlarga ega.

Hugs>Hello
ERROR - Undefineddataconstructor "Hello"

Sonlardan farqlanuvchi harflar, raqamlar va maxsus alomatlar (!$%&*/:=<>?^_~ +-.@)izchilligitimsoldebataladi.Bularning asosiy vazifasi ob'ektlarni nomlashdir. Shuning uchun ushbu timsol bilan nomlangan ob'ekt timsolning qiymatidir. Biroq Hello nomi bilan hozircha hech qanday qiymat bog‘liq bo‘lmaganligi uchun xato to‘g‘risidagi xabarni olamiz.

Qo‘shtirnoqlar yordamida timsollarni avtomnim qo‘llash mumkin. “<timsol>” ifodasini qiymati ushbu timsolning o‘zidir. Bu ulardan masalan, Paskaldagi sanaladigan turlar qiymatiga o‘xshash tarzda foydalanishga imkon beradi.

Hugs>"Hello"
"Hello"

Bu satrli konstanta misolidir. Bular ikkilama qo‘shtirnoqlarda yoziladi va aks ettiriladigan alomatlarning izchilligini namoyon etadi.

Hugs> true
ERROR - Undefined variable "true"
Hugs> TRUE
ERROR - Undefined data constructor "TRUE"
Hugs> True
True
(26 reductions, 51 cells)

Ikki mantiqiy konstantalar True vaFalse haqiqat va soxtaligini anglatadi. Registrga Haskellning sezuvchanligiga e'tibor qarating.

Konstanta va timsollar atomlar umumiy nomiga ega, chunki tilning ifodalar barpo etadigan eng sodda unsurlaridan iboratdir.

An'anaviy matematik notasiyada funksiyaning nomi qavslarga olingan dalillar oldidaturadi. Bundan tashqari, arifmetik ifodalarning infiks yozuvi, turli tuman indekslar va maxsus alomatlardan foydalaniladi.

Matematik yozuv	Haskelldagi yozuv
$f(x)$	f(x)
$g(x,y)$	g(x,y)
$h(x,g(y,z))$	h(x, g(y, z))
$sin\ x$	sin(x)
$x + y$	x + y
$x + y * z$	x + y*z
xy	x^y
$\|x\|$	abs(x)
$x = y$	x == y
$x + y < z$	+ y < z

Haskell – qat'iy turlarga ajratish ega til bo‘lib, buning o‘ziga xosliklarini biz turlarga ajratishga mahishlangan bobda ko‘rib chiqamiz.

Nazorat savollari

1) Dasturlashning funksional tillarga qo‘llanishdagi “qat'iy turlarga ajratish” nima?

2) Oliy tartibdagi funksiyalar funksiyaning kirish parametrlari sifatida olingan funksiyalarga qanday ishlov beradi?

3) Dasturlash funksional tillarining “soflik” va “qat'iy bo‘lmaslik” sifatidagi asosiy xossalari nima bilan xususiyatlanadi?

4) Tur konstruktorining uning selektorlari bilan aloqasi to‘g‘risidagi aksioma qanday ikki ehtimoliy ko‘rinishni qabul qiladi?

5) Funksional dasturlash paradigmasi doiralarida isbotlash taklif etilayotgan funksiyalar xossalariga qanday cheklanish kiritiladi?

VII. BOB. TURLARGA AJRATISH

Turlar tizimi dasturlash tillaridagi xossalarni belgilovchi turlar deb ataluvchi dasturni tashkil etuvchi har xil konstruksiyalar – o'zgaruvchanlar, qiymatlar, funksiyalar yoki modullar singari qoidalar majmuidir. Turlar tizimining asosiy roli dasturning turli qismlari va ushbu qismlar o'zaro harakati muvofiqligini navbatdagi tekshirish o'rtasidagi interfeyslarni aniqlash vositasida dasturlardagi xatolar sonini kamaytirishdan iboratdir. Bunday tekshirish statik (kompilyasiya bosqichida) yoki dinamik (bajarish vaqtida), yuz berishi, shuningdek ikki ko'rinish kombinasiyasi bo'lishi mumkin.

Dasturlash tillarini turlarga ajratish bo'yicha ikki katta lagerga — turlarga ajratilgan va turlarga ajratilmagan (tursiz) lagerga ajratish qabul qilingan. Bularning birinchisiga C, Python, Scala, PHP va Lua kiradi, ikkinchisiga esa — assembler tili , Forth va Brainfuck mavjuddir.

Mohiyatiga ko'ra "tursiz turlarga ajratish" soddadir, endilikda u hech qanday boshqa turlarga taqsimlanmaydi. Turlarga ajratilgan tillar esa, yana bir qancha kesishuvchi kategoriyalarga ajratiladi:

Statik / dinamik turlarga ajratish. Statik turlarga ajratish o'zguruvchan qiymatlar va funksiyalarning yakuniy turlari kompilyasiya bosqichida o'rnatilishi bilan belgilanadi. Ya'ni endilikda kompilyator qaysi tur qaerda joylashganligiga 100% ishonadi, dinamik turlarga ajratishda barcha turlar dasturni bajarish vaqtidayoq aniqlanadi.

Misollar:

Statik: C, Java, C#;Dinamik: Python, JavaScript, Ruby.

- Kuchli / kuchsiz turlarga ajratish (bular ba'zan qat'iy/qat'iy bo'lmagan deb ham ataladi). Kuchli turlarga ajratish til har xil turlarni ifodalarda aralashtirib yuborishga imkon bermasligi hamda avtomatik noaniq o'zgarishlarni bajarmasligi, masalan, satrlardan ko'plikni chiqarib tashlash mumkin emasligi bilan ajralib turadi. Kuchsiz turlarga ajratishga ega bo'lgan tillar, xatto agar aniqlikni yo'qotish yoki bir ma'noli bo'lmagan o'zgarish yuz bersada noaniq o'zgarishlarni avtomatik tarzda bajaradi.

  Misollar:

  Kuchli: Java, Python, Haskell, Lisp;

Kuchsiz: C, JavaScript, Visual Basic, PHP.

- Aniq / noaniqturlarga ajratish. Aniq turlarga ajratilgan tillar yangi o‘zgaruvchanlarni / funksiyalarni / ularning dalillarini aniq berish zarurligi bilan farqlanib turadi. Binobarin, noaniq turlarga ajratishga ega bo‘lgan tillar ushbu vazifani kompilyatorga / interpretatorga yuklaydi.

  Misollar:

  Aniq: C++, D, C#

  Noaniq: PHP, Lua, JavaScript

Shuningdek ushbu barcha kategoriyalar kesishishini ta'kidlash zarur, masalan, S tili statik, kuchsiz, aniq turlarga ajratishga, Python tili esa — dinamik, kuchli, noaniq turlarga ajratishga ega.

Biz dasturlashning deklarativ tillarida turlarga ajratish qanday rol o‘ynatish zarurligini ko‘rib chiqamiz.

7.1. Kuchli va kuchsiz turlarga ajratish

Kuchli turlarga ajratishga ega bo‘lgan tillar ifodalarda har xil turlarning mohiyatini aralashtirib yuborishga imkon bermaydi hamda hech qanday avtomatik o‘zgarishlarni bajarmaydi. Bular “qat'iy turlarga ajratishga ega tillar” deb ataladi. Buning uchun— strong typin inglizcha iborasi mavjud.

Kuchsiz turlarga ajratilgan tillar aksincha dasturchi bir iborada har xil turlarni aralashtirib yuborishiga har tomonlama imkon yaratadi, shu asnoda kompilyator barcha narsani yagona turga olib keladi. Bular “qat'iy turlarga ajratilmagan tillar” deb ataladi. Buning uchun inglizcha ibora — weak typing mavjud.

Kuchsiz turlarga ajratish ko‘pincha dinamik turlarga ajratish bilan chalkashtirib yuboriladi. Bu butkul noto‘g‘ridir. Dinamik turlarga ajratilgan til ham kuchsiz, ham kuchli turlarga ajratilgan bo‘lishi mumkin.

Til shu asnoda yana kuchli turlarga ajratishga ham ega bo‘lishi lozim. Shunsi ham haqiqatki, agar kompilyator xato to‘g‘risidagi xabar o‘rniga shunchaki songa satrni qo‘shib qo‘ysa yoki yana ham yomonrog‘i bir massivdan boshqasini ayrib tashlasa, kompilyasiyaning ushbu bosqichidagi turlarni barcha “tekshirishlardan” bizga qanday ma'no bor?

7.1.1. Kuchli turlarga ajratishning afzalliklari

- Ishonchlilik — Siz noto‘g‘ri xatti-harakat o‘rniga mustasnoga yoki kompilyasiya xatosiga ega bo‘lasiz.
- Tezlik— ancha harajatli bo‘lishi mumkin bo‘lgan, yashirin o‘zgarishlar o‘rniga kuchli turlarga ajratishda ularni aniq yozish zarur, bu dasturchini kamida kodning ushbu qismi sekin bo‘lishi mumkinligini bilishga majbur etadi.
- Dastur ishini tushunish — yana o‘sha turlarni noaniq keltirish o‘rniga dasturchi hamma narsani o‘zi yozadi va demak satrlar hamda sonlarni taqqoslash o‘z-o‘zidan yoki sexrgarlikka ko‘ra yuz bermayotganligini tahminan tushunadi.
- Aniqlik— o‘zgarishni siz o‘z qo‘lingiz bilan yozganingizda nimani nimaga o‘zgartirayotganingizni bilasiz, shuningdek qanday o‘zgarishlar aniqlik yo‘qotilishiga va noto‘g‘ri natijalarga olib kelishini doimo tushunasiz.

7.1.2. Kuchsiz turlarga ajratish afzalliklari

- Aralash ifodalardan foydalanish qulayligi (masalan, yaxlit va moddiy sonlardan).
- Turlarga ajratishdan abstraksiyalashish va vazifaga e'tiborni jamlash.
- Yozuvning qisqaligi.

7.2. Ma'lumotlar va funksiyalarin turlarga ajartish

Tur dasturlashning zamonaviy funksional tillari bilan chambarchas bog‘liq bo‘lgan tushunchadir. Dasturlashning biron bir tili, shu jumladan Haskell tili ham ma'lumotlar va funksiyalarni qat'iy turlarga ajratmasdan ish ko‘ra olmaydi. Dasturlashning ushbu tilida turlarga ajratishning anchagina umumiy tizimi joriy etilgan bo‘lib, u asossiy turlarning katta bo‘lmagan to‘plamini, shuningdek yangilarini ishlab chiqarishi uchun qudratli vositalarga ega. Ammo o‘z yangi turlarini yaratish usullarini ko‘rib chiqishdan oldin Haskell tilida ma'lumotlar tuzilmasining qanday turlaridan foydalanishni tushunish zarur.

Dasturlashning har qanday tilidagi asosiy birliklardan biri - timsoldir. Harflar, raqamlar, va cheklangan yoki cheklanmagan

uzunlikdagi maxsus alomatlar izchilligi an'anaga ko‘ra timsol deb ataladi (ba'zan bunday izchillik “identifikator” deb aham ataladi, ammo timsol tushunchasi identifikatorlar tushnchasidan bir muncha kengroqdir). Ayrim tillarda bosh va kichik harflar farqlanadi, ba'zilarida esa yo‘q. Masalan Lisp tilida kichik va bosh harflar o‘rtasida farq yo‘q, Haskell tilida esa bor.

Har qanday holatda timsollar, ko‘proq identifikatorlar — konstantalar, o‘zgaruvchan qiymatlar, funksiyalar sifatida yuzaga chiqadi. Konstantalar, o‘zgaruvchanlar, va funksiyalarning qiymatlari esa alomatlarning turlarga ajratilgan izchilligi bo‘lib, bular ham tismollardir, biroq identifikatorlarga qaraganda bir muncha boshqacha tartibdagi timsollardir. Har bir timsol muayyan turga ega. Identifikatorlar uchun ushbu identifikatorning extimoliy qiymatlari turi tur hisoblanadi. Masalan, sonli konstantaning qiymati harflardan iborat satr bo‘la olmaydi va xokazo.Funksional tillarda atom degan ajoyib tushuncha mavjud. Dasturlashning aniq tillarini amalga oshirishda timsollar va sonlar atomlar deb ataladi, shu asnoda sonlar uch ko‘rinishda: yaxlit, qayd qilingan va suzuvchi nuqtaga ega bo‘lgan sonlar bo‘lishi mumkin. barcha qolgan ushbu to‘rt asosiy turlar asosida barpo etiladi.

Ba'zida ushbu to‘plamga beshinchi asosiy tur – mantiqiy tur ham qo‘shiladi, unga ikki konstanta — true (“HAQIQAT”) va false (“SOXTA”) kiradi.

Haskell tilida asosiy turlar sifatida quyidagilardan foydalaniladi:

1. Char — xotirada bir baytni (sakkiz bitni) egallovchi literalni namoyish etish uchun tur.
2. Double — ikki yoqlama aniqlik suzuvchi nuqtasiga ega o‘nli sonlarni namoyish etish uchun tur (Double turining qiymati Float turi qiymatiga nisbatan xotirada ikki marta kattaroq o‘rinni egallaydi).
3. Float — suzuvchi nuqtaga ega o‘nli sonlarni namoyish etish uchun tur.
4. Int — [—2147483648, 2147483648] intervalga kiruvchi yaxlit sonlarni namoyish etish uchun tur.
5. Integer — har qanday qiymat (cheksizlikkacha bo‘lgan) yaxlit sonlarini namoyish etish uchun tur.

Qolgan turlarni yuqorida sanab o‘tilganlaridan turlarni barpo etish harakatlari yordamida olish mumkin. Masalan, timsollar satrini namoyish etish uchun tur timsollar ro‘yxati sifatida belgilanadi:

type String =[Char]

ma'lumotlarning har xil turlarini shu tarzda belgilash mumkin. bundan tashqari Haskell tilida ma'lumotlar tuzilmasidan farqlanadigan, asosiy turlar orqali bir muncha murakkab tarzda ifodalanadigan turlarni barpo etish uchun har xil ko‘rinishdagi harakatlar yoramida belgilangan ma'lumotlar turlari o‘rtasida uncha katta bo‘lmagan farq o‘tkaziladi. Masalan, ro‘yxatlar va kortejlar – bu ma'lumotlarning butkul yakunlangan tuzilmasi bo‘lib, ular xotirada ko‘plab narsani namoyon etishga va saqlashga imkon beradi, ammo dasturlarni ishlab chiquvchi ba'zan xotirada o‘z ob'ektlarini saqlash uchun bir muncha qulayroq tuzilmalarni yaratishni istaydi. Masalan, ilgari ma'lumotlar tuzilmasi cho‘qqilardagi erkin turdagi belgilarga ega ikki yoqlama shajaralarini namoyon etish uchun ko‘rib chiqilar edi. Ma'lumotlarning bunday turini garchi buning imkoni bo‘lsada, ro‘yxatlar yoki kortejlar orqali ifodalash qiyindir. Shu bois ushbu maqsadlar uchun Haskell tilida data kalit so‘zi bo‘lib, u taxminan quyidagi tarzda qo‘llaniladi:

data Triple a b s = Triple a b s

Birinchi qarashda g‘alati bo‘lgan bu ta'rif aslida Tripledeb ataluvchi ma'lumotlarning yangi turi tashkil etilishining yozuvidan iboratdir. Shu asnoda buning ichida uch qiymat bo‘lib, bulardan birinchisi qanday turia, ikinchisi b turi, uchinchisi esa, mos tarzda s turidir (bu yerda turlarni parametrlash texnologiyasidan foydalanilgan, bu haqida ushbu qismning oxirida batafsilroq aytib o‘tiladi). Tenglik belgisidan so‘ng yangi turdagi konstruktorga ko‘rsatma keladi. U ham Triple deb ataladi (garchi ma'lumotlar turi va uning konstruktorini bir xil nomlash shart bo‘lmasada, bu turdan foydalanishni bir muncha tushunarliroqqa aylantiradi). bunday konstruktor ko‘rsatib o‘tilgan turlarning uch qiymatini qabul qiladi.

Ma'lumotlarning bunga o‘xshash tuzilmasiga ishlov berish uchun funksiyalarning o‘zini yozib olish mumkin. Masalan, yuqorida belgilangan turning qismlariga kirish uchun selektorlar zarur. Ularni quyidagicha belgilash mumkin:

```
tripleFst (Triplexyz) = x
tripleSnd (Triplexyz) = y
```

tripleThr (Triplexyz) = z

Ammo Haskell tili ma'lumotlar turlarini quyilgan vazifaga bog'liq holda turli vaziyatlarda qo'llaniladigan bir necha konstruktorga ega ma'lumotlar turlarini aniqlashgai mkon beradi. Masalan, Prelude standart modulda foydalanishda ancha oddiy bo'lgan Maybe turi belgilangan:

data Maybe a = Nothing
| Just a

Turning bunday yozuvi o'zida qandaydir a turi qiymatiga ega Maybe turi ikki konstruktorga: birinchisi — xoli Nothing, ikkinchisi esa — xoli bo'lmagan Just ga ega ekanligini ko'rsatmoqda. Bunday turdan masalan, dasturdan chiqmasdanturli xato vaziyatlarni tutib qolish uchun foydalanish mumkin. Masalan, ro'yxatlar uchun head selektorining o'xshashi bo'lgan firstElementfunksiyasi agar xoli ro'yxat ta'rifi quyidagicha ko'rinishda bo'lsa:

firstElement [] = Nothing
firstElement (x:xs) = Justx

xoli ro'yxatga ham ishlov berishi mumkin.

Bunga o'xshash texnologiya yordamida rekursiv turlarni ham belgilash mumkin. Bunday turdagi ma'lumotlar tuzilmasini potensial cheksiz tuzilmalarga aylantiradi. Oldingi qismda (7 misol) binar shajaralarini namoyon etish uchun cheksiz tur ta'rifi ko'rsatilgan. Taxminan shu tarzda ro'yxatlarni taqdim etish uchun o'zining turini ham belgilash mumkin:

data List a = Nil
| Cons a (List a)

Ko'rinib turganidek, bunday ta'rif List (A) turidagi nazariy ta'rifga butkul mos keladi.

data kalit so'zi sanashlarni yaratish uchun ham qo'llanishi mumkin. buning uchun qancha kerak bo'lsa, shuncha xoli konstruktorlarni ko'rsatish etarlidir. Masalan, Bool turi aynan shunday belgilangan:

data Bool = True
| False

Shuning uchun bulev (ikki alomatli) mantiqda haqiqiylik qiymatlari ko'rinishi bo'lgan ushbu tur aslida Haskell tili nuqati nazaridan asosiy emas, u xatto interpretator chuqurliklarida emas, Prelude standart modulida belgilangan. Shu tarzda ish uchun dasturchiga kerak bo'lgan har

qanday sanab o‘tishlarni aniqlash mumkin. Masalan, ranglarni sanab o‘tishni quyidagi tarzda belgilash mumkin:

```
dataColor = Red
          | Orange
| Yellow
          | Green
          | Blue
          | Purple
          | White
          | Black
```

Ta'kidlash zarurki, bunga o‘xshash sanab o‘tishlarda Syoki Object Pascal singari dasturlash tillaridan farqli o‘laroq, konstruktorlarning kelish tartibi butkul muhim emas, bu yerda extimoliy qiymatlarning shunchaki ko‘pligi ko‘rsatiladi. Bunday qiymatlarni qandaydir integral turga (masalan, Int) avtomatik o‘tkazish mavjud emas, bunday maqsadlar uchun maxsus funksiyalarni belgilash zarur.

Bundan tashqari sanab o‘tishlar ta'rifiga ayrim turlarni parametrlashtiradigan qo‘shimcha konstruktorlarini kiritishga hech narsa halal bermaydi. Masalan, Color ma'lumotlar turiga RGB tizimidagi qandaydir rangni ko‘rsatish uchun Custom Int Int Int konstruktorini qo‘shib yozish mumkin bo‘lar edi.

data kalit so‘zi yordamida ma'lumotlarning algebraik turi deb ataluvchi turlar belgilanadi, bular ushbu turlar qiymatlari ustidan algebra belgilanishi mumkinligi bilan xususiyatlanadi.

7.2.1. Turlarning sinonimlari

Haskell tilida yozuvlarni bir muncha qisqa qilish uchun turlar sinonimlarini belgilash imkoniyati mavjud. Masalan, ilgari ko‘rib o‘tilgan String turi o‘z ortida Char turi qiymatlarini yashiruvchi aynan sinomimdir. Haskell tilida turlar sinonimining mavjudligi bu tilni yanada kuchli qilmaydi. Bular tilda faqat qulaylik uchun mavjuddir.

Masalan, qandaydir grafik kutubxonada uch o‘lchamdagi nuqtalarga ega ish uchun funksiya mavjud bo‘lishi mumkin edi. Buning uchun muntazam qandaydir o‘xshashlikni: (Double, Double, Double), ya'ni uch o‘lchamli makonda aynan nuqtani ko‘rsatishi mumkin bo‘lgan uch haqiqiy

sondan iborat kortejni yozib qo‘yish zarur bo‘ladi. Yozuvni qisqartirish uchun turning sinonimini:

type Point3D = (Double, Double, Double)

ta'rifini yozib qo‘yish mumkin bo‘ladi.

Mazkur holatda Point 3D identifikatori ma'lumotlar turining har qanday boshqa nomi singari turdir. Undan turlarning identifikatorlardan foydalanish mumkin bo‘lgan barcha joyida foydalanish mumkin.

Buning ustiga turlar sinonimlari huddi ma'lumotlar turlari konstruktorlari singari parametrlashtirilishi mumkin. Masalan, uch o‘lchamli makonda nuqtani tasavvur qilish uchun bir koordinatani saqlash uchun Double turi foydalanishi ishlanmasidan cheklanmasdan qandaydir turni o‘z navbatida koordinatalarni ko‘rsatish uchun qo‘llaniladigan parametrlashtiruvchi Point 3D turini belgilash mumkin:

type Point3D a = (a, a, a)

Ammo turlar sinonimlariga bir jiddiy cheklanish qo‘yiladi. Bular rekursiv bo‘la olmaydi. Shuning uchun quyidagiga o‘xshash:

type BadType = (BadType, Int)

yozuv xatodir. Ushbu maqsadlar uchun faqat data kalit so‘zidan foydalanish mumkin.

Shunday qilib, Haskell tilida yangi turlarni yaratish uchun data va type kalit so‘zlaridan foydalaniladi. Dasturlashning ushbu tilida turlar bilan bog‘liq yana bir necha kalit so‘zlar mavjud, biroq ular ob'ektli. Yo‘naltirilgan dasturlashni Haskell tilga joriy etish bilan bog‘liq dasturiy ta'minotni amalga oshirishning bir muncha murakkab jihatlariga tegib o‘tadi va shuning uchun 13 bobda ko‘rib chiqiladi.

7.2.2.Funksionaltillardagi funksiyalarning turlari

Haskell tilida turlarni statik tekshirishdan foydalaniladi. Bu demak, har qanday qiymat, har qanday ifoda, har qanday funksiya bajarilish jarayoniga qadaro‘zining muayyan turiga ega bo‘lishi lozimligini anglatadi. Masalan 'a' timsoli Char turiga ega va xokazo. Bundan funksiyalar kirishda funksiyani aniqlashdagi dalillarning tasvirlangan turiga mos keladigan dalillarni kirishda kutishi kelib chiqadi, aks holda dasturni bajarishning ushbu bosqichida turlarning mos kelmasligi xatosi paydo bo‘ladi. Bunday texnologiya dasturlarni yozish jarayonida yo‘l qo‘yiladigan xatolar sonini anchagina kamaytiradi.

Ammo dasturchining ishini osonlashtirish uchun Haskell tilida turlarni chiqarish mexanizmi joriy etilgan, ya'ni aksariyat xolatlarda dasturchi ifodalarning turini aniq ko'rsatishi kerak emas, chunki bunday tur ifoda hisoblab chiqish jarayonida qanday foydalanish asosida avtomatik tarzda hisoblab chiqiladi. Taqqoslash uchun, S singari dasturlash tillarida funksiyalar qaytaradigan o'zgaruvchanlar va qiymatlarni doimo aniq ko'rsatish zarur.

Masalan, Haskellda funksiyalarning dalillari nafaqat asosiy turlarning ob'ektlari balki, boshqa funksiyalar ham bo'lishi mumkin. kirishda boshqa funksiyalarni qabul qiladigan funksiyalar oliy tartib funksiyalari de ataladi. Bularni ko'rib chiqish uchun "funksional tur" tushunchasi, ya'ni ushbu funksiyaning turi juda qulaydir. Bu funksiya qaytaradigan qiymat turi emas, balki hisoblab chiqish jarayonini tasvirlovchi qandaydir ob'ekt sifatidagi funksiya turi ekanligini alohida ta'kidlash zarur.

Aytaylik, funksiya f kirishda A turining bir dalilini olmoqda va uni V turi qiymatini hisoblab chiqish natijasida qaytarmoqda. Mazkur holatda ushbu funksiyaning turi:

$\#(f) : A \rightarrow B$. ekanligi aytilmoqda.

#(f) alomati "f funksiyasining turi"ni anglatadi. Shunday qilib, o'q timsoli (→) bor turlar funksional turlardir. Ba'zan matematikada bularni anglatish uchun yanada nozikroq yozuvdan: B^A foydalaniladi (bundan buyon faqat o'qli yozuvdan foydalaniladi, chunki ayrim funksiya uchun ularning turlarini darajalar yordamida tasavvur qilish o'ta murakkabdir).

masalan:

#(sin) : Double → Double,

#(length) : List(A) → Integer.

Ko'plab dalillar funksiyalari uchun ularning funksional turi ta'rifini dekartcha asar amaliyoti yordamida qo'lga kiritish mumkin(masalan, $\#(add(x,y)) : (Double * Double) \rightarrow Double$). Ammo funksional dasturlashda ko'plab o'zgaruvchilarning funksional turlarini aniqlashning bunday usuli o'z o'rnini topa olmadi.

1924 yilda Mozes Shenfinkel ko'plar argumentlar funksiyalarini bir dalil funksiyalarining izchilligi sifatida tasavvur qilishni taklif etdi.Mazkur

holatda ikki asl sonni jamlaydigan funksiya turi quyidagicha ko‘rinishga ega bo‘ladi:

Double → (Double → Double).

Ya'ni, bunday funksiyalar turi o ‘q (→) timsolini izchil qo ‘llash bilan erishiladi. Ushbu jarayonni quyidagi misolda izohlash mumkin.

Misol. Add funksiyasining turi

Taxminga ko‘ra add funksiyasining dalillaridan har biri belgilangan. Bu x = 5, u = 7 bo‘lsin. Mazkur holatda add funksiyasidan birinchi dalilni ayrish yo‘li bilanyangi funksiya — add5 qo‘lga kiritiladi, u o‘zining yagona daliliga 5 sonini qo‘shadi. Uning turiga osonlik bilan erishiladi. U ta'rifga ko ‘ra quyidagicha: $Double \rightarrow Double$. Endi, orqaga qaytgan holda nima uchun addfunksiyaning turi $Double \rightarrow (Double \rightarrow Double)$ ga teng ekanligini tushunish mumkin.

add5 turining gipotetik funksiyalarini kiritish bilan havolanib ketmasligi uchun (xuddi oldingi misoldagidek) yozuvning “operator — operand” ko‘rinishidagi maxsus aplikativ shakli o‘ylab chiqilgan.funksional dasturlashdagi funksiyani yangicha ko‘rish bu unga sharoit bo‘lib xizmat qildi. Axir an'anaga ko‘ra f(5) ifodasi “5 ga teng agument qiymatiga f funksiyasini qo‘llash”ni anglatar edi (ya'ni faqat bir dalil hisoblab chiqiladi), unda funksional dasturlashda funksiyaning o‘zi ob'ekt sifatida hisoblab chiqiladi deb hisoblanadi. Masalan, 2.13 misolga qaytgan holda, add funksiyasini (add(x))y sifatida yozish mumkin, dalillar aniq qiymatlarga ega bo‘lganda esa ((add(5))7) dastlab so‘nggi dalilga qo‘llaniladigan bir dalil funksiyasi paydo bo‘lguniga qadar barcha ichki qo‘llashlar, hisoblab chiqiladi.

Shunday qilib, agar f funksiyasi $A_1 \rightarrow (A_2 \rightarrow (\ldots(A_n \rightarrow B)\ldots)$ turiga ega bo‘lsa, unda $f(a_1, a_2, \ldots, a_n)$ qiymatini to‘liq hisoblab chiqish uchun$(\ldots(f(a_1)a_2)\ldots)a_n$ ni hisoblab chiqishni izchil o‘tkazish zarur.Hisoblab chiqish natijasi esa V turidagi tob'ekt bo‘ladi.

Binobarin barcha funksiyalar yagona dalil funksiyasi sifatida ko‘rib chiqiladigan ifoda, yagona operasiya esa, applikasiya (qo‘llash) bo‘lganda bular “operator — operand” shaklidagi ifodalar deb ataladi.

Bunday funksiyalar “karrilashgan”, oldingi xat boshida keltirilgan ko‘rinishga funksiya turini olib kelish jarayonining o‘zi esa, karrilash (Karri Xaskell nomi bilan)deb ataldi.

Agarλ-hisoblab chiqish yodga olinsa, unda yozuvlarning applikativ shakllari uchun matematik abstraksiya mavjudligi ko‘zga tashlandi. Masalan:

$$f(x) = x^2 + 5 => \lambda x.(x^2 + 5),$$
$$f(x,y) = x + y => \lambda y.\lambda x.(x + y),$$
$$f(x,y,z) = x^2 + y^2 + z^2 => \lambda z.\lambda y.\lambda x.(x^2 + y^2 + z^2).$$

Va hakozo...

Haskell tilida funksiya turini tasvirlash matematikadagi notasiya singaridir. (→) o ‘q timsoli (→) timsollarining izchilligi ko ‘rinishida kodlashuvi bundan mustasnodir (bu mutloqo mantiqiydir). Shuning uchun kirishga ikkita yaxlit hisoblangan. Dalilni qabul qiladigan va haqiqiy sonni qaytaradigan someFunction qandaydir funksiyasining ta'rifi quyidagicha ko‘rinishga ega bo‘ladi:

$$someFunction :: Integer \rightarrow Integer \rightarrow Double$$

Bu erdan ko ‘rinib turganidek, bunday yozuvlardan qavslar olib tashlangan,chunki (→) amaliyoti huquqiy o ‘xshashligi faraz qilinadi.

λ–abstraksiyasi turi huddi funksiyalar turi singari belgilanishini ta'kidlash qoldi, xolos (chunki λ-abstraksiya o‘z-o‘zidan funksiya hamdir). λx. expr кўриниши λ-turi $T_1 \rightarrow T_2$, sifatida ko‘rinadi, bu erdagi T_1 — bu x o‘zgaruvchan qiymat turi, T_2 esa — expr ifodasi turidir.

7.2.3. Ko‘p qiyofali turlar

Ilgari aytib o‘tilganidek, dasturlash funksional turlaridagi ma'lumotlarning turlari Haskell tilida ham avtomatik tarzda belgilanadi. Ammo ayrim xolatlarda turni aniq ko‘rsatish zarur. Aks holda interpretator noaniqlikda adashib qoladi (aksariyat holatlarda xato to‘g‘risidagi xabar yoki ogohlantirish chiqarib beriladi). Haskell tilida maxsus timsol — (::) (ikki ikki nuqta) qo‘llaniladi, bu “turga ega” sifatida o‘qiladi. Ya'ni agar:

5 :: Integer

deb yozilsa, bunday yozuv “5 ning sonli konstantasi Integer (yaxlit son) turiga ega sifatida o‘qiladi”.

Ilgari ushbu notasiya funksiyalar turlari aniq berilgan misollarda bir necha bor qo‘llanilgan edi. Ammo e'tiborli o‘quvchi bunday misollarda funksiya turlarini tasvirlashda ilgari ko‘rib chiqilgan turlarning hamda til sintaksisi talab qilganidek, katta harflar bilan yozilishi lozim bo‘lgan tur

nomlaridan emas, balki kichik harflar bilan yozilgan qandaydir o‘zgaruvchan qiymatlardan foydalanilganligini ilg‘ashi lozim edi. Funksiya narsalarning bunday xolati osonlik bilan izohlanadi.Haskell tilida polimor turlar yoki turlar qoliplari singari soflikka ega. Bu boradagi ma'lumot 1.2 qismda keltirilgan edi. Ammo Haskell tilining ushbu jihati bu yerda batafsilroq ko‘rib chiqiladi.

Shunday qilib, “o‘zgaruvchan turlar” deb ataluvchi va yunon alfavitining kichik harflari ko‘rinishida an'anaga ko‘ra yozib olinuvchi tur polmorf turi deb ataladi. Masalan:

$$tail :: [a] \rightarrow [a]$$

yozuvi tail funksiyachi [a] → [a] ga ega, bu kirishda mazkur funksiya a qandaydir turi qiymatlarining ro ‘yxat bo ‘lgan bir dalilga ega bo ‘lishini, chiqishda esa, tail funksiyasi turi a ning endilikda aniqlangan turiga ega qiymatlar ro ‘yxati bo ‘lgan qiymatni qaytarishini nazarda tutadi.

Shunday qilib polimorf turdagi yozuvda qo‘llanilgan a timsoli turlarning o‘zgaruvchan qiymati bo‘lib, u aniq xolatda aniq qiymat bilan ko‘rsatiladi, agar oldingi funksiya turishiga yaxlit sonlar ro‘yxati kirib kelgan bo‘lsa, a turlarining o‘zgaruvchan qiymati Integer qiymatini oladi va hokazo.

Turlarning qiymatlangan o‘zgaruvchanlari turning butun yozuvi davomida o‘z qiymatiga ega bo‘lishini ta'kidlash joiz, ya'ni ular aniqlangandir va yozuvning turli joylarida turli qiymatga ega bo‘lishi mumkin emas.

Nomlash bo‘yicha kelishuvga muvofiq turlarning bunday o‘zgaruvchilari nomlari kichik harfdan boshlanishi lorzimligidir, ayni vaqtda sodda turlar nomlanishi bosh harf bilan boshlanishi lozimligini ta'kidlash joiz.

Misol sifatida Prelude standart modulida belgilangan funksiyalarga ega yana bir necha polimorf turlarni ko‘rib chiqamiz.

```
fst :: (a, b) ->a
snd :: (a, b) -> b
length :: [a] -> Int
append :: [a] -> [a] -> [a]
map :: (a -> b) -> [a] -> [b]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

bunga o‘xshash yozuvlar Haskell tili sintaksisi nuqtai nazaridan yo‘l qo‘yiladi, shuning uchun ayrim dasturchilar boshlang‘ich kodlar oson o‘qilishi uchun o‘z funksiyalarining ta'riflarini ularning turlarini shunday aniq ko‘rsatish bilan boshlaydi. Sonlarda dasturlarning kitobga ilova qilinadigan CD-diskka yozilgan boshlang‘ich matnlari misollarida barcha modullar aynan shunday tarzda shakllantirilgan – har bir funksiya ta'rifi uning funksional turini tasvirlashdan boshlanadi.

Ammo shuni ta'kidlash joizki, funksiya turini uning ta'rifida (::) timsollar yordamida aniq ko‘rsatishdan funksiya turiga cheklash qo‘yish uchun ham foydalanish mumkin bunday cheklanish, ba'zan optimallashtiruvchi harakatni bajargan holda funksiyadan foydalanish sohasini toraytirishi mumkin. Masalan, agar faqat cheklangan yaxlit sonlarni turlarga ajratish zarur bo‘lsa, unda funksiya turini quicksort funksiyasini turini:

quicksort :: [Int] -> [Int]

ko‘rinishida belgilash mumkin.

Signaturasi shu tarzda belgilangan funksiya tezroq ishlaydi va kamroq zahiralarni talab qiladi, chunki Haskell tilining kompilyator qiymatini tenglash mumkin bo‘lgan har qanday tur ma'lumotlarini emas, balki yaxlit sonlarni cheklash uchun turlarga ajratishga optimallashtirishni amalga oshiradi. Bu ishda eng muhimi funksiyadan foydalanishni cheklovchi bunday tur parametrlashtiruvchi o‘zgaruvchan qiymatlarni almashtirishga qadar aniqlikdagi funksiyaning umumiy turiga mos kelishidir.

Nazorat savollari

1) Juftlarni barpo etish amaliyoti qanday tuzilgan? Ushbu amaliyot xotirada qanday ob'ektni yaratadi? Ushbu ob'ektni qanday grafik tasavvur qilish mumkin?

2) Haskell tilidagi kichik va bosh harflar bilan boshlanadigan identifikatorlarning farqi nimada? Bunday identifikatorlarni aralashtirib yuborish mumkinmi?

3) “Namuna” nima?

4) O‘z operandlari o‘rtasida ikki dalilga ega bo‘lgan funksiyani qanday yozib olish mumkin? Yozib olishning bunday usuli nima deb ataladi?

5) Haskell tilidagi data va type xizmat so‘zlaridan foydalanish nima bilan farqlanadi?

6) Sanashni qanday ta'riflash mumkin?

7) Karrilashgan funksiyalar nima? Haskell tilida karrilashmagan funksiyalardan foydalanish mumkinmi?

8) Polimorf turlar nima? S tili nuqtai nazaridan bunday turlar ko‘proq nimaga o‘xshadi.?

9) “Parametrik polimorfizm nima”? bunday polimorfizm “ad hoc” polimorfizmidan nimasi bilan farqlanadi?

10) Polimorfizmning ko‘rsatib o‘tilgan ikki turi Haskell tilida qanday foydalaniladi?

11) Ma'lumotlar turlari, funksiyalar turlari va toifalarni tasvirlashda qo‘llaniladigan “kontekst” nima?

12) Xindli-Milner turlarga ajratish modeli nimaga asoslangan? Ushbu model doiralaridagi turlarnii chiqarish algoritmi ishining asosiy tamoyillari qanday?

VIII. BOB. MA'LUMOTLARNING REKURSIV TUZILMALARI

8.1. Ro‘yxatlar– funksional tillarning asosidir

Bir necha bor eslatib o‘tilganidek, ro‘yxat – bu asosiy tuzilma bo‘lib, funksional dasturlash bilan shug‘ullanadiganlar uchun o‘ta maqbuldir. Ishning bunday xolatiga hayratlanib bo‘lmaydi. Chunki ro‘yxatlar yordamida oddiy massivlardan murakkab tuzilmalargacha bo‘lgan ma'lumotlarning har xil tuzilmalari katta miqdorini tasavvur qilish, xususan agar ro‘yxatlarda bir turdagi qiymatlar saqlanadigan bo‘lsa, ularga shunchaki ishlov berish mumkin (bu masalan, Haskell tili uchun xosdir, biroq Lisp tiliga xos emas). ro‘yxatlar yordamida har xil turdagi hisoblab chiqishlarni shakllantirish, shuningdek, masalan cheksiz izchillik singari ma'lumotlar turlarini dasturlashning oddiy tillari uchun endilikda butkul nostandart tasvirlash mumkin. Cheksiz izchilliklar faqat dasturlashning qat'iy bo‘lmagan tillaridagina (ya'ni chetga surilgan hisoblab chiqishiqlarga ega tillarda) mavjuddir .

List(A) турини расмий таърифи

List(A) = NIL + (Ax List(A))

prefix = constructor List(A)

head, tail = selectors List(A)

isNil, isNonNil = predicates List(A)

nil,nonNil = parts List(A)

List(A) turining ushbu ta'rifi mohiyatiga ko‘ra A turining asosiy (atomar) qiymati asosiga barpo etilgan ayrim murakkab qiymatlarning induktiv ko‘pligi tarifidir. Bunday induktivko‘pliklar, shu jumladan mazkur turlar qiymatlariga ishlov berish uchun yaratilgan funksiyalari xossalarini isbotlashga ham imkon beradi.

Mazkur ta'rif ro‘yxatning “nazariy tushunchasini” uni dasturlashning muayyan tilida, shuningdek uning sintaksisida ifodalagan holda amaliy sirtga ko‘chirishga imkon beradi.

8.1.1. Ro‘yxatlarni Haskell tiliga ko‘chirish

Yuqorida keltirilgan va A turidagi unsurlar ustidan ushbu ro‘yxatni namoyon etuvchi List(A) tuzilmasining aniq proeksiyasini ko‘rib chiqish uchun dasturlashning muayyan tilida dastlab ayrim alomatlarni aniqlash va shuningdek, ayrim kelishuvlarni kiritish zarur. Norasmiy tarzda bunday

kelishuvlar List(A) tuzilmasining o‘zini ta'riflashda keltirib o‘tilgan edi, ammo endilikda uni yanada rasmiy amalga oshirish mumkin.

Haskell tilida ro‘yxatlarni taqdim etish uchun tuzilmani dastlab u ro‘yxatda saqlanadigan unsurlar turiga tobe' bo‘lib qolmaydigan tarzda belgilash zarur, ya'ni mohiyatiga ko‘ra List(A) ning ta'rifi a ga bog‘liq bo‘lib qolmasligi lozim. Bu List(A) turidagi konteyner turi ko‘rinishidagi, ya'ni unda qandaydir qiymatlar saqlanadigan ta'rif yordamida amalga oshiriladi.

Ro‘yxatning ichida saqlanadigan (mavjud bo‘lgan) ushbu qiymatlar J. MakKarti tomonidan an'anaga ko‘ra “atomlar” bularning a turi esa shunga mos ravishda “atomar tur” deb atab kelinadi. Shu tarzda ro‘yxat uchun ushbu atomlar, ularning atomar turi butkul muhim emas (va muhim bo‘lishi ham lozim emas).

Atomar ob'ektlar uchun nazariy mulohazalarga mos ravishda ro‘yxatda bunday ob'ektlarni shakllantirish uchun konstruktor va selektorlarning mavjudligi ko‘rsatib o‘tiladi (postuliruetsya). Konstruktor va selektorlar asosiy amaliyotlar hisoblanadi. Bularning yordamida barcha qolgan amaliyotlar ro‘yxatlar ustidan harakatlar va funksiyalar yaratiladi va tasvirlanadi. Dasturlashning har bir funksional turi asosiy amaliyotlarni o‘zicha tasvirlaydi, biroq ularning tili yagona bo‘lishi lozim – bu List (A) turining rasmiy ta'rifida beriladi.

List(A) turi uchun bir konstruktor va ikki selektor belgilangan.Haskell tilida ushbu maqsadlar uchun quyidagi alomatlardan foydalaniladi:

1. Juftlikni (konstruktor) yaratish amaliyoti— (:) (ikki nuqta).

2. Ro‘yxatning boshiga ega bo‘lish amaliyoti (birinchi selektor) — head.

3. Ro‘yxat dumiga ega bo‘lish amaliyoti (ikkinchi selektor) — tail.

Barcha tasvirlangan uch amaliyot (Haskell tili nuqtai nazaridan — bular shubhasiz funksiyalardir) ma'lumotlar turlarining konstruktorlari va selektorlarini bog‘lovchi aksiomadan bevosita kelib chiqadigan quyidagi nisbatlar bilan bir-biriga bog‘liqdir:

1) head (x:y) = x.

2) tail (x:y) = y.

3) (head (x:y)) : (tail (x:y)) = (x:y).

Bu yerda x va u — juftlik yaratiladigan ayrim ob'ektlardir .

Ammo tasvirlangan funksiyalar mohiyatiga ko‘ra List(A) turini tasvirlash uchun etarli emas, chunki ular ko‘rsatib o‘tilgan uch funksiyani erkin qo‘llash yordamida qo‘lga kiritiladigan qiymatlarga ega bo‘lgan S-ifodalar (SExpr(A) turi) bilan ish ko‘rish uchun mo‘ljallangandir. SExpr(A) —List(A)turiga nisbatan bir muncha umumiyroqdir, shuning uchun cheklanish kiritish zarur. Ro‘yxatlar uchun bunday cheklanish bo‘sh ro‘yxat ko‘rinishidagi nol unsurining mavjudligidir. Bu Haskell tilida [] sifatida tasvirlanadi (matematik notasiyada ham shunday alomatdan foydalaniladi,dasturlashning boshqa tillarida boshqa timsollar qo‘llanishi mumkin).

Ushbu unsur ham Aning atomar turiga, garchi bu hech qaerdan kelib chiqmasada, mansub hisoblanadi, bu tasdiq kelgusidagi mulohazalar va ishlar qulayligi uchun shunchaki ko‘rsatib o‘tiladi. Ushbu maxsus atom kiritilganidan so‘ng List(A) turiga rasmiy ta'rif berish uchun barcha narsa tayyordir:

1-ta'rif. A turidagi unsurlar ustidan ro‘yxat

1. $[] \in List(A)$.
2. $x \in A \wedge y \in List(A) \Rightarrow (x : y) \in List(A)$.

Ushbu ta'rifdan ro‘yxatlarning quyidagi formulalar bilan ta'riflanadigan asosiy xossasi keladi:

$x \in List(A) \wedge x \neq [] \Rightarrow headx \in A$; (2.1)

$x \in List(A) \wedge x \neq [] \Rightarrow tailx \in List(A)$ (2.2)

Shu tarzda ro‘yxat bu juftlik bo‘lib, uning birinchi unsuri atomar qiymat, ikkinchisi esa ro‘yxatdir. n unsurlaridan iborat ro‘yxat juftlik ko‘rinishida quyidagi tarzda yoziladi: $(a_1 : (a_2 : (\ldots(a_n : [])\ldots)))$. Bunday yozuv unchalik aniq emas, shuning uchun dasturlashning turli tillarida n unsurlaridan ro‘yxatlarin qisqacha yozib olish uchun notasiyalardan foydalaniladi. Masalan, Lisp turidayozuvning quyidagi usulidan foydalaniladi:

$$(a_1 a_2 \ldots a_n)$$

Shu asnoda uch nuqta (...) o‘tkazib yuborilgan unsurlarni ko‘rsatadi – bu timsol til sintaksisining qismi emas, boshqa tomondan Lisp tilida juftlikni barpo etish amaliyoti — (.) nuqtasi bilan belgilanadi, shuning uchun bunday ro‘yxatni yanada “to‘liq shaklda ham yozish mumkin” (yana uch nuqta qisqartirish uchun qo‘llaniladi):

$$(a_1.(a_2.(\ldots(a_n.NIL)\ldots)))$$

Haskell tilida matematik yozuvga eng ko‘proq yaqinlashtirilgan notasiya qabul qilingan (ro‘yxatdagi vergulni navbatdangi qiymatdan ajratuvchi bo‘shliqlar opsionaldir):

$$[a_1, a_2, \ldots a_n]$$

Yoki juftlik konstruktori Haskell tilida (:) ko‘rinishida anglatilishini e'tiborga olgan holda quyidagicha yozish mumkin:

$(a_1 : (a_2 : (\ldots(a_n : [])\ldots)))$.

Ammo ro‘yxatlar — bu ma'lumotlarning anchagina tor ixtisoslashgan tuzilmasidir, buning ustiga agar bunday ro‘yxatlarga ro‘yxatning barcha unsurlari aynan o‘sha A turini egallashi lozimligidan iborat bo‘lgan cheklanish qo‘yiladi (Lisp tilida bunday cheklanish yo‘q). Vazifalarning qandaydir ko‘pligini hal qilish uchun “A ustidan ro‘yxatli tuzilma” deb ataluvchi ma'lumotlarning qo‘shimcha turini belgilash zarur (alomat — ListStructure(A)).

Ro‘yxatli tuzilmaning ta'rifi quyidagi ko‘rinishdadir.

2-ta'rif. A turidagi unsurlar ustidan ro‘yxatli tuzilma.

1. a ϵ A => a $\epsilon ListStructure(A)$.
2. $List(ListStructure(A))$ $\epsilon ListStructure(A)$.

Ya'ni ro‘yxatli tuzilma — alomatlari ham atomlar, ham boshqa ro‘yxatli tuzilmalar, shu jumladan oddiy ro‘yxatlar ham bo‘lishi mumkin bo‘lgan ro‘yxatdir. Ayni vaqtda oddiy ro‘yxat bo‘lgan ro‘yxatli tuzilmaga quyidagi ifoda misol bo‘lib xizmat qilishi mumkin:

$$[a1, [a2, a3, [a4]], a5].$$

Ro‘yxatli tuzilmalar uchun kiritilganlik darajasi sifatidagi tushuncha kiritiladi, bu o‘zidan oldin ro‘yxatli tuzilmaning qandaydir unsurini olib keladigan ochiruvchi qavslar «[», maksimal miqdoriga tengdir (ammo ushbu miqdorga hisoblab chiqiladigan unsurga qadar yopiladigan qavslar kirmaydi).

Masalan, yuqorida keltirilgan misolda a4 unsuri uchunkiritilganlik darajasi 3 ga, a5 unsuri uchun esa— bor-yo‘g‘i 1 ga tengdir. Binobarin yuqorida keltirilgan misoldagi ro‘yxatli tuzilmaning o‘zidagi kiritilganlik darajasi 3 ga tengdir.

8.1.2. Misollar

Juftliklar va ro‘yxatlar ustidan asosiy harakatlarni amalga oshirish uchun mo‘ljallangan ko‘rib chiqilgan amaliyotlarning mazmuni va mohiyatiga yanada batafsilroq kirish uchun bir nechta misollarni ko‘rib chiqish zarur. Bular dasturlashning funksional tillaridagi dasturlar qanday ishlashini tushunishga yordam beradi. Shuningdek, turli vazifalar uchun funksiyalarni barpo etish usulini tushunish imkon beradi.

1-misol. (:) juftlik konstruktori

Yanada murakkabroq misollarni keltirishdan oldin (:) juftlik konstruktorining o‘zi qanday ishlashini batafsil ko‘rib chiqish zarur. Bu mazkur funksiyani qo‘llash yordamida xoh ular List(A) ro‘yxatlari,ListStructure(A) ro‘yxatli tuzilmalari yoki SExpr(A) 5 ifodasi yordamida olinadigan ma'lumotlar tuzilmasini doimo farqlash imkonini beradi.

1. $a1 : a2 = (a1 : a2)$ — $S-$ ифода
2. $[a1, a2]$: [a3, a4] = $[[a1, a2], a3, a4]$— ro′yxatil tuzilma.
3. $a1 :$ [a2, a3] = $[a1, a2, a3]$ — $ro'yxat.$

Ko‘rinib turganidek (:) konstruktorni qo‘llash vaqtida doimo natijada ro‘yxat emas, balki qandaydir ko‘proq umumiylik olinishi mumkinligini doimo yodda tutish zarur

Ikki ro‘yxatni zanjirlash uchun (:) konstruktoridan foydalanib bo‘lmaydi. buning uchun append (konkatenasiya) funksiyasi qo‘llaniladi, bu Haskell tilida, shu jumladan infikslishaklga (++)ga ega. buning ustiga funksional dasturlashga endigina kirib kelganlar

$$[a1] : [a2, a3]$$

ifodani hisoblab chiqish natijasi [$a1$, $a2$, aZ] ro‘yxatiga tengligini faraz qilgan holda anchagina keng tarqalgan va o‘ta noxush tuzoqqa tushadi. Bu shunday emas, hisoblab chiqish natijasi boshqa hech narsa emas, balki ro‘yxatli tuzilma

[[$a1$] , a2, aZ] bo‘ladi. Ro‘yxat (:) konstruktor birinchi argumenti agar atom ikkinchisi esa ro‘yxat bo‘lgandagina qo‘lga kiritiladi. Bu List(A) ma'lumotlar tuzilmasiga aniq mos keladi.

Ushbu bobga vazifalarda bir necha misollarni hal qilish uchun turli operantlarga (:) konstruktorini qo‘llash keltirilgan.

2-misol. length ro‘yxati uzunligini aniqlash funksiyasi

Length funksiyasini amalga oshirishni boshlashning o‘zidan oldin, unimani qaytarishi lozimligini tushunish zarur. length funksiyasi natijasining tushunchali ta'rifi “funksiyaga parametr sifatida berilgan ro‘yxatdagi unsurlar miqdori” sifatida yangrashi mumkin. Bu yerda ikki xolat yuzaga keladi — funksiyaga bo‘sh ro‘yxat berilgan va funksiyaga bo‘sh bo‘lmagan ro‘yxat berilgan. Birinchi xolatda barcha narsa aniq – natija nol bo‘lishi lozim, ikkinchi xolatda vazifani head va tail selektorlari yordamida ro‘yxatning boshiga va dumiga uzatilgan ikki kichik vazifaga ajratish mumkin.

head selektori ro‘yxatni birinchi unsurini qaytarishi tail selektori esa qolgan unsurlardan iborat unsurlardan iborat ro‘yxatni qaytarishi butkul tushunarlidir.Tail selektori yordamida olingan ro‘yxat uzunligi ma'lum bo‘lsin, unda boshlang‘ich ro‘yxatning uzunligi birlikka orttirilgan ma'lum uzunlikka teng bo‘ladi. Ushbu holatda length funksiyasiningg o‘z ta'rifini osonlik bilan quyidagicha yozish mumkin:

```
length l = if (l == []) then 0
        else 1 + length (taill)
```

length funksiyasi ta'rifining bunday yozuvi u abstraktmatematik notasiyada yozilishi mumkin bo‘lgan xolatga aniq mos keladi. Ammo Haskell tilida funksiyalar ta'rifini yanada ifodaliroq yozish uchun vositalar mavjud. Namunalar va yopiqliklar ushbu vositalarga mansubdir. Bu yerda ushbu tushunchalarni batafsil ko‘zdan kechirmasdan Haskell tilidagi length funksiyasini yanada aniqroq ta'rifi quyidagicha ko‘rinishda bo‘ladi:

```
length [] = 0
length (x:xs) = 1 + lengthxs
```

ko‘rinib turganidek, bunday yozuv, bir muncha nozikroq va oldingidan qisqaroqdir. Funksional dasturlash paradigmasini tushunib ulgurgan dasturchilar uchun esa, bunday yozuv shartli operatorlar yoki ko‘plik tarmoqlanish operatorlari qo‘llaniladiganga nisbatan bunga o‘xshash yozuvlar yanada tushunarliroq bo‘ladi.

3-misol. Appendikki ro‘yxatini qo‘shilish funksiyasi

Ro‘yxatlarni qo‘shilish (zanjirlanish yoki konkatenasiya) funksiyasini ko‘plab usullar bilan amalga oshirish mumkin. Xayolga birinchi keladigan narsa — destruktiv o‘zlashtirishdir, ya'ni birinchi ro‘yxat oxiridagi [] ko‘rsatkichini ikkinchi ro‘yxat boshidagi ko‘rsatkichga almashtirish va bu bilan birinchi ro‘yxatda qo‘shilish

natijasini qo'lga kiritish mumkin. Ammo echimning bunga o'xshash ko'rinishida birinchi ro'yxatning o'zi o'zgaradi. Bunday usullar funksional dasturlashda qat'iy ta'qiqlangan (garchi arim funksional tillarda baribir bunday imkon mavjudligini yana bir bor ta'kidlash zarur).

Ikkinchi usul birinchi ro'yxatdan nusha olish va so'nggi ko'rsatkichga ikkinchi ro'yxat birinchi unsuriga iqtibos nushasini joylashtirishdan iboratdir. Ushbu usul destruktivlik nuqtai nazaridan yaxshidir (buzg'unchi va nuqsonli harakatlarni amalga oshirmaydi), ammo xotira va vaqtni qo'shimcha sarflashni talab qiladi. Vazifa echimining bunday ko'rinishida append funksiya(yoki infiksli shaklda — (++)) quyidagicha ko'rinishga ega:

$$[] ++ 1 = 1$$

$$(x{:}xs) ++ 1 = x : (xs ++ 1)$$

Ushbu yozuv bosqichli barpo etish yordamida ikki berilgan konkatenasiyaga teng bo'lgan yangi ro'yxatni qanday barpo etish mumkinligini ko'rsatmoqda.

4-misol. Barcha kichik ko'pchiliklarning ko'pchiligiga ega bo'lish

Yanada murakkabroq misol berilgan ko'plikning barcha kichik ko'pliklari ko'pligini qaytaradigan funksiyani barpo etishdan iboratdir. Mayli bunday ko'pliklar ro'yxatlar ko'rinishida taqdim etila qolsin, shu asnoda bunday ro'yxatlarda takrorlanuvchi unsurlar bo'lmasligi lozim. Unda kirishda ro'yxatni qabul qilgan holda har biri boshlang'ich ko'plikning kichik ko'pliklarin namoyon etadigan ro'yxatlarning ro'yxatini qaytaradigan funksiyani yozish zarur.

Dastlab matematikada bunday ko'plik qanday aniqlanishini yodga olish zarur. Bularni(ayrim ko'plikning barcha kichik ko'pliklarning ko'pligi shunday deb ataladi) hisoblab chiqish uchun formula quyidagicha ko'rinishga ega:

$$\{\emptyset\}, \quad \text{arap } A = \emptyset$$

$\beta(A) = \{\beta(A\backslash x) \cup \left(\cup |\beta(A\backslash x)|j = 1\{x\} \cup B, B \subseteq \beta(A\backslash x)\right)$ aks holda.

Bunga o'xshash matematik formulalar Haskell tili sintaksisiga osonlik bilan o'tkaziladi. Mazkur holatda o'xshash sodda ko'chirish mavjud, unda funksiyaning ta'rifi quyidagi ko'rinishga ega:

```
getSubsets [] = [[]]
```

```
getSubsets (x:xs) = ss ++ map (x:) ss
where ss = getSubsets xs.
```

Ushbu ta'rifni quyidagicha o‘qish zarur: birinchi satr agar funksiyaning kirishiga bo‘sh ro‘yxat berilgan holatda funksiyaning xatti-harakatini belgilaydi, matematik formulaga muvofiq bo‘sh ko‘plik barcha ko‘pliklarining ko‘pligi faqat bir bo‘sh ko‘plikdan iboratdir va bu funksiyani bajarish natijasi sifatida ham yozib qo‘yilgan(ushbu misolni ko‘rib chiqishda mazkur holatdagi ko‘plik ro‘yxatlar yordamida kodlashtirilishini doimo yodda tutish lozim).

Ta'rifning ikkinchi satri kirishga bo‘sh bo‘lmagan ro‘yxat berilgan holatda funksiya qaytaradigan qiymatni belgilaydi mazkur holatda boshni (x) va bo‘sh bo‘lishi ham mumkin bo‘lgan dumni (xs) doimo ajratib ko‘rsatish mumkin.Unda natija ro‘yxat boshidan va ro‘yxatining har bir unsuridan yana dum uchun getSubsets funksiyasi yordamida olingan juftlikni tuzish amaliyotini qo‘llash yordamida olingan ro‘yxatga ega dum uchun aynan shu funksiyani bajarish natijasining konkatenasiyasiga tengdir. Bunday qo‘llash Prelude standart modulidagi tar funksiyasidan foydalanishdan kelib chiqadi.

Ta'rifning uchinchi satri hisoblab chiqish jarayonlarini optimallashtirish maqsadida cheklangan o‘zgaruvchan qiymatni kiritishdan iboratdir. Ko‘rinib turganidek, ikkinchi satrida ro‘yxatning dumi uchun getSubsets funksiyasini bajarish natijasi ikki marta qo‘llanilmoqda. Ushbu qiymatni ikki marta hisoblab chiqmaslik uchun where xizmat so‘zi yordamida cheklangan o‘zgaruvchi qiymat kiritiladi.

8.1.3. Ro‘yxatni belgilovchilari va matematik izchilliklar

Ehtimol Haskell tili — bu qandaydir sodda matematik formulaga asoslangan ro‘yxatlarni oson va jadal barpo etishga imkon beradigan dasturlashning yagona tili bo‘lsa kerak. Birinchi navbatda bunday formulalarga arifmetik progressiyalar tegishlidir. Ammo bular bilan ish chegaralanib qolmaydi, oddiy sonlar ro‘yxatini, mukammal sonlar ro‘yxatini va hakozlarni juda oson aniqlash mumkin (ushbu ikki misol quyiroqda batafsil ko‘rib chiqiladi).

Bunga o‘xshash yondoshuv ilgari Xoar usuli bilan ro‘yxatlarni jadal turlarga ajratish funksiyasini barpo etishda qo‘llanilgan edi. Ushbu usul

Haskell tilida "ro'yxatlarni belgilovchi" nomiga ega ro'yxatlarni belgilovchilarning eng umumiy turi quyidagi ko'rinishga ega:

$$[x \mid x \leftarrow xs].$$

Bu yozuv "barcha bunday x lardan ro 'hat xs dan olingan" tarzida o 'qilishi mumkin. "$x \leftarrow xs$" tuzilmasi generator deb ataladi.Bunday generatordan so 'ng (u o 'zi belgilaydigan har bir qiymat uchun yagona bo 'lishi va ro 'yxatlarni belgilovchi yozuvda birinchi turishi lozim) vergullar bilan ajratilgan qo'riqchi ifodalarning ayrim soni turishi mumkin. Mazkur xolatda barcha shunday x lar, barcha qo'riqlovchi ifodalarni qiymati bunga to'g'ri bo'lgan ko'rinishda tanlab olinadi, ya'ni

$$[x \mid x \leftarrow xs,\ x > m,\ x < n]$$

ni "Barcha bunday x lar ro'yxati xs dan olingan, u (x m dan kattaroq) va (x n dan kichikroq)" tarzda o'qilishi mumkin.

Ro'yxatlarni aniqlovchilari oddiy qiymatlarga qaraganda ma'lumotlarni yanada murakkab tuzilmalaridan ro'yxatlar yaratilishi uchun qo'lanilishi mumkin. Buning uchun o'zgaruvchi qiymatlardan qancha kerak bo'lsa, shuncha foydalanish mumkin. masalan:

$$[(x, y) \mid x \leftarrow xs,\ y \leftarrow ys,\ x <= y]$$

Bu yozuv quyidagicha o'qiladi: "xs va ys ro'yxatlarning barcha unsurlaridan juftlik ro'yxati, shu asnoda juftlikni birinchi unsuri ikkinchi unsuridan katta bo'lmasligi lozim". Ko'rinib turganidek, bu ish uncha qiyin emas.

5-misol. Oddiy sonlarning cheksiz ro'yxati

Faqat birga va o'z-o'ziga (ya'ni, ikki yaxlit bo'luvchiga ega bo'lgan)butunlay bo'linadigan natural son oddiy son deb ataladi. Quyida oddiy sonlarning cheksiz ro'yxatini qaytaradigan o'ta sodda (har qanday turdagi optimallashtirishsiz) ko'rsatilgan.

```
primes = [x | x <- [2..],
          isPrime x]
  where isPrime x = (dividers x == [1, x])
dividersx = [y | y <- [1..x],
```

mod x y == 0]

primes funksiyasi barcha bunda natural x lar, ya'ni [2..] natural sonlar ko‘pligidan olingan ro‘yxatni qaytaradi (bir ushbu ko‘plikdan chiqarib tashlangan, chunki u ta'rifiga ko‘ra oddiy son emas), shu asnoda x ning har bir qiymati faqat oddiy sonlar uchun “HAQIQAT” qiymatini qaytaradigan isPrime predikatini qondirishi lozim.

isPrime predikati berilgan son bo‘luvchilarini “1” qiymati va sonning o‘zidan iborat bo‘lgan ro‘yxatga taqqoslaydiva agar ta'rifga to‘la mos tarzda u bunday ro‘yxatga teng bo‘lsa predikat “HAQIQAT” qiymatini qaytaradi, aks holda u “SOXTA” qiymatini qaytaradi.

Berilgan son bo‘luvchilari ro‘yxati yana ro‘yxatni belgilovchisini yaratish texnologiyasi yordamida hisoblab chiqiladi. Berilgan son bo‘luvchilari ro‘yxatiga kichik yoki sonning o‘ziga teng bo‘lgan barcha shunday sonlar kiradi, boshlang‘ich sonni bo‘lishdan qoldiq nolga tengdir (yaxlit sonni bo‘lishdan qoldiq Prelude standart modulidagi mod funksiyasi yordamida hisoblab chiqiladi).

Ushbu funksiyani ko‘rib chiqishdan juft sonlarni mustasno etgan holda optimallashtirish mumkin, chunki ulardan faqat birginasi oddiy sondir, shundan so‘ng ayrim yanada nozik sozlashlarni ham amalga oshirish mumkin bo‘ladi. Natijada taxminan quyidagi ko‘rinishdagi funksiya paydo bo‘lishi mumkin:

primes = 2 : [x | x ← [3, 5..],
([y | y ← [2..(*div* x 2)],
mod x y == 0] == [])]

Mazkur ta'rifda yaxlit sonli bo‘lishni amalga oshiradigan Prelude standart modulidan div funksiyasi ham qo‘llaniladi. O‘quvchiga qanday ishlashini mustaqil tagiga etish taklif etiladi.

Ushbu misolni yozish vaqtida HUGS 98 interpretatorida primes funksiyasi optimallashgan ko‘rinishda bajarish uchun ishga tushirilgan edi. Interpretator barcha misollar ustidan ish tugagunga qadar ishni oxiriga etkaza olmadi. 26099 soniga to‘xtagan holda uning ishini qo‘lda to‘xtatishga to‘g‘ri keldi.

Oddiy sonlarni hisoblab chiqish uchun keltirilgan funksiya u qadar optimallashtirilmaganligini ta'kidlash zarur. Anchagina katta sonlar uchun u S tilidagi shunga o‘xshash funksiyaga nisbatan taxminan 40 bor sekinroq

ishlaydi. Masalan, ushbu funksiya yordamida 10 000 li oddiy son hisoblab chiqilgan.

GHC dagi kompilasiyalashgan modul uni 148 soniyada hisoblab chiqdi, ayni vaqtda shu kompyuterdagi S tilidagi dastur bunday oddiy sonni 4 soniyada hisoblab chiqdi.

Biroq, bu g'amga botish uchun sabab emas, oddiy sonlarni hisoblab chiqish uchun funksiyani kompiliyasiyalashgan modul S tilidagi dasturga nisbatan xatto tezroq ishlaydigan qilib, optimallashtirish mumkin. Masalan, navbatdagi kod har qanday hajmdagi oddiy sonlarni hisoblab chiqishda ikki bor ko'proq yutuq berishi mumkin:

```
update [] = []
update ((x, m):xs) = (if (x' >= m) then (x' - m, m)
                          else (x', m)):update xs
  where x' = x + 2
nonZero x m = and $ map (0 /=) $ map fst $ takeWhile less m
where less (_, y) = y * y <= x
nextP x m | nonZero x m = x:(nextP (x + 2) (update m ++ [(2, x)]))
          | otherwise = nextP (x + 2) (update m)
primes :: [Int]
primes = 2:(nextPrime 3 [])
```

Ha, bunday kod o'zining tushunarliligi va ifodaliligi bilan hayratlantirmaydi, ammo u yaxshi translyatorlar bilan ishlov berilgan dasturlashning zamonaviy funksional tillaridagi dasturlar dasturlashning an'anaviy imperativ tillarida yozilgan dasturlardan unumdorligi bo'yicha sira kam emas.

6-misol. Mukammal sonlarning cheksiz ro'yxati

O'zini mustasno etganda o'z bo'luvchilari qiymatiga teng son mukammal son deb ataladi. Masalan, 6 soni mukammal sondir, chunki u butunligicha 1, 2 va 3 ga bo'linadi va o'z navbatida ushbu sonlarning qiymatiga tengdir: 1 + 2 + 3 = 6. Mukammal sonlar anchagina kam (birdan milliongacha bo'lgan sonlar orasida mukammal sonlar bor-yo'g'i - to'rtta, milliardgacha – beshta, trilliongacha esa — oltitadir), va matematiklar xatto ular cheksiz ekanlii yoki yo'qligini bilmaydi.

Mukammal sonlarning cheksiz ro'yxatini hisoblab chiquvchi funksiya ham ro'yxatlarni aniqlovchilardan foydalanishga asoslangan. Uning ta'rifi quyidagicha ko'rinishi mumkin:

```
perfects = [x | x <- [1..],
            sum ([y | y <- [1..(x - 1)],
                  mod x y == 0]) == x]
```

Ushbu ta'rifda o'ziga ro'yxatning dalili sifatida uzatilgan barcha unsurlarni qaytaradigan Prelude standart modulida aniqlangan sum funksiyasidan foydalanilgan. Birdan boshlanadigan berilgan son bo'luvchilar ro'yxati uning dalili sifatida yuzaga chiqadi (bunday ro'yxatning shakli oldingi misolda ko'rib chiqilgan edi). Binobarin barcha natural x, uni bo'luvchilari qiymati x ning o'ziga teng va perfects funksiyasi qaytadigan ushbu cheksiz ro'yxatni tashkil etadi. Bularning barchasi aniq matematik ta'rifga mos keladi.

Ushbu funksiya yordamida milliondan kichik bo'lgan barcha mukammal sonlar hisoblab chiqilgan. Bunday mukammal sonlarning ro'yxati quyidagicha ko'rinadi:

$$[6, 28, 496, 8128]$$

Albatta, keltirilgan funksiya mukammal sonlarni yana butkul ortiqchalik bilan izlaydi, bu uning tez harakatlanishiga o'ta jiddiy ta'sir ko'rsatadi. Ammo ushbu misol iterativ ortiqchaliklarni shu jumladan, qo'shimcha tekshirishlarga ega ahamiyatsiz kiritilgan davriyliklarni amalga oshirish uchun tez funksiya yaratish imkonini ko'rsatmoqda. O'quvchi mustaqil tarzda Mersenning oddiy sonlarini mukammal sonlarga bog'lovchi formula asosida mukammal sonlarni hisoblabchiqish uchun funksiyani amalga oshirishga mustaqil tarzda urinib ko'rishi mumkin:

$$P_p = 2^{p-1} * (2^p - 1) \qquad (2.4)$$

bu yerda — $(2^p - 1)$ Mersenning oddiy sonidir. Bugunga kunga qadar toq mukammal sonlar mavjudligi umuman isbotlanmagan (shuningdek rad ham etilmagan). Ko'rsatilgan formula bo'yicha sonlarni hisoblab chiqish uchun funksiyani amalga oshirish esa, Haskell tilining butun qudratini ko'rsatmoqda, chunki bunday funksiya o'nli alomatlarning ulkan miqdoriga ega sonlarni hisoblab chiqishga imkon beradi.

Keltirilgan misollardan ko'rinib turganide, Haskell tili ancha murakkab tabiatli ro'yxatlarni jamlash uchun hayratli imkoniyat taqdim etadi. Shu ansoda ish qiymatlar sifatidagi faqat natural sonlar bilan cheklanib qolmaydi.

U qiymatlar ro‘yxatlarni aniqlovchilar texnologiyasi yordamida yaratilishi, ma'lumotlarning har qanday tuzilmasidan foydalanishi mumkin. Yuqorida keltirilgan misollarda ilgari ko‘rib chiqilmagan bir notasiyadan foydalanilmagan. Gap bu yerda ikki nuqta — (..) yoziladigan, tugallanmaganlik yordamida yaratiladigan va ro‘yxat ichiga kiritiladigan ro‘yxatlar haqida bormoqda. Ushbu notasiya arifmetik progressiya singari shunday matematik izchillikni shunchaki aniqlashga imkon beradi.

Ikki nuqta yordamida ham yakuniy, ham cheksiz har qanday arifmetik progressiyani aniqlash mumkin. agar izchillik yakuniy bo‘lsa, unda birinchi va so‘nggi unsurlar beriladi. Agar arifmetik progressiya cheksiz bo‘lsa, progressiyaning so‘nggi unsuri tashlab yuboriladi. Arifmektik progressiyaning martaligi birinchi va ikkinchi unsurlar o‘rtasidagi martalik sifatdagi unsurlar yordamida aniqlanadi. Agar ikkinchi unsur bo‘lmasa, ya'ni birinchi unsurdan so‘ng darhol ikki nuqta ko‘rsatilgan bo‘lsa, bunday arifmetik progressiyaning martaligi sukut saqlash bo‘yicha birga tengdip.

Mana bir necha misol:

```
-- 1. Natural sonlarning cheksiz ro‘yxati [1..]
-- 2. ~1 dan~100 gacha natural sonlarning ro‘yxati [1..100]
-- 3. To‘rt natural sonning cheksiz ro‘yxati [2, 4..]
-- 4. ~1 dan ~100 gacha toq natural sonlarning ro‘yxati [1, 3..100]
-- 5. 5 martalik sonlarning cheksiz ro‘yxati [5, 10..]
```

Agar progressiyaning bunday aniqlovchisida berilgan yakuniy unsur progressiyaning o‘ziga kirmasa, (4 ro‘yxatdagidek) u natijaga kiritilmaydi, shuning uchun [1, 3..99] va [1, 3..100] aniqlovchilari aynan bir xil natijani qaytaradi.

7-misol. Binar shajaralarini taqdim etish uchun turni belgilash

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
Branch :: Tree a -> Tree a -> Tree a
Leaf :: a -> Tree a
```

Ushbu misolda cheksiz turni belgilash usuli ko‘rsatilgan. Vaziyatsiz bu yerda ish bitmagani ko‘rinib turibdi, ishlab chiqaruvchiga zarur bo‘ladigan ma'lumotlar har qanday tuzilmasining turini shunga o‘xshash tarzda aniqlash mumkin.

Ro‘yxat ham ma'lumotlarning cheksiz tuzilmasi ko‘rinishida aniqlanganligini eslatib qo‘yish zarur. Bu cheksiz arifmetik

progressiyalarni va yanada murakkabroq matematik izchilliklarni aniqlash uchun yuqorida tasvirlangan qoidadan foydalanishga imkon beradi.

8.2. Shajaralar

8.2.1. Binar shajaralar tasavvuri

Binar shajara chap shajarachaga, o'akka va ng shajarachaga ega sifatida rekursiv belgilanadi. o'ng va chap shajarachalarning binarlaridir. 8.1. rasmda binar shajaraga misol keltirilgan.

8.1-rasm. Binar shajarasi

Bunday shajaralarni bsh (Chsh, O', O'sh) ko'rinishidagi termlar bilan tasavvur qilish mumkin,

Bu yerda Chsh –chap shajaracha, O'-o'zak, O'sh – o'ng shajarachadi.r

Bo'sh binar shajarani ko'rsatish uchun nil atomidan foydalanamiz. 8.1. rasmdagi binar shajara chap shajarachaga bsh (bsh(nil, d, nil), b, bsh (nil, e, nil)) o'ng shajaracha bsh(nil,s, nil) va yaxlitligicha bsh(bsh(bsh(nil,d, nil), b, bsh(nil,e, nil)), a, bsh(nil, s, nil)) ko'rinmshida yozaladi.

8.2.2. Binar shajarachalar yordamida ko'pliklar tasavvuri

Ko'pliklarni ro'yxatlar ko'rinishida yozish ilgari ro'yxatlar uchun belgilangan maqsadli tasdiqdan foydalanishga imkon beradi, ammo unsurlarning katta soniga ega bo'lgan ko'pliklar uchun ro'yxatli yaxlit tasdiqlar samarasizga aylanadi. Masalan, "tegishli" yaxlit tasdig'ini ko'rib chiqamiz.

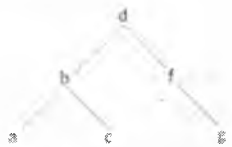
tegishli (R, [R | T]). tegishli (R, [H | T]) :- tegishli (R, T) ko'pdik tegishliligini modellashtirishga imkon beradi.

L — dastlabki 1024 natural sonlar ko'rligini tasivrlovchi ro'yxat bo'lsin, bunda

?- mansub(3000, b)

so‘roviga javob berishda Prologga bunday son yo‘q ekanligi xulosasiga kelishdan oldin barcha 1024 sonni tekshirishga to‘g‘ri keladi.

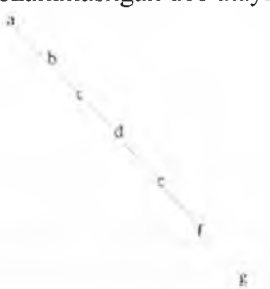
Ko‘plikni binar shajara bilan tasavvur qilish eng yaxshi natijaga erishishga imkon beradi. Shu asnoda binar shajara chap shajaradagi har qanday unsur o‘zak qiymatidan kichik bo‘lishi o‘ng shajaradagi har qanday unsur esa kattaroq bo‘lishi tarzida tartibga solinishi lozim. Biz shajarachani binar shajara sifatida belgilaganimiz bois bunday tartibga solish barcha shajaralarga qo‘llaniladi. 8.2- rasmda tartibga solingan binar shajara misoli keltrilgan.



8.2-rasm. Tartibga soligan binar shajara

Tartibga solish shajara yordamida ko‘plikni tasavvur qilishning yagona ko‘rinishiga olib kelmasligiga e'tibor qarating. Masalan, 8.3 rasmda xuddi 8.2-rasmdagidek ko‘plik tasvirlangan

8.3-rasmdagi bunday ko‘rinishni chiziqli tasavvur deb va 8.2-rasmdagi ko‘rinishni muvozanatlashgan deb ataymiz.



8.3-rasm. Chiziqli tasavvur

Ko‘plikkka tegishlilikni modellashtirish. Binar shajara bilan tasvirlangan ko‘plikka ega bo‘lgan holda biz shajara_ tegishli yaxlit tasdiq yordamida ko‘plikka tegishlilikni modellashtirishimiz mumkin. Shu asnoda “nisbatan kamroq” munosabatini ifodalovchi @< operatordan va “nisbatan kattaroq” munosabatini ifodalovchi @> operatoridan foydalaniladi.

/* chegara shart: X agar o‘zak bo‘lsa .

x/* shajaraga tegishlidir

shajara_ tegishli (X, bsh (chsh, X, O‘sh)),

/* Rekursiv shartlar

Agar X o‘zakning /* qiymatidan katta bo‘lsa va

/* o‘ng shajarachada joylashgan bo‘lsa,

shajara_ tegishli (X, bsh (Chsh, U, O‘sh)) :- X@Y shajara_ga tegishli O‘sh.

Agar X o‘zak qiymati /* dan kichik bo‘lsa, va chap

/* shajarachada joylashgan bo‘lsa /* X shajaraga tegishlidir:

shajara_ ga tegishli (X, bsh (Chsh ,U ,O‘sh)) :-X@Y,

shajara_ (X, Chd).

Agar dastlabki 1024 sonlar ko‘pligini T muvozanatlashgan binar shajarachasi yordamida tasvirlanadigan bo‘lsa

?- tegishli shajara_ga (3000, T)

So‘roviga javob qaytarishda Prolog

Yo‘q

Javobini qaytarishdan oldin 3000 sonini ko‘plikning 11 dan ortiq bo‘lmagan unsurlariga taqqoslaydi.

Albatta, agar T chiziqli tasavvurga ega bo‘lsa, 3000 ni ko‘plikning 1024 unsuriga taqqoslash talab etiladi.

Binar shajarani yaratish. Tartibga solingan binar shajarani boshqa tartibga solingan binar shajaraga X unsurini qo‘shganda yaratish vazifasi quyidagi ko‘rinishda shakllantiriladi.

Chegara shart:

nil ga X ni qo‘shish bsh (nil, X, nil) ni beradi.

Rekursiv shartlar:

X ni bsh (Chsh, O‘, O‘sh) ga qo‘shishda natijadagi shajara tartibga solinganligiga ishonish uchun ikki holatni ko‘rib chiqish zarup.

1. X K dan kichikroq, ushbu xoltada chap shajarachaga erishish uchun X ni Chsh ga qo‘shish zarur. O‘ng shajaracha O‘sh ga teng. Natijadagi shajara o‘zagining qiymati esa K ga teng.

2. X K dan kattaroq bunday holatda o‘ng shajarachaga erishish uchun X ni O‘sh ga qo‘shish xarur. Chap shajaracha Chsh ga teng. O‘zakning qiymati esa K ga teng.

Vazifani bunday shakllantirishga quyidagi dastur mos keladi:

/* chegara shart:

qo‘shilgan_bsh (nil, X, bsh(nil, X, nil)).

/* Rekursiv shartlar:

/*(1)

qo‘shilgan_ bsh (bsh (Chsh, O‘, O‘sh), X, bsh(Chsh, O‘, O‘sh)) :-

X@K,

qo‘shilgan_bsh(Chsh,X,Chsh).

/*(2)

qo‘shilgan_bSh(bsh(ChSh, O‘, O‘Sh), X, bsh(Chsh, O‘, O‘sh)) :-

X@K,

qo‘shilgan_bsh(Chsh, X, O‘sh).

?- qo‘shilgan_bsh(nil, d, T1), qo‘shilgan_bsh(T1, a, T2) so‘roviga .

T1=bsh(nil, d, nil)

T2=bsh(bd(nil, a, nil), d, nil) qiymatlari olinadi

qo‘shilgan_bsh () tartibotini ro‘yxatdagi tartibga solingan shajarani yaratish uchun qo‘llash mumkin:

/* Chegara shart:

ro‘yxat_ shajara ([], nil).

/* Rekursiv shartlar:

ro‘yxat_ shajara ([N | T], Bsh) :-

ro‘yxat_ shajara (T, Bsh2),

qo‘shilgan_bsh(N, Bsh2, Bsh).

qo‘shilgan_bsh muvozanatlashgan shajara yaratishni ta'minlamasligini ta'kidlaymiz, ammo bunday yaratishni kafolatlovchi algoritmlar mavjud.

Nazorat savollari

1. Haskell tilida ro‘yxatlarni yaratish va ularning qismlariga kirish uchun qaysi asosiy amaliyotlar joriy etilgan?

2. Funksional dasturlash doiralarida ishlab chiqilgan qaysi texnologiyadan foydalangan holda cheksiz ro‘yxatlarni hisoblab chiquvchi funksiyalarni ishlab chiqish mumkin?

IX. BOB. IZLASH ALGORITMLARI

9.1. Cheklanishlarni qondirish uchun vazifalar

Izlash vazifalari quyidagi tarzda shakllantiriladi:

Berilgan X da K(x) berilgan shartlarni qondiruvchi x unsur(lari) ini toping

Bunday turdagi vazifani hal qilishning ko‘rinib turgan usuli X ko‘plik unsurlarini navbatma-navbat tanlash va har bir unsur uchun K(x) shartlarini bajarishdir. Agar X ko‘plik (ortiqchalik makoni) ko‘p unsurlarga va hatto cheksiz unsurlarga ega bo‘lsa, bu usul unchalik yaxshi emasligiga ham, yaqqol ko‘rinib turibdi.

Ba'zan ortiqchalik makoni vazifani hal qilish ko‘pligini soddalashtirish maqsadida vazifa ancha qo‘pol, katta “zahira” bilan yozilganda vazifa muvaffaqiyatsiz shakllantirilganligidan kelib chiqadigan o‘ta katta bo‘lib chiqadi. Bunday holatlarda vazifa shaklini o‘zgartirilgan holda odatda ortiqchalik makonini ancha qisqartirish va shu zahoti xatto uni “bevosita” usul bilan hal qilish mumkin bo‘ladi. Ba'ribir, ortiqchalikdan qochishning imkoni yo‘q. Vazifalar muntazam uchrab turadi. Hamda ortiqchalikni qisqartirish muammosi birinchi o‘ringa chiqadi. Bunday holatlarda ortiqchalik yoki kombinatorlik vazifalari haqida so‘z yuritiladi. Aniq vazifalarning o‘ta katta miqdori ushbu toifaga mansubdir va faqat kamginasi uchungina ortiqchalikdan qochish usullari topilgan.

Ko‘pincha X ortiqchalik makoni ko‘pliklarning dekartcha asari X_1, ... ,X_n dir. Mazkur holatda vazifa odatda cheklanishlarni qondirish muammosi deb ataladi va quyidagicha tarzda shakllantiriladi.

Qiymatlari sohasi ko‘plik bo‘lgan, bir vaqtning o‘zida barcha berilgan cheklanishlarni qondiradigan $K_i(x_1 \ldots, x_n)$ $x_1 \ldots, x_n$ X_1, ... ,X_n o‘zgaruvchan qiymatlarini toping.

9.2. Kriptoarifmetika

Bunday turdagi ajoyib misol – qiziqarli matematikaning kriptoarifmetik vazifalaridir. Mana eng mashhur misol:

SEND

```
 +MORE
 -----
 MONEY
```

Eng sodda echim "yaratish va tekshirish" usuliga asoslangan. Biz o'zgaruvchan qiymatlar uchun barcha imkondagi qiymatlarni tanlab olamiz va berilgan shartlar bajarilishini tekshirib ko'ramiz.

```
solve([S,E,N,D]+[M,O,R,E]=[M,O,N,E,Y]) :-
   digit(S), digit(E), digit(N), digit(D),
   digit(M), digit(O), digit(R), digit(Y),
   S\==0, M\==0,
   all_different([S,E,N,D,M,O,R,Y]),
         1000*S + 100*E + 10*N + D
   +     1000*M + 100*O + 10*R + E
   =:=
   10000*M + 1000*O + 100*N + 10*E + Y.
digit(D) :- between(0,9,D).
```

all_different predikati notengliklar ($E\==S$, $N\==S$, $N\==E$, i t.d.) ning uzun qatorini yozish zaruratidan qochish maqsadida kiritilgan.

```
all_different([]).
all_different([X|Xs]):-
   \+ member(X,Xs), all_different(Xs).
```

Biz ushbu dasturni ishga tushurishimiz va oxir-oqibatda quyidagi javobni olishimiz mumkin.

```
?- solve(X).
X = [9, 5, 6, 7]+[1, 0, 8, 5]=[1, 0, 6, 5, 2]
Yes
```

Ammo javobni kutish uchun anchagina sabr kerak bo'ladi. Mazkur holatdagi ortiqchalik makoni o'ta muvaffaqiyatsiz tanlangan – u 10^8 unsurlarga ega. Biroq, generasiya jarayonidagi cheklanishlardan ayrimlari hisobga olinsa, uni osonlik bilan qisqartirish mumkin. Eng avvalo, - o'zgaruvchanlarning barcha qiymatlari turlicha ekanligi shartidir. Bu ortiqchalik makonini 10!/2! gacha, ya'ni taxminan 50 marta qisqartiradi, binobarin dasturning ishlash vaqti ham kamayadi.

```
solve([S,E,N,D]+[M,O,R,E]=[M,O,N,E,Y]) :-
range(0,9,L0),
   selects([S,E,N,D,M,O,R,Y],L0),
```

```
S\==0,  M\==0,
        1000*S + 100*E + 10*N + D
 +      1000*M + 100*O + 10*R + E
 =:=
10000*M + 1000*O + 100*N + 10*E + Y.
```

selects predikati oldin uchragan, a range (A,B,L) esa L sonlar ro'yxatini A dan B gacha qaytaradi.

```
selects([],_).
selects([X|Xs],Ys) :- select(X,Ys,T), selects(Xs,T).
range(N,N,[N]).
range(M,N,[M|Ns]):- M < N, M1 is M+1, range(M1,N,Ns).
```

Keraksiz ko'rinishlarni imkon qadar ertaroq kesib qo'yish barchasidan yaxshi ekanligi tushunarlidir, masalan, generasiya jarayonidagi S\==0 va M\==0 cheklanishlarni hisobga olgan holda ayrim tezlanishga erishish mumkin.

```
range(0,9,L0),
select(S,L0, L1), S\==0,
```

ва ноказо.

Cheklanishlar tuzilmasini qayta ko'rib chiqish yanada jiddiyroq takomillashuvdir. Buning uchun ustuncha qo'shish qanday bajarilishini yodga olamiz va adc predikatini - ko'chirib o'tkazishning ikki bir qiymatli soni qo'shilishi tarzida belgilaymiz.

```
adc(A, B, Res, Cin, Cout) :-
   Sum  is A + B + Cin,
   Res  is Sum mod 10,
   Cout is Sum // 10.
```

Endi, generasiya jarayoniga cheklanishlarni kiritish va bu bilan ortiqchalik makonini 1000 martadan ko'proqqa qisqartirish mumkin.

```
solve([S,E,N,D]+[M,O,R,E]=[M,O,N,E,Y]) :-
   range(0,9,L0),      select(D,L0,L1),
                       select(E,L1,L2),
   adc(D, E, Y, 0, C1), select(Y,L2,L3),
                       select(N,L3,L4),
                       select(R,L4,L5),
   adc(N, R, E, C1, C2), select(O,L5,L6),
   adc(E, O, N, C2, C3), select(S,L6,L7), S\==0,
```

```
select(M,L7,L8), M\==0,
```

adc(S, M, O, C3, M).

9.3. Farzinlar to‘g‘risidagi vazifa

NxN andozasiga ega shatranj taxtasiga N farzinlarni ular bir birbirini urmaydigan qilib joylashtirish talab etiladi.

Agar barcha farzinlarni x va y koordinatlari to‘plami ko‘rinishida joylashuvi (pozisiya) tasavvur qilinsa, cheklanishlarni qondirish sifatidagi vazifa shakllantirilishi mumkin.

1. $[1..N]$ ko‘pligidan barcha turli i va j quyidagi shartlar bajarilishi $x_1, y_1, x_2, y_2, \ldots, x_N, y_N$ uchun qiymatlar to‘plamini topish:
2. x_i =/= x_j (farzinlar bir vertikalga joylashmagan);
3. y_i =/= y_j (farzinlar bir gorizontalga joylashmagan);
4. $|x_i - x_j|$ =/= $|y_i - y_j|$ (farzinlar bir diagonalga joylashmagan).

Ushbu vazifani ortiqchalik usuli bilan hal qilishga urunib ko‘rish mumkin.

```
queens(N,Ys-Xs) :-
   range(1,N, Ns),
   length(Xs, N), members(Xs,Ns),
   length(Ys, N), members(Ys,Ns),
safe(Ys-Xs).
```

Ammo N^{2N} ortiqchalik makonining andozasi bizni ogoxlantirishi lozim. Mumtoz holat N=8 uchun ko‘rinishlarning deyarli 10^{14}ni ko‘rib chiqishga to‘g‘ri keladi. X farzinlar koordinatalari orasida $x_1, x_2, \ldots, x_N$ sonlarning 1,2,...,N har biri faqat bir marta uchrashini ta'kidlaymiz. Shu tarzda farzinlarning raqamini x koordinatalari sifatida ko‘rib chiqish mumkin va y koordinatalarini tanlab olish etarlidir, vazifaning yangi ko‘rinishi:

Barcha turli i va j lar uchun $(y_1, y_2, \ldots, y_N), 1 < y_i < N$ to‘plamni topish.

1. y_i =/= y_j
2. $|y_i - y_j|$ =/= $|i - j|$.

Ortiqchalikni yangi sxemasi quyidagicha ko‘rinishga ega bo‘ladi:

```
queens(N,Ys) :-
```

```
range(1,N, Ns),
length(Ys, N),
members(Ys, Ns),
safe(Ys).
```

Endi, ortiqchalik makoni ko‘rinishlar N^N ($N = 8 - 10^7$ uchun) ga ega. Bu yaxshiroq, biroq hali ham ko‘proqdir. Agar y farzinlari koordinatalari orasida $y_1, y_2, \ldots, y_N$ sonlar 1,2,...,N har biri faqat bir marta uchrashini hisobga oladigan bo‘lsak, vazifani yana bir qayta shakllantirish mumkin.

1. $i =/= j$ bo‘lganda $|y_i - y_j| =/= |i - j|$. Bo‘ladigan turli $(y_1, y_2, \ldots, y_N), 1 < y_i < N$ to‘plamini toping.
2. Albatta, bu mohiyatiga ko‘ra aynan shuning o‘zi, shunchaki biz $y_i =/= y_j$ shartini cheklanishlar ko‘pligidan ortiqchalik makoni ta'rifiga ko‘chiramiz. Ortiqchalik tartiboti arzimagan tarzda o‘zgaradi.

```
queens(N,Ys) :-
  range(1,N, Ns),
  length(Ys, N),
selects(Ys, Ns),
  safe(Ys).
```

Ortiqchalik makoni N^N dan N! gacha qisqaradi, bu N=8 uchun 40 320 anglatadi va bu endilikda maqbuldir, endi tekshirish tartibotini yozib olish mumkin.

```
safe([Y|Ys]) :- noattack(Y,Ys), safe(Ys).
safe([]).
noattack(Y,Ys) :-  noattack(Y,Ys,1).
noattack(_,[],_).
noattack(Q,[Y|Ys],D) :-
   abs(Q-Y) =\= D,
   D1 is D+1,
noattack(Q,Ys,D1).
```

Ammo faktorial faktorialdir va agar N=8 uchun barcha javoblarni izlash soniyalar bilan o‘lchansa, N=10 uchun, bu endi daqiqalar, N=12 uchun esa soatlardir. Vazifa tabiati bilan biz hech narsa qila olmaymiz – cho‘chqadan chopqir ot yaratib bo‘lmaydiku, biroq tez chopadigan

cho‘chqani yaratish mumkin. Biz huddi oldingi misoldagidek, cheklanishlarni tekshirishni generasiya tartibotiga o‘tkazishimiz mumkin. Navbatdagi farzinning o‘rnini tanlagan holda uning oldingi joylashtirilgan farzinlar bilan muvofiqligini tekshirib ko‘ramiz. Bu bilan biz selects va safe tartibotini yagona tartibot place_queens ga birlashtiramiz. Bu tartibot joylashtirib bo‘lingan farzinlarningg ro‘yxatiga ega Board qo‘shimcha daliliga egadir.

```
queens(N,Ys) :-
  range(1,N,Ns),
  length(Ys,N),
  place_queens(Ys,Ns,[]).
place_queens([Y|Ys], Ns, Board) :-
   select(Y, Ns, Ns1),
   noattack(Y, Board),
   place_queens(Ys, Ns1, [Y|Board]).
place_queens([],_,_).
```

Oldingiga taqqoslaganda ushbu dastur N=8V bo‘lganda 20 marta tezroq bajariladi, bundan tashqari hisoblab chiqishlar vaqti $N!^{0.7}$. ga proporsionaldir.

place_queens tartibotining o‘zi generator va test juftligiga egaligini ta'kidlaymiz. Biz dastlab select vositasida farzinning o‘rnini tanlaymiz, va so‘ngra, noattack ni chorlagan holda, uning mosligini tekshirib ko‘ramiz. Ushbu ikki tartibotni ham birlashtirish jozibalidir, ammo bunday birlashtirish ma'noga ega bo‘lishi uchun faqat yo‘l qo‘yiladigan o‘rinlarni tanlashga o‘rganib olish zarur. Buning uchun joylashtirilgan farzinlarning shunchaki ro‘yxatidan ko‘ra kattaroq narsa talab qilinadi. O‘zining mashhur maqolasida [1] Niklaus Virt /-diagonal i \- diagonaldan iborat bo‘lgan ikki bulev massividan foydalangan. Biz ushbu g‘oyani salgina o‘zgartiramiz va dioganallarni “asrovchi” o‘zgaruvchan qiymatlarga ega Us va Ds ro‘yxatidan foydalanamiz, har bir o‘zgaruvchi qiymat, bir dioganalni namoyon etadi. Navbatdagi farzinni joylashtirishda Ys ro‘yxatidagi o‘zgaruvchan qiymat farzinning raqami bilan bog‘lanadi, biroq biz bir vaqtning o‘zida Us va Ds ro‘yxatlarida tegishli o‘zgaruvchilarni bog‘laymiz. O‘zgaruvchan qiymat bu bog‘lanishi bilan faqat bir qiymatga ega bo‘lishi bois hech qanday boshqa farzin endilikda aynan shu dioganalda bo‘la olmaydi (ushbu usuldan biz xaritalarni

bo‘yashda foydalangan edik). place_queens tartiboti farzin uchun joy tanlanganidan so‘ng Us va Ds ni bir o‘ringa turli tomonlarga “surib” qo‘yadi.

```
queens(N,Ys) :-
   range(1,N, Ns),
   length(Ys, N),
   place_queens(Ns,Ys,_,_).
place_queens([N|Ns], Ys, Us, [D|Ds]):-
   place1(N, Ys, Us, [D|Ds]),
   place_queens(Ns, Ys, [U|Us], Ds).
place_queens([], _, _, _).
place1(N,[N|_], [N|_],[N|_]).
place1(N,[_|Cs],[_|Us], [_|Ds]):- place1(N,Cs,Us,Ds).
```

Bunday takomillashtirish 2-3 marta tezlikka erishiga imkon beradi, biroq umuman u qadar ahamiyatli bo‘lib chiqmaydi. Ortiqchalik makoni mohiyatiga ko‘ra aynan shunday qolaveradi. Faqat tanlash imkonini tekshirish tezlashadi, xolos. Shunday bo‘lsada, ushbu dastur cheklanishlarning tarqalishi (constraints propagation) deb ataluvchi vazifalarni hal qilishning muhim usulini ko‘rsatmoqda. Navbatdagi o‘zgaruvchan qiymat uchun qiymatni tanlash boshqa o‘zgaruvchan qiymatlar uchun tanlash imkoniyatlarini cheklaydi.

Cheklanishlarning tarqalishi – turli joriy etishlarga yo‘l qo‘yuvchi ana umumiy tamoyildir. Ushbu usulni joriy etishning muhim tarzi – har bir o‘zgaruvchan qiymat uchun imkondagi qiymatlarni aniq hisobini olib borish, bizning holatimizda esa, har bir farzin uchun imkondagi o‘rinlarni hisobini yuritishdir. Bu usul “oldinga qarash” (forward checking) deb ataladi. Juftliklar ro‘yxatini (farzinning raqami: yo‘l qo‘yiladigan o‘rinlar ro‘yxatini)ni kiritamiz. Dastlab har bir farzin to‘la erkinlikka ega bo‘ladi. Uning ro‘yxati 1,2,...*N* ko‘rinishiga ega. Navbatdagi farzinni joylashtirgan holda biz qolgan farzinlarning yo‘l qo‘yiladigan o‘rinlaridan u hujum qilgan o‘rinlarni chiqarib tashlaymiz. Bu bilan tanlash imkoniyatni cheklaymiz.

```
queens(N, Queens) :-
   range(1,N, Ns),
   init_vars(Ns, Ns, V),
   place_queens(V,Queens).
```

```
init_vars([X|Xs],Ys,[X:Ys|V]) :-
   init_vars(Xs,Ys,V).
init_vars([], _, []).
place_queens([X:Ys|V],[X-Y|Qs]) :-
   member(Y, Ys),
   prune(V, X, Y, V1),
   place_queens(V1,Qs).
place_queens([],[]).
prune([Q:Ys|Qs], X,Y, [Q:Ys1|Ps]) :-
   sublist(noattacks(X,Y,Q), Ys, Ys1),
   Ys1 \== [],
   prune(Qs, X,Y, Ps).
prune([], _,_, []).
noattacks(X1,Y1,X2,Y2) :-
   Y1 \== Y2,
   abs(Y2 - Y1) =\= abs(X2 - X1).
```

sublist o‘rnatilgan predikati–filter funksiyasiga o‘xshashdir. U ko‘rsatilgan predikat bajariladigan ro‘yxat unsurlaridan ro‘yxatni qaytaradi. Mohiyatiga ko‘ra bu

findall(T, (member(T,Ys), noattacks (X,Y,Q,T)) , Ys1) bajarishning eng jadal usulidir.

Ys1 \== [] ni tekshirishga e'tibor qaratamiz. Agar hali joylashtirilmagan farzinlardan qaysi biri uchundir bo‘sh joy qolmagan bo‘lsa, prune muvaffaqiyatsiz tugaydi. Shu tarzda farzinning taxmin qilinayotgan o‘rni darhol rad etiladi.

Endi biz ko‘rsatib o‘tilgan tekshiruvni prune ga qo‘shmasligimizni faraz qilamiz. Unda farzinlarni bir bir-birining ortiga shunchaki axmoqona joylashtirish o‘rniga dastlab joylashtirilgan farzilanlar orasida joylashtirib bo‘lmaydigani yo‘qligiga ishonch xosil qilish oqilonadir. So‘ngra joylashtirishning faqat bir ko‘rinishiga ega bo‘lgan farzinlar bor yoki yo‘qligiga qarash zarur. Agar bor bo‘lsa, aynan shundayini tanlash lozim. Xuddi shuningdek, umumiy holatda bo‘sh o‘rinlarning eng kami qolgan farzinni tanlash yaxshiroqdir. Shunday qilib biz farzinlarni joylashtirish tartibini o‘zgartirgan holda dasturni takomillashtirishimiz mumkin. Endi, farzinlarni tartib bo‘yicha joylashtirish shart bo‘lmaganligi bois natijani X-Y juftligi ko‘rinishda tasavvur qilamiz.

```
place_queens(V, [X-Y|Qs]) :-
    queen_to_place(V,X:Ys,V1),
    member(Y, Ys),
    prune(V1, X, Y, V2),
    place_queens(V2,Qs).
place_queens([], []).
```

queen_to_place tartiboti joylashtirilmagan ro'yxatdan joylashtirish zarur bo'lgan farzinni tanlaydi. Agar biz shunchaki,

```
queen_to_place([V|Vs],V,Vs).
```

ni yozadigan bo'lsak, oldingi ko'rinishga ega bo'lamiz, biz esa, "eng cheklangan" farzinni, ya'ni extimoliy o'rinlari ro'yxati eng qisqa bo'lgan farzini tanlamiz.

```
queen_to_place([V|Vs], Q, Vx) :- select_boundest(Vs, V, Q, Vx).
select_boundest([], Q, Q, []).
select_boundest([X|Xs], Q, Qm, [Q|R]) :-
    shorter(X, Q), !,
    select_boundest(Xs, X, Qm, R).
select_boundest([X|Xs], Q, Qm, [X|R]) :-
    select_boundest(Xs, Q, Qm, R).
shorter(_:L1, _:L2) :- length(L1,N1), length(L2,N2), N1<N2.
```

Ortiqchalik makonini qisqartirishda ushbu qisqartirish andozasini uni joriy etishga sarflar bilan taqqoslash zarur. Uncha katta bo'lmagan taxtalarda ushbu dastur xatto sekinroq bo'lib chiqadi - bir muncha murakkab sxemani joriy etishga sarflar qisqartirishdan bo'ladigan barcha yutuqni eb yuboradi. Shunday bo'lsada, N o'sishi bilan qisqartish kattaroq rol o'ynay boshlaydi. Dasturni bajarish vaqti N! Bo'lgan kvadrat ildiz sifatida o'sadi, bu esa biz katta hajmli taxta uchun ham vazifani hal qila olishimizni anglatadi.

9.4. Grafalardan izlash

Makonda izlashning boshqa muhim holati – qandaydir grafdagi yo'llar ko'pligidir. Ko'plab vazifalarni quyidagi tarzda shakllantirish mumkin:

- Qandaydir boshlang'ich holat berilgan.
- Ko'plab maqsadli holatlar belgilangan. Bu ko'plik bir unsurdan iborat bo'lishi va balki, maqsadga erishilganligini belgilovchi qandaydir qoida bilan berilishi mumkin.

- Joriy holatni o‘zgartirish uchun foydalanish mumkin bo‘lgan amaliyotlarning ko‘pligi belgilangan.
- Maqsadli holatga olib boruvchi amaliyotlar izchilligini topish talab etiladi.

Biz vazifalar holatini yo‘naltirilgan graf ko‘rinishida tasavvur qilishimiz mumkin. Undagi cho‘qqilar holatlarga, amaliyotlar esa – yoylarga mos keladi. Shu tarzda vazifani yo‘naltirilgan grafdagi yo‘lni izlash vazifasi sifatida qayta shakllantirish mumkin.

Graf cho‘qqilar ko‘pligidagi shunchaki binar munosabatni tasvirlaydigan eng sodda holatdan boshlaymiz, ya'ni biz yoylarga xos bo‘lgan turli atributlardan abstraksiyalashamiz va grafda A dan V gacha yoy borligini anglatadigan arc(A,B) binar munosabatini ko‘rib chiqamiz. Sodda misollarda barcha yoylarni aniq sanab chiqish mumkin. Bir muncha murakkablarida bu munosabat dastur bilan belgilanadi. Yo‘naltirilmagan grafni har bir qobirg‘aga (rebru) ikki yoy mos kelishini qo‘ygan holda yo‘naltirilgan graf sifatida tasavur qilish mumkin.

```
arc(A,B) :- edge(A,B); edge(B,A).
```

Boshqa amaliy muhim tasavvur cho‘qqining cheklanganligi g‘oyasiga asoslanadi. Graf cho‘qqini ro‘yxatning uning qo‘shnilari bilan bog‘laydigan neighborhood(A,NB) munosabati bilan belgilanadi, biz xaritaga rang berayotganda grafni shu tarzda tasavvur qilgan edik. Bir tasavvurdan boshqa tasavvurga o‘tishga imkon beruvchi tartibotni belgilash qiyin emas.

```
arc(A,B):- neighborhood(A,NB),member(B,NB).
neighborhood(A,NB):- setof(B,arc(A,B),NB).
```

9.5. Refleksiv-tranzitiv birlashish

arc iboralarida biz ikki cho‘qqi qandaydir yo‘l bilan bog‘langanligini anglatuvchi boshqa binar munosabat connected ni belgilashimiz mumkin. matematik nuqtai nazardan biz arc munosabatini refleksiv-tranzitiv birlashuvini, ya'ni arc ni o‘z ichiga oluvchi va refleksivlik hamda tranzitivlik xossalariga ega bo‘lgan eng kichik munosabatni barpo etamiz. Bu shartlarni bevosita Prologga yozish mumkin.

```
connected(A,A) .
connected(A,B) :- connected(A,C), connected(C,B).
```

```
connected(A,B) :- arc(A,B).
```

Ko‘rinib turganidek, ushbu dasturni ishga solishga urinish befoydadir1. Biroq uncha murakkab bo‘lmagan mulohazalar bilan dastur sifatida butkul yaroqli bo‘lgan mantiqiy ekvivalent ta'rifga ega bo‘lish qiyin emas.

```
connected(A,A).
connected(A,B) :- arc(A,C), connected(C,B).
```

Bu dastur Prologning hisoblab chiqish strategiyasiga asoslangan holda izlashni chuqurlik tomon (Depth-First Search, DFS) amalga oshiradi. U yakuniy shajaralarda yaxshi ishlaydi hamda izlash jarayonida yana yakuniy shajaralarga borib taqaladigan, yakuniy kontursiz graflarda maqbuldir. Biroq izlash shajarasi cheksiz bo‘lsa, dastur cheksiz shoxga tushib qolishi va u erda mangu qolib ketishi mumkin.

Agar graf yakuniy, biroq davriyliklarga ega bo‘lsa, izlash shajarasida cheksiz shohlar vujudga keladi.

```
arc(a,b). arc(b,a).
arc(b,c). arc(c,d).
?- connected(a,d).
```

Graf cheksiz yoki shunchaki o‘ta katta bo‘lgan va amalda aynan bir narsa bo‘lgan holatning ham imkoni bor. Ayni vaqtda echim anchagina yaqin bo‘lishi mumkin.

```
arc(X,Y) :- succ(X,Y).
arc(X,Y) :- succ(Y,X).
?- connected(1,0).
```

Shunday qilib, bizga izlash shajarasidagi cheksiz shoxlarni bartaraf etishning qandaydir usuli talab etiladi.

Yakuniy grafdar uchun biz ko‘rib chiqilgan cho‘qqilar hisobini yuritishimiz mumkin. A va V S ko‘plik cho‘qqilarining birontasi ham orqali o‘tmaydigan yo‘l bilan bog‘langanligini anglatuvchi connected(A,B,S) yangi munosabatini belgilaymiz, ko‘plikni tasavvur qilish uchun ko‘rib chiqilgan cho‘qqilarni biz qo‘shadigan ro‘yxatidan foydalanish hamma narsadan osonroqdir.

```
connected(A, A, _).
connected(A, B, S) :-
  arc(A,C),
  \+ member(C, S),
```

```
connected(C,B,[C|S]).
connected(A,B) :- connected(A,B,[]).
```

Ushbu dastur uchun agar graf davriyliklarga ega bo'lganda ham izlash shajarasi yakuni ekanligini ko'rish oson. S ro'yxatidagi cho'qqilar soni grafdagi cho'qqilarning umumiy sonidan yuqori bo'lishi mumkin emas.

9.6. Chuqurlikka cheklanishlar bilan izlash

Cheksizlikka qarshi kurashning boshqa usuli – izlash chuqurligini cheklashdir. connected_b(A, B, N) A va B N dan uzun bo'lmagan yo'l bilan bog'langanligini anglata qolsin.

```
connected_b(A, A, _).
connected_b(A, B, N) :-
    N>0,
    arc(A,C),
    N1 is N-1, connected_b(C,B,N1).
```

Endi agar bir cho'qqidan boshqasigacha (graf diametri) eng qisqa yo'l uzunligining yuqori chegarasi ma'lum bo'lsa, biz uni N uchun qiymat sifatida ko'rsatishimiz mumkin. Masalan,

```
connected(A,B) :- connected_b(A,B,1000).
```

Ushbu dasturdan cheksiz yoki juda katta graflarni izlash uchun foydalanish mumkin. Agar echim mavjud bo'lsa va uncha uzoqda joylashmagan bo'lsa, u topiladi. U davriyliklarga ega graf uchun ham mos keladi – berilgan chuqurlikka etishgach cheksiz shohlar kesib tashlanadi.

9.7. Izchil chuqurlashtirishlar usuli

Izlash chuqurligi uchun bizda hech qanday aprior baholash bo'lmasa nima qilishimiz kerak? Mazkur holatda izchil yoki iterasion chuqurlashtirish deb aylanadigan uslubiyatdan foydalanish mumkin. N ning kichik qiymatidan, loaqal 1 dan boshlagan holda biz barcha katta qiymatlarni ulardan ayrimlari echim bermaguncha izchil sinab ko'ramiz. Natijada "iterasion chuqurlashtirish bilan chuqurlikni izlash" (Depth-First Iterative Deepening, DFID) qo'lga kiritiladi. Cheklanishga ega tartibotni iterasion chorlaydigan va echim topilgach to'xtaydigan tashqi tartibotni yozib olish mumkin. Masalan,

```
connected_i(A,B,N) :- connected_b(A,B,N).
connected_i(A,B,N) :- N1 is N+1, connected_i(A,B,N1).
```

```
connected(A,B) :- connected_i(A,B,1).
```

Ammo, agar biz bog'liq bo'lmagan cho'qqilar uchun connected_ ni chorlasak nima bo'ladi? ko'rinib turganidek, connected_b hamma vaqt muvaffaqiyatsizlik bilan tugaydi, connected_i esa qaysarlik bilan izlash chuqurligini ortirib boradi va yana connected_b chorlayveradi. Gap shundaki, dastur nima uchun: barcha imkoniyatlar tugagani uchunmi yoki izlash chuqurligidagi chegaraga etilganligi uchunmi izlash muvaffaqiyatsiz yakunlanganligini farqlamaydi. Tegishli tekshirishni qo'shish mumkin.

```
connected_i(A,B,N) :- connected_b(A,B,N).
connected_i(A,B,N) :- reach_bound(A,B,N), N1 is N+1,
connected_i(A,B,N1).
reach_bound(_, _, 0).
reach_bound(A, B, N) :-
  N>0,
  arc(A,C),
  N1 is N-1, reach_bound(C,B,N1).
```

biroq bu ish hajmini ikki barobar oshiradi.

Boshqa imkoniyat – nuqsonli samaradan foydalanishdir. Muvaffaqiyatsizlik, ehtimol izlash chuqurligi cheklanganligidan yuzaga kelishi mumkinligni anglatuvchi dinamiy o'zgaruvchan holatni (shunchaki bayroqni) bound, belgilaymiz va izlashni chuqurlashtirishdan oldin uni tekshirib ko'ramiz.

```
:- dynamic bound/0.
connected_i(A,B,N) :-
  connected_b(A,B,N).
connected_i(A,B,N) :-
bound, retract(bound),
  N1 is N+1, connected_i(A,B,N1).
connected_b(A, A, _).
connected_b(_, _, 0) :-
    assert(bound), fail.
connected_b(A, B, N) :-
  N>0,
  arc(A,C),
  N1 is N-1, connected_b(C,B,N1).
```

Bu endilikda mantiqiy dasturlashni kamroq eslatsada, biroq afsuski, ushbu axborotni uzatishning tabiiy usuli mavjud emas, yana bir muammo - biz izlash chuqurligidagi cheklashni olib tashlaganligimiz bois grafdagi davriyliklar yana noxushliklar keltirishi mumkin. Albatta, ko'rib chiqilgan cho'qqilarni eslab qolishni qo'llash yoki chuqurlikka yana bir cheklanish o'rnatish mumkin.

9.8. Kenglikni izlash

Biz chuqurlikda izlashdan butkul voz kechishimiz va izlashning boshqa strategiyasini amalga oshirishimiz mumkin. Kenglikda izlash (Breadth-First Search, BFS) ko'p va'da beruvchan bo'lib ko'rinmoqda. Ikki yoqlama shajarani ko'zdan kechirishda qo'llanilgan o'sha sxemani qo'llash barchasidan osonroqdir. Bizga barcha qo'shni cho'qqilarni darhol olishimiz zarur bo'lganligi bois graflarni tegishli tarzda qayta qo'yishdan qulayroqdir.

S ko'pligi cho'qqilaridan biri V bilan bog'langanligini anglatuvchi reach(S, B) munosabatini belgilaymiz. A S ning ichida bo'la qolsin, shu asnoda A\=B, ko'rib turganimizdek, agar biz A ning o'rniga uning qo'shnilari ko'pligini kiritsak, nisbat buzilmaydi.

```
connected(A,B) :- reach([A],B).
reach([A|_], A).
reach([A|As], B) :-
   neighbours(A,Ns),
   append(As,Ns,Zs),
   reach(Zs,B).
```

Har bir cho'qqining qo'shnilari cho'qqilar ro'yxatining oxiriga qo'shiladi. Shuning uchun muayyan cho'qqining barcha qo'shnilari ushbu qo'shnilarning qo'shnilaridan oldin ko'rib chiqiladi. Bu kenglikda izlashni anglatadi. Agar append(As,Ns,Zs) o'rniga append(Ns,As,Zs) kiritilsa, cho'qqining barcha qo'shnilari ro'yxat boshiga tushib qolishini va biz chuqurlikdagi oddiy izlashga, biroq endilikda orqaga qaytish mexanizmidan foydalanmaydigan izlashga ega bo'lishimizni ta'kidlash zarur.

Ko'rinib turgan takomillashtirish ro'yxatda mavjud bo'lgan cho'qqilarga cho'qqilar qo'shmaslikdir. Bu kuchli bog'langan graflarda dastur xatti-harakatini yaxshilaydi. Biroq davriylar muammosi qolaveradi.

Agar echim mavjud bo‘lsa, u topiladi. Aks holda cheksiz shajarani to‘la ko‘rib chiqishga to‘g‘ri keladi. Albatta, barcha ko‘rib chiqilgan cho‘qqilarni eslab qolish ushbu muammoni hal qilishga yordam beradi.

9.9. Chuqurlikka izlashni kenglikka izlash bilan taqqoslash

Kenglikka izlash huddi izchil chuqurlashtirish bilan izlash singari cho‘qqilar o‘rtasidagi eng qisqa yo‘lni topadi. Darhaqiqat, dastlab bir qadam bilan erishiladigan, so‘ngra ikki va hakozo qadamlar bilan barcha cho‘qqilar ko‘rib chiqiladi. Buning kamchiligi xotiraga katta talablar qo‘yishdir.

Izlash makoni har bir cho‘qqisi b ajdodlarga ega bo‘lgan shajaradan iborat deylik. Unda k chuqurlikda cho‘qqilarning bk joylashadi. Shajaradagi n darajaga ega cho‘qqilarning umumiy soni esa

$$1 + b + b^2 + \ldots b^n = (b^{n+1} - 1)/(b-1)$$

Kenglikda izlashda shajaraning har bir darajasi navbatdagisi ko‘rib chiqilguniga qadar eslab qolinishi lozim. Shuning uchun bn cho‘qqilarni bir vaqtda izlashga to‘g‘ri keladi. Chuqurlikda izlash faqat bn cho‘qqilar uchun joy talab qiladi. Ammo biz iterasion chuqurlashuvda oldingi iterasiyalarda amalga oshirilgan barcha hisoblab chiqishlarni har safar takrorlaymiz. Bu g‘ashga tegsada, aslida ishlar unchalik yomon emas.

Oddiy ortiqchalik har bir cho‘qqiga bir marta ishlov beradi. Iterasion ortiqchalik esa k, n-k chuqurlikda joylashgan cho‘qqini bir marta ko‘rib chiqadi. Jami

$$n + (n-1)b + (n-2)b^2 + \ldots + b^n < b^{n+2}/(b-1)^2$$

cho‘qqilarni ko‘rib chiqishga to‘g‘ri keladi, ya'ni ancha katta bo‘lgan n da izchil chuqurlashuvlar usuli $b/(b-1)$ da bir bor ko‘proq vaqtni talab qiladi. Bu endi unchalik katta emas, shunday qilib, BFS $O(b^n)$ vaqt va xotirani DFID - $O(b^n)$ vaqtni va O(n) xotirani talab qiladi. Xozirda juda qimmatli zahira bo‘lganligi bois DFID barcha ma'noda optimaldir. Ushbu mulohazalardagi zaif joy graf shajara ekanligi borasidagi taxmindir. Boshqa graflardagi natijalar bu qadar bir qiymatli bo‘lmasligi mumkin. Masalan, “rombik graf" uchun natijalar aniq .BFS foydasiga bo‘lsa, davriyliklarga ega graflar uchun ko‘rib chiqilgan cho‘qqilarni hisobga olish borasida qo‘shimcha xotira sarflashga to‘g‘ri keladi.

Albatta, ko‘rib chiqilgan usullar faqat graflarda izlash uchun emas, balki Prolog hisoblab chiqishlar strategiyasi qoniqarsiz bo‘lgan holatlardagi barcha dasturlar uchun qo‘llash mumkin. Masalan, biz dasturni har bir tartibot ikki qo‘shimcha dalilga: qadamlar sonini cheklash va ushbu cheklashlar orasidagi bir matabalik hamda amalda foydalanilgan qadamlar soniga ega bo‘ladigan qilib qayta shakllantirishimiz mumkin.

```
p(X, Z):- q(X, Y), r(Y, Z).
    p(X, Z, B, D):-
B>0,
  B1 is B-1, q(X, Y, B1, D1),
  r(Y, Z, D1, D) га ўзгаради.
```

Dastlab biz kichraytirilgan cheklanish ($B1=B-1$) ga ega bo‘lgan birinchi kichik maqsad q ni chorlaymiz. Muvaffaqiyatli holatda q qadamlarning qolgan sonini (D1) qaytaradi. Bu son navbatdagi kichik maqsad r ga yangi cheklanish sifatida uzatiladi. Dasturning har bir taklifini shu tarzda o‘zgartirgan holda biz izlash chuqurligiga cheklanishga ega bo‘lgan dasturning yangi ko‘rinishini qo‘lga kiritamiz.

Bunday o‘zgartirishlarni avtomatik tarzda amalga oshirgan yaxshiroq va ayrim Prolog-tizimlar buning uchun tegishli vositalarga ega. Masalan, Ciao ni

```
:- iterative(connected/2,10,next).
```

деб эълон қилиш мумкин.

```
connected(A,A).
connected(A,B) :- arc(A,C), connected(C,B).
next(X,Y) :- Y is X + 5.
```

Bu predikat connected/2 10 sahifadan boshlab, izchil chuqurlashish bilan bajarilishini anglatadi, navbatdagi cheklanish esa next predikati bilan belgilanadi, ya'ni 5 ga ortadi. Navbatdagi ta'rif esa, kenglikka izlashga olib keladi.

```
connected(A,A) <- .
connected(A,B) <- arc(A,C), connected(C,B).
```

9.10. Graflarda izlashni amalga oshiruvchi dasturlarga misollar

Holatlar makonidagi graflarda izlashni ko‘rib chiqamiz. Holatlar makoni graflari vazifalarni taqdim etish uchun qo‘llaniladi. Graflar Cho‘qqilari vazifalarning holatlaridir. Grafning ikki cho‘qqisi agar

o‘tishning (move) qandaydir qoidasi mavjud bo‘lsa, qovurg‘a bilan bog‘langandir. Bu qoidaga muvofiq bir holatni boshqasiga o‘zgartirish amalga oshiriladi. Vazifaning echimi berilgan boshlang‘ich holatdan o‘tish qoidalarining izchilligini qo‘llash vositasida tegishli izlanayotgan echimga o‘tish yo‘lini izlashdan iboratdir.

Dastur 9.6 qismda keltirilgan chuqurlikka izlashni qo‘llashga ega holatlar makonini graflarda izlash yo‘li bilan hal qilish uchun asosiy dasturdir.

Holatlarni tasavvur qilish uchun hech qanday cheklanishlar kiritish kerak emas, o‘tishlarni binar predikat move(State, Move) bilan tasvirlaymiz, bu yerda Move – State holatiga qo‘llaniladigan o‘tish qoidasidir. update(State, Move, State1) predikati State holatiga Move qoidasini qo‘llash yordamida erishiladigan State1 holatini izlash uchun qo‘llaniladi. Bir qator holatlarda move i update tartibotlarini birlashtirish maqsadga muvofiqdir. Bu yerda ular extimol samaradorlikka zarar etkazgan holda bayon qilishning osonligi va dasturlarning moslashuvchanligi hamda modulliligini saqlab qolish uchun qoladi.

Ehimoliy o‘tishlarga yo‘l qo‘yilishi State vazifasining holati vazifaning cheklanishlarini qondira olishini tekshiruvchi legal(State) predikati bilan baholanadi. Takrorlashlarning oldini olish uchun dastur oldingi o‘tilgan holatlarni saqlab qoladi. Boshlang‘ich holatdan yakuniy holatga o‘tishlarning izchilligi uchinchi dalilda solve_dfs/3 predikatini orttirish yo‘li bilan barpo etiladi

solve_dfs(State, History, Moves)

Moves - State ning joriy holatidan talab qilinayotgan yakuniy holatga erishishga qadar o‘tishlar izchilligidir. History odingi o‘tilgan holatlarga ega.

Holatlar makonida chuqurlikka izlashni tashkil etishning asosiy dasturi

```
solve_dfs(State, History, []) :-
    final_state(State).
solve_dfs(State, History, [Move|Moves]):-
    move(State, Move),
    update(State, Move, State1),
    legal (State1),
    not member(State1, History),
```

```
    solve_dfs(State1, [State1|History], Moves).
```

Асосий дастурни синовдан ўтказиш учун

```
test_dfs(Problem, Moves) <-
    initial_state(Problem, State),
    solve_dfs(State, [State], Moves).
```

Vazifani hal qilishda izlashni tashkil etishning asosiy dasturidan foydalanish uchun dasturchi move, update va legal tartibotlarining holati hamda aksiomatizasiya taqdim etish to‘g‘risidagi qaror qabul qilishi lozim. Asosiy dasturni qo‘llash muvaffaqiyati holatlarni munosib taqdim etishni tanlashga sezilarli darajada bog‘liqdir.

Aytaylik asosiy dastur, bo‘ri, echki va karam to‘g‘risidagi mashhur vazifani hal qilish uchun qo‘llanilmoqda. Ushbu vazifani noformal shakllantiramiz.

Fermerning bo‘risi, echkisi hamda karami bor va bularning barchasi daryoning chap qirg‘og‘ida turibdi. Fermer ushbu “uchlik”ni o‘ng qirg‘oqqa olib o‘tishi lozim, biroq qayiqqa bulardan faqat 1 tasi – bo‘ri, echki yoki karam sig‘adi, xolos. Bu vazifadagi eng muhimi bo‘rini echki bilan qoldirib bo‘lmaydi (bo‘rilar echki go‘shtiga o‘ch bo‘ladi), huddi shuningdek echkini karam bilan qoldirib bo‘lmaydi (karam echkining jonidili) fermer yuzaga kelgan holatdan jiddiy tashvishga tushgan va yo‘lovchini yo‘qotish hisobiga ekologik muvozanatni buzishni istamaydi.

Bu holatlar wgc(B,L,R) uchligi bilan taqdim etiladi, bu yerda u B – qayiq turgan joy (chap yoki o‘ng qirg‘oq), L – chap qirg‘oqda turganlar ro‘yxati, R – o‘ng qirg‘oqda turganlar ro‘yxati. Shunga mos tarzda wgc(left,[wolf, goat, csbbage],[]) va wgc(right,[],[wolf, goat, csbbage]) boshlang‘ich va yakuniy holatlardir..

Aslida o‘ng va chap qirg‘oq axolisini kiritish zarurati yo‘q. Hozirgi lahzada chap qirg‘oqda kim turganligini bilgan holda o‘ng qirg‘oqda turganlarni aniqlash oson va aksincha ikki ro‘yxatdan foydalanish o‘tishlarni tasvirlashni soddalashtiradi.

Takrorlanishni tekshirish uchun yashovchilar ro‘yxatini turlarga ajratilgan ko‘rinishda saqlash qulaydir. Shu tarzda bo‘ri ro‘yxatda echkidan oldin va ularning ikkalasi karamdan oldin turadi. Albatta, agar barcha yo‘lovchilar bir qirg‘oqda turgan bo‘lsa. Holatdan holatga o‘tishlar - bu yashovchilarni bir qirg‘oqdan boshqasiga olib o‘tishdir va shuning uchun biz yuk (Cargo) deb ataydigan aniq “yo‘lovchiga”

ixtisoslashtirilishi mumkin. Fermer daryo orqali suzib o'tadigan vaziyat alone (yakka, ya'ni yuksiz) yukka ixtisoslashtiriladi. member predikatining determinizasiyalashmagan xatti-harakati pastdagi dasturda ko'rsatilganidek barcha imkondagi o'tishlarni 3 gap bilan qisqacha tasvirlashga imkon beradi: chap qirg'oqdan o'ng qirg'oqqa olib o'tish uchun o'ng qirg'oqdan chap qirg'oqqa olib o'tish uchun va fermerning har qanday yo'nalishda yolg'iz suzishi uchun.

Bo'ri , echki va karam to'g'risidagi vazifadagi holatlar wgc (Boat, Left, Right) ko'rinishi tuzilmasi bilan beriladi, bu yerda Boat – hozir qayiq turgan qirg'oq, Left – chap qirg'oq yashovchilarining ro'yxati, Right– o'ng qirg'oqda yashovchilarning ro'yxati.

Bo'ri va echki to'g'risidagi vazifani hal qilish uchun dastur.

```
initial_state(wgc, wgc(left,[wolf, goat, cabbage], [])).
final_state(wgc, [], wgc(right,[wolf, goat, cabbage])).
move(wgc(left, L, R), Cargo) :- member(Cargo, L).
move(wgc(right, L, R), Cargo) :- member(Cargo, R).
move(wgc(B, L, R), alone).
update(wgc(B, L, R), Cargo, wgc(B1, L1, R1)) :-
update_boate(B, B1), update_banks(Cargo, B, L, R, L1, R1).
update_boat(left, right).
update_boat(right, left).
update_banks(alone, B, L, R, L, R).
update_banks(Cargo, left, L, R, L1, R1) :- select(Cargo, L, L1),
insert(Cargo, R, R1).
update_banks(Cargo, right, L, R, L1, R1) :-
select(Cargo, R, R1), insert(Cargo, L, L1).
insert(X,[Y|Ys],[X,Y|Ys]) :-
precedes(X,Y).
insert(X,[Y|Ys],[Y|Zs] :-
precedes(Y,X), insert(X, Ys, Zs).
insert(X,[],[X].
precedes(wolf,X).
precedes(X, cabbage).
legal(wgc(left, L, R)) :- not illegal(R).
legal(wgc(right, L, R)) :- not illegal(L).
illegal(L) :- member(wolf, L), member(goat, L).
```

illegal(L) :- member(goat, L), member(cabbage, L).

Ko‘rsatib o‘tilgan har bir o‘tishlar uchun update_boat/2 qayiq turgan joyni, update_banks qirg‘oqlarida yashovchilar tarkibini o‘zgartirishi mumkin bo‘lgan tartibot ixtisoslashtirilishi lozim. select predikatidan foydalanish modifikasiyalashtiruvchi tartibotlarni ixcham tasvirlashga imkon beradi. Qirg‘oqda yashovchilar ro‘yxatini tartibga solingan holda saqlab turish uchun takrorlanishni tekshirishni engillashtiruvchi update_banks/3 tartibotidan foydalaniladi, unda qirg‘oqdagi yashovchilar tarkibini kengaytirishni barcha ko‘rinishlari hisobga olingan va nihoyat, o‘tishlarga yo‘l qo‘yishni tekshirish ixtisoslashtirilishi lozim. Bu erdagi mavjud cheklanishlar soddagina bo‘ri va echki, xuddi shuningdek karam bilan, fermer yo‘q paytida bir qirg‘oqda qola olmaydi.

Nazorat savollari va mashqlar

1. Chuqurlikka izlash yondoshuvlari va kenglikka izlash yondoshuvlaridan keskin farqi nimadan iborat?
2. U yoki bu yondoshuvga bog‘liq holda izlanish samaradorligini taqqoslash to‘g‘risida so‘z yuritish mumkinmi?
3. Chuqurlikka izlash vazifasini hal qilish iboralarida farzinlarni joylashtirish to‘g‘risidagi vazifa echimiga yondoshuvni tasvirlang

X. BOB. TABIIY TILGA ISHLOV BERISH USULLARI VA ALGORITMLARI

10.1. Intellektual muloqot tizimlarining umumlashtirilgan sxemasi

Sun'iy intellekt tizimlarini ishlab chiqishda bunday tizimning foydalanuvchi bilan tabiiy tilda muloqot qilishi anchagina ahamiyatli jihatdir. Buning katta qismi shundan kelib chiqadiki, sun'iy intellekt tizimlaridan odatda foydalanuvchilar o‘z fan sohalaridagi ekspert olimlardir, biroq ular dasturiy ta'minot va dasturiy tizimlarni ishlab chiqish texnologiyalarini kamroq tushunadilar.

Aynan shu bois asoslari N. Xomskiy tomonidan yaratilgan formal grammatikalar nazariyasi sun'iy intellekt bo‘yicha tadqiqotlar doirasida qo‘shimcha ko‘rib chiqiladigan bo‘ldi. Ammo kompyuter lingivistikasi bo‘yicha tadqiqotlar uning muammolari u qadar soddda emasligini va chuqurroq ishlab chiqish hamda rivojlanishni taqozo etayotganligini ko‘rsatdi. Agar formal (sun'iy) tillar tahlili unchalik murakkab emas (6 bobga qarang) bo‘lsa, tabiiy tillar bilan ishlash hozircha sezilarli natijalar bermadi.

Tabiiy tilda muloqotdan foydalanuvchi tizimlarini barpo etish masalalarini ko‘rib chiqish uchun tabiiy intellekt nazariyasida tadqiqotlarning maxsus yo‘nalishi – intellekt muloqot tizimlarini o‘rganish va barpo etish ajralib chiqdi. Ushbu yo‘nalishni mazkur qismda qisqacha ko‘rib chiqiladi.

Intellektual muloqot tizimini tashkil etish vazifasi odatda aks ettirishni yaratish bilan bog‘liqdir: α: $\omega_{in} \rightarrow \omega_{out}$, где $\omega_{in} \leftarrow \Omega_{in}$ и $\omega_{out} \leftarrow \Omega_{out}$. (← alomati tegishlini anglatadi). Ω_{in} ko‘pligi — muayyan intellekt muloqot tizimi ishlaydigan muammoli sohaga xizmat ko‘rsatuvchi tabiiy tilning cheklangan tilchasidagi kirish matnlari ko‘prligidir. Ω_{out} ko‘pligi — bu qandaydir formal til bo‘lib, u ma'lumotlar bazasiga so‘rovlar ro‘yxati yoki amaliy dasturlar jildidan yoki yana shunga o‘xshashlardan qandaydir tartibotlarni chorlash ko‘pligi bo‘lishi mumkin.

Sun'iy intellekt tizimlari bilan muloqot uchun tilcha sifatida odatda, ish nasri (ba'zan – ilmiy nasr) deb ataluvchi ya'ni, o‘ta qisqaligi maksimal bir ma'noligi va mazmunni ifodalashni qisqaligi bilan xususiyatlanuvchi tabiiy til doiralaridagi muayyan tarzdan foydalaniladi.

Ish nasrining vazifasi shundan iboratki, qandaydir fan tomoni qandaydir ish uchun o‘ta aniqlik va to‘liqlikda bayon qilinadigan qandaydir mazmunni imkon qadar aniq ikki ma'noli bo‘lmagan tarzda shakllantirishdan iboratdir. Aniqlik—ish tarzining eng muhim jihati bo‘lib, uning xolisligini, muayyan holatlarda esa nutq vositalaridan foydalanishdagi standartlikni keltirib chiqaradi. Ishning mohiyatini va uning ahamiyatli tafsilotlarini aniq uzatishning muqarrar shartlari ish tarzining to‘liqligi va bir vaqtning o‘zida qisqaligidir. So‘ngra aynan ish nasri tarzi intellektual tizimi bilan muloqot uchun kirishdagi tilcha ostida tushuniladi.

Kirishdagi tilchadagi matnlarni tushunish (ishlov berish jarayonini) uch tasvirning izchil bajarilishi sifatida tushunish mumkin:

$\psi^* : \gamma_j^* \rightarrow \gamma_j$, бу ерда γ_j^* kirishdagi cheklangan tildagi muammoli sohaning qandaydir γ_j holatini tasvirlashdir. Bu kirish matni tahlilining tasviridir;

$\psi^{**} : \gamma_j \rightarrow \gamma_j^{**}$, bu yerda γ_j^{**} sun'iy intellekt tizimida qo‘llaniladigan bilimlarni taqdim etish tilidagi γ_j holatini tasvirlashdir;

$\psi^{***} : \gamma_j^{**} \rightarrow R_i$ — бу ерда R_i cheklangan tilchada ifodalangan foydalanuvchining so‘roviga tizimning qandaydir munosabatidir.

Uchinchi tasvir majburiy emas, tizim foydalanuvchining so‘roviga qayta amalga oshirmasligi, balki shunchaki aynan shunday cheklangan tilda javob qaytarishi mumkin (ammo mazkur holatda R_i munosabat deyilganda javoblarning chiqarilishi tushunilishi mumkin)

Yuqorida keltirilgan tasvirlar qo‘llaniladigan intellekt muloqot tizimining umumlashtirilgan sxemasi quyidagi rasmda keltirilgan .

Insonning intellektual muloqot tizimi muloqot qilish jarayonida tabiiy tilni tushunish muammolari paydo bo‘ladi. Bunday muammolarni qisqagina bir so‘z bir ma'noli emaslik bilan ifodalash mumkin. Kirish tabiiy tilida qanday cheklanishlar bo‘lmasin, baribir mazmundan tashqarida ma'noning bir necha ko‘rinishlariga ega bo‘lmagan iboralarni yaratish mumkin. An'anaviy misollar sifatida, odatda “ruhoniy ularga etti gunohni ta'na qildi”, “to‘qimadan oqsilni ajratib olish”, “usta pishirishni boshladi”, “time flies like arrow” iboralari keltiriladi.

Umumiy holatda tabiiy tilni tushunish muammolari ko‘p jihatdan muammoli sohadagi bilimga bog‘liq. Tilni tushunish so‘zlovchining maqsadlari va mazmun to‘g‘risidagi bilimlarni talab qiladi. Shuningdek, gap oxirigacha aytilmaganini yoki boshqacha aytilganligini hisobga olish zarur. Masalan, quyidagi “Ivan Maryani maydonda gullar bilan kutib oldi” gapida kim gullar bilan ekanligi: Ivanmi, Maryami yoki maydonmi tushunarsiz bo‘lib qolgan.

Masalan, mashhur rus tilshunosi, akademik L.V.erbaning “Glokaya kuzdra shteko budlanula bokra i kurdyachit bokrenka” iborasi bunday “tushunarsiz” ibora rus tilining butun qoidalari bo‘yicha yaratilganligi, ushbu gapni grammatik tahlilida muammo uyg‘otmasligi, biroq uni tushunishda muammolar keltirib chiqarishi haqida so‘z yuritiladi. Shu tarzda tabiiy tilni tushunishdagi muammolarni sanab o‘tish mumkin.

1. “Mazmun –matn” muammosi. Ushbu muammo haqida hozirgina so‘z yuritiladi.

2. Rejalashitrish muammosi muloqot olib borish zaruratida vujudga keladi. Buning uchun muammoli sohani chuqur bilish zarur.

3. Sinonimlarning teng ma'nologi muammosi.

4. Muloqot ishtirokchilarining modellari muammosi. Muloqot ishtirokchilarida bilimlarni ko‘rsatishning taqqoslanadigan modellari, tegishli chuqur tushunish, mantiqiy xulosa imkoniyati, harakat imkoniyati bo‘lishi lozim.

5. Elliptik konstruksiyalar, ya'ni, muloqotdagi tushirib qoldirilgan unsurlar muammosi.

6. Vaqtinchalik ziddiyatlar muammosi.

Ushbu muammolarni hal qilish oson ish emas va ularni hal qilishning bir ma'noli universal usullari ham yo‘q. Har bir sun'iy intellektual tizimida ushbu muammolarni qandaydir usul bilan chetlab o‘tish yoki ularni hal qilish uchun o‘z uslubiyatlaridan foydalaniladi, shu asnoda ushbu uslubiyatlar ko‘pincha muammoli sohaga bog‘liq.

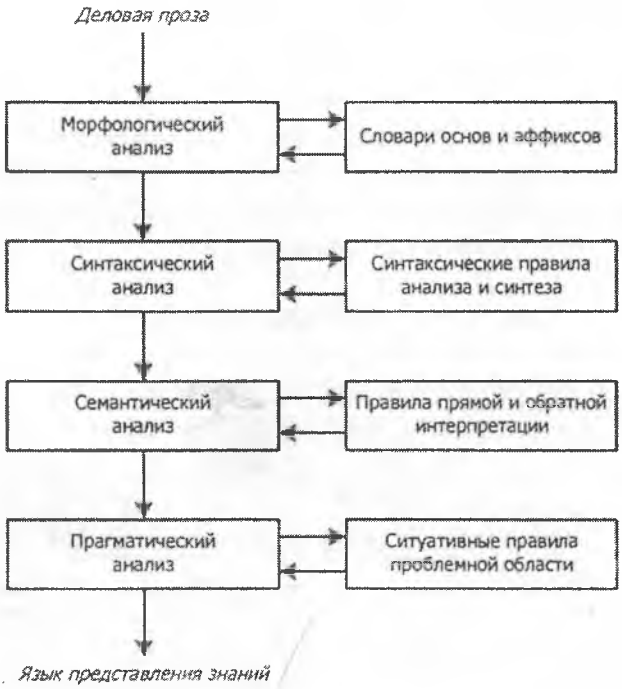
10.2. Kirish matni tahlil sxemasi

Ilgari ko‘rsatib o‘tilganidek, tabiiy til sifatida bunday tilning cheklangan qandaydir ko‘plikchasi, ya'ni tabiiy tilning kasbiy yo‘naltirilgan ko‘plikchasi, yohud (ish nasri) ko‘rib chiqiladi. Ish nasridagi kirish matnlarini tahlil qilish uchun kirish matni tahlili sxemasini amalga oshiradigan lingvistik prosessorlar deb ataluvchi dasturi majmualaridan foydalaniladi. Quyida bunday lingvistik prosessorning extimoliy tuzilmali sxemasi keltirilgan.

Ushbu rasmdan ko‘rinib turganidek, tabiiy tildagi matnlar tahlili to‘rt bosqichdan – morfologik tahlil, sintaktik tahlil, semantik tahlil va pragmatik tahlildan iborat, ba'zan ushbu sxemaga leksik tahlil, ya'ni kirish iborasini leksemalar ro‘yxatiga parchalash ham kiradi, biroq odatda ushbu vazifani morfologik analizator o‘z ishining boshida bajaradi.

Javobni umumlashtirish vazifasi shunga o‘xshash sxema bilan tasvirlanadi. Undagi o‘zaro harakat o‘qi bilimni tasavvur qilish tilidan ish nasri tili tomon tahlilning barcha bosqichlari orqali teskari tartibda yo‘naltiriladi (ular umumlashtirishda tabiiyki, umumlashtirish bosqichlari deb ataladi).

Morfologik tahlil bosqichida kirish leksemalariga ishlov berish mulohaza mazmuni bilan aloqadan tashqarida ishlov beriladi. Morfologik tahlilning asosiy vazifasi leksemalarni taqqoslash hamda navbatdagi bosqich bo‘lgan sintaktik tahlil uchun kirish ma'lumotlari bo‘lib xizmat qiluvchi morfologik axborotni ularga taqdim etishdir.



Morfologik axborot deyilganda sun'iy intellektning muayyan tizimida muhim bo‘lgan kirish leksemalarining muayyan grammatik alomatlari tushuniladi. Odatda tabiiy tildan kelib chiqadigan barcha grammatik alomatlarni ajratib ko‘rsatuvchi to‘liq morfologik tahlil amalga oshiriladi. Ba'zan vaqt va zahiralarni tejash maqsadida ayrim ahamiyatsiz bo‘lgan grammatik alomatlar ajratib ko‘rsatilmaydi.

Grammatik alomatlar misoli sifatida rus tilining grammatik kategoriyalarini ajratib ko‘rib chiqish mumkin. Eng birinchi kategoriya bu gap bo‘lagidir. Har bir gap bo‘lagi uchun o‘zgarmas alomatlar va o‘zgaruvchi alomatlar ajratib ko‘rsatiladi. Masalan, rus tilidagi ot uchun jins va turlash o‘zgarmas alomatlar, kelishik va son esa o‘zgaruvchi alomatlardir. Sifat nomi uchun masalan sifatlilik o‘zgarmas alomat, jins, son va kelishik esa o‘zgaruvchan alomatlardir.

Umumiy ko‘rinishda morfologik tahlil o‘zgarmas so‘zlarni ajratib ko‘rsatishdan (buning uchun tayyor so‘z shakllari lug‘ati zarur) va

shuningdek, o‘zgaruvchan so‘zlardagi asoslarni va affikslar to‘plamini ajratib ko‘rsatishdan iborat. Olingan affikslardan bir ma'noli bo‘lmasligi mumkin bo‘lgan morfologik axborot yaratiladi. Ushbu maqsadlar uchun asoslar lug‘ati va har biriga muayyan affiksga mos kelishi mumkin bo‘lgan grammatik kategoriyalar ro‘yxati berilgan affikslar lug‘ati zarur. Masalan “Ko‘lda baliqlarning har xil turlari yashaydi” iborasi chiqishda quyidagi axborotni qaytaradi.

1.DA: ko‘shimcha.

2.KO‘L: ot, birlik son.

3.YaShAMOQ: fe'l, ko‘plik son, uchinchi shaxs, tugallanmagan ko‘rinish.

4.HAR-XIL: sifat, ko‘plik son, bosh kelishik.

5.KO‘RINISh: ot, ko‘plik son, bosh kelishik yoki chiqish kelishigi.

6.BALIQ: ot, ko‘plik son, qaratqich kelishigi.

Shu tarzda, morfologik tahlilning o‘zi garchi lug‘atlarni yaratish bo‘yicha o‘ta katta tayyorgarlik ishi talab qilinsada unchalik og‘ir jarayon emas, toleimizga, aksariyat keng qo‘llaniladigan tabiiy tillar uchun bunday lug‘atlar yaratib qo‘yilgan. Masalan, rus tili uchun A. A. Zaliznyakning grammatik lug‘atiga asoslangan lug‘atlardan foydalaniladi, unda 100 mingdan ortiqroq eng ko‘p qo‘llaniladigan so‘zlar jamlangan bo‘lib, ularning har biriga batafsil grammatik axborot – gap bo‘lagi, so‘z o‘zgarish qolipi, extimoliy affikslar ro‘yxati keltirilgan.

Sintaktik tahlil deyilganda kirish gaplardagi leksemalar o‘rtasidagi munosabatlarni barpo etish tushuniladi. Ushbu munosabatlar morfologik tahlil bosqichida olingan axborot asosida barpo etiladi, chiqish natijasi sifatida sintaktik tahlil gapning sintaktik tuzilmasini qaytaradi.

Gapdagi leksemalar o‘rtasidagi munosabatlar barcha tabiiy tillar uchun ozmi-ko‘pmi universaldir. Birinchi navbatda u ega va kesim, to‘ldiruvchi, hol va inkorning munosabatidir. Ba'zan ushbu munosabatlar ro‘yxatiga boshqa kompletiv munosabatlar (olti turga qadar) bir turli a'zolar, yordamchi so‘zlar qo‘shiladi.

Rus tilida sintaktik munosabatlarni barpo etish uchun ayrim evristik qoidalarni ajratib ko‘rsatish mumkin. Odatda, masalan bu ot, sifat yoki bosh kelishikda (qaysi sondaligi muhim emas) turgan olmoshdir. Kesim – bu shaxsda va sonda ega bilan moslashadigan fe'ldir (yoki o‘ta kamdan-kam holatda qisqa shakldagi sifatdir). To‘ldiruvchi – bu kelishikda kesim

bilan moslashadigan otdir: bevosita to'ldiruvchi chiqish kelishigidagi otdir. Bilvosita to'ldiruvchi chiqish kelishigida bo'lmagan ko'pincha old qo'shimchaga ega to'ldiruvchidir. Aniqlovchi — bu ega yoki to'ldiruvchi bilan bog'liq bo'lgan sifatdir (bu bog'liqlik jinsda, sonda va kelishikda moslashgan bo'lishi lozim).

Hol — bu, odatda kesim bilan faqat semantik bog'liq bo'lgan old qo'shimchaga ega kesim yoki otdir. Har qanday holatda ushbu qoidalar faqat umumiy qolipni tasvirlaydi, xolos va hech qachon sintaktik tahlilni amalga oshirishda dogma bo'la olmaydi.

Kirish matnining semantik tahlili yoki semantik talqin gap mazmunining semantik tasavvurini belgilaydi. Mazmunning qandaydir ko'rinishidagi modeli semantik tahlil natijasidir. Semantik tahlilning maqsadi sun'iy intellekt tizimida ma'lum bo'lgan ichki tushunchalar munosabatlar va holatlardagi gap mazmunini bir ma'noda ifodalash, shuningdek, buyruq gaplar uchun "yangi" deklarativ axborot, so'roq gaplar uchun so'roq unsuri tushunchalarini ajratib ko'rsatishdir.

Semantik tahlil quyidagi bosqichlarni o'z ichiga oladi.

1. Bo'laklarga ajratilgan unsurlarga ega navbatdagi tahlil qilinayotgan unsurning grammatik va semantik nisbati. Muayyan testlarni o'tkazish bilan bog'langan so'zlar guruhidagi unsurlarni birlashtirish. Bunday testlar yordamida kirish lug'atida axborot saqlanayotgan qayd qilingan sintaktik konstruksiyalar mavjudligini tekshirib ko'rish mumkin. Munosabatlarning binar jadvali aniqlanayotgan va tobe leksik unsurlarni ularni grammatik –semantik alomatlari va tobe so'zning semantik o'rnini ko'rsatgan holdagi aniqlovchi juftligiga ega.

2. Bog'liq so'zlarni bosh so'zni ajratib ko'rsatish, guruh ichidagi semantik rollarni aniqlash va lug'atdan olingan axborot asosida guruhning umumiy grammatik alomatlarini (jins, son, kelishik va x.k.) aniqlash bilan unsurlarni shakllantirishni yakunlash.

3. Lug'atga ko'ra predikatni va uning alomatlarini aniqlash hamda predikatlar guruhi holatida "bo'lmoq" fe'lining bosh, bog'lovchi predikatlarini, prediativ unsurlarini aniqlashdir, grammatik-semantik alomatlarga ko'ra predikatlar predikat shaklini tanlash uchun zarur bo'lan lug'atda ko'rsatilgan bir necha toifalarga bo'linadi.

4. So'zlarning kirish izchilligi nihoyasiga etgach predikatning toifasi va turiga ko'ra qolipni tanlash amalga oshiriladi. Turiga (shaxsli, shaxssiz,

egasiz gap majhul nisbatga bog‘liq holda shablonning tegishli ko‘rinishi tanlab olinadi. Predikatlar va otlarning binar munosabatlari jadvallari yordamida qolipni to‘ldirish amalga oshiriladi. Noaniqlik holatida aynan shu binar jadvallari yordamida ot guruhlari o‘rtasidagi qo‘shimcha bog‘liqliklarni ajratib ko‘rsatish yuz beradi.

5. Синтактик ва семантик таҳлилларни амалга ошириш воситаси сифатида кўпинча бошқариш модели сифатидаги формализмдан фойдаланилади. Бошқариш модели – бу табиий тил предикатининг гапнинг бошқа унсурларини ўзига бўйсундириш салоҳиятидир.

Berilganni boshqarish modelida berilgan predikat gapning boshqa a'zolari bilan ega bo‘lishi mumkin bo‘lgan sintaktik va semantik munosabatlar to‘g‘risidagi axborot yozib olinadi. Binobarin boshqarishning har bir modelida ikki valentlik: sintaktik va semantik valentlik mavjuddir sintaktik valentlik deyilganda son va predikatga tobe bo‘lgan qo‘shimchalar xususiyati tushuniladi. Semantik valentlik deyilganda son va predikat tasvirlaydigan vaziyatlar aktantlari xususiyati tushuniladi, shu asnoda har bir aktant qandaydir chuqurlik kelishiklaridan birida turadi.

Agar “Ko‘lda baliqlarning har xil turi yashaydi” misolini ko‘rib chiqadigan bo‘lsak, yashamoq fe'li ushbu gapdagi predikatdir. Boshqarish modelidagi ushbu predikatga agent muayyan joyda yashashi mumkinligi to‘g‘risidagi axborot yozib qo‘yilishi lozim. Bunday axborotdan kelib chiqqan holda muayyan gapning semantik tuzilmasi quyidagi rasmda ko‘rsatilgandek barpo etilishi lozim.

Lingvistik freym deb ataluvchi ko‘rinishda bu quyidagi tarzdagi ko‘rinishga ega bo‘ladi:

(predikat (yashamoq)
  (agent (tur)
    (aniqlashtirish (baliq))
    (detal (har xil))
  (joy (ko‘l))

Ko‘rinib turganidek, bu yerda bog‘liq so‘zlarning ikki guruhi mavjud, ular 7.10 rasmda kiritilganlikning ikki turli darajasida ko‘rsatilgan. Birinchi guruh “yashamoq” predikatiga, ikkinchisi shunga mos tarzda “tur” agentiga bo‘ysunadi.

Kirish matnining pragmatik tahlili bu sun'iy intellekt tizimining o‘ziga, uning bilimlar bazasiga va tizim joylashgan holatga kirish iborasining munosabatini tahlil qilishdan iborat. Pragmatik tahlil kirishda kirish iboralarining ma'nosi taqdim etilgan lingvistik freymlarning ro‘yxatini oladi, shundan so‘ng bular asosida sun'iy intellekt tizimidagi bilimlarni ichki tasavvuri uchun xizmat qiluvchi pragmatik freymni yaratish amalga oshiradi.

Har bir muammoli soha uchun pragmatik freymlarning abstrakt tarmog‘ini o‘z ichiga oluvchi bilimlarning o‘z bazasi ishlab chiqilishi lozim. Bunday freymlarning yuqori darajasi qayd etilgandir va faraz qilinayotgan vaziyatda doimo haqiqiy bo‘lgan holatlarga ega. Quyi bosqichlar ko‘plab terminal unsurlarga, ya'ni aniq ma'lumotlar bilan to‘ldirilishi zarur bo‘lgan “uyachalarga” ega. Har bir terminal unsurda qiymatlarning bunday taqdim etilishini qondirishi zarur bo‘lgan shartlar sanab o‘tilishi mumkin.

Pragmatik tahlil — bu sog‘lom fikr darajasidagi modellar to‘g‘risidagi mulohazalardir. Aynan shu erda tabiiy tilni tushunish muammolari fikrlash muammolari sohasiga kiradi va bilimlarning katta bazasiga murojaat qilish yuz beradi. Aynan shu sohada ko‘plab qiyin vazifalarni hal qilish hali oldinda turadi.

Ayrim yakuniy izohlar

Tabiiy tildagi sun'iy intellekt tizimlari bilan muloqot qilish masalalarining ushbu qisqa ko‘zdan kechirishni nihoyasiga etkazgan holda bunday muloqot uchun mamlakatimizdagi dastlabki samarali tizimlarga misol sifatida “ShOIR” va “ADALIT” tizimlarini keltirish mumkin. tabiiy

tilga lingivistik ishlov berish oshasida so‘nggi vaqtda, shubhasiz muvaffaqiyatlar ko‘zga tashlanmoqda: mashinada tarjima qilishning tijorat tizimlari, tabiiy tildagi matnlardan axborot izlash va bunday matnlarni taqrizlash va xakozolar paydo bo‘ldi. Matnlarga ishlov berish tizimlarining keng turlari yaratildi.

Ammo kelgusida ishlov berish lozim bo‘lgan vazifalarning keng toifasi ham mavjud, bular: xatboshi doiralarida va undan kattaroq bog‘liq matnlarni uzatish, matnni to‘laqonli lingivistik umumlashtirish, modellarni to‘ldirish jarayonini avtomatlashtirish, lingivitsik ishlov berishni tekshirish, va lingivistik modellarni to‘liqligi, to‘g‘riligi va turli tumanligini tekshirish usullaridir. Shuningdek, muammoli muhim modellarini, lingivistik prosessorlar uchun chiqarish mexanizmlarini ixchamlashtirish masalalari etarli ishlab chiqilmaganligini ta'kidlash joiz.

Nazorat savollari

1. Intellektual muloqot tizimi nima? Bunday tizim tabiiy tilda foydalanuvchi bilan muloqotni tashkil etish uchun qanday mexanizmlardan foydalanadi?
2. Morfologik tahlil nima, u nima uchun qo‘llaniladi?
3. Sintaktik tahlil nima, u nima uchun qo‘llaniladi, qanday kirish ma'lumotlarini oladi?
4. Semantik tahlil nima, u qanday maqsadlarni ko‘zlaydi?
5. Pragmatik tahlil nima, buni amalga oshirishda qanday muammolar mavjud?

XI. BOB. ZAMONAVIY YONDOSHUVLARIDAN FOYDALANISH

11.1. Meroslash

Dasturlashning ob'ektli yo'naltirilgani paradigmasi uch asosga—inkapsulyasii, meroslash va polimorfizmga tayanadi. Ushbu qismda meroslash ko'rib chiqiladi.

Haskell tilidagi toifalar—bu dasturlashning ob'ektli yo'naltirilgan toifalarga nisbatan qandaydir kattaroq narsadir. Garchi ular bir birlarining xossalarini meros qilib olsada Haskell tilida ma'lumotlarining muayyan turida toifalarni zarurati mavjud. Ushbu ikki munosabatlar –meroslash va joriy etish dasturlashning ko'rib chiqilayotgan turida birgalikda kechadi.

Shu tarzda ob'ektli yo'naltirilgan dasturlashning uch asosini to'la saqlab qolgan holda Haskell tilini ob'ektli yo'naltirilgan til deb atash mumkin, ya'ni ikki paradigmaning sinergizmi yaqqol ko'rinib turibdi. Bu tilning o'zini yanada moslashuvchanroq hamda vazifalarning eng ko'p qirralarini hal qilish uchun butkul mos keladigan tilga aylantiradi.

Takror va takror aytib o'tilganidek Haskell tilida meroslash konsepsiyasi mavjud. Bu butkul tabiiy holdir – hamonki, dasturlash tilida toifa tushunchasi bor ekan nima uchun ushbu toifalarni bir–biridan meroslanadigan qilish mumkin emas? Garchi Haskell tilida toifa deyilganda standart ob'ektli yo'naltirilgan paradigmaga nisbatan qandaydir kattaroq abstrakt narsa tushunilsada, meroslashni ham mazkur holatda tasniflash mumkin. Shunisi ham tabiiyki, agar bir toifa meroslashda boshqa toifa bilan munosabatda turgan bo'lsa, bu meros oluvchi toifani joriy etishda bunday toifaning ekzemplyari asosiy toifaning barcha usullarini saqlab qolishi lozim (ya'ni mohiyatiga ko'ra meros oluvchi toifa ekzemplyari bo'lgan tur asosiy toifaning ekzemplyari bo'lishi lozim).

Masalan, Prelude standart modulida ma'lumotlarning taqqoslanadigan turlarini namoyon etuvchi Ord toifasi belgilangan, bu toifa Eq tofiasidan meros qilib olingan, chunki ma'lumotlarning taqqoslanadigan turi ham (==) (teng) yoki (/=) (teng emas) amaliyotlari yordamida ekvivalentlik toifalariga taqsimlanishi mumkinligi butkul tushunarlidir. Ord toifasini belgilash quyidagicha ko'rinishga ega:

```
class (Eqa) =>Ordawhere
  (<), (>), (<=), (>=) :: a -> a -> Bool
```

min, max :: a -> a -> a

Ko'rinib turganidek, bir toifa boshqasidan meros qilinib olinish holatini tasvirlash Haskell tilida standart tarzda – kontekst yorlamida amalga oshiriladi. Bu funksiyalarning signaturalarida qo'llaniladigan turlarga cheklanishlarni ko'rsatish uchun amalga oshirilgan edi.

Parametrik o'zgaruvchan turlarni aniqlash konteksti bunday o'zgaruvchanlarni shu tarzda shunchaki cheklaydiki, ular muayyan toifalarning faqat ekzemplyari bo'lib qoladi.

Eng hayratlisi shundan iboratki, Haskell tili ko'plik meroslashga ega (dasturlashning har bir ob'ektga yo'naltirilgan tili ham meroslashning bunday turiga ega bo'la olmaydi). Bir necha asosiy toifalardan meroslashdan foydalanish holatida ularning barchasini kontekstga ega tegishli seksiyada vergul orqali shunchaki sanab chiqish etarlidir.

Yuqorida keltirilgan Ord ta'rifini quyidagicha tarzda o'qish mumkin: "a turini parametrlovchi Ord toifasi Eq toifasining barcha usullarini meros qilib oladi, biroq shu asnoda o'z usullarini belgilaydi va aynan (usullar va ularning turlarini sanab o'tishdir)".

Ord toifasining barcha ekzemplyarlar "kichikroq", "kattaroq", "kichikroq yoki teng", "kattaroq yoki teng", "minimum» yoki "maksimum" amaliyotlaridan tashqari tenglik amaliyotlarini ham (==) va (/=) amaliyotlarini ham belgilashi lozim, chunki bunday turlar ayni vaqtda Eq ekzemplyarlari ham bo'lishi lozim. Ya'ni qandaydir asosiy toifadan (shu jumladan bir nechta) meros qilib olingan toifalarning ekzemplyarlari tabiiyki, asosiy toifalarning barcha usullarini ham saqlab turishi lozim.

Haskell tilida toifalarni meros qilish munosabatiga bir eng muhim cheklanish qo'yiladi. Meroslash munosabatlarining silsilasi har qanday murakkablikdagi tranzitiv yopib qo'yishlarga ega bo'lmasligi lozim, ya'ni bunday munosabatlar grafik asiklik bo'lishi lozim. Toifa hatto boshqa toifalar orqali vositalanganda o'z–o'zini meros qilib ololmaydi. Aks holda uzatish xatosi yuz beradi.

Prelude standart modulida ko'plab sodda vazifalarni hal qilish uchun mos keladigan ko'plab toifalar belgilangan bunday toifalarning batafsil tasviri Haskell tiliga kirishda keltiriladi.

Toifani oddiy belgilash qolipida u usullar ta'rifiga ega qismga ega bo'lmasligini, Ya'ni where qismi shunchaki yo'q bo'lishini ta'kidlash qoldi, xolos. Bu ishlab chiqaruvchi taxminan quyidagi toifalar:

class (Reada, Showa) =>Textuala

ni belgilash huquqiga ega ekanligini anglatadi.

Bunday e'lon boshlang'ich toifalarning barcha usullarini meros qilib oladigan katta toifaga majmuini birlashtirish uchun foydali bo'lishi mumkin, ammo mazkur holatda qandaydir tur barcha yaratuvchi toifalarning (ushbu misolda - Read va Show) ekzemplyari bo'lganda ham u avtomatik tarzda meros oluvchi toifaning (Textual) ekzemplyariga avtomatik tarzda aylana olmasligini hisobga olmaslik zarur. Mazkur holatda joriy etishni aniq ko'rsatish zarur.

11.2. Joriy etish

Ammo, agar OYD da toifalar - bu ma'lumotlar turlari bo'lsa, Haskelldagi toifalar ma'lumotlar turlari emas. Xotirada turi qandaydir toifa bo'lgan va unga ushbu toifa tomonidan belgilangan ishlov berish usullari tegishli bo'lgan ob'ektni yaratish mumkin emas. Bunday imkoniyatga ega bo'lish uchun ma'lumotlarning aynan qaysi turlari toifalarning ekzemplyarlari ekanligini belgilash zarur buning uchun instance kalit so'zidan foydalaniladi.

Masalan, Prelude standart modulida Int turi Eq toifasi ekzemplyari ekanligi belgilangan, ya'ni Int (yaxlit sonlar) qiymati bir-biri bilan tenglik (==) va (/=) amaliyotlari yordamida taqqoslanishi mumkin. Bu holat quyidagi tarzda yozilgan:

instance Eq Int where

(==) = primEqInt

Mazkur yozuvni quyidagicha o'qish mumkin: "Int turi Eq toifasining ekzemplyaridir. Shu asnoda usul (==) primEqInt funksiyasi orqali belgilanadi". where kalit so'zidan so'ng muayyan ekzemplyarga qo'llashdagi toifa usullarini sanab o'tiladi. Masalan, ushbu misolda sodda bo'lgan hamda translyatorning ichida joriy etilgan primEqInt ayrim funksiyasi orqali faqat bir usul (==) belgilanadi.

Butkul tabiiyki, primEqInt funksiyasi Int -> Int -> Bool turiga ega bo'lishi lozim (ta'kidlash zarurki, ushbu ta'rifdagi a turidagi parametrik o'zgaruvchan qiymat Eq toifasi ta'rifidan toifaning ekzemplyari, ya'ni Int bo'lgan aniq turga alamashtiriladi) agar ushbu funksiyada boshqa tur bo'lgan holatda xato yuz beradi.

Ushbu ta'rifda (/=) usuli uchun, shuningdek Eq toifasi uchun belgilangan joriy etish ko'rsatilmaganligi hayratlantirmasligi lozim,

chunki bunday ta'rifni Haskell tili translyatori Eq toifasini aniqlashda ham ko‘rsatib o‘tilgan usullar (==) va (/=) usullari o‘rtasidagi bog‘liqlik to‘g‘risidagi axborot asosida mustaqil barpo etadi.

Toifalar va ma'lumotlar turlari bir-biridan mustaqil ekanligini, ular bir-biri bilan har qanday aloqasiz belgilanishi mumkinligini ta'kidlash joiz. Faqat instance kalit so‘zidan foydalanishgina ma'lumotlar turlari va toifalarini bog‘lashga imkon beradi. Alohida tarzda toifalarning mavjudligi va ma'lumotlar turlarining mavjudligi hech narsani anlatmaydi. Qandaydir toifaning ekzemplyari bo‘lmagan ma'lumotlar turi ham baribir dasturda qo‘llanilishi mumkin. Joriy etilishi yo‘q bo‘lgan toifa quruq bayon bo‘lib, u umuman befoydadir.

Toifani ta'riflashning boshqa murakkab misoli 2.5 qismda ikkiyoqlama shajaralarni tasavvur qilish uchun qo‘llaniladigan ma'lumotlarning cheksiz turini sodda ta'riflash keltirilgan. Ma'lumotlarning bunday turi ham ayrim toifalarning ekzemplyari bo‘lishi mumkin. Masalan Prelude standart modulida Functor toifasi belgilangan bo‘lib, u ma'lumotlarning bir tuzilmasini boshqasiga ko‘chirish uchun fmap usulini tasvirlaydi. Ushbu usul ro‘yxatlar uchun tar funksiyasining umumlashmasidir (shu o‘rinda “a turi unsurlari ustidan ro‘yxat” Functor toifasi ekzemplyaridir).

Binar shajara uchun muayyan funksiya yordamida bir shajarani boshqasiga ko‘chirish uchun funksiya butkul mavjud bo‘lishi mumkin. Shuning uchun Tree a turini Functor toifasi ekzemplyari sifatida butkul tabiiy ta'riflash mumkin:

```
instance Functor Tree where
fmap f (Leaf a) = Leaf (fa)
fmap f (Branch left right) = Branch (fmap f left) (fmap f right)
```

xuddi shu tarzda Tree a turi Eq toifasi ekzemplyari ekanligini belgilashi mumkin (axir ikkiyoqlama shajaralarni bir-biriga taqqoslash mumkinku):

```
instance (Eq a) => Eq (Tree a) where
 Leaf a == Leaf b = a == b
 (Branch 11 rl) == (Branch 12 r2) = (11 == 12) && (rl == r2)
_ == _ = False
```

Bu yerda toifalar ekzemplyarlari ta'riflarida kontekstdan foydalanish imkoni, ya'ni parametrik o‘zgaruvchan turlar qiymatlariga cheklanish

ko‘rinib turibdi. Bu o‘zlari tur – konteynerlar ekanligini ko‘rsatish uchun parametrik o‘zgaruvchan qiymatlardan foydalanadigan turlar uchun butkul yaroqlidir (masalan, Tree a turi o‘z ichida a tur qiymatini saqlaydigan tur-konteynerdir).

Dasturiy ta'minot ishlab chiqaruvchisining ishini sal engillashtirish uchun Haskell tilida ma'lumotlarning ancha sodda turlari belgilangan toifalarning ekzemplyarlari ekanligini translyatorga avtomatik ko‘rsatish uchun vositalar mavjud. Eq, Ord, Enum, Bounded, Show va Read tofialari shunday toifalardir. O‘z navbatida ushbu toifalarning ekzemplyarlarini umuman olganda har qanday tur bo‘lishi mumkin hamda suzuvchi nuqtaga ega sonli qiymatlarga ishlov berish bog‘liq bo‘lmagan ma'lumotlarning ancha sodda turlari uchun avtomatik tarzda belgilash mumkin.

Toifalar ekzemplyarlarini avtomatik tarzda barpo etish uchun ushbu toifalarni dumaloq qavslarda ma'lumotlar turi ta'rifidagi deriving kalit so‘zidan so‘ng vergul orqali sanab o‘tish zarur:

```
data Color = Red
           | ...
           | Custom Int Int Int — RGB components
deriving (Eq, Ord, Show, Read)
```

Xuddi shunday tarzda, parametrik o‘zgaruvchan qiymatlarga ega turlar uchun toifalar ekzemplyarlarini ham avtomatik tarzda barpo etish mumkin:

```
data Maybe a = Nothing
             | Just a
deriving (Eq, Ord, Show, Read)
```

Mazkur holatda Haskell tili translyatori agar tur buning uchun yaroqli bo‘lgan holatda ekzemplyarlarni avtomatik tarzda barpo etish mumkin.

Nazorat savollari

1. Dasturlashning ob'ektli yo‘naltirilgan paradigmasidagi qaysi tushuncha Haskell tilidagi toifa tushunchasiga o‘ta o‘xshashdir?

2. Ob'ektli-yo‘naltirilgan dasturlashning uch ustuni qanday? Haskell tilida ushbu ustunlarning qaysi joriy etilishidan foydalaniladi?

3. Haskell tilida toifalarning ko‘plik meroslanishidan foydalanish imkoniyati mavjudmi?

4. “Toifani joriy etish” nima?

5. Ma'lumotlarning muayyan turida toifa usullarini joriy etish nima uchun zarur?

6. Ma'lumotlarning bir turida birinchi toifalarni joriy etish mumkinmi?

XII. BOB. SUN'IY INTYELLEKT TIZIMLARINI YARATISH USULLARI VA VOSITALARI

12.1. Sun'iy intellektning asosiy vazifalari

Sun'iy intellekt — bu bunday usullar inson ongi tomonidan qo'llaniladigan formallashtirilmagan yoki zaif formallashtirilgan echim usullarini algoritmik joriy etishni o'rganuvchi axborotlashtirishning eng yangi qismlaridan biridir. Boshqacha so'zlar bilan aytganda sun'iy intellekt doiralarida aniq algoritmik echimga ega bo'lmagan vazifalarni kompyuterda hal qilish usullari o'rganiladi. Ta'kidlash joizki, so'nggi vaqtda "sun'iy intellekt" iborasi ostida garchi inson ongini modellashtirish sun'iy intellektni rivojlantirishning yo'nalishlaridan biri deb hisoblash mumkin bo'lsada, endilikda inson ongini modellashtirish tushunilmaydigan bo'lib qoldi.

Garchi bu borada bir qator gipotezalar, masalan Nyuella-Saymon gipotezasi mavjud bo'lsada, intellektuallikka erishishning zaruriy va etarli shartlari sifatida aynan nimani hisoblash mumkinligi nazariyada aniq belgilanmagan.

Sun'iy intellekt doiralarida ikki asosiy yo'nalish farqlanadi:

1) Timsolli (semiotik, pastlab boruvchi) yo'nalish, bu inson tafakkurining yuksak darajali jarayonlarini modellashtirishga, bilimlarni taqdim etish va foydalanishga asoslangandir.;

2) Neyrokibernetik (neyrotarmoqli, yuksalib boruvchi) yo'nalish, bu miyaning (neyronlarning) ayrim past bosqichli tuzilmalarini modellashtirishga asoslangandir.

Shu tarzda, sun'iy intellektning eng oliy vazifasi formallashtirilmagan vazifalarni inson bilan taqqoslanadigan yoki undan ustun bo'lgan hal qilish samaradorligi darajasiga ega bo'lgan kompyuterli intellekt tizimni yaratishdir. Intellektuallikning mezoni va asosli taxriri sifatida Tyuring testi sifatida mashhur bo'lgan fikran o'tkaziladigan eksprement taklif etilgan .

Tyurin testi 1950 yilda Alan Tyuring tomonidan kompyuter so'zining insoniy ma'nosidagi ongli ekanligini tekshirish uchun "Hisoblab chiqish mashinalari va ong" maqolasida taklif etilgan edi. Test shundan iboratki, hakam (inson) ikki suhbatdosh bilan tabiiy tilda yozishma olib boradi. Bulardan biri – inson, boshqasi kompyuterdir. Agar hakam kim

ekanligini ishonchli tarzda belgilay olmasa, kompyuter testidan o‘tgan hisoblanadi. Suhbatdoshlardan har biri aynan uni inson deb e'tirof etishlariga intilishi tahmin qilinadi. Testni sodda universal qilish maqsadida yozishma matnli habarlar bilan almashishga borib taqaladi.

Yozishma hakam javoblarining tezligidan kelib chiqqan holda xulosa qila olmasligi uchun vaqtning nazorat qilinadigan oralig‘ida amalga oshiriladi (Tyuring zamonlarida kompyuterlar insondan sekinroq ishlagan. Hozirgi vaqtda bu qoida zarurdir, chunki ular insonga nisabatan ancha tez ishlamoqda).

A.Tyuring uning fikriga ko‘ra “mashina o‘ylay oladimi?” be'mani savolini almashtirish uchun bir muncha aniq testni taklif etgan.

Hozirgi vaqtda biron bir apparatli –dasturiy tizim Tyuring testidan muvaffaqiyatli o‘tish uchun rivojlanishning bunday darajasiga eta olmadi. Shuning uchun so‘nggi vaqtda sun'iy intellekt mafkurasi doiralarida tizimlarni yaratishga uchinchi yondoshuv va aynan aralash inson-mashina yoki yana aytayotganlaridek, tabiiy va sun'iy intellekt imkoniyatlarini birlashtirishni amalga oshiruvchi interfaol itellektual tizimlarni yaratish ko‘rib chiqilmoqda. Tabiiy va sun'iy intellekt o‘rtasida funksiyalarni oqilona taqsimlash va inson va mashina o‘rtasidagi muloqotni tashkil etish ushbu tadqiqotlardagi eng muhim muammolardir.

Ushbu tushuncha sun'iy intellektni ishlab chiqish vazifasini qo‘yishning eng zamonaviy jihatlaridan biri va aynan sun'iy intellektni uning oliy intellektual tushunishdagi singulyarlikka yaqinlashtirish, ya'ni inson ongining rivojlanishi natijasida kelgusidagi o‘zgarishlar jadal harakatning yanada yuqori darajasiga va tafakkurning yangi sifatiga ega ong paydo bo‘lishiga olib keladigan nano texnologiya, biotexnologiya va sun'iy intellekt rivojlanishi natijasida kelajakdagi taxmin qilinayotgan nuqtaga erishish uchun vosita sifatidagi sun'iy intellektni ko‘rib chiqishdir.

12.1.1. Sun'iy intellekt rivojlanishi tarixi

Qadim zamonlardan beri inson sun'iy hayotni yaratishga intilib kelgan. Qadimgi Yunoniston faylasuflari inson shuurining tabiati va mexanik vositalar yordamida uni modellashtirish imkoniyatlari ustida bosh qotirgan. O‘rta asrdagi alkimyo huddi inson sifatidagi tafakkurlash qobiyaliyatiga ega bo‘lgan sun'iy mavjudotni yaratishga urinishlar rivojlanishiga yangi turtki berdi. Alkimyogarning teskari idishsi

gomunkulus — mitti sun'iy inson tug'ilish mumkin bo'lgan idish sifatida ko'rib chiqqan.

O'rta asrda Yevropada olam to'g'risidagi mexanik tasavurlar gullab-yashnagan davrda sun'iy hayotga erishish istagi yanada rivojlandi. Buning ustiga murakab bo'lmagan harakatlarni (aynan bir matnni yozish, g'ijjakda aynan bir musiqani chalish va hokazolar) bajaruvchi ayrim mexanik harakatlar yaratilganidan so'ng ilmiy olamda yaqin orada sun'iy inson yaratilishi umidalridan iborat bo'lgan o'ziga xos ko'tarinkilik paydo bo'ldi. Ammo bu umidlar amalga oshmadi.

Dastlabki hisoblash mashinalari yaratilgan zamonda sun'iy ong borasidagi navbatdagi ko'tarinkilik paydo bo'ldi. O'sha davrda barchaga xuddi insonlar singari fikrlaydigan dasturlar yaratilishi uncha uzoqda emasdek bo'lib ko'ringan,

Ammo, dastlabki kompyuterlar insondan tezroq xatto hisoblashni ham amalga oshrishga imkon berishi mumkin bo'lgan mahorat darajasini ko'rsata olmadi, shuning uchun ko'p o'tmay dastlabki umidlar xafsalasizlik va ishonchsizlikka aylandi.

Oradan muayyan vaqt o'tgach sun'iy intellektni vazifalarini ko'rib chiqish yo'nalishi o'zgardi, chunki aksariyat olimlar vazifalarni hal qilishning insoniy usulini va umuman ong va intellektuallikning tabiatini yanada aniqroq tushunmasdan turib, insoniy ongni modellashtirish behudaligini anglab etdilar. Aynan shu bois, ushbu qismning boshida tasvirlangan sun'iy intellektni yangicha tushunish ishlab chiqila boshlandi. Shunday bo'lsada, mulohaza yuritish va ongning insoniy usullarini to'liq yoki qisman modellashtirishga urinishlar hanuzgacha to'xtagan emas.

Shuning uchun sun'iy intellektning rivojlanishi formallashtirilmagan vazifalarni echimini izlash tomon rivojlana boshladi. Ushbu qatorda, eng avvalo, mantiqiy o'yinlarni joriy etish (shashka, shatranj va boshqalar) vazifasi hamda teoremalarni isbotlashga kirishildi. So'ngra, robot texnika sohasidagi tadqiqotlar boshlandi - ushbu yo'nalishdagi dastlabki ishlanma murakkab releli sxema bilan boshqariladigan K.Shennonning "elektron sichqon"i edi. Ushbu sichqon labirintlarni "tadqiqot" qila olar va undan chiqish yo'lini topishi mumkin edi. bundan tashqari u o'ziga ma'lum bo'lgan labirintga joylashtirilsa, chiqishni izlab o'tirmasdan, darhol boshi berk ko'chalarga nazar tashlamasdan labirintdan chiqar edi.

A.Samuel kompyuter uchun unga shashka o'ynashga imkon beradigan dasturni yaratdi, shu asnoda mashina o'yin chog'ida o'rganadi yoki loaqal to'plangan tajriba asosida o'z o'yinini yaxshilayotganligi to'g'risidagi tasavvur yaratar edi. 1962 yilda ushbu dastur o'sha zamonda AQShning eng kuchli shashkachisi R. Nili bilan bellashdi va g'olib chiqdi.

Ushbu shashka dasturiga o'yin qoidalari dasturda shunday kiritilgan bo'lib, navbatdagi yurishni tanlash qandaydir evristik qoidalarga bo'ysundirilgan edi. O'yinning har bir bosqichida mashina shashkada (xuddi shunday shatranjda ham) odatda o'z donachalarini yo'qotish manfaatli bo'lmagan va aksincha raqibning donasini qo'lga kiritish manfaatli bo'lgan o'yin sifati mezonlariga bog'liq imkoniy yurishlar ko'pligidan navbatdagi yurishni tanlar edi. O'yinchi (u xoh inson, yoki mashina bo'lsin) o'z donachalarining harakatchanligini va yurishni tanlash huquqini saqlab qoladi va ayni paytda taxtadagi maydonlarning katta sonini jang bilan ushlab turadi va o'yinning ushbu unsurlariga ahamiyat bermaydigan o'z raqibidan odatda yaxshi o'ynaydi. Yaxshi o'yinning tasivrlangan mezonlari o'z kuchini butun o'yin davomida saqlab turadi, biroq uning ayrim bosqichlariga mansub bo'lgan boshqa mezonlar ham mavjud.

Bunday mezonlarni oqilona birlashtirgan holda (masalan, tajribada tanlanadigan koeffisientlar yoki yanada murakkab chiziqli kombinasiya ko'rinishida) mashinaning navbatdagi yurishini baholash uchun samaradorlikning qandaydir son ko'rsatkichini – baholash funksiyasini qo'lga kiritish mumkin. Unda mashina navbatdagi yurishlar samaradorlik ko'rsatkichlarini o'zaro taqqolagan holda eng katta ko'rsatkichga mos keluvchi yurishni tanlaydi. Navbatdagi yurishni tanlashni bunday avtomatlashtirish oqilona tanlovni, albatta tanlashi shart emas, biroq bu baribir qandaydir tanlashdir va uning asosida mashina o'tmish tajribadagi o'qish jarayonidagi o'z strategiyasini (harakat tarzini) takomillashtirgan holda o'yinni davom ettirishi mumkin. Rasman o'qish amalga oshirilgan yurishlar va o'yinlarning yakunini hisobga olgan holda tahlil asosida baholash funksiyasining parametrlarini (koeffisientlarini) yaratishdan iborat, xolos.

A.Samuelning fikriga ko'ra, o'qishning ushbu ko'rinishdan foydalanadigan mashina nisabatan qisqa davr ichida o'rtacha o'yinchidan yaxshiroq o'ynashga o'rganib olishi mumkin.

Shaxmat o‘yini jarayonida mashina namoyish etgan intellektning barcha ushbu unsurlari unga dastur muallifi tomonidan kiritilgan deb aytish mumkin. Qisman shunday. Biroq ushbu dastur barcha tafsilotlari oldindan o‘ynalgan “qat'iy” dastur emasligini unutmaslik lozim. U mustaqil o‘qish jarayonida o‘z o‘yin startegiyasini takomillashtirmoqda. Garchi mashinadigi tafakkur jarayoni shashka o‘ynayotgan inson miyasida yuz beradigan holatdan sezilaril darajada farqlansada, mashina insonni yutib qo‘yishi mumkin.

Shu tarzda shatranj yoki qandaydir boshqa mantiqiy o‘yinlarni o‘ynash yuz beradi. Bugunning o‘zidayoq kompyuterlar va ular uchun dasturlar shu qadar kuchayib ketdiki, har qanday insonni yutib chiqishi mumkin bo‘lib qoldi: shatranj bo‘yicha jahon chempionlaridan birini mag‘lub etgan Deep Blue dasturi har biri 4 Gb diskdagi 128 Mb tezkor xotiraga ega bo‘lgan 256 prosessorli kompyuterda ishlagan. Barcha ushbu majmua bir soniyada 100 000 000 dan ko‘proq yurishlarni hisoblab chiqishi mumkin edi. Yaqin zamonlargacha bir soniyada yaxlit sonli amaliyotlarning bunday miqdorini hisoblab chiqa oladigan kompyuter kam bo‘lgan, bu yerda esa to‘planishi lozim bo‘lgan, hamda ular uchun baholash funksiyasi hisobalab chiqilgan yurishlar to‘g‘risida so‘z bormoqda.

Hozigi vaqtda kompyuterlarga katta amaliy ahamiyatga ega bo‘lgan ishdagi yoki harbiy o‘yinlarni o‘ynashga imkon beradigan dasturlar mavjud va muvaffaqiyatli qo‘llanilmoqda. Bu yerda dasturlarga isnonga xos bo‘lgan o‘qish va moslashish salohiyatini baxsh etish o‘ta muhimdir.ulkan amaliy ahamiyatga ega bo‘lgan eng qiziqarli intellektual vazifalardan biri – qiyofalar va vazifalarni tanib olish vazifasidir. Bu vazifani hal qilish bilan turli fanlar vakillari — fiziologlar, psixologlar, matematiklar, muhandislar shug‘ullanib kelgan va shug‘ullanishda davom etmoqdalar. Ushbu vazifaga bunday qiziqish nazariy tadqiqotlar natijalaridan keng amalda foydalanishning fantastik istiqbollari: o‘qiydigan avtomatlar, tibbiy tashxis qo‘yadigan, jinoyatshunosik ekspertizasini o‘tkazadigan va hokazo sun'iy intellekt tizimlari, shuningdek murakkab sensorli vaziyatlarni tanib olish va tahlil qilishga qodir bo‘lgan robotlarni yaratish imkoni mavjudligidan yanada kuchaygan.

1957 yilda amerikalik fiziolog F. Rozenblatt nazar bilan idrok etish va tanib olish modeli — perseptronni taklif etdi. Tushunchalarga o‘rganishga va ko‘rsatilgan ob'ektlarni tanishga qodir bo‘lgan qurilmaning yaratilishi nafaqat fiziologlar, balki bilimning boshqa sohalari vakillariga ham o‘ta qiziqarli bo‘lib chiqdi va nazariy hamda eksperemental tadqiqotlarning katta oqimini yuzaga keltirdi.

Perseptron yoki tanib olish jarayoniga taqlid qiluvchi har qanday dastur ikki tartibda: o‘qish tartibida va tanib olish tartibotida ishlaydi. O‘qish tartibotida o‘quvchi rolini o‘ynovchi kimdir (inson, mashina, robot yoki tabiat) mashinaga ob'ektlarni namoyon etadi va ularning har biri borasida ular qaysi toifaga mansubligini etkazadi, ushbu ma'lumotlar bo‘yicha mohiyatiga ko‘ra tushunchalarning formal tasviri bo‘lgan hal qiluvchi qoida barpo etaladi. Tanib olish tartibida mashinaga yangi ob'ektlar (umuman aytganda, oldingi ko‘rsatilganlardan farq qiluvchi ob'ektlar) ko‘rsatiladi va mashina imkon qadar ularni to‘g‘ri tasniflashi lozim. O‘qish jarayonida qurilmaning (dasturning) koeffisientlarini sozlash yuz beradi, shu asnoda ayrim o‘quvchi tizimlar uchun qanday tarzda va qanday sabablarga ko‘ra koeffisientlar aynan shunday tarzda shakllanganligini tushunish imkoniyati yo‘q. Bu o‘quvchi tizimlarni insonga va uning ongiga kundan kunga ko‘proq o‘xshatmoqda.

Tanib olishga o‘qish muammosi boshqa bir intellektual vazifa bir tildan boshqa tilga tarjima qilish muammosi, shuningdek kompyuterni tabiiy tilga o‘rgatish muammosidir. Asosiy grammatik qoidalarni etarli darajada hamda lug‘atdan to‘g‘ri foydalanish usullarini formal ishlov berish va tasniflashda tarjima uchun aytaylik ilmiy yoki ish matnini tarjima qilish uchun butkul qoniqarli algoritmni yaratish mumkin. Ayrim tillar uchun bunday tizimlar 60 yillarning oxiridayoq yaratilgan, ammo ancha katta suhbat matnini ravon tarjima qilish uchun uning ma'nosini tushunish zarur. Bunday dasturlar ustidagi ish allaqachondan beri olib borilmoqda, biroq to‘la muvaffaqiyatga xali uzoqdir. Shuningdek cheklanmagan tabiiy tilda inson va mashina o‘rtasidagi muloqotni ta'minlovchi dasturlar ham mavjud.

Mantiqiy tafakkurni modellashtirishga kelganda esa, bu yerda teoremalarni isbotlashni avtomatlashtirish vazifasi yaxshi modellashtiruvchi vazifa bo‘lib xizmat qilish mumkin.

1960 yillardan boshlab birinchi tartibdagi predikatlarni hisoblab chiqishda teoremalar isbotini topishga qodir bo‘lgan bir qator dasturlar ishlab chiqilgan.

Ushbu dasturlar Lisp funksional tilini yaratuvchisi J.MakKarti so‘zlariga ko‘ra “sog‘lom fikrga”, ya'ni deduktiv xulosalar qilish salohiyatiga ega.

Robottexnika sun'iy intellekt tizimlarini ishlab chiqishdagi o‘ta katta yo‘nalishdir. Robot intellektining universal hisoblab chiqish mashinalari intellektidan asosiy farqi nimada? Ushbu savolga javob berish uchun buyuk rus fiziologi I.M.Sechenovga tegishli bo‘lgan mulohazani eslatib o‘tish o‘rinlidir: “Miya faoliyatining tashqi ko‘rinishlaridagi barcha cheksiz turli tumanlik oxir-oqibatda faqat bir hodisaga – mushak harakatiga borib taqaladi...”. boshqacha so‘zlar bilan aytagnda insonning butun intellektual faoliyati oxir-oqibatda harakatlar vositasida tashqi olam bilan faol o‘zaro harakatga yo‘naltirilgandir. Xuddi shuningdek, robot intellekti unsurlari, eng avvalo, uning maqsadga yo‘nalitrilgan harakatlarini shakllantirishga xizmat qiladi. ayni vaqtda sun'iy intellektning sof kompyuterli tizimlarining asosiy vazifasi, odatda his etishning sun'iy a'zolari yordamida atrof-muhitni idrok etish va ijro mexanizmlari harakatini tashkil qilish bilan bog‘liq bo‘lmagan abstrakt yoki yordamchi tusga ega bo‘lgan intellekt ual vazifalarni hal qilishdan iboratdir.

12.1.2. Sun'iy intellekt tizimlarini barpo etishga turli yondoshuvlar

Sun'iy intellekt tizimlarini barpo etishga turli yondoshuvlar mavjud. bunday farqlanish bir fikrni o‘rniga asta-sekin boshqasi kelishi bois tarixiy emas. Chunki hozir ham turlicha yondoshuvlar mavjud. Bundan tashqari hozirgi vaqtda sun'iy intellektning haqiqiy to‘liq va asl tizimlari yo‘qligi bois qandaydir yondoshuvni to‘g‘ri yoki xato deb aytib bo‘lmaydi. Hozirgi vaqtda quyida qisqacha ko‘rib chiqadigan quyidagi yondoshuvlar ajratib ko‘rsatilmoqda:

1) mantiqiy yondoshuv;
2) tuzilmali yondoshuv;
3) evolyusion yondoshuv;
4) imitasion yondoshuv.

Sun'iy intellekt tizimlarini ishlab chiqishdagi birinchi yondoshuv - mantiqiy yondoshuvdir. Bu yondoshuv shuning uchun ham paydo bo'lganki, aynan mantiqiy fikrlash salohiyati insonni ongsiz mavjudotlardan etarli darajada kuchli farqlab turadi. Bulev algebrasi mantiqiy yondoshuv uchun asosdir. Har bir dasturchiga u shartli konstruksiya if-then-else ni o'zlashtirgan zamondan beri ushbu formalizm bilan tanishdir. Bulev algebrasi birinchi tartibdagi predikatlarni hisoblab chiqish ko'rinishida yanada rivojlandi. Bunda u ashyoli timsollarni kiritish, ular o'rtasidagi munosabatlar, mavjudlik va umumiylik kvantorlari hisobiga kengaytrildi. Mantiqiy tamoyilda barpo tetilgan sun'iy intellektning deyarli har bir tizimi ozmi-ko'pmi teoremalar isbotining universal mashinasidir. Shu asnoda boshlang'ich ma'lumotlar bilimlar bazasida aksiomalar ko'rinishida, mantiqiy xulosa qoidalari esa, ular o'rtasidagi munosabatlar sifatida saqlanadi. Bundan tashqari bunday har bir mashina maqsadlarni jamlash blokiga ega va chiqarish tizimi muayyan maqsadni teorema sifatida isbotlashga urinadi. Agar maqsad isbotlansa, o'zgaruvchan qoidalarni yo'lga solish belgilangan maqsadni amalga oshirish uchun zarur bo'lgan harakatlar silsilasini qo'lga kirtishga imkon beradi. Bunday tizimning quvvati maqsadlar generatorining imkoniyatlari va teoremalar isboti mashinasi bilan belgilanadi.

Tabiiyki, mantiqiy yondoshuv hisoblab chiqishning ikki yoqlama tizimiga asoslangan – bu butkul mantiqiydir, chunki bulev algebrasida ham zamonaviy kompyuterlar arxitekturasida ham aynan ikki yoqlama mantiqdan foydalaniladi. Ammo sun'iy intellektning butun kuchini faqat ikki timsol — 0 yoki 1^1 orqali ifodalanishini faraz qilish g'alatidir. Katta ifodalilikka erishish mantiqiy yondoshuvga noaniq mantiq sifatidagi nisbatan yangi yo'nalishga imkon beradi. Bunday mantiq ushbu bobning 7.2 qismida qisqa tasvirlangan. Uning asosiy farqi shuki, undagi o'zgaruvchan haqiqiyliklar qiymatlarning cheksiz miqdorini qabul qilishi mumkin. ushbu yondoshuv inson tafakkuriga ko'proq o'xshaydi, chunki u savollarga kamdan kam to'liq aniqlikda javob beradi.

Ifodalash xuddi bu har qanday axborot tilga olingan timsollar yordamida fon Neymanning an'anaviy arxitekturasi vositalarida kodlashtiriladigan axborotlashtirish va amaliy dasturlashda amalga oshirilgani singari ifodalash albatta, mumkin, ammo bu yerda gap ko'proq qanday yozib olish haqida emas, balki fanning u yoki bu yo'nalishida

qanday aksiomatika va chiqarish qoidalaridan foydalanish to‘g‘risida bormoqda.

Aksariyat mantiqiy usullar katta mehnat talab qiladi. Chunki isbotni izlash vaqtida ko‘rinishlarning to‘liq ortiqchaligi bo‘lishi mumkin. Shuning uchun ushbu yondoshuv hisoblab chiqish jarayonini samarali amalga oshirishni taqozo etadi. Yaxshi ish esa odatda, bilimlar bazasining nisbatan katta bo‘lmagan hajmi bilan kafolatlanadi.

Tuzilmali yondoshuv sun'iy intellekt oqimida rivojlanayotgan ikkinchi yondoshuv bo‘lib, buning ostida inson miyasini tuzilmasini modellashtirish yordamida sun'iy intellekt tizimlarini barpo etishga urinishlar tushuniladi. Ilgari tilga olib o‘tilgan F.Rozenblatning perseptroni ana shunday birinchi urinishlardan edi. Perseptronlardagi asosiy modellashtiriladigan tuzilmaviy birikma (xuddi shuningdek, miyani modellashtirishning aksariyat boshqa ko‘rinishlarida) asab to‘qimasi — neyrondir.

Keyinchalik ko‘plab boshqa modellar paydo bo‘ldi, bular odatda “neyronli tarmoqlar” (“neyrotarmoqlar”) deb ataladi. Ushbu modellar alohida neyronlar tuzilishi bo‘yicha, ular o‘rtasidagi aloqalar makoni bo‘yicha va o‘qish algoritmi bo‘yicha farqlanadi. Bugungi kundagi neyronli tarmoqlar turlari orasidagi eng mashhurlaridan biri sifatida xatoni qayta tarqatishga ega neyrotarmoqni, Xopfild tarmog‘ini va stoxastik (ehtimoliy) neyronli tarmoqlarni tilga olish mumkin.

Neyrotarmoqlar namunalarini tanib olish vazifalarida, shu jumladan kuchli shovqinlangan namunalarni tanlab olish vazifalarida muvaffaqiyatli qo‘llanilmoqda, ammo bulardan sun'iy intellekt tizimlarini o‘zini, shu jumladan robototexnikani barpo etish uchun qo‘llash misollari ham mavjud.

Neyrotarmoqli modellar uchun uncha katta bo‘lmagan ifodalilik, algoritmlarni oson parallellikdan chiqarish hamda shunga bog‘liq parallel amalga oshirilgan neyronli tarmoqlarning yuksak unumdorligi xosdir. Shuningdek bunday tarmoqlar uchun ularni inson miyasiga juda yaqinlashtiradigan bir xossa tegishlidir – neyronli tarmoqlar xatto kirish axboroti to‘liq bo‘lmagan sharoitda ham ishlaydi.

Evolyusion model ham ancha keng tarqalgan. Sun'iy intellekt tizimlarini barpo etishda boshlang‘ich modelni barpo etishga va u o‘zgaradigan (evolyusiya qilandigan) qoidalarga asosiy e'tibor qaratiladi.

Shu asnoda model o‘ta turli usullar bo‘yicha tuzilishi mumkin, bu neyrotarmoq ham, mantiqiy qoidalar to‘plami ham va boshqa har qanday model bo‘lishi mumkin. shundan so‘ng evolyusion algoritm yaratiladi, u modellarni tekshirish asosida bulardan eng yaxshilarini tanlab oladi, bular vositasida muayyan qoidalar bo‘yicha yangi modelllar to‘planadi, bulardan esa yana eng yaxshilar tanlab olinadi va xokazo.

Asl ma'nodagi evolyusion modellar yo‘qligini, faqat boshqa yondoshuvlar doiralaridagi o‘qishning evolyusion algoritmlari mavjudligini aytib o‘tish mumkin. Ammo evolyusion yondoshuvda olingan modellar ayrim o‘ziga xosliklarga ega

Bu ularni alohida toifaga ajratishga imkon beradi. Masalan, shunday o‘ziga xosliklardan biri ishlab chiqaruvchining asosiy ishini modelni barpo etishdan uni modefikasiyalash algoritmiga ko‘chirib o‘tkazish mumkinligi va olingan modellar amalda sun'iy intellekt tizimini qamrab olgan muhit to‘g‘risidagi yangi bilimlarni chiqarib olishga hamrohik qilmasligidir.

Va nihoyat, sun'iy intellekt tizimlarini barpo etishda keng qo‘llaniladigan yondoshuvlardan biri — imitasion yondoshuvdir. Mazkur yondoshuv kibernetika uchun uning asosiy tushunchalaridan biri – “qora quti” tushunchasiga ega bo‘lgan mumtoz yondoshuvdir, ya'ni ma'lumotlarning qurilmasi, dasturiy modeli yoki to‘plami bo‘lib, bularning ichki tuzilmasi va mazmuni to‘g‘risidagi axborot butkul mavjud emas, biroq kirish va chiqish ma'lumotlarining o‘ziga xosliklari ma'lumdir. Xatti-harakati taqlid qilinayotgan ob'ekt aynan shunday “qora quti”dan iborat. Unda va model ichida nimaligi va u qanday amal qilishi butkul muhim emas, asosiysi, model o‘xshash vaziyatlarda xuddi modellashtirilayotgan ob'ekt tarzida harakat qilishidir.

Amalda sun'iy intellekt tizimlarini barpo etishga tasvirlangan yondoshuvlar o‘rtasida aniq chegara mavjud emas. Juda ko‘p holatlarda ishning bir qismi bir turga ko‘ra, boshqa qismi esa boshqa turga ko‘ra bajariladigan aralash tizimlar uchraydi.

12.1.3. Sun'iy intellektda deklarativ dasturlashning o‘rni

Dasturlashning deklarativ paradigmalari doiralarida olimlar taklif etayotgan uslubiyatlar sun'iy intellekt bo‘yicha mutaxassislarning qalbida keng qo‘llab-quvvatlanayotganligini faraz qilish tabiiydir. Bu shuning uchun ham yuz berdiki, deklarativ dasturlash olimga ma'lumotlarni

kodlash usullar va kodlashning bunday usuliga yanada ishlov berishni o‘ylab o‘tirmasdan hisoblab chiqish jarayonlarini abstraksiyani yanada yuksak darajasida tasvirlashga imkon beradi hamda tadqiq qilinayotgan jarayonni ushbu ko‘rib chiqilayotgan jarayonning muammoli sohaning iboralarida bevosita tasvirlashni taklif etadi (tabiiyki bunday tasvirlash dasturlash tili sintaksisi doiralarida bo‘lishi lozim).

Masalan, tuzilmali, evolyusion va imitasion yondoshuvlar doiralarida sun'iy intellekt tizimlarini ishlab chiqarishni dasturlashning funksional tillarida muammosiz amalga oshirish mumkin. shuningdek mantiqiy yondoshuv ham garchi buning uchun alohida paradigma – mantiqiy dasturlash yaratilgan bo‘lsada, funksional dasturlash uslubiyatiga u qadar yomon bo‘lmagan darajada ko‘chirilishi mumkin.

Sun'iy intellekt tizimlaridagi zarur bo‘lgan har qanday formalizmlar va modellarni uncha katta bo‘lmagan zahira harajatlari bilan dasturlashning deklarativ tillarida amalga oshirish mumkin. Insoniy neyron modeli muammosiz hal bo‘ladi. Genetik algoritmalrni joriy etish (evolyusion yondoshuvlardan biri) —uchun ikki soat ishlash etarlidir. Tizim bilan muloqot uchun cheklangan tabiiy til uchun parser boshlang‘ichning kodning bir juft o‘nlik kilobayti etarlidir.

Deklarativ dasturlash va sun'iy intellektning birgalikda rivojlanishini to‘xtatib turgan yagona narsa – tizimni barpo etish uchun jiddiy apparat – texnik ta'minotning yo‘qligidir. Axborotlashtirish va kompyuter fanini rivojlantirishning ikki yo‘nalishi ilgari umuman mavjud bo‘lmagan yoki o‘ta qimmat bo‘lgan etarli darajada quvvatga ega hisoblab chiqish zahiralarini taqozo etadi. shuning uchun ko‘plab ilmiy laboratoriyalar ishlab chiqishning kamroq zahira sarflanadigan tizimlaridagi sun'iy intellekt sohasidagi tadqiqotlar bilan shug‘ullangan.

Ammo bugungi kunda kompyuterlar kundan kunga qudratli bo‘lib borayotgan davrda jahonning barcha mamlakatlaridagi sun'iy intellekt o‘yicha tadqiqotlar olib borilayotgan aksariyat ilmiy laboratoriyalar funksional tillarga, shu jumladan Haskell tiliga o‘tmoqda. Bu til uchun kundan kunga ko‘proq kutubxonalar va kengaytirishlar yaratilmoqda. Bularning ayrimlari sun'iy intellektning muayyan vazifalarini hal qilish uchsun mo‘ljallangandir. Shuning uchun ham ushbu tilni ham funksional paradigmani o‘rganish, umuman olganda, o‘z vaqtidagi va butkul istiqbolli ishdir.

Quyida yanada jiddiryroq dasturiy ta'minotni yaratishga yordam berishi mumkin bo‘lgan sun'iy intellekt tizimlarining ayrim amaliy jihatlari qisqa ko‘rib chiqiladi.

12.2. Bilimlardagi mantiqiy xulosa

Sun'iy intellekt doiralaridagi tadqiqotning eng muhim yo‘nalishlaridan biri bilimlarga ishlov berishdir, chunki ushbu yo‘nalish sun'iy intellektning ko‘plab amaliy vazifalarida qo‘llanilmoqda. bilimlarga ishlov berish, birinchi navbatda bilimlardagi xulosa, ekspertli, muloqotli, intellektual, ko‘p agentli tizimlar singari sun'iy intellekt tizimlarining ish layoqatini va yuksak amaliy ahamiyaini ta'minlaydi. Xuddi shuningdek, sun'iy intellektning ozmi ko‘pmi har qanday murakkab tizimida tizimning muammoli sohasi to‘g‘risida muayyan xulosalar olish uchun bilimlarga ishlov berishdan foydalanadi.

Shuning uchun bilimlarni taqdim etish va ishlov berish bilan bog‘liq masalalarni ko‘rib chiqish sun'iy intellektni o‘rganishda muhimdir.

12.2.1. Ma'lumotlar va bilimlar

Bilimlar — bu sun'iy intellekt tizimlari texnologiyasining eng muhim tushunchalaridan biridir. Muammoni o‘rganish yillarida mutaxassislar tomonidan ushbu tushunchani “ma'lumotlar” tushunchasi bilan taqqoslashga imkon beruvchi qator o‘ziga xos alomatlar orqali turli talqinlar ko‘pligi taklif etildi. Bilimlarning bunday alomatlari (turlari) quyida sanab o‘tilgan.

1) Inson xotirasidagi bilimlar.

2) Moddiylashtirilgan bilimlar (darsliklar, ma'lumotnomalar)

3) Bilimlar maydoni (oldingi ikki turning tuzilmalangan, yarim formallashtirilgan ta'rifi).

4) Bilimlarni taqdim etish tillaridagi bilimlar (bilimlar maydonini formallashtirish).

5) Kompyuter uchun bilimlar bazasi (axborotni mashinali olib yuruvchilarda).

Қуйида билимлар учун сифат хоссалари мажмуи, яъни “билимлар” иборасининг ўзини белгилашга ва тасвирлашга имкон берувчи билимларнинг ўзига хос аломатлари келтирилади:

1) Bilimlar ma'lumotlarga nisbatan ancha murakkab tuzilmaga ega;

2) Bilimlar ham ekstensional (ya'ni, ko‘rib chiqilayotgan tushunchaga mos keluvchi aniq holatlar to‘plami orqali), ham intensional (ya'ni, ko‘rib chiqilayotgan tushunchaga mos keluvchi xossalar orqali) beriladi, ma'lumotlar doimo ekstensional beriladi;

3) Bilimlarning ichki talqinliligi — xotirada ma'lumotlar unsuri bilan birgalikda nomlar “ortiqcha” tizimini saqlash imkoniyati mavjudligi;

4) Bilimlarning rekursiv tuzilmalanganligi – “matreshka” tamoyili bo‘yicha ajralish va birlashish imkoniyatining mavjudligi;

5) Bilimlar birliklarining o‘zaro aloqasi (bog‘liqligi) — alohida hodisalar va holatlar aloqasining semantikasi va progmatikasini aks ettiruvchi turli munosabatlarning, shuningdek umuman tizim ma'nosini aks ettiruvchi munosabatlarni aniqlash imkoniyati mavjudligi;

6) Bilimlarda metrikaga ega semantik makonning — axborot birliklarining bir-biridan yaqinligi/uzoqligini aniqlash imkoniyati mavjudligi;

7) Bilimlarning faolligi — xatti-harakat sababalarini shakllantirish, maqsad qo‘yish, ularni hal qilish tartibotini barpo etish imkoniyati mavjudligi;

8) Bilimlarning funksional yaxlitligi — istalayotgan natijani tanlash, natijani olish vaqti va vositalarini, olingan natija etarligini tahlil qilish vositalarini tanlash imkoniyati.

Bilimlar bilan ishlovchi nazariyalar doirasida sun'iy intellekt tizimlari va ulani barpo etish bilan bevosita bog‘liq bo‘lgan uch jdarayon ajratib ko‘rsatiladi. Bu jarayonlar bilimlarni olish, taqdim etish va ishlov berish mohiyatidir.

Bilimlarni olish deyilganda bilimlarni ekspertdan yoki qandaydir boshqa manbalardan olish va ushbu bilimlardan kelajakda bilimlarga asoslangan tizimlarda foydalanish uchun ushbu bilimlarda formallashtirish jarayoni tushuniladi. Bilimlarni olish bilimlarga ega bo‘lish jarayoni ko‘rinishlaridan biridir. Shu asnoda bilimlarga ega bo‘lishning o‘zi ostida bilimlar manbalaridan bilimlar bo‘yicha muhandis va ekspert faoliyatini qo‘llab quvvatlashning dasturiy vositalaridan foydalanish yordamida bilimlarni olish tushuniladi.

Bilimlarni olishning boshqa ikki ko‘rinishi bilimlarni chiqarib olish (ekspertlardan yoki bilimlarning boshqa manbalaridan ushbu jarayonni qo‘llab-quvvatlashning kompyuterli vositalaridan foydalanmasdan balki,

bilimlar bo'yicha muhandisning va bilimlar manbaining bevosita muloqoti yo'li bilan bilimlarni olish) va bilimlarni shakllantirish (manbalardan bilimlarni ko'rib chiqilayotgan fan sohasidagi qarorlar qabul qilish misollarining reprezentativ tanlanmasi mavjud bo'lganda o'qitish dasturidan foydalanish yordamida olish) dir.

Mazkur kitobda ushbu jarayon ko'rib chiqilmaydi. Shu asnoda muammoli soha to'g'risidagi bilim qandaydir tarzda sun'iy intellekt tizimiga kiritilganligi faraz qilinadi.

Bilimlarni taqdim etish — bu ikki: "Nimani taqdim etish?" va "Qanday taqdim etish?" savoliga javoblarni amalga oshiruvchi jarayondir. Birinchi jarayon — bu bilimlar tarkibini aniqlash masalasidir, buning muhimligi aynan shu muammoning echimi modellashtrilayotgan muammoli sohani munosib aks ettirishini ta'minlashi bilan belgilanadi. Ikkinchi savol o'z navbatida kattagina darajada mustaqil bo'lgan ikki vazifaga: bilimlarni qanday tashkil etish (tuzilmalash) va tanlagan formalizmda bilimlarni qanday taqdim etish vazifalariga bo'linadi.

Bilimlarni taqdim etishning ikki asosiy masalasi bir-biriga tobe' emasligini ta'kidlash zarur. Darhaqiqat taqdim etishning tanlangan formalizmi umuman yaroqsiz bo'lib chiqishi yoki qandaydir muammoli sohalarda bilimlarni ifodalash uchun samarasiz bo'lib chiqishi mumkin.

Bilimlarni taqdim etish uchun formalizmlar sifatida turli uslubiyatlar qo'llaniladi. birinchi navbatda bu produksion model, freymlar va semantik tarmoqlardir. Eng sodda va ayni vaqtda katta ifodalovchi salohiyatga ega bo'lgan ma'lumotlarni taqdim etishning produksion modelidir va u ushbu qismda ko'rib chiqiladi.

Zamonaviy tushunchadagi mahsulot (produksiya) iborasi — bu bilimlarni quyidagi eng oddiy ko'rinishda tasavvur qilsh usulidir.

$$(i): Q;\ \mathrm{P};\ \mathrm{C};\ \mathrm{A} \Rightarrow \mathrm{B};\ N, \qquad (7.14)$$

bu erda

1) i — mahsulotning o'z nomi (belgisi);
2) Q — fan sohasidan uning qandaydir qismini ajratib oluvchi mahsulotni qo'llash sohasi. Unudagi mahsulotga biriktirilgan bilimlar ma'noga ega. Mahsulotni qo'llash ma'nosi;
3) P — boshqaruv strategiyalarida bajarish uchun ushbu mahsulot tanlovi uchun xulosalarda foydalanadigan muayyan mahsulotning

haqiqiyligi, uning ustuvorligi va xokazolar to‘g‘risida axborotga ega dastlabki shart;

4) C — predikatdan iborat bo‘lgan shart bo‘lib, uning asl ma'nosi qandaydir qadamda ushbu mahsulotdan foydalanishga ruxsat beradi;
5) A => B — mahsulot negizi. Mahsulot negizini talqin qilish turlicha bo‘lishi mumkin, masalan: “Agar A haqiqiy bo‘lsa, V haqiqiydir”, “Agar A joriy vaziyat bo‘lsa, V ni qilish zarur” va h.k;
6) N — mahsulotning joriy mahsulot bajarilishganidan so‘ng mahsulotlar tizimiga kiruvchi ushbu mahsulotga yoki boshqa mahsulotlarga qanday o‘zagritirishlar kiritish zarurligi to‘g‘risidagi axborotga ega bo‘lgan mahsulotning keyingi sharti.

Bilimlarni mahsulotli taqdim etish afzalliklariga quyidagilar kiradi.

1. Modullilik — har qanday mahsulot mahsulot tizimining har qanday joyiga joylashtirilishi mumkin. Chunki mahsulot tizimidagi bilmmlarni tashkil etish tabiiy modullilikka ega. Har bir mahsulot — bu fan sohasi to‘g‘risidagi bilimlarning tugallangan bo‘lagi bo‘lganligi uchun mahsulotlarning barcha ko‘pligini ayrim ob'ekt tasviriga mos keluvchi kichik ko‘plikliklarga bo‘lish mumkin.

2. Tuzilmaning bir qiyofaliligi — mahsulot tizimining asosiy bo‘g‘inlari turli muammoli yo‘nalishga ega bo‘lgan sun'iy intellekt tizimlarini barpo etish uchun qo‘llanilishi mumkin.

3. Mahsulotli tizimlarga xos bo‘lgan deklarativlik, faqatgina axborotni o‘zgartirish dasturlarini barpo etishgagina emas, balki mavzu sohasini tasvirlashga imkon beradi. Bundan tashqari, chiqishni boshqarish va chiqishning o‘zi o‘rnatilgan mexanizmdan foydalanish bilan amalga oshiriladi.

4. Tabiiylik — mahsulot tizimidagi xulosaning chiqishi ko‘p jihatdan insonning mulohaza yuritish jarayoniga o‘xshashdir.

5. Mahsulotlarning mustaqilligi mahsulot tizimlarini parallel kompyuterlar joriy etish uchun xususan, mahsuldorlik qoidalariga mo‘ljallangan ixtisoslashtirilgan kompyuterlarni ishlab chiqish uchun o‘ta istqbolliga aylantiradi.

6. Tushunchalarning tur ko‘rinishli ierarxiyasi moslashuvchanligi, bu faqat qoidalar o‘rtasidagi aloqa sifatida saqlab turiladi (qoidalarning o‘zgarishi ierarxiyada ham o‘zgarishni keltirib chiqaradi).

7. Reaktivlik — ma'lumotlar o'zgarishiga darhol munosabat.

8. Tushunarlilik — mahsulotlar insonga xissiy tushunarli bo'lgan ancha yirik birliklardir.

9. Kengayuvchanlik — mahsulotlar bilimlar bazasiga qo'shilishi yoki bilimlar bazasi tuzilmasini o'zgartirmasdan uzoq vaqt davomida o'zgartirilishi mumkin. Kengayuchanlik modullilik va deklarativlik natijasidir.

12.2.2. Bilimlardan xulosa chiqarish

Bilimlardan xulosa chiqarish — bu bilimlarga ishlov berish hamda bilimlar va ma'lum holatlar bazasi asosida qandaydir natijani qo'lga kiritish jarayonidir. Bilimlardan xulosa chiqarish sun'iy intellekt tizimlarini yaratishning eng muvaffaqiyatli texnologiyalaridan biridir, chunki insoniy mulohazalarni qandaydir darajada modellashtirishga imkon beradi. Bu o'z ortidan boshlang'ich bazaga kiritilmagan natijani olish imkoniyati sifatidagi muhim xosani keltirib chiqaradi. Aynan shu bilan bilimlardan xulosa chiqarish oddiy dasturlashdan farqlanadi.

Buning ustiga, agar xulosa chiqarish jarayonida olingan natijalar asosida bilimlar bazasini modifikasiya qilish uchun andaydir evristik qoidalar qo'llanilsa, o'qishga ega bo'lgan tizimni qo'lga kiritish mumkin, bu bilimlardan xulosa chiqarish tizimlarini sun'iy intellekt sohasidagi tadqiqotchilar uchun yanada fqiziqarliga aylantiradi.

Bilimlardan xulosa chiqarish texnologiyasi etuklikning bir necha bosqichidan o'tgan holda rivojlanib va kengayib borgan avval boshdanoq, ushbu g'oya endigina olimlarning miyasida paydo bo'lganda xulosa sodda va oddiy edi. Mahsulotlar "agar..." " unda ..." oddiy qoidalardan iborat bo'lgan. Bu ishonchli xulosa deb ataluvchi xulosa edi. Chunki xulosa jarayonida olinadigan natijalar to'liq ishonchlilikka ega edi. Biroq buning qiymati ham ancha yuqori bo'lgan — mahsulotlar o'ta cheklangan ko'rinishga ega edi. Qiziqarli, muammoli sohani ozmi ko'pmi tarzda maqbul tasvirlash uchun esa minglab va o'n minglab qoidalarni yaratish zarur edi.

Navbatdagi qadam isholnchsiz xulosani, yoki ishonchsizlik koeffisientiga ega bo'lgan xulosani ishlab chiqish bo'lgan. Mazkur holatda har bir holatga muayyan koeffisientlar berilgan, bular muammoli sohaga bog'liq holda u yoki bu tarzda talqin qilingan. Bu esa o'xshash qoidalarni

bunday qoidalarining haqiqiyligiga ishonchni muayyan tarzda taqdim etgan holda guruhlarga birlashtirishga imkon berdi, bu esa o‘z navbatida natija chiqarish jarayonini nozik sozlash imkonini berdi.

Vanihoyat, bilimlarda mashina xulosasini chiqarishni rivojlantirish yo‘nalishidagi uchinchi qadam mahsulotlardagi holatlarni tasvirlash uchun noaniq matematika formalizmlaridan foydalanishlardan iborat edi, bu ma'lumotlar bazasidagi qoidalar sonini o‘nlab va yuzlab marotaba qisqartirishga, xulosaning o‘zini esa, o‘ta moslashuvchan qilishga imkon bergan. Aynan, noaniq xulosa ma'lumotlar bazasida yozilmagan yangi natijaga erishishning ulkan salohiyatlariga ega – bu mahsulotlarda lingivistik va noaniq o‘zgaruvchan qiymatlardan foydalanish oqibatidir. Quyida bilimlardan xulosa chiqarish aynan shu jihatdan noaniq matematikadan foydalanish bilan ko‘rib chiqiladi.

Sun'iy intellekt tizimlarining asosiy bo‘g‘ini xulosa chiqarish mashinasi yoki hal qiluvchi bo‘lib, shuning o‘zi bilimlardagi mashinali xulosa chiqarishni amalga oshiradi, shaklan hal qiluvchi to‘rtlik ko‘rinishida taqdim etilishi mumkin:

$$I = (V, S, K, W), \qquad (7.15)$$

Bu erda:

1) V — Bilimlar bazasidan hamda mahsulot tizim xotirasidan interpretator ishining navbatdagi davriyligida foydalanish mumkin bo‘lgan faol mahsulotlar kichik ko‘pligi va faol ma'lumotlar (holatlar) kichik ko‘pligi mahsulot tizimining ishchi xotirasidan tanlash jarayonidir;
2) S — anglatishning ko‘pligini, ya'ni juftliklarning ko‘pligini taqqoslash jarayonidir: p_i qoidasi — d_i ma'lumotlar, shu asnoda p_i har bir qoidasi faol qoidalarning kichik ko‘pligiga mansubdir, d_i ma'lumotlar esa V jarayonida olingan faol ma'lumotlar kichik ko‘pligining kichik ko‘pligidir;
3) K — ziddiyatlarni hal qilish jarayoni (boshqachasiga – rajalashtirish jarayoni) bo‘lib, anglatishlarning qaysi biri bajarilishini belgilaydi;
4) W —K jarayonidagi anglatish uchun tanlangan qoidani bajarishni amalga oshiruvchi jarayon. Mahsulot tizimining ishchi xotirasini modifikasiya qilish yoki kirish/chiqish amaliyoti bajarilish natijasidir.

Bilimlardagi mashinada xulosa chiqarishning o‘zi uch ko‘rinishda to‘g‘ri, qayta va aralash turda bo‘lishi mumkin. to‘g‘ri va qayta chiqishlar Modus Ponens va Modus Tollens chiqishlarining mantiqiy qoidalariga to‘liq mos keladi:

$$((x \to y) \wedge x) \to y; \qquad (7.16)$$

$$((x \to y) \wedge |y) \to |x \qquad (7.17)$$

Chiqarishning aralash strategiyasi bevosita chiqishning muayyan bosqichlarida bundan so‘ng shajara bo‘yicha ilgari surish imkoniyatsizligida turli imkoniyatlarni ko‘rib chiqish va muqobilliklarni tanlash uchun qayta chiqish qo‘llaniladigan mantiqning ikkala qoidasidan foydalanadi.

Noaniq mashinali xulosa ModusPonens va ModusTollensning salgina o‘zgartirilgan qoidalaridan foydalanadi. O‘zgartirish shundan iboratki, Modus Ponens qoidasi agar x o‘zgaruvchan qiymati butkul haqiqiy bo‘lmaganda ham ishlab ketadi. Modus Tollens qoidasi esa agar undagi parametr qiymati butkul soxta bo‘lmagan holatda ham ishlab ketadi. Bu noaniq mantiq xizmatdir. Unda olamni haqiqat va soxtalikka aniq taqsimlash yo‘q. o‘zgartirilgan qoidalarni quyidagi tarzda yozish hammasidan osonroqdir:

$$((if\ x = A\ then\ y = B) \wedge (x = A')) \to (y = B'); \qquad (7.18)$$

$$((if\ x = A\ then\ y = B) \wedge (y = B')) \to \{x = A'). \qquad (7.19)$$

Mahsulotli noaniq xulosa muammoli soha to‘g‘risidagi bilimlarni tasvirlash ekspertlar tomonidan:

$Rule_1$: Если x_1 есть A_1, то y_1 есть B_1;

$Rule_2$: Если x_2 есть A_2, то y_2 есть B_2;

. . .

ko‘rinishdagi qoidalar to‘plami ko‘rinishida shakllantirilganligini nazarda tutadi.

$Rule_n$: agar x_n A_n bo‘lsa, unda y_n B_n dir.

Qoidalarning ushbu to‘plamidagi $x_i \in X$ — kiruvchi o‘zgaruvchanlar nomlarining ko‘pligi, $y_i \in Y$ — chiqish o‘zgaruvchanlari nomlarining ko‘pligi (X va Y ko‘pligi kesishishi mumkin), A_i va B_i esa — mos tarzda X va Y ko‘pliklari uchun belgilangan noaniq o‘zgaruvchilarning nomlaridir.

Noaniq bilimlardagi mahsulotli xulosa chiqarishning umumiy uslubiyati bosqichlarining quyidagi ro'yhatiga muvofiq amalga shiriladi

1. Noaniqlikni kiritish. Fazzifikasiya.

Har bir kirish amaliy parametri (chiqarish jarayonida ishtirok etuvchi o'zgaruvchan qiymat) uchun mansublikning tegishli funksiyasi belgilanadi. Buning uchun ma'nosi joriy kirish parametri bo'lgan lingvistik o'zgaruvchanning sintaktik yoki semantik tartiboti qo'llaniladi.

2. Taqqoslash.

Chiqarishning navbatdagi qadamida ishtirok etuvchi har bir qoidani oldingi jo'natish uchun haqiqiylik qiymati hisoblab chiqiladi. U kelgusida qoida izchil xulosasiga nisbatan qo'llaniladi. Bu qoidalar konsekventidagi mansublik funksiyasining ko'rinishi bir muncha o'zgarishiga olib keladi va ko'rinishning bu o'zgarishi chiqarishning qo'llaniladigan usuliga bog'liqdir.

3. Kompozisiya.

Taqqoslash jarayonida olingan hamda chiqarishning aynan bir o'zgaruvchisiga mansub bo'lgan mansublikning barcha funksiyalari mansublikning yagona funksiyasini shakllantirish uchun birlashtiriladi. Kompozisiya usuli ham noaniq chiqarishning qo'llaniladigan usuliga bog'liqdir.

4. Toq sonni juft songa almashtirish (Defazzifikasiya).

Defazzifikasiya — bu mansublik funksiyalarini aniq qiymatlarga olib kelishdir. Ushbu tartibotdan mansublik funksiyalarining ko'pligidan iborat bo'lgan chiqarilgan qiymatlarni muammoli sohaning iboralarida talqin qilish mumkin bo'lgan aniq qiymatlarga o'zgartirish foydali bo'lgan holatda qo'llaniladi. Mazkur tartibot o'z navbatida noaniq chiqarish usuliga bog'liq emas, ammo o'zi o'zgarishi mumkin, shuning uchun noaniq chiqarish jarayonida defazzifikasiya usulini tanlash mexanizmiga ega bo'lish maqsadga muvofiqdir.

12.2.3. Mashinali xulosa to'g'risidagi ayrim yakuniy xulosalar

Mantiqiy dasturlash paradigmasi mashinali xulosani amalga oshirish uchun tabiiy mexanizm ekanligini ta'kidlash qoldi, xolos. Prolog singari dasturlash tillarining talqinlari mashinali xulosa uchun universaldir. Buning ustiga mantiqiy dasturlashda xatto dasturning o'zi ba'zan "bilimlar

bazasi" deb ataladi. Bu masalan, aynan o'sha Prolog tilidagi dastur mahsulotlar ko'pligi ekanligidan kelib chiqadi.

Funksional dasturlash paradigmasi mantiqiy dasturlashga nisbatan bir muncha umumiy paradigma sifatida ko'rib chiqish mumkin. Mantiqiy tillar interpretatorlarini funksiyalar tillariga joriy qilish o'ta oson ishdir. Masalan, HUGS 98 interpretatorini standart etkazib berishda Prolog tili interpretatorini joriy etish misoli mavjud, undagi modullarning umumiy hajmi 20 Kb ni tashkil etadi.

Buning ustiga, sof funksional tillar interpretatorlarini xulosa chiqarishning universal mashinalari sifatida ko'rib chiqish mumkin, chunki bular kirishga har bir yopiqlik bir mahsulotga mos keladigan funksiyalar ta'rifining ko'pligi beriladi. Kirish o'zgaruvchanlarining amaldagi qiymatlarini esa, hal qiluvchi kirishga beradigan holatlar sifatida tushunish mumkin. Boshqacha so'zlar bilan aytganda dasturlash funksional tilining har bir interpretatori bevosita strategiyani amalga oshiruvchi xulosa chiqaruvchi mashinasidan iboratdir. Bilimlardan xulosa chiqarish jarayoni asoslarini tushunish dasturlash funksional tillarining samarali tillarini qanday joriy etishga va tushunishga yordam berishi mumkin.

Nazorat savollari

1. "Sun'iy intellekt" nima? Sun'iy intellekt sohasida tadqiqotchilar oldiga qaysi asosiy vazifalar qo'yilmoqda?
2. Sun'iy intellekt tizimlarini barpo etishda qanday yondoshuvlar mavjud? Ushbu yondoshuvlar nima bilan xususiyatlanadi, nima bilan farqlanadi?
3. Bir tizimni yaratishda bir necha yondoshuvlardan foydalanish mumkinmi?
4. Sun'iy intellekt tizimlarini ishlab chiqish ishida dasturlashning funksional va mantiqiy tarzlari qanday o'ringa ega?
5. Noaniq mantiq nima, u an'anaviy bul mantiqidan nimasi bilan farqlanadi?
6. Nima uchun noaniq mantiq doiralarida haqiqiylikning noaniq qiymatlari ustidan asosiy amaliyotlarning ko'rinishlari cheksiz miqdorini belgilash mumkin?

Foydalanilgan adabiyotlar

1. R. A. Kowalski Algorithm = Logic + Control. CACM 22(7), 1979
2. J. A. Robinson : Editor's Introduction. Journal of Logic Programming Vol.1 1984
3. J. W. Lloyd Practical Advantages of Declarative Programming. Proceedings of the 1994 Joint Conference on Declarative Programming.
4. D.A. Turner Programs as executable specifications. In C. A. R. Hoare and J. C. Shepherdson, Mathematical Logic and Programming Languages, 1985.
5. Herbrand J. Une methode de demonstration. Thesis, 1931.
6. Horn A. On sentences which are true of direct unions of algebras. Journal of Symbolic Logic, 1951.
7. Robinson A. J. A machine oriented logic based on the resolution principle. Journal of the ACM vol 12 1965. [perevod: Robinson Dj. Mashinno-orientirovannaya logika, osnovannaya na prinsipe rezolyusii. Kiberneticheskiy sbornik vp.7, 1970.]
8. Colmerauer A., Kanoui H., Pasero R., Roussel P. Un Systeme de Comunication Homme-machine en Francais. Universite d'Aix Marseille, 1973.
9. Fild A., Xarrison P. Funksionalnoe programmirovanie. M., Mir, 1993.
10. Xenderson P. Funksionalnoe programmirovanie. Primenenie i realizasiya. M., Mir, 1983.
11. Maurer U. Vvedenie v programmirovanie na yazke Lisp. - M.: Mir, 1976.
12. Morozov M.N. Funksionalnoe programmirovanie - kurs leksiy.
13. Xarper R. Vvedenie v Standartny ML.
14. Dushkin R.V. Leksii po funksionalnomu programmirovaniyu.
15. Dushkin R.V. Laboratorny praktikum po funksionalnomu programmirovaniyu.
16. Kovalski R. Logika v reshenii problem. M.:Nauka,1990.
17. Xogger K. Vvedenie v logicheskoe programmirovanie. - M.: Mir, 1988.
18. Sterling L., Shapiro E. Iskusstvo programmirovaniya na yazke Prolog. - M.: Mir, 1990.

19. Kloksin U., Mellish K. Programmirovanie na yazke Prolog. - M.: Mir, 1987.
20. Bratko I. Programmirovanie na yazke Prolog dlya iskusstvennogo intellekta. - M.: Mir, 1990.
21. Malpas Dj. Relyasionny yazk Prolog i ego primenenie. - Novosibirsk: Nauka,1990.
22. Morozov M.N. Logicheskoe programmirovanie - kurs leksiy .
23. Nishanov A.X., Akbaraliev B.B., Xan I.V., Ruzibaev O.B. Deklarativnoe programmirovanie, Uchebnoe posobie. Tashkent, «IMPRESS MEDIA», 2021g. Str.-216.
24. Nishanov A.X. Babadjanov E.S., Ergashev A.K., Risnazarov A.M. Ma'lumotlar bazasi. O'quv qo'llanma. Toshkent, «IMPRESS MEDIA», 2021g. Str.-298.
25. Nishanov A.X. Raximbaev X.J., Ergashev A.K., Xujaev O.K. Baz dannx. Uchebnoe posobie. Tashkent, «IMPRESS MEDIA», 2021g. Str.-260.
26. In S., Solomon D. Ispolzovanie Turbo-Prologa. - M.: Mir, 1993.ivnoe
27. Dextyarenko I.A. Deklarativnoe programmirovanie. Uchebnoe posobie. 2003
28. Will Kurt, Get Programming with HASKELL, 2018 by Manning Publications Co
29. Simon Thompson, Haskell the craft of functional programming, © Pearson Education Limited 2011
30. Christopher Allen and Julie Moronuki, Haskell Programming from first principles. Sep 15, 2016.
31. Bird R. Thinking Functionally with Haskell, Cambridge University Press, 2015.
32. Abelson H., Sussman G. Structure and Interpretation of Computer Programs. - MIT Press, Boston, 1986.
33. Fild A., Xarrison P. Funksionalnoeprogrammirovanie. M., Mir, 1993.

Mundarija

A.X.NISHANOV

DEKLARATIV DASTURLASH

DARSLIK

Muharrir *M.Talipova*
Tex. muharrir *G.Ibragimova*
Sahifalovchi *B.Haydarov*

Bosishga ruxsat etildi 10.09.2021. Бичими 60x84 $^{1}/_{16}$
Ofset qog'ozi. «Tayms» garniturasi.
Shartli bosma tabog'i 11.34. Nashr hisob tabog'i 9,44.
Adadi 100 nusxada. Buyurtma № 21-12.

«IMPRESS MEDIA» MCHJ kosmaxonasida chop etildi
Manzil: Toshkent sh., Qushbegi ko'chasi, 6-uy.

ISBN 978-9943-7506-5-4