ON CYCLE INVARIANT

Dainis ZEPS
Riga, 1986
University of Latvia

Abstract
Taking the programming paradigm with the cycle invariant as a base notion there for a ground cycle paradigm, in a more general setting here these things are considered. Epistemological aspects with reference to Rene Descartes methods of reasoning are considered. Particularly analogy between Descartes method of specification and top-down programming is considered. A general principle of reconstruction in epistemology is suggested and discussed, firstly specifying it within programming and then trying it to generalize to whatever else.

Submitted as a paper in philosophy for doctoral studies.

PĒTERA STUČKAS LATVIJAS VALSTS UNIVERSITĀTE
DIALEKTISKĀ UN VĒSTURISKĀ MATERIĀLISMA
MARKSISMA-ĻEŅINISMA FILOZOFIJAS KATEDRA

PAR CIKLA INVARIANTU

Zeps Dainis Aloiza R.,
LVU Skaitļošanas centrs,
matemātiķis - spec:fiziķis

RĪGA 1986

SATURS

IEVADS 4

CIKLISKUMS DABĀ UN SKAITĻOŠANAS PROCESĀ 6

Invarianta jēdziens fizikā 6

Invarianta jēdziens matemātikā un skaitļošanā 7

PROGRAMMĒŠANAS UN SKAITĻOŠANAS PROCESI 8

Programmēšana kā mērķtiecīga darbība 8

Skaitļošanas process 9

REKONSTRUKCIJAS PRINCIPS 10

Dekarts un skaitļošana 10

Rekonstrukcijas princips 13

Matemātiskā semantika 14

Rekonstrukcijas piemērs 15

Grīsa balona teorija 16

Par inicializāciju 18

Rekonstrukcijas princips fizikā 19

METODOLOĢIJA 20

Subjekta un objekta mijiedarbība izziņas procesā 20

Inkrementalitāte 22

Tehnoloģiskās sistēmas 23

Par optimizāciju 24

NOBEIGUMS 25

LITERATŪRA 26

IEVADS

Jau programmēšanas kā cilvēka darbības sfēras rītausmā tika noskaidrots, ka skaitļotāja programmēšana ir, vienkārši runājot, ciklu programmēšana. Jebkura reāla skaitļotāja programma, kas protams ir galīga, apraksta tā virtuālu darbību, kuras konkretizācija vispārīgā gadījumā ir bezgalīga. Tāpat bezgalīgais programmas neatkārtojošos darbināšanas variantu skaits ir tikai pateicoties atklātajiem vai slēptajiem cikliem tajā.

Skaitļotāja programmēšana kā praktiska darbība ir attīstījusies straujāk nekā atbilstošais teorētiskais pamatojums. Agri radās doma ekspluatēt skaitļotāja spēju veikt lielus rēķinus programmu izveidošanas atbalstīšanai. Viena no optimistiskām prognozēm mākslīgā intelekta laukā bija programmu automātiska verifikācija. Tomēr cerības anotēt programmas automātiski, t.i. sintezēt priekšnosacījumus katrai programmas instrukcijai no sākotnējā vai beidzamā apgalvojuma, izrādījās nepamatotas. Par iemesli izrādījās nespēja sintezēt tieši tos apgalvojumus, kuri ieiet ciklos vai netieši ciklu veidojošās programmēšanas valodu konstrukcijās, piemēram, rekursīvās procedūrās, kurām jāsintezē priekšapgalvojumi [10]. Ciklu sistemātiska aizvietošana ar rekursīvām procedūrām vai otrādi anotētām programmām ir izdarāma automātiski [28] un tātad attiecināma vairāk vai mazāk uz valodas sintakses līmeņa manipulācijām.

Šis ciklā ieejošais apgalvojums, kurš nepadevās automātiskai sintēzei pēc programmas teksta, tika nosaukts par cikla invariantu, un šis apgalvojums tiešām, triviālā kārtā, ir cikla invariants. Šoreiz lieta savā vārdā tika nosaukta tieši šo apgalvojumu svarīguma dēļ. Izdarītais secinājums bija sekojošs: izstrādājot programmu, programmistam pašam tā ir daļēji jāanotē un konkrēti, ar precizitāti, lai pārējie iztrūkstošie apgalvojumi var tikt sintezēti automātiski. Un , pirmām kārtām, katram ciklam vajadzīgs vismaz viens, tikai tam piederošs apgalvojums - šī cikla invariants.

Attīstoties programmēšanas tehnoloģijai, vismaz divi tās aspekti papildināja šo situāciju.Pirmkārt, tehnoloģijas pamatā ir jābūt hierarhiskai projektēšanai. Topošās programmas apgalvo-

jumi, kuri tagad pildīja programmas fragmentu specifikācijas, t.i. apraksta, lomu, bija tie, kas jāizstrādā un jādetalizē pa priekšu, un programmas teksts būs tikai šī detalizācijas procesa nobeiguma produkts. Otrkārt, cikla(invarianta) projektēšanā jāiziet no tā, ka vispirms mums ir "jāatpazīst" pats cikliskais process,t.i. jānoskaidro tā invariants, un ,izejot no tā, jārepprezentē šis cikliskais process programmas tekstā. Cikla projektēšana kļuva par hierarhiskās projektēšanas metodoloģijas patstāvīgu disciplīnu [3, 26, 27, 33, 37, 39].

Šajā referātā aplūkosim un mēģināsim noskaidrot cikla invarianta vietu skaitļošanas matemātikā un programmēšanas prakses metodoloģijā.

CIKLISKUMS DABĀ UN SKAITĻOŠANAS PROCESĀ

Cikliskums kā kategorija ir sastopams jau antīkajā filozofijā, piemēram, Heraklīta pasaules ugunsgrēks ir ciklisks ar periodu 10800 gadi [1].Stoiķu visuma attīstība noris lielos ciklos, kur kārtējais periods sākas ar dievišķu uguni un beidzas ar vispārēju ugunsgrēku [40]. Nostiprinoties Galileja un Ņutona izveidotajai laika un telpas fizikālajai izpratnei un Ņutona un Leibnica atklātajiem fizikas likumiem un izstrādātajām matemātikas metodēm, ciklisks process iegūst skaidru un noteiktu vietu mehāniskajā pasaules ainā. Mehāniskais cikliskais process ir aprakstāms ar konkrētiem fizikāliem (vai matemātiskiem) lielumiem - periodu, amplitūdu, fāzes nobīdi, kā arī pašu kustības līkni un to apraksto̶šo kustības vienādojumu. Visi šie lielumi ir aprakstāmi ar vienu jēdzienu - šī cikliskā procesa invariantu. Pats jēdziens "invariants" ir ieviests tikai 19. gadsimtā, un to ir izdarījis angļu matemātiķis Silvestrs [9].

Invarianta jēdziens fizikā

Mainīgais pretstatīts patstāvīgajam jau Heraklīta filozofijā ieņem vienu no galvenajām vietām, kur nemainīgajam logosam pretstatīts uguns vienmēr tā mainīgajās izteiksmēs, kuru vienība veido universālo visuma substanci [1, 13]. Visa mainīgā pretstatītais nemainīgais kā vienīgais būtiskais un reāli eksistējošais pasaules ainā ir absolutizēts Parmenīda filozofijā [1, 40].

Cikliska vai vispārīgāk mainīga procesa "atpazīšana" izziņas procesā var notikt divējādi. Subjekts var no kustības invarianta atklāt pašu kustību vai arī, otrādi, kustībā atklāt nemainīgo. Pirmā varianta piemērus varam saskatīt Heraklīta "upē, kurā nevar iekāpt divreiz" vai dienas un nakts jēdzienos kā Ptolemaja sistēmas invariantos. Otrā gadījumā varam minēt jebkuru novērojamu ciklisku procesu, pretstatot tam tam piemītošos fizikālos lielumus. Minēsim uzreiz to, kas izziņas procesā ir prasījis lielākas pūles, proti, saglabāšanās likumus fizikā un citās dabaszinātnēs. Cieša saistība starp saglabāšanās likumiem un kustības vienādojumiem fizikā noskaidrijās tikai 20. gadsimtā. 1904. gadā Hamels konstatēja [2],ka saglabāšanās likumi seko no laika un telpas fundamentālajām simetrijas īpašībām. Trīsdesmitajos gados E. Netere pierādīja teorēmu, kura ļauj no matemātiskās fizikas vienādojumiem un no to izrietošajiem invariantiem, simetriskajām laika un telpas transformācijām, iegūt saglabāša-

nās likumus kustības integrāļu veidā, proti, enerģiju, impulsu, kustības daudzuma momentu,utt. Tātad telpas un laika transformāciju grupa ir tā, kas "akceptē" jebkuras (mehāniskās) kustības invariantus, starp kuriem vispārīgākie ir saglabāšanās likumi. Tieši invarianta jēdziens ir tas, kurš ļauj tik cieši saistīt telpu un laiku no vienas puses un kustību no otras puses, proti, saglabāšanās likumu līmenī.

Invarianta jēdziens matemātikā un skaitļošanā

Nils Bors ir izteicis pārliecību, ka "invariantu ideja ir īstenības racionālas izpratnes atslēga un ne tikai fizikā, bet jebkurā izziņas aspektā vispār" [14]. Matemātikā šī ideja jau sasniedza pilnību 1872. gadā Kleina formulētajā Erlangenas programmā [7, 8]. Tās būtība īsumā ir formulējama tā, ka ģeometrijas ir klasificējamas un pētāmas pēc to simetrijas grupām. Arī algebraiskā topoloģija, kura ir attīstījusies Erlangenas programmas garā, darbojas tikai ar objektiem, kuri ir invarianti attiecīgajās simetriju grupās vai arī, runājot citādi, ir aplūkojami tikai ar precizitāti līdz homeomorfām transformācijām.

Skaitļošanā, kur invarianti algebraiskajās izteiksmes formās kļūst gaužām netverami, mēs nonākam atpakaļ pie cikliska procesa tā mehāniskajā izpratnē. Ja skaitļotāja (galīgu) programmu uzskatām par (bezgalīga) skaitļošanas rpocesa aprakstu, tad šis apraksts jau ir šī procesa invariants. Tomēr šāda abstrakcija ir gandrīz informatīvi tukša, un mums ir izdevīgi šo invariantu maksimāli sašaurināt, atstājot iespēju pilno invariantu, t.i. programmas tekstu, reģenerēt no tā. Pateicoties programmēšanas valodu vienkāršajai sintaksei, liela (hierarhiskā sintakses) daļa atkrīt automātiski. Ja mēs iedomājamies, ka esam radījuši absolūti heirarhisku programmu, kura nesatur nevienu semantisku saiti, tad šis reducētais invariants būtu triviālais apgalvojums. Visu programmu mēs varam tātad iedomāties kā hierarhisku struktūru, kur katrai semantiskai saitei ir piekārtots apgalvojums, kuri kopā sastāda šo reducēto invariantu. Bez tam, ja kāda semantiskā saite, darbinot programmu, "nostrādā" tikai vienu (vai arī konstantu skaitu) reizi, tad tai atbilstošais invariants ir izvedams no beigu nosacījuma, kuram protams tādā gadījumā jābūt. Tātad paliek tikai tie invarianti, kuri "pārgriež" daudzkārtējās, "virtuālās" semantiskās saites.

Ar šādu ļoti neformālu un vienkāršotu spriedumu mēs esam

nonākuši līdz cikla invariantam. Šajā kontekstā mums šķiet interesanti fiksēt īpašību pāri - hierarhisks un ciklisks, kuri mehāniskā procesā izslēdz viens otru, bet radošās konstruēšanas procesā veido kategoriju pāri.

PROGRAMMĒŠANAS UN SKAITĻOŠANAS PROCESI

Radošās konstruēšanas subjekts izgatavo programmas procesā, kuru sauksim mums pierastā vārdā par programmēšanu. Šīs programmas darbināšana skaitļotājā savukārt ir skaitļošanas process. Ja pirmajā procesā programma pilda objekta lomu, tad otrajā procesā to varam uzskatīt par subjektu, ja paša skaitļotāja funkcijas mēs piedēvējam videi, sistēmai. Pamatots ir jautājums, vai šos divus procesus saista tikai pragmātiskā prasība pēc "galarezultāta".

<u>Programmēšana kā mērķtiecīga darbība</u>

Viena no auglīgākajām atziņām, kāda ir fiksēta programmēšanas kā subjekta darbības metodoloģijā, ir tas, ka programmēšana ir mērķtiecīga darbība (goal-oriented-activity), kur "mērķtiecīgs ir jāsaprot mehāniskā,teleologiskā nozīmē [27, 33]. Ja, izmantojot mehānikas kustības vienādojumus, rēķinām lielgabala lādiņa kustības līkni, tad virzāmies no agrākā laika momenta uz vēlāku. Ja mums šī kustība ir "jāprogrammē", tad izrādās, ka dabīgāk ir no beigu nosacījuma virzīties laikā atpakaļ. E.Deikstra ieviesa vājākā priekšnosacījuma predikātu wp ,kurš programmai P un beigu nosacījumam R piekārto vājāko priekšnosacījumu wp(P,R) ,t.i. anotētai programmai $\{Q\}$ P $\{R\}$ vienmēr izpildās $Q \Rightarrow wp(P,R)$ [27]. Predikāts wp ir viegli definējams vienkāršām programmēšanas valodām, definējot wp strukturāli katrai valodas sintakses konstrukcijai atsevišķi. Cikla konstrukcijai gan atkal vajadzēs cikla invariantu pēc būtības. Ja programma P tiek izpildīta un strādā laika intervālā (t_0,t_1) ,tad varam iedomāties, ka pretējā virzienā laikā izpildās cita programma , proti, wp ar argumentu P ,kura rēķina vājākos priekšnosacījumus katrai P instrukcijai. Precīzāk, ja instrukcija i stāvoklī s_1 transformē stāvoklī s_2 ,tad $wp(i,s_2) = s_1$,kur

a_1, a_2 apraksta attiecīgi stāvokļus s_1, s_2. Šī iemesla dēļ programmas tiek sauktas par predikātu transformatoriem un programmēšanu - par predikātu transformēšanu. Pāreja no instrukciju kodēšanas uz predikātu transformēšanu ir grūti pārvērtēt, jo beidzamo mēs darām matemātikas valodā. Nākošais solis, strukturizētšo procesu savādāk, nekā to prasa (pretēji laikā) izpildīšanas virziens, jau liekas gluži dabīgi.

Skaitļošanas process

Programmēšanas kā "goal-oriented activity" aplūkošana dod mums pieeju līdzīgi apskatīt arī skaitļošanas procesu, t.i. procesu skaitļotāja stāvokļu telpā, tam darbojoties pēc noteiktas programmas.

Lai programma P, izpildot to k reizes, ir beigusies ar stāvokļiem $t_1, \ldots, t_k$, un predikāts R izpildās visiem šiem stāvokļiem. Skaitļošanas process izrēķina predikātu Q tādā nozīmē, ka, izpildot P kā predikāta transformatoru pa P instrukcijām, iegūstam $s_i = wp(P, t_i)$, kur s_i, $1 \leq i \leq k$, apmierina Q. Tā kā mēs aplūkojam tikai tos P darbināšanas paraugus, kuri beidzas pēc galīga soļu skaita, tad nepieciešamos invariantus wp "izrēķina", kopējot instrukcijas cikla tikreiz, cikreiz tās izpildās.

Skaitļošanas procesam šīs definīcijas nozīmē, kuru mēs varētu saukt par logisko skaitļošanas procesu, mēs varētu pretstatīt fizisko skaitļošanas procesu, proti, kas notiek ar skaitļotāja stāvokļiem. Ja logiski skaitļošanas process virzās laikā atpakaļ, tad fiziski - tā dabiskajā virzienā. Šajā vietā mēs fiksēsim būtisku momentu mūsu izpratnē par šo fizisko procesu. Ja iedomājamies pietiekoši ātrdarbīgu skaitļotāju ar lielu atmiņu, teiksim, kurā tikai savu stāvokļu sanumurēšanai patērētu gadsimtus, tad reālo skaitļošanas procesu šādā skaitļotājā mēs saprotam kā haotisku ceļošanu stāvokļu telpā bez atkārtošanās kādos stāvokļos. Tātad no novērotāja viedokļa, kas "neko nesaprot, kas tiek rēķināts", fiziskais skaitļošanas process ir haotisks klejojums stāvokļu telpā no kāda zināma sākuma stāvokļa.

Atbildot uz jautājumu nodaļas sākumā par saistību starp programmēšanu un skaitļošanu, nonākam pie secinājuma, ka varam tās attiecināt tāpat kā logisko skaitļošanas procesu pret fizisko, t.i. notiekošus pretēji laikā. Tamdēļ mums nav obligāti jāmēģina stādīties priekšā, kā programmēt pretēji laika virzienam,

jo mūs interesē šos procesus vienojošās īpašības, atšķirīgās uztverot par tādāk, kā mēs tās novērojam praksē.

REKONSTRUKCIJAS PRINCIPS

Šajā nodaļā ievedīsim un lietosim metodi, kuru esam atvasinājuši no Dekarta metodes [5, 17].

<u>Dekarts un skaitļošana</u>

Renē Dekartam (1596 - 1650), izcilajam franču dabaszinātniekam un filozofam, ir milzīgi nopelni matemātikas un tās metodoloģijas attīstībā. Pirmo reizi matemātikas vēsturē jauna matemātikas nozare nav izveidojusies evolūcijas procesā, darbojoties daudz domātājiem, bet gan kā viena matemātiķa apzinātu pūliņu rezultāts. Tas notiek Renē Dekarta personā , praksē lietojot savu izstrādāto metodi analītiskās ģeometrijas izveidošanā un pamatošanā.

Renē Dekarta ietekme uz 20. gadsimta dabaszinātņu attīstību ir tik jūtama, ka Heizenbergs [4] ir spiests Einšteina neizpratni par jaunajām kvantu mehānikas idejām attiecināt kā karteziānisma relikta sekas. Heizenberga izpratnē tieši kvantu mehānika ir tā, kas kartezianiskо sadalījumu "domāju" un "eksistēju" noved līdz izziņas procesu traucējošai nepilnībai. Bet mēģināsim palūkoties uz Dekartu no cita viedokļa un atdalīt no viņa kartezianismu, t.i. ietekmi, ko viņš atstāja uz vairākiem gadsimtiem dabaszinātņu attīstībā. Dekarts saka [5, 94.lpp.]:

> Es, apzinoties savu vājību, nolēmu izziņas meklējumos cieši pieturēties sekojošai kārtībai: visu sākt ar visvienkāršākajām un visvieglākajām lietām un nekad nepāriet pie citām, iekāms neredzēšu, ka ar tām neko vairāk izdarīt nevar.

Atliek atvērt akadēmiska stila matemātikas monogrāfiju un pārliecināties, ka mūsdienu matemātika ir caur un cauri kartezianiska šī citāta nozīmē.

Divdesmitā gadsimta dabaszinātnes, centienos atbrīvoties no karteziānisma vienā izpratnē, ir spiestas pēc tam, kad ir parādījusies kibernētika un skaitļošanas zinātne, vēlreiz paklanīties Renē Dekartam un "akceptēt" jaunu karteziānismu, šoreiz viņa metodes izskatā. Mēģināsim to pakāpeniski parādīt. [8]

Aplūkosim likumu V no Dekarta "Likumiem prāta vadīšanai":

> Visa metode kādas patiesības atklāšanai slēpjas tā kārtībā un izvietojumā, uz ko ir jāvērš domas asums. Mēs to stingri ievērosim, ja centīsimies reducēt tumšākās un neskaidrākās lietas uz vienkāršākām un tālāk, izejot no vienkāršāko intuitīvas skaidrības, mēģināsim pacelties pa tiem pašiem pakāpieniem, lai izprastu visu pārējo.

Ja ar "patiesības atklāšanu" saprotam likumsakarību atklāšanu kāda projekta rāmjos, tad ar šo citātu skaidri ir izteikta hierarhiskās projektēšanas ideja, kas šodien ir pilnīgi un galīgi akceptēta kā vienīgais universālais līdzeklis skaitļotāju programmu izstrādāšanai. Galvenokārt tas attiecās uz ļoti lielām programmām, kuras tikai tā (hierarhiski) var radīt. Šajā vietā mēģināsim precizēt situāciju un citēsim programmēšanas klasiķi E.Deikstru [11] :

> Lidzko programmēšana pārsniedza pieļaujamās sarežģītības robežas, notika pagrieziens uz disciplīnu, kuras mērķis kopš gadusimtiem ir lietot efektīvu strukturēšanu ar nolūku pārvarēt šķietami nepārvaramu sarežģītību. Šī disciplīna, kas mums visiem ir vairāk vai mazāk pazīstama, saucas matemātika. Ja mēs akceptēsim kā pareizu spriedumu, ka matemātiskās metodes ir visefektīvākais līdzeklis sarežģītību pārvarēšanai, mums nebūs citas iespējas, kā tikai pārorientēt programmēšanas jēmu tādā veidā, lai šīs metodes kļūtu pieejamas, jebšu citi līdzekļi neeksistē.

Bez kā tad nevar iztikt? Bez hierarhiskās projektēšanas vai bez matemātikas? Atbilde ir vienkārša un, mūsu skatījumā, vienīgā. Proti, gan vienas gan otras vienībā. Tikai tad mums ir jāsamierinās, ka šī matemātika ir karteziāniska (likuma V izpratnē). Šeit protams var iebilst, ka strukturējamība matemātikā var būtiski atšķirties kā noveselā attieksmes pret tā daļu. Matemātikā vispār - jā, bet mums diemžēl nāksies šīs elegantās matemātiskās mašīnas "sadauzīt" mūsu nepilnīgo programmēšanas valodu instruk-

cijās.

Ņemot vērā, kādu vietu Dekarta mehāniciskais domāšanas veids objektīvi ieņem mūsu mehānisko skaitļotāju laikmetā, citēsim viņa metodi paša Dekart vārdiem [17, 27.lpp.] :

Pirmais no tiem bija neuzskatīt nekad par patiesu to, par ko man nav acīmredzami zināms, ka tas tāds ir, citiem vārdiem sakot, rūpīgi izvairīties no pārsteidzības un aizspriedumiem un neietvert savos spriedumos neko vairāk kā vienīgi to, kas parādās manam prātam tik skaidri un noteikti, ka man nav nekādu iespēju to apšaubīt.

Otrais, sadalīt katru pētāmo problēmu tik daudz daļās, cik vien iespējams un cik nepieciešams, lai šo problēmu labāk atrisinātu.

Treškārt, virzīt manas domas noteiktā kārtībā, sākot ar visvienkāršākajām un visvieglāk izzināmajiem objektiem, lai maz pamazām kā pa pakāpieniem paceltos līdz vissarežģītāko zināšanai; pieļaujot domu, ka pastāv kārtība pat starp tiem, kas dabiski neseko viens otram.

Un pēdējais, it visur izdarīt tik aptverošu uzskaiti un tik vispārējus apskatus, lai es būtu drošs, ka nekas nav izlaists.

Dekarts pats mēģināja lietot savu metodi dažādos līmeņos, sākot ar izziņas kategoriju līmeni un beidzot ar mehānisku aprēķinu, kas tomēr, jādomā, bija galvenais avots viņa metodei vispār. Šodien skaitļošanas zinātnē Dekarta metodes "precedentu" uzskaitījumu ir grūti pārvērtēt. Minēsim tikai dažus kā piemērus, sadalot tos trīs grupās:

I. Izziņas kategoriju līmenis - lejupejošā (hierarhiskā) projektēšana [3, 11, 23, 30] ;

II. Problēmu kategoriju līmenis - visas sintakses vadītās (sintax directed) metodes, procedūru un datu abstrakciju metode [30, 32] ;

III. Pārlases (zemkategoriju) līmenis - "dalīt un valdīt" metode, balansēšana, "zaru un robežu" metode, dinamiskā programmēšana [18] .

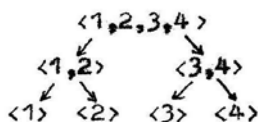
Vienkāršāk tas būtu izsakāms tā, ka programmēšanā bez dalīšanas pa daļām nekas nav izdarāms. Kompjūteri uzrodas 300 gadus pēc Dekarta, bet pats Dekarts acīmredzot bija spējīgs veikt milzīga apjoma aprēķinus, kas viņam ļāva tik ļoti attīstīt savu domāšanu šajā virzienā.

Rekonstrukcijas princips

Atcerēsimies, ka mūsu sākotnējais nolūks bija noskaidrot cikla invarianta lomu un vietu skaitļošanas metodoloģijā. Nedaudz modificējot Dekarta metodi, formulēsim rekonstrukcijas principu, kas arī kalpos kā galvenais rīks mūsu jautājumu noskaidrošanai.

Mūsu metodes pamatā ņemsim Dekarta metodi tās mehāniskā, teleologiskā izpratnē. Kāds ir galvenais arguments pret Dekarta metodi? Mūsu izpratnē tas ir fakts, ka veselā rekonstrukcija no daļām, par kurām mēs visu it kā zinām, nedod iespēju teikt, ka mēs iegūsim tikpat skaidru priekšstatu par veselo. Mēģināsim situāciju simetrizēt: ja a ir b sastāvdaļa vienā nozīmē, un b ir a sastāvdaļa citā (duālā) nozīmē, tad teiksim, ka a un b veido rekonstrukciju. Protams, telpiskie objekti mums pierastā Eiklīda telpā nekad šādu rekonstrukciju neveido. (Ja iekļaušanās ietver sevī sakritību, tad dabūjam ekvivalentu objektu pārus.)

Aizstāsim iekļaušanās īpašību ar aproksimācijas īpašību. Tiešām, ja a ir b sastāvdaļa, tad informācija par a aproksimē informāciju par b, un rakstām $a \sqsubseteq b$. Aplūkosim saraksta dalīšanas procesu (bultiņu virzienā) 1.zīm.



1.zīm.

Kādas īpašības piemīt šādam dalījumam? Varam veidot aproksimējošu virkni

$$\langle 1,2,3,4\rangle \sqsubseteq \langle 1,2\rangle \langle 3,4\rangle \sqsubseteq \langle 1\rangle\langle 2\rangle\langle 3\rangle\langle 4\rangle ,$$

kura aproksimē saraksta dalīšanas procesa galarezultātu. Iedomāsimies, ka šis process notiek pretēji laikā. Tad aproksimējošā virkne

$$\langle 1\rangle\langle 2\rangle\langle 3\rangle\langle 4\rangle \sqsubseteq \langle 1,2\rangle \langle 3,4\rangle \sqsubseteq \langle 1,2,3,4\rangle$$

aproksimē sarakstu vienošanas procesa galarezultātu. Šādu dalīšanas procesu vienībā ar apvienošanas procesu mēs sauksim par rekonstrukciju.

Šāds divu simetrisku procesu pāris ir pēc būtības neinformatīvs procesa nozīmē, jo mēs būtībā ignorējam cēloņsakarības principu. Rekonstrukcijai, kura ir neinformatīva, mēs pretsta-

tīsim rekonstrukcijas principu, ar kura palīdzību mēs centīsimies iegūt metodologiska satura informāciju, lietojot rekonstrukciju.

Rodas jautājums, ko var dot šāds vienkāršs dalīšanas process un ar to saistītais apvienošanas process, izņemot to, ka kaut kas ir izjaukts un atkal salikts no jauna? Tomēr sistēmās, kur nepieciešams liels skaits mehānisku manipulāciju, šāds process var būt izšķirošs, kas pilnībā ienes skaidrību. Aplūkosim, kādas funkcijas pilda rekonstrukcija. Izdalīsim šādas:

1) realizē pieeju katram elementārajam objektam;
2) izdala procesa iterācijas (atkārtošanās) komponentes;
3) fiksē nedeterminismu.

Aplūkosim vienkāršus matemātiskus piemērus. Visvieglāk rekonstruējamie objekti ir kopas. Jau izteiksme $x \in S$ satur visu informāciju kopas S rekonstruēšanai. Šādu rekonstrukciju atklāti var atspoguļot ar formulu $x \in S \Rightarrow \{x\} \cup S' = S$ .Skaidrāk tas ir attēlojams specifikāciju valodās, piemēram META IV [19, 30]:

$$S' = \{x \mid x \in S\} .$$

Būtiski cits kopas rekonstrukcijas piemērs ir

$$S' = \underline{\text{union}}\, \{X \mid X \in \text{subsets}(S)\} .$$

Lai M ir tabula un $\underline{\text{def}}\ M$ ir tās definīcijas apgabals. Izteiksme

$$M' = [M(i) \mid i \in \underline{\text{def}}\ M]$$

rekonstruē tabulu M .Lai L ir saraksts un $L(i)$ - i-tais tā elements. Izteiksmes

$$L' = \langle L(i) \mid 1 \leqslant i \leqslant \underline{\text{len}}\ L \rangle ,$$

$$L' = \langle \underline{\text{hd}}\ L \rangle \frown \underline{\text{tl}}\ L$$

rekonstruē sarakstu L .Pirmā no izteiksmēm fiksē nedeterministisku pieeju saraksta elementiem, turpretī otrā ļauj rekonstruēt sarakstu pēc būtības tikai sekvenciāli.

Matemātiskā semantika

Matemātiskā semantika, kā to pasniedz Skota teorija [15, 19, 20] ,funkciju f, $f{:}D \to D$,kuru mēs gribam izrēķināt ar skaitļotāja programmu P ,meklē kā kādas funkcionāļa Y nekustīgo punktu: $Y(f) = f$. Funkcijas f aprēķināšanu ar rekursijas metodi pamato Klīni teorēma, un tas notiek sekojoši:

$$\text{(i)} \qquad Y(f) = \bigsqcup_{i=0}^{\infty} f^i(\perp) .$$

Tātad virkne

$$\text{(ii)} \qquad \perp \sqsubseteq f(\perp) \sqsubseteq \ldots \sqsubseteq f^i(\perp) \sqsubseteq \ldots$$

aproksimē atrisinājumu f .Uzskatīsim izteiksmi (ii) par funkci-

jas f rekonstrukciju. Tiešām, ja teorija garantētu, ka Y nekustīgais punkts eksistē, tad izteiksmes (i) labā puse vai virkne (ii) parāda, kā to rekonstruēt. Skota teorijā ievestie domēni ar režģa struktūru tajos un prasība pēc funkciju monotonitātes un nepārtrauktības tajos pilnībā atrisina tai uzstādītās prasības , tādējādi piešķirot teorijai matemātiskās semantikas statusu.

Skota teorijas konstruktīvākais moments ir tas, ka ar skaitļotāju (teorētiski) izrēķināmo vērtību apakškopa domēnā precīzi sakrīt ar tā bāzi, kuras ieviešanas nepieciešamību diktē jebkuras funkcijas nepārtraukības prasība, kas tieši ir garantējošais faktors tipa (ii) virkņu robežu eksistencei. Tātad Skota teorijā domēni ir izveidoti tā, ka vienmēr ir lietojama Klīni teorēma, jo citu funkciju vienkārši neeksistē. Domēnu konstruktori, kuri atkārto relāciju valodas sintaksi, garantē visu "ar skaitļotāju izrēķināmu" funkciju izrēķināmību. Tākā visu daļējo funkciju kopa ir nesanumurējama, Skota teorija mums nosprauž robežu reālai izrēķināmībai uz skaitļotāja.

<u>Rekonstrukcijas piemērs</u>

Aplūkosim rekonstrukcijas ideju vienkāršā piemērā. Z.Manna [12] parāda, ka funkcionāļa P ,kur

P: F(x,y) = <u>if</u> x = ⟨⟩ <u>then</u> y <u>else</u> F(<u>tl</u> x, y^⟨<u>hd</u> x⟩),

nekustīgais punkts, funkcija reverse,apmierina inv ,kur

inv: reverse(x^y) = reverse(y)^ reverse(x)

Ko varam izsecināt mēs? Abas šīs izteiksmes ir rekonstrukcijas. Protams rodas jautājums,1)ko tās rekonstruē,un 2)vai rekonstruē vienu un to pašu funkciju. Funkcionālis P rekonstruē funciju reverse, t.i. savu nekustīgo punktu. Pārrakstīsim P līdzīgi, kā to dara Terners [38] :

F ⟨⟩ y = y
F x y = F <u>tl</u> x y^⟨<u>hd</u> x⟩ .

un vēlreiz

F ⟨⟩ = id
F a:x y = F x y:a,

kur id - identiskā funkcija. Šeit visas sarakstu operācijas izņemot "divnozīmīgā" (cons) operācija ":", kura kreisajā pusē ir atdalīšana, bet labajā pusē - pievienošana. Šādā ,no loģiskās programmēšanas aizņemtā semantikā [31, 41] , P vēl vairāk mums atklājas kā rekonstrukcija. Pārrakstīsim [38] stilā arī inv :

reverse ⟨⟩ = ⟨⟩

reverse ⟨a⟩ = ⟨a⟩

reverse append x y = append reverse y reverse x,

kur esam pievienojuši inicializāciju, lai iegūtu nobeigtu programmu. Ievērosim ka, ja "append" kreisajā pusē ir dalīšana, bet labajā pusē - apvienošana, tad trešā rindiņa, kura atbilst inv, ir rekonstrukcija. Bet tieši tas tiek garantēts augšminētajā semantikā. Tādā līmenī, kādā aplūkojam mēs, P un papildinātā inv ekvivalence ir triviāla. Atšķirība tā, ka inv ietver sevī nedeterminismu, proti, to kuru slēpj sevī funkcija append.

<u>Grīsa balona teorija</u>

Aplūkosim Deivida Grīsa cikla konstruēšanas tehnologiju [26, 27], modificējot to mūsu nolūkam iegūt tikai vispārēju ainu. Lai mums jākonstruē predikātu transformators, kurš no (sākuma) stāvokļiem Q ļauj nokļūt (beigu) stāvokļos R (2.zīm.).

Q   R

2.zīm.

Kā mēs redzējām, mums reāli jāpārvietojas pretēji laikā, t.i. no R uz iespējamiem stāvokļiem kopā Q. Uzreiz fiksēsim, ka mums pēc būtības ir "dabīgs" avots R, no kura izplatīsies rēķināšanas process. Stāvokļi kopā Q šim procesam nepieder, bet tikai apmierina mūsu nepieciešamības pēc "rēķināšanas vispār". Lai atspoguļotu šo momentu, uzdevumu, kurš ir atkarīgs no Q un R, vispārināsim uz tādu, kurš atkarīgs tikai no R un sauksim to par virtuālo akciju.

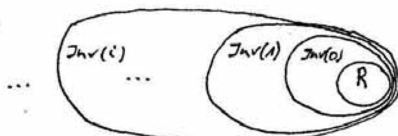
Lai ir anotēts programmas fragments u:

u: {Q} I ; <u>while</u> B <u>do</u> {Inv} S <u>od</u> {R},

kuram atbilst virtuālā akcija a:

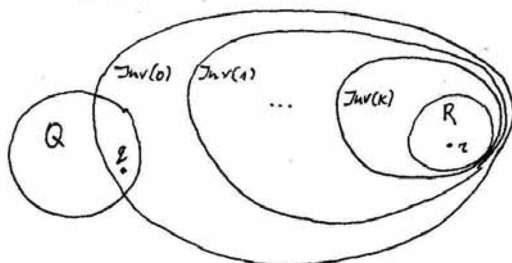
a: {P} <u>while</u> B <u>do</u> {Inv(i)} S <u>od</u> {R}.

Nosacījumi Inv un Inv(i) ir attiecīgo ciklu invarianti, kur Inv(i) ir atkarīgs no kāda parametra i. 3zīm. attēlosim virtuālās akcijas invarianta izplatīšanās logiskā rēķināšanas procesa laikā.



3. zīm.

Virtuālā uzdevuma konkretizācijai ar $i = 0$ atbilst tukšā akcija. Mūs praktiski interesē tie rēķināšanas procesa gadījumi, kad $Inv(i)$ galīgā soļu skaitā sasniedz stāvokļus kopā Q. Dažādiem $r \in R$ šis soļu skaits ir protams atšķirīgs, tāpēc zīmēt attiecīgu diagrammu ir jēga tikai fiksētam r (4.zīm.).



4.zīm.

4.zīm. mēs esam pārnumurējuši i, sākuma vērtību piešķirot k un samazinot to līdz nullei. Tādējādi mēs dabisko rēķināšanas procesu esam savienojuši ar tā pragmatiku. Atzīmēsim, ka vispārīgi i nav ierobežots un tā pārnumurēšana nav iespējama. Mūs interesējošā rekonstrukcija ir attēlojama ar virknēm

$$Inv(0) \subseteq Inv(1) \subseteq \ldots \subseteq Inv(k) \subseteq R ,$$

un

$$a_k \subseteq a_{k-1} \subseteq \quad \ldots \quad \subseteq a_0 \subseteq \ldots ,$$

kur a_i - virtuālā akcija ar $Inv(i)$. Pirmā virkne aproksimē beigu nosacījumus, turpretī otrā - nepieciešamo akciju, lai veiktu pragmātisko uzdevumu.

Īsumā attēlosim cikla konstruēšanas tehnologiskos soļus, izmantojot mūsu terminologiju. Tātad, ja ir doti nosacījumi Q, R un Inv, jāatrod instrukcijas I un S un nosacījums B. To izdarām četros soļos:

1) atrast parametru i, ka invariantu virkne $Inv(i)$ aproksimē R;
2) atrast nosacījumu B, kuram atbilst tukšā akcija;
3) atrast instrukciju S, kura, samazinot i, atjauno invariantu Inv, t.i. realizē pāreju $Inv(i) \rightarrow Inv(i-1)$;
4) inicializēt ciklu, t.i. atrast instrukciju I.

Cikla projektēšana praksē sākas ar cikliskā procesa atpazīšanu, t.i. ar cikla invarianta atklāšanu. Šie četri tehnologiskie soļi rāda, ka pārējais ir izdarāms vairāk vai mazāk formāli.

Aplūkosim vienkāršu piemēru. Lai mums jāizrēķina masīva elementu summa, t.i. sum(0 ▭ n-1) .To izdara šāda programma:

```
sum := 0; i := 0;
while B: i  n  do
{rec: sum(0 ▭ n-1) = sum(0 ▭ i-1) + sum(i ▭ n-1)}
{inv:  i }
 sum := sum + ▭i;
 i   :=  i  +  1; od
{sum  =  sum(0 ▭ n-1)}  .
```

Šeit mēs izdarījām mazu atkāpi no augšminētās tehnologijas.Mēs cikla invariantu aizvietojām ar rekonstrukciju rec ,atstājot par invariantu inv to informāciju, kura ir minimāli nepieciešama, t.i. šajā gadījumā parametru i .Tālāk pa soļiem demonstrējam tehnologiju:

1) i ir iebūvēts jau rekonstrukcijā;
2) pie i = n-1 vēl ir jāpieskaita pēdējais masīva elements, tātad pie i = n akcija kļūst tukša;
3) pieskaitam sum i-to masīva elementu, vienlaicīgi palielinot i ;
4) rekonstrukcija trivializējas pie i = 0 ,un formāli izpildās pie sum = 0 .

Par inicializāciju

Naivi programmējot var likties, ka programma jāsāk rakstīt no tās sākuma. Tātad, programmējot ciklu, tas vispirms ir jāinicializē. No iepriekš aplūkotā mēs viegli varam izsecināt, ka tā rīkoties ir aplami.

Tiešām, ja mēs ciklisku procesu papriekšu inicializējam un tikai pēc tam atklājam tā dabu, fiksējot tā invariantu, tad šis invariants būs atkarīgs no izdarītās inicializācijas. Tātad to būs vienkārši grūtāk atklāt, ja vispār to varēsim izdarīt. Ar kļūdu un mēģinājumu metodi, mainot šo inicializāciju, mēs varbūt nonāksim līdz labam invariantam, bet visbiežākā situācija būs tāda, ka šis invariants pēc piespiedu inicializācijas būs kļuvis mums pārlieku sarežģīts, un mēs to "vienkāršosim", ielaižot programmā kļūdu.

Mūsu ievestā virtuālā akcija atšķiras no pragmātiskā uzdevuma vienīgi ar to, ka tā nav inicializēta, bet tieši tās konkretizācijas ir tās, kuras rekonstruē invariantu. No šā viedokļa konstatējam, ka programmēšanas valodu sintakse, kura izdala atklāti ciklu (vai rekursīvu procedūru, kas ir viens un tas pats,)

nenojauc iespēju ciklisko procesu rekonstruēt, bet semantiskajai rekonstrukcijai mums jāseko pašiem.

Inicializācija pēc būtības tiek atdalīta būvējot moduļus, kas sakrīt ar virtuālā akciju, t.i. dinamiskus moduļus. Tomēr šādi "principā aizmukt" no inicializācijas nevar, jo moduļu dinamizācija nav izdarāma viennozīmīgi pēc būtības. Patiesībā šāda dinamizācija ir"izklīstoša",t.i. kā skaitļošanas process vispār. Liela daļa no inicializācijas ir saistīta ar optimizāciju un ir jau ietverta kompilēšanas fāzē, kuru mēs neapzināmies. Lai iedomājamies, ka mēs shēmu (compiler; executer) aizvietojam ar (initializer; compiler; executer). Pagaidām mēs tā programmēt nemākam.

Rekonstrukcijas principe fizikā

Skaitļošanas procesu izziņas procesā mums bija izdevīgi aplūkot kā izklīstošu no beigu nosacījuma, t.i. izklīstošu pretēji laikā. Hierarhiskā rekonstrukcija ir neinformatīva, turpretī jebkurš cikls "uzstājas" ar savu vispār netriviālu, tomēr mehāniski interpretējamu rekonstrukciju, Analogiski fizikā, sekojot cēloņsakarības principam, jebkurš process ir izklīstošs no saviem specifiskajiem cēloņiem - sākuma nosacījumiem. (Atcerēsimies fizikālo skaitļošanas procesu.) Tātad fizikā hierarhiskajai rekonstrukcijai mēs varam pretstatīt cēloņsakarības principu, kurš ir neinformatīvs ar precizitāti, ka sakārto (relativistiskajā fizikā daļēji) notikumus laikā. Katra semantiskā saite mums fizikā reprezentējas kā likumsakarība starp fizikālajiem lielumiem. Šo likumsakarību aprakstošais kustības vai, vispārīgāk lauka vienādojums ir tas, kas rekonstruē parādību kā matērijas atspoguļojumu subjektā izziņas procesā. Šo likumsakarību noteikšanas secības nepieciešamība izziņas procesā ir Dekartu, acimredzot, uzvedinājušas uz domu par metodi, ņemot vērā viņa vienīgo legālo (ne acīmredzama) sprieduma veidošanas līdzekli - dedukciju.

Mūsdienu fizikālā pasaules aina rāda, ka aplūkotais rekonstrukcijas princips ļauj,piemēram,konstatēt, ka "daba savās likumsakarībās rekonstruē pati sevi", bet šādi konstatējumi ir tikpat neinfirmatīvi kā cēloņsakarības princips. Kas attiecās uz metodoloģijas jautājumiem skaitļošanas matemātikā, tad mums ir būtiski tikai noskaidrot, vai šāds mūsu ievests princips saskaņējas ar fizikālo pasaules ainu vai nē.

METODOLOĢIJA

Subjekta un objekta mijiedarbība izziņas procesā

Iepriekš aplūkotajam kategoriju pārim "process dabā" un "skaitļošanas process" pretstatīsim atbilstošo kognitīvo kategoriju pāri "daba kā izziņas objekts" un "humanizētā niša", ar pēdējo saprotot tieši to dabas daļu, kuru pārveido un rada cilvēks. Sāksim ar to, ka ievedīsim divus izteikumus, proti:

D: viss, kas dabā iespējams, eksistē;

H: viss, subjekta radītais, eksistē,

kuru patiesīgumu mēģināsim noskaidrot.

Subjektam iedarbojoties uz objektu izziņas procesā aktīvs ir subjekts, bet objekta aktivitāte izpaužās tādējādi, ka konstatētajiem dabas likumiem jāsaskaņojas ar pašu dabu, t.i. šīs sistēmas ietvaros šis dabas likums itkā pieder objektam.

Subjektam iedarbojoties uz objektiem, proti, "humanizēto nišu" savas darbības procesā aktīvs, protams, ir pats subjekts, bet objekta aktivitāte, mūsu skatījumā, izpaužas (lokāli) nepieciešamībā ievērot dabas likumus un (globāli) subjekta pašapzināšanās nepieciešamībā būt primārajam agentam šīs sistēmas ietvaros. Šeit ar subjekta darbību mēs saprotam šīs darbības organizatorisko, izziņas fāzi. Tādējādi globālā prasība mūsu darbībai izpaužas pēc nepieciešamības būt mūsu projektos cik iespējams vispārīgiem un dabas atspoguļojumu izvērst tajos maksimāli iespējami, noliedzot prgmātismu kā nepieciešamību pilnībā.

Apgalvojums H šķiet acīmredzams, ja ar subjekta radīto mūs interesējošās darbības rezultātu. Aplūkojot izteikuma D patiesumu, tas mūs novedīs pie neauglīgiem spriedumiem,ja izziņu nesaistīsim ar darbību. Tieši tāpēc mēs ņemsim abus izteikumus, D un H ,kopā un no šā viedokļa izteiksim savu pārliecību, ka izteikums D ir patiess. P.Diraks [6] ,izstrādājot relativistisko kvantu mehānikas vienādojumu, nonāca pie secinājuma, ka iespējama elementārdaļiņa, kas analoga elektronam, bet ar pretēju lādiņu. Dabā šī daļiņa, pozitrons "atradās" dažus gadus vēlāk. Hamiltons savus vienādojumus izveidoja simts gadus pirms kvantu mehānikas, apmierinot savu iekšējo nepieciešamību pēc skaistas matemātiskās teorijas. Bez šiem vienādojumiem Heizenbergs nebūtu izveidojis savu kvantu mehānikas variantu. Ilgu laiku algebraiskā topoloģija tika uzskatīta par tik abstraktu matemātikas disciplīnu, kurai nav nekā kopēja ar pielietojumu.Relatīvi nesen noskaidrojās, ka kvarku mijiedarbību simetrijas grupa ir invariants saslāņotajās telpās. Šādu uzskaitījumu varētu tur-

pināt, bet tas protams nepierāda, ka katram matemātiskam konstruktam eksistē kaut kāds analogs dabā. Ar izteikuma D palīdzību mēs izsakām pārliecību, ka, ja arī kāds "matemātisks kalamburs" dabā nerod sev apstiprinājumu, precīzāk, tas nav dabas atspoguļojuma rezultāts, tad tas izrādīsies pārejošs. Lai reāli apšaubītu šo viedokli, varētu, piemēram, konstatēt, ka kāda simetrijas grupa dabā nerealizējas, bet nespēja izrauties no izziņas procesa, kuras ietvaros mums matērija vienmēr parādīsies kā neizsmeļama, mums droši vien liks atteikties no šī nolūka.

Tai vietā, lai uztvertu D un H kā sholastiskus izteicienus, mēs ar to gribam izteikt estētisku un reizē metodoloģisku prasību matemātikai, lai viss tajā parādītos pēc dabas "ģīmja un līdzības".

Matemātiķu starpā jau sen ir izvērsusies diskusija, kāda ir matemātikas loma izziņas procesā [7]. Angļu matemātiķis H.Hārdi pirmais izvirzīja lozungu par matemātiku bez pielietojumiem. Var izdalīt, mūsuprāt, trīs galvenos viedokļus, kuri valda matemātikā vēl šodien:

1) tīrajai matemātikai nav nekā kopēja ar lietišķo matemātiku, un tai ir jāiet savs ceļš;
2) matemātikai savās tendencēs jāiet pret iedalījumu tīrajā un lietišķajā;
3) matemātikai, neatkarīgi no iedalījuma, jākalpo lietišķajiem uzdevumiem.

Skaitļošanas matemātikai, mūsuprāt, jādod galvenie kritēriji matemātikas lomas izpratnei. Jau tāpēc vien, ka mēs nevaram iedomāties tādu matemātisku konstruktu, kurš, jāatceras, kļūst "taustāms" caur tā invariantiem, kurā nebūtu nekā izrēķināma. Izgatavojot abstraktu mašīnu kādas problēmas risināšanai, mēs kā aksiomu pieņemsim, ka pirmām kārtām mums jāņem vērā telpas īpašības, kurā šis rēķināmais lielums ir tās invariants. Vēl vairāk, pašai šai mašīnai ir jābūt šīs telpas invariantam tajā nozīmē, ka tā fiksē, rekonstruē šīs telpas invariantu. Tieši šis moments, mūsu izpratnē, noņem augstāk minētās diskusijas jēgu vismaz skaitļošanas matemātikas metodoloģijas laukā.

Augstāk izdarītais spriedums gan neko nepierāda, jo vienmēr pastāvēs šķirojums starp vispār izrēķināmo un to, kas praksē jārēķina. Tomēr, atgriežoties pie iepriekš teiktā par izteikumiem D un H, šajā spriedumā saskatīsim apstiprinājumu nepieciešamībai atdalīt metodoloģiju no pragmatikas.

Inkrementalitāte

Cilvēka psiholoģija sarežģītas shēmas akceptē unsistemātiski ļauj lietot labāk nekā vienkāršās un darbietilpīgās, piemēram, loģiskos izvedumus. Bet tajā pašā laikā subjektam radošās konstruēšanas procesā piemīt tendence ienest patvaļu šajā procesā. Tiešām, modernās programmēšanas valodas ar to "sintactical sugar" [21] ļauj lietot sarežģītas struktūras, piemēram, abstrakto datu hierarhijas, realizēt tās netriviālās programmās, un tas tiešām ir izdarāms "ērti", t.i.sekojot tikai intuīcijai.

Lai gan jebkura matemātiska problēma uzstājas ar savu "dabīgu mašīnu", kura tai ir unikāla, tā tiek sistemātiski "sadauzīta", lietojot vienkāršu algebraisko valodu, sastāvošu no trīs operācijām:secības, cikla un alternācijas [11]. Lai kaut cik situāciju glābtu, sekvenciālā valoda tiek aizstāta ar relāciju valodu, ievedot nedeterminismu (Grīss, Deikstra, Parnass [26, 27, 36]).Struktūrprogrammēšanas paradigma ir ar to izsmelta. Kāda vēl ir iespēja, ņemot vērā, ka vispār no šīs strukturizēšanas nepieciešamības izvairīties nevar.? Atbilde ir - ņemt jau gatavas matemātiskās teorijas. Ņemsim, piemēram, matemātisko semantiku Skota teorijas izskatā un jautāsim, vai šo elganto teoriju nevar pielietot ne tikai mūsu rēķināšanas procesa legalizācijai, bet arī tieši, izmantojot tās dažādos konstruktīvos elementus. Konstatēsim, ka tas ir jau izdarīts un atspoguļots programmēšanas valodu semantikas definēšanas metodoloģijās, proti, Oksfordas skolā un Vīnes metodē(Viena Development Method) [19, 20, 30] .Teiksim dažus vārdus par pēdējo. Uzskaitīsim VDM domēnu konstruēšanas paņēmienus, kuri ir Skota teorijas attiecīge paņēmienu "sintactical sugar":

1)iezīmēts Dekarta reizinājums;
2)disjunkts vairāku domēnu apvienojums;
3)(galīgu) kopu konstruktors;
4)virkņu konstruktors;
5)funkcionālais konstruktors;
6)pietiekamības (optionality) konstruktors, kurš pieļauj tukšo objektu kā objekta iespējamību.

VDM šie paņēmieni tiek apvienoti zem jēdziena "abstraktā sintakse".

Mūs interesē, kā rekonstrukcija vispārējā veidā, kādu dod Klīni teorēma, transformējas uz abstrakto sintaksi un objektiem.

Mēģināsim mūsu izpratni atspoguļot tā. Lai lielākā daļa mūsu sistēmas jau eksistē un mūsu programmējamais produkts ir papildinājums tai. Programmas īpašību ~~tikt~~ būt"neierobežoti" palielināmai sauksim par inkrementalitāti. Šādu inkrementalitāti daļēji garantē jau VDM augstā strukturizācijas pakāpe.Kā atpazīt visus iespējamos ciklus šādās sistēmās? Protams, tas nav iespējams. Sistēmas pieeja ir šāda. Jau eksistējošā sistēmas daļa uzstājas ar savu stāvokli, kuru mēs nedrīkstam izjaukt, bet tikai papildināt. No katra moduļa, ko ienesīsim sistēmā, prasīsim apmierināt kādu unversālu īpašību (net effect property)[28].šo universālo īpašību fiksēsim ar datu invariantu palīdzību. Tātad lielās sistēmās, kur cikla atpazīšana kļūst neiespējama, cikla invarianta rekonstrukcija mums būs jāaizvieto ar attiecīgo datu invariantu rekonstrukciju, kas izpaudīsies acīmredzot to algebraiskās īpašībās.

Tehnologiskās sistēmas

Matemātiķim, it sevišķi programmu izstrādāšanas jomā, ir jādarbojas ar slēgtām sistēmām, kuras mēs sauksim par tehnologiskām sistēmām. Ar tehnologisku sistēmu mēs sapratīsim tādu slēgtu sistēmu, kurā ir izslēgti, t.i. nedarbojas, viens vai vairāki jebkurā vaļējā sistēmā darbojošies likumi. Piemēri šādām sistēmām varētu būt sākot ar bezsvara stāvokli kosmosa kuģī un beidzot ar ledusskapi. Ar šo definīciju mēs gribam izteikt to, ka sistēmai nevar piešķirt jaunus ,nekur neesistējošus likumus, jo tas nav iespējams. No šejienes seko metodologisks secinājums: katrai eksistējošai tehnologiskai sistēmai, lai mēs to labāk izprastu, jānosaka šo izslēgto likumsakarību kopums.

Iedomāsimies programmēšanu kā procesu, kur sākumā ir iespējami visi stāvokļi, bet reizē tas ir arī neinformatīvs. Ievedīsim aizliegumus, vienu pēc otra, tādā veidā aizvien sašaurinot iespējamo stāvokļu apgabalu. Tāda ir unversālo algebru pieeja [20, 22, 42] .šo procesu varētu nobeigt tad, kad būtu palicis viens vienīgs stāvoklis - problēmas atrisinājums. Tas, izrādās, nav konstruktīvi. Var apstāties jebkurā solī, un par atrisinājumu atzīt vienu no stāvokļiem. Par tādu izvēlas stāvokli, kurš kategoriāli ir visbrīvākais, t.i. satur minimumu iekšējo likumsakarību. Sākotnējo algebru (initial agebra) pieeja ir izrādījusies ļoti produktīva un,bez teorētiskā nozīmīguma, ir devusi vēl specifikāciju valodu CLEAR [21],augsta līmeņa programmēšanas

valodu OBJ [25] un jaunu programmēšanas tehnoloģiju – parametrizēto programmēšanu [24], kura kā jebkura programmēšanas tehnoloģija nav atkarīga no programmēšanas valodas.

Par optimizāciju

Atgriezīsimies pie struktūrprogrammēšanas paradigmas. Tātad, jebkura funkcija programmējot ir jāsaskalda programmēšanas valodas instrukcijās. Kādu lomu šai procesā spēlē optimizācija? Mūsuprāt, vairāk negatīvu. Ja pati problēma prasa savu dabīgo optimālo strukturizējamību, tad to mēs varam atklāt tikai "no augšas uz leju", t.i. izpildot šo optimalitātes meklējumu tajā virzienā, kā projektējam. Ļoti vienkārši runājot, visdabiskākais funkcijas realizācijas veids ir arī visoptimālākais[*]. Protams, praktiski šāds konstatējums mums gandrīz neko nedod. Tas ļauj tomēr izdarīt ļoti svarīgu metodoloģisku secinājumu: ir jāoptimizē tad, kad to prasa skaitļotāja ierobežotie resursi, un, otrādi, nav attaisnojama optimizācija, ja pēc tās attiecīgajā momentā nav izšķirošas nepieciešamības. Citādi runājot, pamatojums ir vajadzīgs optimizācijai un nevis otrādi. No tā tbiski izriet, piemēram, ka ar optimizāciju nedrīkst upurēt programmu skaidrību un atvieglotu saprotamību.

---

[*] Šī tēze ir pamatīgi analizējama un, iespējams, var būt tikai vadmotīvs līdzīgi iepriekš aplūkotajiem.

NOBEIGUMS

Šajā referātā mēginājām aplūkot cikla invarianta vietu skaitļošanas matemātikā un noskaidrojām, ka tas nav lokalizējams kādā vienā tās disciplīnā vai programmēšanas tehnoloģijā. Ja programmēšanas process jāveic lejupejošā virzienā, tad radošās konstruēšanas subjekts izgatavo programmu pretējā virzienā, t.i. no apakšas uz augšu. Vienīgā universālā pieeja ir balstīties uz pareiziem (saprātīgiem) metodologiskiem pieņēmumiem, kurus mēs varam iegūt tikai tad, ja mums ir pietiekoši skaidra vispārējā aina. Reāli šo ainu jau veido radošais evolūcijas process pats par sevi, jo šajā radošās konstruēšanas procesā piedalās milzīgs daudzums matemātiķu, un katra jauna programmēšanas tehnologija uzstājas ar savu šī procesa metodologisko izpratni. Mūsu uzdevums ir tās atpazīt.

LITERATŪRA

1. Богомолов А.С. Античная философия. Изд.МГУ,М.,1985.
2. Вигнер Е. Этюды о симметрии. Мир,М.,1971.
3. Вирт Н. Систематическое программирование.Введение. Мир. М.,1977.
4. Гейзенберг В. Физика и философия. Изд.иностр.лит.М.,1963.
5. Декарт Рене. Избранные произведения. Гос.изд.полит.лит. М.,1950.
6. Дирак П.А.М. Пути в физике. Атомэнергоиздат.М.,1983.
7. Клайн М. Математика. Утрата определенности. Мир,М.,1984.
8. Комацу Мацуо. Многообразие геометрии. Знание.М.,1981.
9. Кондаков Н.И. Логический словарь-справочник.Наука,М.,1976.
10. Косовский Н.К. Элементы математической логики и ее приложения к теории субрекурсивных алгоритмов.ЛГУ,Л.,1981.
11. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования. Мир,М.,1982.
12. Манна З. Теория неподвижной точки программ, в кн.Кибернетический сборник.Новая серия,вып.15,Мир,М.,1978.
13. Материалисты древней Греции. Собрание текстов Гераклита, Демокрита и Эпикура. Госизд.полит.лит.,М.,1955.
14. Мостепаненко А.М. Пространство-время и физическое познание. Атомиздат,М.,1975.
15. Скотт Дана. Теория решеток, типы данных и семантика, в кн.Данные в языках программирования,Мир,М.,1982.
16. Философский энциклопедический словарь.Советская энциклопедия,М.,1983.

17. Dekarts Renē. Pārruna par metodi.Zvaigzne,R.,1978.

18. Aho A., Hopcroft J., Ullman J. The analysis and design of computer algorithms.Addison-Wesley P.C.,1974.
19. Bjørner Dines, Jones Cliff B. Formal specification & software development, Prentice-Hall Inc.Englewood Cliffs,1982.
20. Blikle Andrzej. Notes of the mathematical semantics of programming languages, Warszawa,1981.
21. Burstall R.M., Goguen J.A. An informal introduction to specifications using CLEAR, in The correctness problem in computer science,ed.R.S.Boyer,J.S.Moore, Academic Press,London,1981.

22. Burstall R.M., Goguen J.A. Algebras, theories and freeness: an introduction for computer scientists, in Proc. 1981 Marktoberdorf NATO Summer School, Reidel,1982.

23. Floyd Robert W. The paradigms of programming. Comm.ACM, Vol.22,No 8,1979,424-436.

24. Goguen Joseph A. Parameterized programming.IEEE tr.Soft.Eng. Vol SE-10,No 5,1984,528-543.

25. Goguen J.A., Meseguer J., Plaisted D. Programming with parameterized abstract objects in OBJ, in Theory and practice of software techology,North-Holland,1982.

26. Gries David. Educating the programmer: notation, proofs and the development of programs. Information Processing 80,North-Holland Publ.Co.1980,935-944.

27. Gries David. The science of programming.Springer Verlag, N.Y.,1981.

28. Henderson Peter. Functional programming. Application and implementation, Prentice-Hall Inc.,Englewood Cliffs,1980.

29. Hoare C.A.R. The Emperor's old clothes,Comm.ACM,Vol.24,No 2, 1981,75-83.

30. Jones C.B. Software development: a rigorous approach, Prentice-Hall Inc.,Englewood Cliffs,1980.

31. Kowalski Robert. Algorithm = logic + control,Comm.ACM, Vol.22,No 7,1979,424-436.

32. Liskov Barbara. Modular program construction using abstractions.LNCS,Springer Verlag,No 86,1980;354-389.

33. Manna Zohar, Waldinger Richard. The logic of computer programming.IEEE tr.Soft.Eng.,Vol.SE-4,No 3,1978,199-229.

34. Naur Peter. Formalization in programming development. BIT, Vol.22,1982,437-453.

35. Parnas D.L. A technique for software module specification with examples,Comm.ACM,Vol.15,No 5,1972,330-336.

36. Parnas David Lorge. A generalized control structure and its formal definition,Comm.ACM,Vol.26,No 8,1983,572-581.

37. Reynolds John C. The craft of programming.Prentice-Hall inc. Englewood Cliffs,1981.

38. Turner D.A. Recursion Equations as programming language, in Functional Programming, ed.Darlington et al.

39. Wand M. Induction, recursion and programming.1980.
40. Warner Rex. The Greek philosophers.A Mentor Book.N.Y.
41. Warren David. Logic programming and compiler writing, Soft.Pract.Exp.,Vol.10,1980,97-125.
42. Zilles Stephen N. An introduwtion to data algebras.LNCS, Springer Verlag,No 86,1980,248-272.