ლოგიკური დაპროგრამება
1-2 ლექცია

შესავალი

ლოგიკური პროგრამა არის რაიმე ლოგიკური აღწერა, რომელშიც ჩადებულია რაიმე ოპერაციული სემანტიკა და ამიტომ იგი შეიძლება შესრულდეს კომპიუტერზე. როცა ეს ოპერაციული სემანტიკა კარგად მოფიქრებულია, მაშინ ლოგიკური პროგრამის შესრულებას აქვს შემდეგი თვისებები – იგი კორექტულია და ეფექტური. კორექტულია ნიშნავს, რომ პროგრამის შესრულება ითვალისწინებს ლოგიკურ სემანტიკას ე.ი. გამოთვლის ყველა შედეგი არის აქსიომათა რაიმე სიმრავლის სწორი შედეგი. ეფექტურია ნიშნავს, რომ შესაძლებელია დაიწეროს პროგრამები, რომლებიც შესრულდებიან ნავარაუდევ დროში და გამოიყენებენ ნავარაუდევ მეხსიერების მოცულობას.

პირველ რიგში დავაზუსტოთ ლოგიკა, რომელშიც ჩვენ დავწერთ ლოგიკურ პროგრამებს. არსებობენ სხვადასხვა ლოგიკები. მაგალითად, წინადადებათა(პროპოზიციული) აღრიცხვის ლოგიკა. პროპოზიციული ფორმულა შედგება გამოსახულებებისაგან, რომლებიც აკავშირებენ ერთიმეორესთან სიმბოლოებს როგორიცაა p, q, r და ა.შ.. ამ სიმბოლოებს ჰქვიათ ატომები. ატომები ერთიმეორესთან დაკავშირებულია მაკავშირებლებით (კონექტორებით). კონექტორებია: ¬(უარყოფა), ∧(კონიუნქცია), ∨(დეზიუნქცია), ↔(ექვივალენტობა) და →(იმპლიკაცია). პროპოზიციული ლოგიკა არის სუსტი ლოგიკური დაპროგრამებისათვის, რადგან მისი საშუალებით არ შეგვიძლია გამოვსახოთ მონაცემთა სტრუქტურები. უფრო სასურველია გამოვიყენოთ პირველი რიგის პრედიკატთა აღრიცხვა. პირველი რიგის პრედიკატთა აღრიცხვა აფართოებს პროპოზიციულ ლოგიკას ცვლადების, თერმებისა და ქვანტორების შემოტანით.

გავიხსენოთ ზოგიერთი ცნებები პირველი რიგის პრედიკატთა აღრიცხვიდან. ლოგიკური ფორმულა განისაზღვრება შემდეგი გრამატიკით:

<f> ::= <a>
| <x> = f(l_1 : <x>$_1$, . . . , l_n : <x>$_n$)
| <x>$_1$ =<x>$_2$
| <f>$_1$ → <f>$_2$ | <f>$_1$ ∧ <f>$_2$ | <f>$_1$ ∨ <f>$_2$ | <f>$_1$ ↔ <f>$_2$ | ¬ <f>
| ∀ <x> <f> | ∃<x> <f>
<a> ::= p(<x>$_1$, . . . , <x>$_n$)

1.1 დიაგრამა

სადაც <a> არის ატომი და <f> არის ფორმულა. <x> არის ცვლადი, l_i (i=1, …,n) არის თვისება, f არის თერმის ჭდე, p არის პრედიკატული სიმბოლო, ∀ და ∃ არიან ქვანტორები. ლოგიკურ ფორმულაში რაიმე ცვლადს,

რომელიც გამოიყენება რაიმე ქვანტორში და მის განსაზღვრის არეშია ჰქვია დაბმული ცვლადი, ხოლო ყველა სხვა ცვლადებს ჰქვიათ თავისუფალი ცვლადები. ლოგიკურ ფორმულას, რომელიც არ შეიცავს თავისუფალ ცვლადებს ჰქვია ლოგიკური წინადადება. მაგალითად, $\forall x.\exists y.(a(x, y)\wedge b(y))$ არის ლოგიკური წინადადება, ხოლო - $\exists y.\ a(x, y)\wedge b(y)$ არა. რომ დავუნიშნოთ პრედიკატული ლოგიკის რაიმე წინადადებას ჭეშმარიტული მნიშვნელობა, უნდა განვსაზღვროთ ლოგიკური მოდელი, რომელიც ძალზე განსხვავებულია გამოთვლების მოდელისაგან. იგი შედგება ცვლადების შესაძლო მნიშვნელობებისაგან, რომლებსაც ეწოდებათ საუბრის არე და დამოკიდებულებისაგან. ყოველ პრედიკატს აქვს დამოკიდებულება, რომელიც არის წყვილების სიმრავლე და რომლებისთვისაც პრედიკატი არის ჭეშმარიტი. წყვილის კომპონენტები აღებულია საუბრის არედან. ლოგიკური მოდელი არჩეულია ისე, რომ ზოგიერთი წინადადებები არიან ყოველთვის ჭეშმარიტი და მათ ეწოდებათ აქსიომები. ასეთ მოდელს ეწოდება აქსიომათა ლოგიკური სემანტიკა. საზოგადოდ არსებობს მრავალი მოდელი, რომელთათვისაც აქსიომები არიან ჭეშმარიტი. განვიხილოთ მაგალითი: აქსიომებად ავიღოთ შემდეგი წინადადებები:

$\forall x,y$. მეტია-ან-ტოლი$(x,\ y) \leftrightarrow \exists z.$ მეტია-ან-ტოლი$(x,z) \wedge$ მეტია-ან-ტოლი(z, y)
$\forall x, y, z$. მეტია-ან-ტოლი $(x, y) \wedge$ მეტია-ან-ტოლი$(y,x) \rightarrow x = y$

ამ აქსიომებისათვის არსებობს მრავალი მოდელი. ავიღოთ ერთ-ერთი მათგანი:

საუბრის არე: (a, b) ნამდვილ რიცხვთა ღია ინტერვალი, სადაც a და b ფიქსირებული ნამდვილი რიცხვებია.
მეტია-ან-ტოლი დამოკიდებულება: $\{$მეტია-ან-ტოლი$(x,y) \mid x >= y$ და $x, y \in (a,b)\}$
ტოლობის დამოკიდებულობა: $\{x = x \mid x \in (a,b)\}$

დამოკიდებულებები შეიცავენ მხოლოდ იმ წყვილებს, რომელთათვისაც დამოკიდებულება ჭეშმარიტია. ყველა სხვა წყვილებისათვის დამოკიდებულება მცდარია. ამ მოდელის საშუალებით ჩვენ შეგვიძლია დავადგინოთ ჭეშმარიტობის მნიშვნელობა პრედიკატული აღრიცხვის ნებისმიერი წინადადებისათვის.
ლოგიკური პროგრამა ჩვენ შეგვიძლია განვიხილოთ ორი სხვადასხვა თვალსაზრისით - ლოგიკური და ოპერაციული თვალსაზრისით. ლოგიკური თვალსაზრისით იგი არის ლოგიკური ინსტრუქცია, რომელიც არის ჭეშმარიტი ან მცდარი. ოპერაციული თვალსაზრისით ლოგიკური პროგრამა არის ინსტრუქცია, რომელიც განსაზღვრავს, თუ როგორ გამოითვლება იგი კომპიუტერზე. ამიტომ ლოგიკური პროგრამის ყოველ ინსტრუქციას ჩვენ შეგვიძლია შევუსაბამოთ ლოგიკური ინსტრუქცია, რომელიც განსაზღვრავს მისი ჭეშმარიტობის მნიშვნელობას. ლოგიკური

პროგრამის დაწერა შედგება ორი ნაწილისაგან – დავწეროთ ლოგიკური სემანტიკა და შემდეგ ავირჩიოთ რაიმე ოპერაციული სემანტიკა მისთვის. ლოგიკური დაპროგრამების ხელოვნება შედგება იმისაგან, რომ დავაბალანსოთ ლოგიკური პროგრამის ლოგიკური და ოპერაციული სემანტიკა ისე, რომ ლოგიკური სემანტიკა იყოს ადვილად გასაგები და ოპერაციული სემანტიკა იყოს ეფექტური.

3-4 ლექცია

ახლა ჩვენ შეგვიძლია დავაზუსტოთ თუ რა არის ლოგიკური დაპროგრამება. ლოგიკური პროგრამა შედგება აქსიომების სიმრავლისაგან, რაიმე მოდელისაგან ამ აქსიომებისათვის და რაიმე წინადადებისაგან, რომელსაც ჰქვია შეკითხვა (მოთხოვნა). არსებობს სისტემა, რომელსაც ჰქვია თეორემის დამამტკიცებელი და მას შეუძლია შეასრულოს ლოგიკური პროგრამა და ამ პროგრამიდან გამოიყვანოს შეკითხვის ჭეშმარიტული მნიშვნელობა. სისტემა რომ იყოს სასარგებლო მისი გამოყვანა უნდა იყოს კონსტრუქციული, ე.ი. თუ შეკითხვა აცხადებს, რომ არსებობს ისეთი x, რომელიც აკმაყოფილებს რაიმე ფორმულას, მაშინ სისტემამ უნდა ააგოს x–ის შესატყვისი მონაცემთა სტრუქტურა. თეორემის ზოგადი დამამტკიცებელი, რომელსაც შეეძლოს დაამტკიცოს ნებისმიერი სწორი შედეგი არ არსებობს. ეს გამომდინარეობს გეოდელის არასისრულის ცნობილი თეორემიდან. ე.ი. არსებობს სწორი ინსტრუქცია, რომელიც არ შეიძლება გამოყვანილ იქნეს აქსიომათა მოცემულ სისტემიდან. ამ თეორემიდან გამომდინარე ჩვენ უნდა დავადოთ ზოგიერთი შეზღუდვები ლოგიკურ პროგამებსა და თეორემათა დამამტკიცებელზე, რომ გავხადოთ ლოგიკური დაპროგრამება შესაძლებელი. ჩვეულებრივ ასეთი შეზღუდვებია: მხოლოდ ზოგიერთი სახის შეზღუდვებია დასაშვები და დასაშვებია, რომ თეორემათა დამამტკიცებელი იყოს არასრული. ე.ი. არსებობდეს ისეთი შეკითხვები, რომლებსაც თეორემათა დამამტკიცებელი ვერ ამტკიცებს. ჩვენ შემთხვევაში ნაგულისხმევია მხოლოდ მეორე შეზღუდვა. დასაშვებია ნებისმიერი სახის აქსიომები და შეკითხვები, მაგრამ მოდელი არის საკმაოდ სუსტი როგორც თეორემათა დამამტკიცებელი. პროგრამისტის პასუხისმგებლობის ქვეშ არის თუ როგორ უნდა შესრულდეს გამოყვანა. ამის გამო ლოგიკური და ოპერაციული სემანტიკები განსხვავდებიან ერთიმეორესაგან. ოპერაციული სემანტიკა არის მარტივი და გამოცნობადი ისე, რომ პროგრამისტს შეუძლია განსაზღვროს ალგორითმები. გამოყვანის პროცესის მეგზურობა დაპროგრამების ენის ტერმინებში ნიშნავს, განვსაზღვროთ ალგორითმი, რომელიც შესრულებული იქნება გამომთვლელი მოდელის მიერ. პრაქტიკული დაპროგრამების ენის თვალსაზრისით ეს არის უფრო სასურველი ვიდრე თეორემათა დამამტკიცებელი. რაიმე დაპროგრამების ენა ძლიერ განსხვავდება თეორემათა დამამტკიცებელისაგან. იმისათვის, რომ სრულად განისაზღვროს ლოგიკური დაპროგრამება ჩვენ დაგვრჩა

განვსაზღვროთ გამომთვლელი მოდელი, რომელსაც განვსაზღვრავთ შემდეგ პარაგრაფებში.

დეკლარაციული გამომთვლელი მოდელი

დაპროგრამება შედგება სამი ნაწილისაგან:

1. გამომთვლელი მოდელი, რომელიც არის ფორმალური სისტემა და განსაზღვრავს თუ როგორ შევასრულოთ გამოთვლები. ჩვენი გამომთვლელი მოდელი შედგება მონაცემთა ტიპებისა და მათ მნიშვნელობათა სიმრავლისაგან, ოპერაციათა სიმრავლისაგან და ენისაგან, რომელიც განსაზღვრავს პროგრამებს ოპერაციათა გარკვეული მიმდევრობის შესასრულებლად. ასეთ ენას უნდა ჰქონდეს ზუსტად განსაზღვრული სინტაქსი და სემანტიკა.
2. დაპროგრამების ხერხებისა და დაპროექტების პრინციპებისაგან, რომლებიც გამოსახავენ კონკრეტულ ამოცანებს პროგრამების სახით გამომთვლელი მოდელის ენაზე და მას ეწოდება დაპროგრამების მოდელი.
3. დამტკიცების ტექნიკა, რომელიც საშუალებას გვაძლევს ვიმსჯელოთ შედგენილი პროგრამის კორექტულობაზე, გავზარდოთ იმის რწმენა, რომ პროგრამა ასრულებს იმას, რაც ჩაფიქრებული იყო ალგორითმით და გამოთვლები ეფექტურად სრულდება.

დაპროგრამების ენის განსაზღვრა

დაპროგრამების ენის განსაზღვრა მოიცემა ფორმალური სისტემის სახით. სინტაქსმა უნდა განსაზღვროს ისეთი პროგრამები, რომლებიც შეიძლება წარმატებით შესრულდეს კომპუტერზე. პირველ რიგში ჩვენ განვსაზღვრავთ თუ რისგან შედგება პროგრამა და როგორ უნდა ჩაიწეროს სწორი პროგრამის ტექსტი. ჩვენ ვიგულისხმებთ, რომ პროგრამა არის ფორმალური ენის წინადადება, რომელიც შედგება სიმბოლოთა მიმდევრობისაგან. იგულისხმება, რომ ზოგი სიმბოლო ცნობილია აპრიორულად ფორმალურ სისტემაში, ხოლო დანარჩენი კი განისაზღვრება სინტაქსის გარკვეული წესებით. აპრიორულად ცნობილ სიმბოლოებს ჩვენ ვუწოდებთ ტერმინალურ სიმბოლოებს, ხოლო დანარჩენს – არატერმინალურ სიმბოლოებს. სიმბოლო არის ასონიშნების მიმდევრობა, რომლის საშუალებით იგი ცალსახად განისაზღვრება. გრამატიკა არის წესების მიმდევრობა, რომელიც საბოლოოდ განსაზღვრავს პროგრამას(წინადადებას), როგორც ამ გრამატიკის საწყის სიმბოლოს. დაპროგრამების ენის განსასაზღვრავად ჩვეულებრივ გამოიყენება კონტექსტისაგან თავისუფალი გრამატიკები. ვისაც სურს უფრო დაწვრილებით გაეცნოს ფორმალურ გრამატიკათა თეორიას ვურჩევ შემდეგ ლიტერატურას: A.Aho, J.Ullman. The Theory of Parsing, Translation and Compiling, Vol.1,

Parsing, 1972(არსებობს რუსული თარგმანი). გრამატიკის წესების ჩასაწერად ჩვენ გამოვიყენებთ Backus-Naur-ის გაფართოებულ ფორმას EBNF. დაპროგრამების ენად ჩვენ გამოვიყენებთ Oz ენას. იგი არის დაპროგრამების მრავალპარადიგმიანი თანამედროვე ენა, რომელიც საშუალებას გვაძლევს განვახორციელოთ დაპროგრამების სხვადასხვა მოდელები და იგი რეალიზებულია სხვადასხვა პლატფორმებზე., კერძოდ, WINDOWS ოპერაციულ სისტემაში. მისი სრული დოკუმენტაცია, ტრანსლირებული კოდი და საინსტალაციო ინსტრუქცია შეგიძლიათ გადმოტვირთოთ http://www.mozart-oz.org/ ან http://www.ps.uni-sb.de/ საიტიდან.
პირველ რიგში ჩვენ დავიწყებთ დაპროგრამების მიმდევრობითი სტილის აღწერით. დაპროგრამების მიმდევრობითი სტილის დროს იგულისხმება, რომ ინსტრუქციები სრულდება მიმდევრობით, ე.ი. იმ რიგით, რა რიგითაც ისინი გვხვდებიან პროგრამაში. დაპროგრამების ამ სტილს Oz–ში ჰქვია ძაფი(thread). ძაფს შეუძლია, როგორ ჩაწეროს ან შეცვალოს ინფორმაცია მეხსიერებაში, ასევე აიღოს ინფორმაცია მეხსიერებიდან. ძაფი მეხსიერებასთან კავშირს ამყარებს ცვლადების საშუალებით. ეს ცვლადები არიან ლოგიკური ცვლადები, რადგან მათ შეიძლება მიანიჭოთ ერთხელ მნიშვნელობა და ამ მნიშვნელობის შეცვლა შემდეგ აღარ შეიძლება. ინსტრუქციებს აქვთ წვდომა ინფორმაციასთან გამოსახულებების საშუალებით და მათი გამოთვლის შედეგად მიიღება მნიშვნელობა. გამოსახულების უმარტივესი სახეა კონსტანტები და ცვლადები. ამიტომ ენის აღწერას მათი აღწერით დავიწყებთ. კონსტანტები ერთიმეორესაგან განსხვავდებიან, როგორც კონკრეტული მნიშვნელობით ასევე ტიპებით. ტიპი არის მნიშვნელობათა კონკრეტული სიმრავლე. ცვლადის ტიპი მიუთითებს თუ რომელი სიმრავლიდან შეიძლება იყოს აღებული ამ ცვლადის მნიშვნელობა. ცვლადი პროგრამაში შემოიტანება მისი სახელისა და განსაყღვრის არის მითითებით. სახელით განსაზღვრული ცვლადი პროგრამაში ცნობილია მხოლოდ მისი განსაზღვრის არის შიგნით და მისი გამოყენება განსაზღვრის არის გარეთ იწვევს პროგრამაში შეცდომას. პროგრამაში ცვლადების შემოტანა ხდება სპეციალური ინსტრუქციებით. ცვლადის განსაზღვრა, მეორენაირად მისი გამოცხადება არ ნიშნავს, რომ ამ ცვლადს უკვე გამოეყო ადგილი მეხსიერებაში და აქვს მნიშვნელობა. ცვლადისათვის მნიშვნელობის მინიჭება (ინიციალიზაცია) ხდება სპეციალური ინსტრუქციებით და ამ დროს ხდება მისთვის ადგილის გამოყოფა. ამავე დროს ხდება მისი ტიპის დადგენა მისი მნიშვნელობის ტიპის მიხედვით. ცვლადისთვის გამოყოფილი მეხსიერების მოცულობა განისაზღვრება მნიშვნელობის ტიპით. მეხსიერების ასეთ განაწილებას ჰქვია მეხსიერების დინამიკური განაწილება, რადგან ცვლადისათვის მეხსიერების გამოყოფა ხდება პროგრამის შესრულების პროცესში. თუ ორ ან რამოდენიმე ცვლადს აქვს ერთიდაიგივე მნიშვნელობა, მაშინ ისინი მიუთითებენ ერთსადაიგივე

ადგილს მეხსიერებაში. ასეთი მიდგომა იძლევა მეხსიერების ეკონომიას. Oz ენაში ცვლადის სახელი იწყება დიდი ასოთი ან ქვედა ხაზით.

5-6 ლექცია

Oz ძირეული ენის ინსტრუქციები და მნიშვნელობების ტიპები

Oz ენის განსაზღვრას ჩვენ დავიწყებთ ძირეული ენის განსაზღვრით და შემდეგ მას გავაფართოებთ იმ იმ ინსტრუქციებით, რომლებიც აუცილებელია ლოგიკური დაპროგრამების განსახორციელებლად. Oz ენის გაფართოება გაკეთებულია იმ მეთოდებით, რომლებსაც ჰქვიათ ლიგვისტური გაფართოება და სინტაქსური შემოკლებები. ლინგვისტური გაფართოება გულისხმობს, ძირეულ ენას დაემატოს ახალი ინსტრუქციები და ეს ინსტრუქციები აღიწერება ძირეული ენის ინსტრუქციების საშუალებით. შემდეგ ეს აღწერები გამოიყენება ძირეული ენის ინსტრუქციებისა და დამატებული ინსტრუქციების ხელმეორედ კომპილაციისათვის, რის შედეგადაც მიიღება გაფართოებული კომპილატორი. ასეთი პროგრამა გააჩნია MOZART სისტემას. ასე, რომ ვისაც შეუძლია ამ პროგრამის გამოყენება მას შეუძლია Oz ენის გაფართოება და გაფართოებული ენის გამოყენება შემდგომში დაპროგრამებისათვის. სინტაქსური შემოკლებები გულისხმობს ინსტრუქციების შემოკლებულ ჩაწერას და იმის მითითებას თუ ამ შემოკლებულ ჩაწერას Oz ენის რა ტექსტი შეესატყვისება. ასეთი პროგრამა მაკროგენერატორის მიერ გადაიყვანება გაფართოებულ ჩაწერაში და შემდეგ მოხდება მისი კომპილაცია. ორივე მეთოდი გამოყენებულია Oz ძირეული ენის გასაფართოებლად. ჩვენ დავიწყებთ Oz ძირეული ენის აღწერას მისი გრამატიკის წესებისა და მნიშვნელობის განსაზღვრის დიაგრამის მოცემით. შემდეგ გავეცნობით ძირეული ენის ოპერაციულ სემანტიკასა და დაპროგრამების ტექნიკას.

ძირეული ენა. ძირეული ენის ინსტრუქციის განმარტებას აქვს შემდეგი სახე:

<s> ::= $\langle s \rangle_1 \langle s \rangle_2$
| X = f(l1 : Y1 … ln : Yn)
| X = <რიცხვი>
| X = <ატომი>
| X = <ლმ>

| {NewName X}
| X = Y
| **local** X1 . . . Xn **in** <s> **end**
||| **proc** { X Y1 . . . Yn} <s> **end**
| **{** X Y1 . . . Yn **}**
| **{** NewCell Y X }
| { Exchange X Y Z }
| { Access X Y }
| **if** B **then** $<s>_1$ **else** $<s>_2$ **end**
| **thread** <s> **end**
| **try** $<s>_1$ **catch** X **then** $<s>_2$ **end**
| **raise** X **end**

3.1 დიაგრამა

სადაც <s>, $<s>_1$ და $<s>_2$ აღნიშნავენ ინსტრუქციის ცნებას, X, Y და Z ცვლადებია, B ლოგიკური გამოსახულებაა, ხოლო გამუქებული სიტყვები აღნიშნავენ გასაღებ სიტყვებს. f, l1, . . . , ln სახელებია, ხოლო <რიცხვი>, <ატომი> და <ლმ> შესაბამისად რიცხვის, ატომისა და ლოგიკური მნიშვნელობის ცნებებია, რომლებსაც განვმარტავთ მოგვიანებით.

Oz პროგრამული გარემო (OPI)
OPI-ს ვუწოდებთ იმ გარემოს, რომელშიც ხდება Oz პროგრამების შექმნა. ამ გარემოს გამოძახება WINDOWS სისტემაში ხდება Oz პროგრამის გამოძახებით, რაც გამოიძახებს Oz სისტემას. ეკრანზე გამოვა ფანჯარა, რომელიც არის EMACS ტექსტური რედაქტორის ფანჯარა და მასში შეგვიძლია ავკრიფოთ პროგრამის ტექსტი, გავუშვათ კომპილაციაზე და შემდეგ კომპილირებული პროგრამა გავუშვათ თვლაზე ან პროგრამის ტექსტი პირდაპირ გავუშვათ თვლაზე. ყველაფერი ეს კეთდება დიალოგურ რეჟიმში EMACS–ის ბრძანებების გამოყენებით. დიალოგური რეჟიმი ნიშნავს, რომ თქვენ შეგიძლიათ აკრიფოთ პროგრამის ტექსტი და გაუშვათ თვლაზე. შემდეგ დაუმატოთ ამ ტექსტს ინსტრუქციები და კვლავ გაუშვათ თვლაზე, მთელი ტექსტის თავიდან გადაწერის გარეშე და OPI–დან გამოუსვლელად. EMACS არის მრავალფანჯრიანი რედაქტორი და თქვენ შეგიძლიათ ერთ ფანჯარაში აკრიფოთ პროგრამის ტექსტი, მეორეში - მიიღოთ კომპილაციის შედეგი და მესამეში - თუ როგორ დამთავრდა თვლის პროცესი.

გაგრძელება

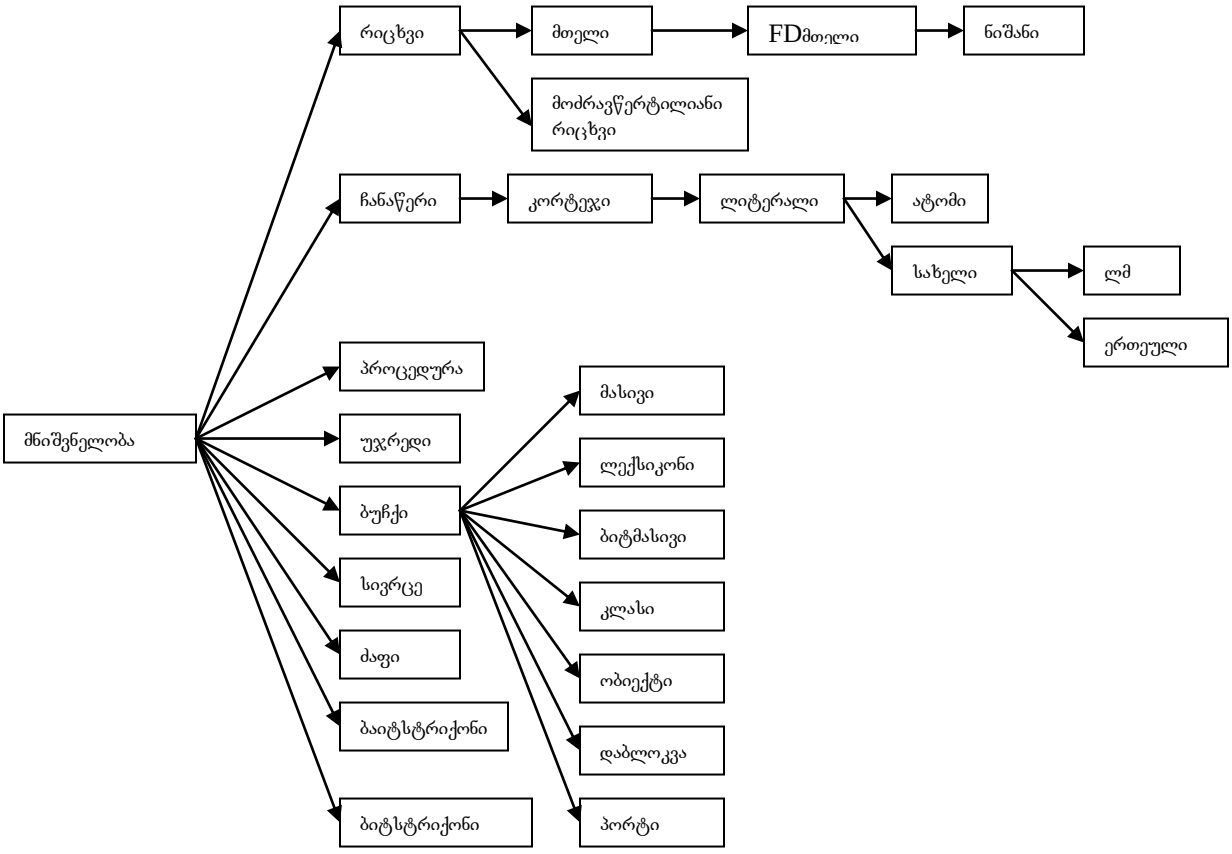
Oz-ში ინსტრუქციები სრულდებიან მიმდევრობით და შესრულების ასეთ პროცესს ჰქვია ძაფი.

```
local A B C in S end
```

ინსტრუქციის შესრულებას შემოჰყავს A, B და C ცვლადები, რომელთა განსაზღვრის არეა S ინსტრუქცია, ხოლო

```
declare A B C in S
```

ინსტრუქციას შემოჰყავს A, B და C ცვლადები, რომელთა განსაზღვრის არეა S ინსტრუქცია და ყველა სხვა მომდევნო ინსტრუქცია, რომელიც ხელმეორედ არ განსაზღვრავს ამ ცვლადებს. ამ შემთხვევაში A, B და C ცვლადებს ეწოდებათ გლობალური ცვლადები.



4.1 დიაგრამა. ტიპების იერარქია.

4.1 დიაგრამაზე წამოდგენილია Oz–ს ტიპების იერარქია. ყველაზე ზოგადი ტიპია მნიშვნელობა. ყველა სხვა ტიპები წარმოადგენენ მის ქვესიმრავლეებს. Oz არის დინამიკურად ტიპირებული ენა, რაც იმას

ნიშნავს, რომ ცვლადის ტიპი განისაზღვრება ცვლადის ინიციალიზაციის დროს. ე.ი. პროგრამის შესრულების დროს. ცვლადების ინიციალიზაცია ხდება გატოლების(=) ორადგილიანი ოპერატორის საშუალებით, რომელიც ახდენს მისი ოპერანდების სტრუქტურულ გატოლებას. ორი მნიშვნელობა ერთიმეორეს ტოლია თუ მათ აქვთ ერთნაირი სტრუქტურა, ერთერთის ტიპი წარმოადგენს მეორეს ქვეტიპს და მისი მნიშვნელობა მიიყვანება პირველის ტიპზე. მაგალითად, მთელი ტიპი მიიყვანება რიცხვი ტიპზე. ინსტრუქცია A=7 შესრულება ხდება შემდეგნაირად: თუ A დაუკავშირებელი ცვლადია, მაშინ იგი დაუკავშირდება მთელ რიცხვს 7, ხოლო თუ დაკავშირებელი ცვლადია, მაშინ მათი მნიშვნელობები უნდა იყვნენ ტოლნი და მთელი ტიპი უნდა იყოს A-ს ქვეტიპი. სხვა შემთხვევაში წარმოიშვება შეცდომა, რომელსაც ჰქვია განსაკუთრებული შემთხვევა.

რიცხვები.

როგორც 4.1 დიაგრამიდან სჩანს, რიცხვებს მიეკუთვნებიან მთელი რიცხვები, რიცხვები მოძრავი წერტილით და ნიშნები. FDმთელი-ს განვიხილავთ მოგვიანებით. რიცხვის უარყოფითი ნიშანი აღინიშნება ~ თი. რიცხვს დადებითი ნიშანი არ ეწერება. გამოიყენება მთელი რიცხვების ათობითი, თექვსმეტობითი, რვაობითი და ორობითი ჩაწერა. რვაობითი ჩაწერა იწყება O ასოთი, თექვსმეტობითი – Ox ან OX ასოებით. ნიშნები არიან მთელი რიცხვების ქვეტიპი და შეიცავენ 0-დან 255-ის ჩათვლით მთელ რიცხვებს. ნიშნებისათვის გამოიყენება ISO 8859-1 კოდირების სტანდარტი. ნებისმიერი ნიშანი შეიძლება ჩაიწეროს ასე - &\ოოო, სადაც ო არის რვაობითი ციფრი. &u აღნიშნავს u ასოს კოდს. ოპერაციები ნიშნებზე, მთელ რიცხვებზე და რიცხვებზე მოძრავი წერტილით მოთავსებულია სტანდარტული ბიბლიოთეკის მოდულებში Char, Int და Float შესაბამისად ან Number მოდულში. ავიღოთ შემდეგი პროგრამის ფრაგმენტი:

```
local A B C in
A =  ~3.5
C =  5.2e~2
B =  1
{Browse [A B C]}
end
```

ეს პროგრამა გამოიტანს ეკრანზე -3.5 0.052 1 რიცხვებს.

ლიტერალები

ლიტერალები არიან ელემენტალური სიდიდეები და წარმოადგენენ თავიანთ თავს. ლიტერალები შედგებიან ასოებისა და ციფრების მიმდევრობისაგან და იწყებიან პატარა ასოთი ან ნებისმიერი ბეჭდვადი ნიშნების მიმდევრობა მოთავსებული ერთმაგ ბრჭყალებში. მაგალითად,

```
ab  v35  '='  'programa #5'
```

შემდეგი ელემენტალური სიდიდეებია <სახელი>. მათი მიღება შეიძლება მხოლოდ ბიბლიოთეკის პროცედურით {NewName A}, სადაც A–ს მნიშვნელობა იქნება საყოველთაოდ უნიკალური სახელი, რომელიც არ დაემთხვევა სხვა სახელს.