ივ. ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტის
გამოყენებითი მათემატიკისა და კომპიუტერულ მეცნიერებათა ფაკულტეტის
კომპიუტერების მათემატიკური უზრუნველყოფისა და ინფორმაციული
ტექნოლოგიების კათედრა

დავით მიშელაშვილი

ინსტრუმენტული საშუალებები ბუნებრივი ენის ტექსტების სინტაქსური და მორფოლგიური ანალიზატორის შესადგენად

ფიზიკა-მათემატიკის მეცნიერებათა კანდიდატის სამეცნიერო ხარისხის
მოსაპოვებლად წარმოდგენილი

დ ი ს ე რ ტ ა ც ი ა

05.13.11 - გამოთვლითი მანქანების, სისტემების, კომპლექსებისა და ქსელების
მათემატიკური და პროგრამული უზრუნველყოფა

სამეცნიერო ხელმძღვანელი:
ფიზიკა-მათემატიკის მეცნიერებათა კანდიდატი,
უფროსი მეცნიერ თანამშრომელი ჯემალ ანთიძე

ს ა რ ჩ ე ვ ი

გამოყენებული ლიტერატურა .

შესავალი

ბუნებრივი ენის ტექსტების კომპიუტერული დამუშავება თანამედროვე საინფორმაციო ტექნოლოგიების ერთერთ მნიშვნელოვან მიმართულებას წარმოადგენს. მასში ჩვენ ვხვდებით ისეთი სახის ამოცანებს, როგორიცაა: მართლწერის ავტომატური შემოწმება; მანქანური თარგმანი; საძიებო სისტემები, რომელთა მეშვეობითაც შესაძლებელია სიტყვის სხვადასხვა ფორმების მოძებნა ბაზაში; ბუნებრივ ენაზე დიალოგურ რეჟიმში მომუშავე პროგრამები; ტექსტების მრავალმიზნობრივი მორფოლოგიური, სინტაქსური და სემანტიკური ანალიზი; ენის სწავლებისათვის განკუთვნილი სავარჯიშო ტრენაჟორები და სხვა. გამოთვლითი ტექნიკის დღევანდელი მასშტაბებით განვითარების პირობებში, ძალზედ მნიშვნელოვანია შეგვეძლოს ამგვარი ამოცანების სწორად და ეფექტურად გადაჭრა.

ნაშრომში განხილულია ბუნებრივი ენის ტექსტების სინტაქსური და მორფოლოგიური, კომპიუტერული ანალიზის პრობლემები. მათ გადასაჭრელად შემუშავებულია ახალი მიდგომები. შეიქმნა სპეციალური ფორმალიზმები ბუნებრივი ენის სინტაქსური და მორფოლოგიური წესების ჩასაწერად. მათი კომპიუტერზე რეალიზაციის შედეგად კი მიღებულ იქნა პროგრამული სისტემა, რომლის მეშვეობითაც შესაძლებელია ბუნებრივი ენის ტექსტების სინტაქსური და მორფოლოგიური ანალიზი. სინტაქსური ანალიზის შედეგად კომპიუტერი ადგენს საწყისი წინადადების სტრუქტურას, ურთიერთდამოკიდებულებებს მის შემადგენელ წევრებს შორის. მორფოლოგიური ანალიზატორი გამოიყენება სიტყვაფორმების მორფემებად დაშლისა და ამ დაშლის საფუძველზე განსახილველი სიტყვაფორმის გრამატიკული კატეგორიების დადგენისათვის. სინტაქსური და მორფოლოგიური კომპიუტერული ანალიზის შედეგად ხერხდება ბუნებრივ ენაზე ჩაწერილი წინადადების სტრუქტურის ფორმალიზაცია, მისი წარმოდგენა კომპიუტერისათვის მისაღები სახით.

თავდაპირველად, სანამ მოხდებოდა ახალი ფორმალიზმების შემუშავება და პროგრამული სისტემის რეალიზაცია, დაწვრილებით იქნა შესწავლილი თანამედროვე მიდგომები ბუნებრივ ენათა დამუშავების თეორიაში. განხილული იქნა სხვა მსგავსი სისტემების მუშაობის ძირითადი პრინციპები (PC PATR, PC PARSE [9]). ეს სისტემები გამოიცადა ქართული ენისათვის შედგენილი გრამატიკული

წესებით. დაწვრილებით იქნა გამოკვლეული მათი მუშაობის ძლიერი და სუსტი მხარეები, მათი ძირითადი ნაკლოვანებები ზოგიერთ ბუნებრივ ენასთან მიმართებაში.

პრაქტიკული ექსპერიმენტების შედეგად, აშკარა გახდა რომ ქართული ენისათვის და მრავალი სხვა ენებისათვის რომლებსაც ახასიათებთ მდიდარი მორფოლოგია, წინადადებაში სიტყვათა თავისუფალი წყობა და სინტაქსის სირთულე, გამოყენებული უნდა ყოფილიყო განსხვავებული მიდგომები. საჭირო იყო უფრო ზოგადი, მოქნილი და მძლავრი პროგრამული სისტემის შექმნა. პირველ რიგში კი აუცილებელი იყო შეგვემუშავებინა ახალი ფორმალიზმები, რომელთა ფარგლებშიც მოხერხდებოდა სხვადასხვა ბუნებრივი ენებისათვის გრამატიკის მაქსიმალურად ლაკონურად და გასაგებად ჩაწერა.

ძირითადი, ცნობილი მეთოდები, რომლებიც საფუძვლად უდევს ნაშრომში წარმოდგენილ სინტაქსური და მორფოლოგიური ანალიზის პროგრამებს, გახლავთ:

- თვისებათა სტრუქტურები და მათზე განსაზღვრული ოპერაციები (უნიფიკაცია, ტოლობაზე შემოწმება და სხვა)
- ზოგადი ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმი
- არადეტერმინისტული ძიების ალგორითმი (მორფოლოგიური ანალიზისათვის)
- შეზღუდვები

პროგრამაში გამოყენებული ალგორითმები ემყარება ამ ძირითად მეთოდებს, თუმცაღა ისინი წარმოადგენენ პრინციპულად ახალ ალგორითმებს.

უნდა აღინიშნოს, რომ ფორმალიზმების და პროგრამული სისტემის შემუშავება მიმდინარეობდა პარალელურად მათ პრაქტიკულ შემოწმებასთან ერთად. ამის შედეგად დროულად ხდებოდა შემუშავებულ ალგორითმებში სუსტი მხარეების აღმოჩენა და მათი გასწორება, იხვეწებოდა ფორმალიზმები მორფოლოგიის და სინტაქსის წესების ჩასაწერად, რათა ეს ფორმალიზმები უფრო მოსახერხებელი ყოფილიყვნენ ადამიანისათვის.

ნაშრომის მიზანი

ნაშრომის ძირითადი მიზანი გახლავთ პროგრამული სისტემის შექმნა, რომლის გამოყენებაც შესაძლებელი იქნება ბუნებრივი ენაზე ჩაწერილი ტექსტების სინტაქსური და მორფოლოგიური, კომპიუტერული ანალიზისათვის. ბუნებრივი ენის მაგალითად აღებულ იქნა ქართული ენა, მისი გრამატიკის წესები და ლექსიკური მარაგი. თავისთავად, ასეთი პროგრამული სისტემის შექმნის პროცესი შემდეგ ცალკეულ მიზნებად შეგვიძლია დავყოთ:

1. თეორიული ბაზის მომზადება. ახალი ალგორითმული მიდგომების შემუშავება.
2. ბუნებრივი ენის გრამატიკული წესების ჩასაწერი ფორმალიზმების შექმნა.
3. პროგრამული სისტემის რეალიზაცია, რომელიც ეფუძნება 1 და 2 პუნქტში მიღებულ შედეგებს.
4. რომელიმე ბუნებრივი ენისათვის ამ სისტემის პრაქტიკული გამოყენება.

პროგრამული სისტემა ორი ნაწილისაგან შედგება, სინტაქსური და მორფოლოგიური ანალიზატორებისაგან. სინტაქსური ანალიზატორის მუშაობის მიზანია ბუნებრივ ენაზე ჩაწერილი წინადადების ანალიზი, მისი სტრუქტურის დადგენა (გარჩევის ხის აგება), ურთიერთკავშირის დადგენა მის შემადგენელ წევრებს შორის და წინადადების შესახებ ძირითადი გრამატიკული ინფორმაციის მოპოვება (მაგალითად: ინორმაცია სუბიექტის, ობიექტების და პრედიკატის შესახებ). მორფოლოგიური ანალიზატორის დანიშნულება გახლავთ სიტყვაფორმის დაშლა მის შემადგენელ მორფემებად და ამ დაშლის საფუძველზე სიტყვაფორმის ძირითადი გრამატიკული კატეგორიების დადგენა.

პროგრამული სისტემის შექმნის ძირითადი მოტივებია:

- იყოს იგი მაქსიმალურად გამოსადეგი ისეთი ბუნებრივი ენისათვის რომლებსაც გააჩნიათ მდიდარი მორფოლოგია, აქვთ რთული სინტაქსი, წინადადებაში დაშვებულია სიტყვათა თავისუფალი წყობა და სხვა
- აღმოიფხვრას ზოგიერთი ნაკლოვანება, რაც გააჩნდათ აქამდე არსებულ მსგავს პროგრამულ სისტემებს
- რაც შეიძლება გამარტივდეს ბუნებრივი ენის გრამატიკის წესების ჩაწერა

სიახლე და ძირითადი შედეგები

წარმოდგენილ ნაშრომში სიახლე და ძირითადი შედეგი არის პროგრამული სისტემა, ინსტრუმენტარი, რომელიც საშუალებას გვაძლევს სპეციალური ფორმალიზმების მეშვეობით მივაწოდოთ მას ბუნებრივი ენის გრამატიკის წესები,

ლექსიკონი და გამოვიყენოთ იგი შესაბამისი ბუნებრივი ენის ტექსტების სინტაქსური და მორფოლოგიური ანალიზისათვის.

დეტალური სახით, ნაშრომში მიღებული სიახლეები და ძირითადი შედეგებია:

- სინტაქსური და მორფოლოგიური ანალიზისათვის შექმნილია ახალი, კომპლექსური ალგორითმები.
- შემუშავებულ იქნა ახალი ფორმალიზმები, რომელთა ფარგლებშიც მარტივად შეგვიძლია ჩავწეროთ ბუნებრივი ენის გრამატიკის შემადგენელი წესები.
- ამ ფორმალიზმების რეალიზაციის შედეგად, დაპროგრამების ენა C++_ზე დაიწერა პროგრამული სისტემა, რომელიც განკუთვნილია ბუნებრივ ენაზე ჩაწერილი ტექსტების სინტაქსური და მორფოლოგიური ანალიზისათვის.
- მიღებული სისტემა გამოიცადა ქართული ენისათვის. შემუშავებული ფორმალიზმების ფარგლებში ჩაიწერა ქართული ენის სინტაქსის და მორფოლოგიის წესების გარკვეული ნაწილი. შედგა სიტყვის ძირების ლექსიკონი (მცირე მოცულობით).

ნაშრომის დაწვრლებითი გაცნობის შემდეგ, შეგვიძლია რამოდენიმე კონკრეტული სიახლის გამოყოფა რომელიც აქამდე არ გვხვდებოდა სხვა ნაშრომებში. ეს სიახლეებია:

- კონტექსტისაგან თავისუფალი გრამატიკის წესების ჩაწერის გაფართოებული ვარიანტი, რომელიც ითვალისწინებს წესში არსებულ სიმბოლოთა თავისუფალ, ან ნაწილობრივ თავისუფალ წყობას.
- შეზღუდვების მუშაობის უფრო ზოგადი პრინციპი. წესზე დადებული შეზღუდვები წარმოიდგინება როგორც ლოგიკური გამოსახულებები, რაც უფრო რთული გრამატიკული წესების ჩაწერის საშუალებას გვაძლევს.
- სინტაქსური და მორფლოგიური ანალიზატორების ფორმალიზმში შემოღებულია მრავალი მოსახერხებელი კონსტრუქცია: ცვლადები, კონსტანტები, თვისებათა სტრუქტურის ინიციალიზაციის ნაწილი, მრავალარგუმენტიანი ოპერაციები თვისებათა სტრუქტურებზე და სხვა.
- მორფოლოგიური ანალიზის პრინციპულად ახალი ალგორითმი, წესის შიგნით შეზღუდვების გადანაწილების შესაძლებლობით.
- პრეპროცესორი მორფოლოგიური ანალიზატორის გრამატიკის ფაილისათვის. მისი მეშვეობით შესაძლებელია პარამეტრიანი მაკრო ჩასმების გამოყენება. რაც გვეხმარება გრამატიკის ფაილში გამეორებადი (ან ერთმანეთის მსგავსი) ტექსტური ფრაგმენტების მოკლედ ჩაწერაში.

პრაქტიკული მნიშვნელობა

ნაშრომს გააჩნია პრაქტიკული მნიშვნელობა, ვინაიდან მასში წარმოდგენილია პროგრამული სისტემა რომელიც განკუთვნილია ბუნებრივ ენაზე ჩაწერილი ტექსტების სინტაქსური და მორფოლოგიური ანალიზისათვის. ამ სისტემის პრაქტიკაში გამოყენება მოხდა ქართული ენის მაგალითზე. ტესტირების მიზნით, სისტემაში გამოყენებული ფორმალიზმის ფარგლებში ჩაიწერა ქართული ენის გრამატიკის ნაწილი და მიღებულ იქნა წინადადებების და სიტყვაფორმების ანალიზი.

შეგვიძლია გამოვყოთ რამოდენიმე ძირითადი მომენტი ნაშრომის პრაქტიკული მნიშვნელობის შესახებ:

- მიღებული პროგრამულ სისტემას თავისთავად გააჩნია პრაქტიკული მნიშვნელობა ვინაიდან იგი არის ინსტრუმენტარი, რომელის გაშვებაც შეგვიძლია კომპიუტერზე და საშუალება გვეძლევა ჩავატაროთ ბუნებრივი ენის ტექსტების სინტაქსური და მორფოლოგიური ანალიზი.
- შემუშავებული ფორმალიზმები, რომელებიც საფუძვლად უდევს სინტაქსურ და მორფოლოგიურ ანალიზატორებს, ასევე პრაქტიკული მნიშვნელობის მატარებელნი არიან, ვინაიდან მათი მეშვეობით სპეციალისტი (ლინგვისტი) ჩაწერს ბუნებრივი ენის გრამატიკას ფორმალური, კომპიუტერისათვის მისაღები სახით.
- პროგრამული სისტემა დაწერილია დაპროგრამების ენა C++_ის სტანდარტზე, მოდულების სახით, ობიექტზე ორიენტირებული დაპროგრამების სტილის გამოყენებით. ის ასევე შეგვიძლია განვიხილოთ როგორც კლასების ბიბლიოთეკა და გამოვიყენოთ იგი სხვადასხვა პროგრამების შიგნით, სადაც საჭიროა ბუნებრივ ენაზე ჩაწერილი ტექსტების ანალიზი.

ნაშრომის აპრობაცია

დისერტაციის შედეგები გამოქვეყნებულია 10 სამეცნიერო ნაშრომში და მოხსენებულია სხვადასხვა სამეცნიერო შეკრებებსა და სემინარებზე:

- საქართველოს მეცნიერებათა აკადემიის ენათმეცნიერების ინსტიტუტის კონფერენციებზე (2003, 2004 წწ.) [2, 4, 5]

- ვეკუას სახელობის გამოყენებითი მათემატიკის ინსტიტუტის სემინარები (2004, 2005 წწ.) [19, 20, 21]
- ენა, ლოგიკა, გამოთვლები - მეოთხე ინტერნაციონალური სიმფოზიუმი (2001 წ.) [17]
- ქართველ მათემატიკოსთა კონგრესი (2001 წ.) [18]

ძირითადი შედეგები მოხსენებული იქნა ვეკუას სახელობის გამოყენებითი მათემატიკის ინსტიტუტის კონფერენციაზე (2005 წ.)

დისერტაციის მოცულობა და სტრუქტურა

სადისერტაციო ნაშრომი შედგება შესავლის, 3 თავის, დანართისა და ციტირებული ლიტერატურისაგან. მოიცავს 117 ნაბეჭდ გვერდს. ასევე ცალკე არის აკინძული დანართი, რომელშიც გაფორმებულია პროგრამის და მასში გამოყენებული მოდულების კოდი, ასევე დამატებითი ფაილები Bison, Lex და Coco/R ინსტრუმენტებისათვის.

დისერტაციის მოკლე შინაარსი თავების მიხედვით

დისერტაციის პირველი თავში - ”ბუნებრივი ენის კომპიუტერული ანალიზი” - განხილულია ბუნებრივი ენის კომპიუტერული ანალიზის საკითხები. დასმულია ბუნებრივი ენის ტექსტების კომპიუტერული ანალიზის პრობლემა და შემადგენელი ნაწილები: მორფოლოგიური, სინტაქსური და სემანტიკური ანალიზი. მორფოლოგიური და სინტაქსური კომპიუტერული ანალიზისათვის განხილულია პრობლემის გადაწყვეტისადმი ძირითადი თეორიული და პრაქტიკული მიდგომები.
პირველ პარაგრაფში დასმულია ბუნებრივ ენათა კომპიუტერული ანალიზის პრობლემა. ბუნებრივი ენის ტექსტების კომპიუტერული ანალიზის პროცესი და მისი შედეგის ფორმალიზაცია. ჩამოთვლილია ის ძირითადი პრაქტიკული ამოცანები, რომელთა გადაჭრისათვისაც აუცილებელია ბუნებრივ ენათა კომპიუტერული ანალიზის პრობლემის გადაწყვეტა. მაგ: მართლწერის ავტომატური შემოწმება, მანქანური თარგმანი და სხვა.

მეორე პარაგრაფში ჩამოთვლილია ბუნებრივი ენის კომპიუტერული ანალიზის შემადგენელი ნაწილები (მორფოლოგიური, სინტაქსური და სემანტიკური ანალიზი) და დაწვრილებით არის განხილული თითოეული მათგანი: დასმულია პრობლემა, მოყვანილია მარტივი მაგალითები და მითითებულია შესაძლო გადაჭრის გზები. აღნიშნულია რომ სადისერტაციო ნაშრომში განიხილება ბუნებრივი ენის მორფოლოგიური და სინტაქსური ანალიზის პრობლემა. სქემის სახით ნაჩვენებია ბუნებრივი ენის მორფოლოგიური და სინტაქსური ანალიზატორებისადან შემდგარი ერთიანი პროგრამული სისტემა. განხილულია მისი მუშაობის ზოგადი პრინციპი.

მესამე პარაგრაფში განხილულია ბუნებრივი ენის კომპიუტერული ანალიზისათვის გამოყენებული ძირითადი მიდგომები:

- თვისებათა სტრუქტურები და მათზე განსაზღვრული ოპერაციები
- ზოგადი, ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმი
- შეზღუდვები
- არადეტერმინისტული ძიების ალგორითმი (მორფოლოგიური ანალიზისათვის)

თითოეული მათგანი სათითაოდაა განხილული. აგრეთვე განხილულია უნიფიკაციის ოპერაციის მუშაობის პრინციპი. შეფასებულია სინტაქსური და მორფოლოგიური ანალიზის ალგორითმების სისწრაფე, როგორც განყენებულად, ასევე შეზღუდვების შემოწმების ალგორითმთან კომბინირებული სახით. ჩამოთვლილია მათი ძირითადი თვისებები. აღნიშნულია შეზღუდვებისადმი განზოგადოებული მიდგომა, ისინი წარმოიდგინება როგორც ლოგიკური გამოსახულებანი.

მეოთხე პარაგრაფი დათმობილი აქვს თვისებათა სტრუქტურების და მათზე განსაზღვრული ოპერაციების დაწვრილებით განხილვას. ამის მიზეზი არის ის, რომ ნაშრომში თვისებათა სტრუქტურები ძალზედ ფართოდაა გამოყენებული. ისინი გამოიყენება ინფორმაციის შესანახად და მისი მანიპულაციისათვის, როგორც სპეციფიკური მონაცემთა სტრუქტურები. თვისებათა სტრუქტურა წარმოადგენს თვისებების სიმრავლეს. თითოეულ თვისება არის ”სახელი” – ”მნიშვნელობა” ტიპის კონსტრუქცია. რაც მთავარია, თვისების მნიშვნელობა შეიძლება თავად იყოს თვისებათა სტრუქტურა. შესაბამისად, თვისებათა სტრუქტურა წარმოადგენს რეკურსიული ტიპის მონაცემთა სტრუქტურას, რაც მას უნივერსალურობას და

მოქნილობას მატებს. წარმოდგენილია თვისებათა სტრუქტურების მაგალითები, მათ შორის ერთმანეთში ჩალაგებულ თვისებათა სტრუქტურების მაგალითებიც. ნაჩვენებია თუ როგორ ჩაიწერება თვისებათა სტრუქტურები და მათზე განსაზღვრული ოპერაციები ფორმალური სახით. დაწვრილებით არიან განხილულნი თვისებათა სტრუტურებზე განსაზღვრული ოპერაციები:

- მინიჭება
- ტოლობაზე შემოწმება
- უნიფიკაცია
- უნიფიკაციაზე შემოწმება
- შემოწმების მრავალარგუმენტიანი ოპერაციები

განსაკუთრებული ყურადღება აქვს დათმობილი უნიფიკაციის ოპერაციის განხილვას. ორი თვისებათა სტრუქტურა ერთმანეთთან უნიფიცირდება მაშინ და მხოლოდ მაშინ თუკი მათი ერთნაირი სახელის მქონე თვისებები არ ეწინააღმდეგებიან ერთმანეთს.

დისერტაციის მეორე თავში - "მორფოლოგიური გამრჩევი" - განხილულია მორფოლოგიური გამრჩევი, მისი მუშაობის პრინციპი და პროგრამული რეალიზაცია. დაწვრილებით არის განხილული მორფოლოგიური გამრჩევისათვის შემუშავებული ფორმალიზმი. განხილულია მაგალითები და ნაჩვენებია ქართული ენის მორფოლოგიის ფრაგმენტის ჩაწერა შემუშავებული ფორმალიზმის მეშვეობით.

პირველ პარაგრაფში დასმულია მორფოლოგიური ანალიზის ამოცანა. სიტყვაფორმის მორფემებად დაშლა განხილულია როგორც კომბინატორული ამოცანა. განხილულია არადეტერმინისტრული ძიების ალგორითმი, რომლის მეშვეობითაც ხდება დასმული პრობლემის გადაწყვეტა. ამ ალგორითმის მუშაობა ნაჩვენებია კონკრეტულ მაგალითზე. საწყის მათემატიკურ მოდელს ვაფართოებთ შეზღუდვების დამატებით, რათა მისი მეშვეობით შესაძლებელი გახდეს ბუნებრივი ენის სიტყვაფორმების სწორ კომბინაციებად დაშლა. გაფართოებული ამოცანის ამოსახსნელად გამოყენებულია იგივე ალგორითმი ოღონდ კომბინირებული სახით შეზღუდვების შემოწმების ალგორითმთან ერთად. განხილულია ალგორითმების ეფექტურობა.

მეორე პარაგრაფში განხილულია მორფოლოგიური გამრჩევისათვის შემუშავებული ფორმალიზმი. აღწერილია თუ როგორ განისაზღვრება მორფემათა კლასები, წესები და შეზღუდვები. დაწვრილებით არის განხილული შეზღუდვების მუშაობის

პრინციპი. შეზღუდვების წესის შიგნით გადანაწილების შესაძლებლობა. შეზღუდვები წარმოადგენენ ლოგიკურ გამოსახულებებს. განხილულია მათი გამოთვლის პროცესი, ოპერაციათა პრიორიტეტები, დეკლარაციული და იმპერატიული ბუნება. განხილულია ფომრლიზმში შემოღებული შემდეგი საშუალებები: ცვლადებისა და კონსტანტების განსაზღვრა, ფაილის ჩართვის დირექტივა, კომენტარები.

მესამე პარაგრაფში აღწერილია მორფოლოგიური გარჩევის პროგრამა. მისი მუშაობის პრინციპი. მომხმარებლის ინტერფეისი (ტექსტური რეჟიმი). ნაჩვენებია როგორ ხდება პროგრამის გაშვება, პროგრამიდან გამოსვლა და სიტყვაფორმების გარჩევა. განხილულია პროგრამის სტრუქტურა.

მეოთხე პარაგრაფი დათმობილი აქვს პრეპროცესორის განხილვას. ნაჩვენებია ის მიზეზები თუ რატომ გახდა საჭირო პრეპროცესორის დამატება მორფოლოგიური ანალიზის პროგრამისათვის. მოყვანილია მაგალითები, სადაც ნაჩვენებია თუ როგორ ხდება მაკრო-ჩასმების დეკლარაცია და მათი გამოძახება. განხილულია პარამეტრიანი მაკრო-ჩასმები. განხილულია რეკურსიული და ურთიერთრეკურსიული მაკრო-ჩასმების პრობლემა და მისი გადაჭრის ერთერთი მეთოდი.

მეხუთე პარაგრაფში განხილულია სხვადასხვა სირთულის მაგალითები, რომლებიც გვაცნობენ მორფოლოგიური ანალიზის პროგრამის ძირითად ასპექტებს. განხილულია, თუ როგორ ხდება გრამატიკის ფაილის შედგენა მორფოლოგიური გამრჩევისათვის, როგორ გამოიძახება პროგრამა და როგორ მიეწოდება გასარჩევი სიტყვაფორმები.

მეექვსე პარაგრაფში ზუსტი, ფორმალური სახით არის აღწერილი მორფოლოგიის ფაილის სტრუქტურა. სინტაქსური კონსტრუქციების აღსაწერად გამოყენებულია ბეკუს-ნაურის გაფართოებული ფორმალიზმი. მისი მეშვეობით აღწერილია მორფოლოგიის ფაილის ყველა კონსტრუქცია: მორფემათა კლასების განსაზღვრა, წესების განსაზღვრა, შეზღუდვები, თვისებათა სტრუქტურები და სხვა.

მეშვიდე პარაგრაფში მოყვანილია ქართული ენის მორფოლოგიის ფრაგმენტი რომელიც ჩაწერილია შემუშავებული ფორმალიზმის მეშვეობით და რომლის საფუძველზეც მორფოლოგიური გარჩევის პროგრამის გამოყენებით მიღებულ იქნა ექსპერიმენტული შედეგები. ქართული ენის მორფოლოგიის ფრაგმენტი 4 ფაილად არის დაყოფილი: არსებითი სახელის და ზედსართავი სახელის შემადგენელი

მორფემათა კლასები, არსებითი სახელის და ზედსართავი სახელის განმსაზღვრელი წესები, ზმნების შემადგენელი მორფემათა კლასები, ზმნების განმსაზღვრელი წესები.

დისერტაციის მესამე თავში - ”სინტაქსური გამრჩევი” - განხილულია სინტაქსური გამრჩევი, მისი მუშაობის პრინციპი და პროგრამული რეალიზაცია. დაწვრილებით არის განხილული სინტაქსური გამრჩევისათვის შემუშავებული ფორმალიზმი. განხილულია მაგალითები და ნაჩვენებია ქართული ენის სინტაქსის ფრაგმენტის ჩაწერა შემუშავებული ფორმალიზმის მეშვეობით.

პირველ პარაგრაფში განხილულია სინტაქსური ანალიზის პრობლემა და სინტაქსური გამრჩევისათვის შემუშავებული ფორმალიზმი. განხილულია კონტექსტისაგან-თავისუფალი გრამატიკის წესები, მათი გამოყენების შესაძლებლობა ბუნებრივი ენისათვის. დაწვრილებით არის განხილული წესში სიტყვათა თავისუფალი წყობის შემთხვევა და მის ჩასაწერად შემოტანილია სპეციალური კონსტრუქცია, სიმბოლოების ურთიერთმდებარეობათა მარეგულირებლები, რომელიც ბოლოში ემატება კონტექსტისაგან-თავისუფალი გრამატიკის წესს. ნაჩვენებია ბუნებრივი ენისთვის სტანდარტული კონტექსტისაგან-თავისუფალი წესების გამოყენებისას მათი არასაკმარისობა. რაც გამოწვეულია წინადადების წევრებს შორის შეთანხმებების პრობლემით. განხილულია ამ პრობლემის მოგვარების შესაძლებლობა შეზღუდვების გამოყენებით. წარმოდგენილია ფორმალიზმში გამოყენებული ძირითადი კონსტრუქციები.

მეორე პარაგრაფში აღწერილია სინტაქსური გარჩევის პროგრამა. მისი მუშაობის პრინციპი. მომხმარებლის ინტერფეისი (ტექსტური რეჟიმი). ნაჩვენებია, თუ როგორ ხდება გრამატიკის ფაილის მომზადება, პროგრამის გაშვება და წინადადებების გარჩევა. განხილულია პროგრამის სტრუქტურა. გარჩეულია პროგრამის ყველა ბრძანება და მოყვანილია მათი გამოყენების მაგალითები. ნაჩვენებია როგორ ვიღებთ ანალიზის შედეგს, გარჩევის ხეები და ინფორმაცია მათ შესახებ. განხილულია პროგრამის რეალიზაციის ზოგიერთი დეტალი.

მესამე პარაგრაფში განხილულია სხვადასხვა სირთულის მაგალითები, რომლებიც გვაცნობენ სინტაქსური ანალიზის პროგრამის ძირითად ასპექტებს. განხილულია, თუ როგორ ხდება გრამატიკის და ლექსიკონის ფაილის შედგენა სინტაქსური გამრჩევისათვის, როგორ გამოიძახება პროგრამა და როგორ მიეწოდება გასარჩევი წინადადებები.

მეოთხე პარაგრაფში აღწერილია სინტაქსური გამრჩევის ლექსიკონის ფაილის სტრუქტურა, მასში ინფორმაციის წარმოდგენა თვისებათა სტრუქტურების მეშვეობით. განხილულია სხვადასხვა მაგალითები.

მეხუთე პარაგრაფში ზუსტი, ფორმალური სახით არის აღწერილი სინტაქსური გამრჩევის გრამატიკის ფაილის სტრუქტურა. სინტაქსური კონსტრუქციების აღსაწერად გამოყენებულია ბეკუს-ნაურის ფორმალიზმი. მისი მეშვეობით აღწერილია სინტაქსური გამრჩევის გრამატიკის ფაილის ყველა კონსტრუქცია: წესების განსაზღვრა, შეზღუდვები, კონსტანტები და სხვა.

მეექვსე პარაგრაფში მოყვანილია ქართული ენის სინტაქსის ფრაგმენტი (სამპირიანი ზმნებით განსაზღვრული წინადადებები) რომელიც ჩაწერილია შემუშავებული ფორმალიზმის მეშვეობით და რომლის საფუძველზეც სინტაქსური გარჩევის პროგრამის გამოყენებით მიღებულ იქნა ექსპერიმენტული შედეგები.

დასკვნა და ძირითადი შედეგები

წარმოდგენილ სადისერტაციო ნაშრომში განხილულია ბუნებრივ ენაზე ჩაწერილი ტექსტების კომპიუტერული დამუშავების საკითხები, შემუშავებული ახალი მიდგომები და მათი რეალიზაციის საუძველზე მიღებულია სინტაქსური და მორფოლოგიური ანალიზისათვის განკუთვნილი პროგრამული სისტემა.

იმისათვის რომ პროგრამას შეეძლოს ტექსტების ანალიზი, სპეციალურად ამ პროგრამული სისტემისათვის შემუშავებული ფორმალიზმების ფარგლებში, უნდა ჩავწეროთ კონკრეტული ენის გრამატიკული წესები (სინტაქსური და მორფოლოგიური) და შევქმნათ სიტყვის ძირების ლექსიკონი. პროგრამული სისტემის შემუშავებისას ბუნებრივი ენის მაგალითად აღებულ იქნა ქართული ენა, თუმცა სისტემა სხვა მრავალი ბუნებრივი ენისთვისაც შეგვიძლია გამოვიყენოთ.

პროგრამისათვის აუცილებელი საწყისი ინფორმაცია, გრამატიკის და ლექსიკონის ფაილი, უნდა მოამზადოს შესაბამისი ენის სპეციალისტმა (ლინგვისტმა), რომელიც ღრმად ერკვევა ენის სინტაქსურ და მორფოლოგიურ სტრუქტურაში. მას შემდეგ რაც სპეციალისტი გაეცნობა ფორმალიზმს, რომელიც საკმარისად მარტივი ასათვისებელი და გამოყენებისათვის მოხერხებულია, იგი

მზადაა ენის სინტაქსური და მორფოლოგიური წესები ფორმალური სახით ჩაწეროს შესაბამის გრამატიკის ფაილებში. ამის შემდეგ უკვე შეგვეძლება სისტემის გამოყენება კონკრეტულ ენაზე ჩაწერილი ტექსტების ანალიზისათვის. გრამატიკის ფაილების შექმნა, მათი ტესტირება და კორექტირება შეიძლება გარკვეული პერიოდი გაგრძელდეს, ვიდრე არ მივიღებთ მაქსიმალურად ზუსტ შედეგს. წარმატებული ანალიზისათვის საჭიროა, რომ ფორმალური სახით ჩაწერილი გრამატიკა დიდი სიზუსტით ასახავდეს ბუნებრივი ენის გრამატიკას.

პროგრამული სისტემა გამოიცადა ქართული ენისათვის. ქართული ენის გარკვეული ნაწილისათვის, შემუშავებული იქნა გრამატიკის ფაილები სინტაქსური და მორფოლოგიური ანალიზისათვის. გრამატიკული წესები ძირითადად ჩაიწერა ა. შანიძის გრამატიკის მიხედვით, ხოლო ზმნების მორფოლოგიური ანალიზისათვის გამოყენებული იქნა დ. მელიქიშვილის მიერ შედგენილი ქართული ზმნების კლასიფიკაცია. მიღებულმა შედეგებმა ცხადყო რომ შექმნილი პროგრამული სისტემის გამოყენება საკმარისად ეფექტურია პრაქტიკული ამოცანების გადასაწყვეტად.

თავი 1

ბუნებრივი ენის კომპიუტერული ანალიზი

§1. პრობლემის დასმა

ბუნებრივ ენათა კომპიუტერული ანალიზი წარმოადგენს ბუნებრივ ენათა დამუშავების ერთერთ ძირითად ამოცანას. მისი მიზანია, კომპიუტერის მეშვეობით გაანალიზოს ბუნებრივ ენაზე ჩაწერილი ტექსტები და მოგვცეს მიღებული შედეგის ფორმალიზაცია იმ სახოთ, რომ იგი გამოსადეგი გახდეს შემდგომი პროგრამული მანიპულაციისთვის გამომთვლელ მანქანაზე. ანალიზის შედეგად უნდა მივიღოთ ძირითადი გრამატიკული ინფორმაცია საკვლევი ტექსტის შესახებ. მორფოლოგიური ანალიზის შემთხვევაში, ეს ინფორმაცია არის სიტყვაფორმის მორფემებად დაშლის შედეგად მიღებული გრამატიკული კატეგორიები. სინტაქსური ანალიზის დროს

დგინდება წინადადების სტრუქტურა და მოიპოვება ინფორმაცია მისი ძირითადი წევრების შესახებ. სემანტიკური ანალიზის შემთხვევაში უნდა მოვახერხოთ გარკვეული ფორმალური სახით ტექსტის შინაარსის ჩაწერა. წარმოდგენილ ნაშრომში განხილულია ბუნებრივ ენათა მორფოლოგიური და სინტაქსური, კომპიუტერული ანალიზის პრობლემები და მათი გადაჭრის გზები.

დღესდღეობით მრავალი მნიშვნელოვანი ამოცანა დგას, რომელთა გადაწყვეტაც მოითხოვს ბუნებრივ ენათა კომპიუტერულ ანალიზს. ერთერთი კლასიკური ამოცანა არის მართლწერის (ორთოგრაფიის) კომპიუტერული შემოწმების ამოცანა. ეს ამოცანა სწორედ ბუნებრივ ენათა კომპიუტერული ანალიზის მეშვეობით გადაიჭრება. თანამედროვეობის ერთერთი ყველაზე რთული ამოცანის, მანქანური თარგმნის გადაწყვეტაც, აგრეთვე ბუნებრივ ენათა კომპიუტერულ ანალიზს ემყარება. თუმცა ამავდროულად საპირისპირო (სინთეზის) ამოცანაც უნდა იყოს გადაწყვეტილი. როდესაც ერთი ბუნებრივი ენიდან მეორე ენაზე ხდება წინადადების თარგმანი, პირველ რიგში ანალიზი უკეთდება საწყის ენაზე ჩაწერილ წინადადებას, დგინდება მისი სინტაქსური კონსტრუქცია, წინადადების წევრები და ა.შ.. შემდეგ ეტაპზე ამ სინტაქსურ კონსტრუქციას მოეძებნება შესატყვისი სინტაქსური კონსტრუქცია მეორე ენაში და გადაითარგმნება წინადადების შემადგენელი წევრები. საბოლოოდ ამ ინფორმაციის საუძველზე მოხდება წინადადების სინთეზი მეორე ენისათვის. ეს გახლავთ ეგრედ წოდებული სინტაქსურად მართვადი თარგმნის პრინციპი. თუმცა წარმატებული შედეგის მისაღწევად იგი არ არის საკმარისი, საჭიროა გათვალისწინებულ იქნეს აგრეთვე ბუნებრივი ენის სემანტიკა. გარდა ამისა, არის სხვა მრავალი ამოცანაც რომელთა გადაწყვეტაც მთლიანად დამოკიდებულია ბუნებრივ ენაზე ჩაწერილი ტექსტების კომპიუტერული ანალიზის ამოცანის გადაჭრაზე. ასეთი ამოცანებია:

- კომპლექსური საძიებო სისტემები (ინტერნეტში), რომლებშიც გათვალისწინებულია სიტყვაფორმების მორფოლოგიური ანალიზი. მარტივად რომ ვთქვათ, თუკი სისტემაში ვეძებთ დოკუმენტებს რომლებიც მაგალითად შეიცავენ სიტყვას ”კანონი”, სისტემამ უნდა მოძებნოს არა მხოლოდ ის დოკუმენტები რომლებშიც ამ სიტყვის ზუსტი თანხვედრაა, არამედ ისეთი დოკუმენტებიც სადაც გვხვდება სიტყვები: ”კანონები”, ”კანონთა”, ”კანონით” და ა.შ..

- ენის სასწავლი კომპიუტერული ტრენაჟორები. აქაც მოითხოვება რომ სისტემას შეეძლოს ბუნებრივი ტექსტების კომპიუტერული ანალიზი, რათა გააანალიზოს კითხვაზე მომხმარებლის მიერ შეტანილი პასუხის გრამატიკული სისწორე.
- ბუნებრივ ენაზე მომუშავე სხვადასხვა სახის დიალოგური სისტემები.

იმისათვის რომ წარმატებით გავართვათ თავი ბუნებრივ ენაზე ჩაწერილი ტექსტების კომპიუტერული ანალიზის პრობლემას, საჭიროა გავეცნოთ ძირითად თეორიულ მიმართულებებს ამ სფეროში, მაგალითად: სასრული ავტომატები, უშუალო შემადგენელთა მეთოდი, ატრიბუტული გრამატიკები, სემანტიკური ქსელები და ა.შ.. შევაფასოთ მათი ძლიერი და სუსტი მხარეები, მათი კომპიუტერზე რეალიზაციის სიმარტივე. ამავდროულად მიღებული უნდა იქნას ექსპერიმენტალური შედეგები.

§2. ბუნებრივი ენის კომპიუტერული ანალიზის შემადგენელი ნაწილები

ბუნებრივ ენაზე ჩაწერილი ტექსტების ანალიზი იყოფა რამოდენიმე შემადგენელ ნაწილად. ეს ნაწილებია:

- მორფოლოგიური ანალიზი
- სინტაქსური ანალიზი
- სემანტიკური ანალიზი

შევნიშნოთ რომ სინტაქსურ და სემანტიკურ ანალიზს შორის დამატებით რამოდენიმე საშუალედო ეტაპს გამოარჩევენ, მაგრამ ჩვენ მათ აქ არ განვიხილავთ.

მორფოლოგიური ანალიზის ამოცანაა, დაშალოს საწყისი სიტყვაფორმა შემადგენელ მორფემებად და ამ დაშლის საფუძველზე დაადგინოს სიტყვაფორმის ძირითადი გრამატიკული კატეგორიები. მაგალითად ავიღოთ სიტყვაფორმა ”ვაშლები”. იგი შემდეგნაირად დაიშლება: ”ვაშლ” – ”ებ” – ”ი”. ამ დაშლის შედეგად ჩვენ ვიგებთ მნიშვნელოვან გრამატიკულ ინფორმაციას დასაშლელი სიტყვაფორმის შესახებ. კერძოდ, ”ებ” მორფემა (რიცხვის ნიშანი) მიუთითებს, რომ სიტყვაფორმა მრავლობით რიცხვშია, ხოლო ”ი” მორფემა (ბრუნვის ნიშანი) გვეუბნება, რომ

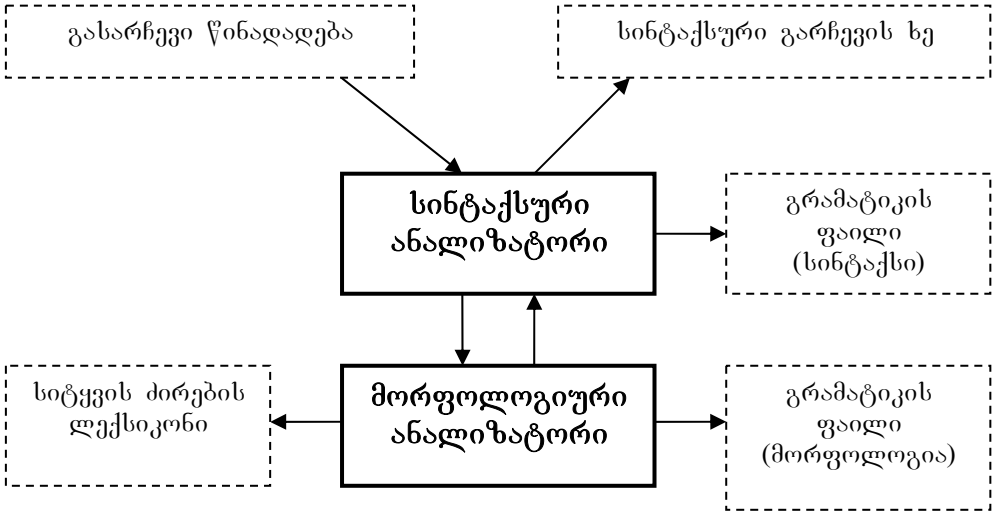
სიტყვაფორმა სახელობით ბრუნვაში დგას. მორფოლოგიური ანალიზის გამოყენება, საშუალებას გვაძლევს, რომ თავი ავარიდოთ ლექსიკონში ყველა სიტყვაფორმის შეტანას და შევიტანოთ მხოლოდ სიტყვის უცვლელი ნაწილები. ამის შედეგად ისეთი ბუნებრივი ენებისათვის, როგორიცაა ქართული, რუსული, უნგრული და ა.შ., ლექსიკონის მოცულობა მკვეთრად მცირდება. ზოგიერთი ისეთი სუსტად განვითარებული მორფოლოგიის მქონე ენებისათვის, როგორიცაა ინგლისური ენა, მორფოლოგიური ანალიზი პრაქტიკული თვალსაზრისით არ გვაძლევს საგულისხმო შედეგს. იმისათვის, რომ მოხერხდეს სიტყვაფორმის მორფოლოგიური ანალიზი კომპიუტერის მეშვეობით, საჭიროა ფორმალური სახით ჩავწეროთ შესაბამისი ენის მორფოლოგიის წესები. ჩვეულებრივ, ბუნებრივ ენათა მორფოლოგიის წესების უმრავლესობა ექვემდებარება ფორმალური სახით ჩაწერას.

სინტაქსური ანალიზი გამოიყენება ბუნებრივ ენაზე ჩაწერილი წინადადების გასარჩევად, მისი სტრუქტურის დასადგენად და სინტაქსური გარჩევის ხის ასაგებად. სინტაქსური ანალიზის შედეგად ჩვენ ვგებულობთ თუ რა ურთიერთდამოკიდებულებაში არიან წინადადების შემადგენელი წევრები ერთიმეორესთან. არსებობს რამოდენიმე სხვადასხვა მიდგომა წინადადებაზე სინტაქსური ანალიზის ჩასატარებლად. მაგალითად: კონტექსტისაგან თავისუფალი გრამატიკის წესები, თვისებათა გრამატიკები, ვალენტობის მეთოდი და ა.შ.. ყველაზე ზოგადი სახით, ბუნებრივი ენის გრამატიკის წესები შეგვიძლია დავწეროთ, როგორც კონტექსტისაგან თავისუფალი გრამატიკის წესები, თუმცა ეს მიახლოება არ არის საკმარისი, რათა ზუსტად ავსახოთ ბუნებრივი ენის მთელი გრამატიკა. ბუნებრივ ენაში ვხვდებით ისეთ მომენტებს, როგორიცაა მაგალითად წინადადების წევრებს შორის რიცხვში ან ბრუნვაში შეთანხმება. იმისათვის, რომ ეს ყველაფერი ავსახოთ კონტექსტისაგან თავისუფალი გრამატიკის წესებში, გვაქვს ორი საშუალება. პირველი - დამატებითი წესების შემოტანით უფრო გავამკაცროთ გრამატიკა, რაც არაეფექტურია (მკვეთრად იზრდება საჭირო წესების რაოდენობა) და მეორე - შემოვიტანოთ თვისებების ცნება, ყოველ სიმბოლოს შევუსაბამოთ თავისი თვისებები და ამ თვისებების გამოყენებით კონტექსტისაგან თავისუფალი გრამატიკის წესს დავადოთ გარკვეული შეზღუდვები. ეს მეთოდი ეფექტურია, რადგან გვაძლევს საკმარისად ზუსტ მიახლოებას ბუნებრივი ენის გრამატიკასთან. ამავე დროს, სინტაქსური ანალიზისათვის, შეგვიძლია გამოვიყენოთ კარგად შესწავლილი

სინტაქსური გარჩევის ალგორითმები და მათ დავამატოთ შეზღუდვების შემოწმების მექანიზმი.

სემანტიკური ანალიზის მიზანია, მოახდინოს ბუნებრივ ენაზე ჩაწერილი ტექსტების შინაარსის ფორმალიზაცია. პრობლემა მეტისმეტად რთულია და დღესდღეობით არ არსებობს მისი მეტნაკლებად დამაკმაყოფილებელი გადაწყვეტა. თუმცა არის გარკვეული მეთოდები რომელთა გამოყენებით ზოგიერთ შემთხვევაში შესაძლებელია შედეგების მიღება, ასეთი მეთოდების მაგალითია სემანტიკური ქსელები. წარმოდგენილ ნაშრომში სემანტიკური ანალიზის პრობლემა არ არის განხილული.

როგორც უკვე ავღნიშნეთ წარმოდგენილ ნაშრომში განხილულია ბუნებრივ ენათა სინტაქსური და მორფოლოგიური, კომპიუტერული ანალიზის საკითხები. გაერთიანებული სახით, სინტაქსური და მორფოლოგიური ანალიზატორებისაგან შემდგარი სისტემა შემდეგი სახით შეგვიძლია წარმოვადგინოთ:



სისტემა შემდეგი პრინციპით მუშაობს: გასარჩევი წინადადება მიეწოდება სინტაქსურ ანალიზატორს. ანალიზატორი წინადადებაში შემავალ სიტყვებს გადასცემს მორფოლოგიურ ანალიზატორს. მორფოლოგიური ანალიზატორი, მორფოლოგიის წესებზე და სიტყვის ძირების ლექსიკონზე დაყრდნობით, მორფემებად დაშლის სიტყვაფორმებს. მისცემს მათ შესაბამის თვისებებს (გრამატიკულ კატეგორიებს) და ამ ინფორმაციას დაუბრუნებს სინტაქსურ ანალიზატორს. რის შემდეგაც სინტაქსური ანალიზატორი გაარჩევს წინადადებას ბუნებრივი ენის გრამატიკის წესების გამოყენებით და მოგვცემს გარჩევის ხეს. თუკი გარჩევა ვერ მოხერხდა ანალიზატორი გამოიტანს შესაბამის შეტყობინებას.

§3. ძირითადი მიდგომები

ნაშრომში განხილულია და პროგრამული სისტემის შექმნისას გამოყენებულია შემდეგი ძირითადი მეთოდები:

- თვისებათა სტრუქტურები და მათზე განსაზღვრული ოპერაციები
- ზოგადი, ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმი
- შეზღუდვები
- არადეტერმინისტული ძიების ალგორითმი (მორფოლოგიური ანალიზისთვის)

თვისებათა სტრუქტურა წარმოადგენს წყვილების სიმრავლეს. თითოეულ წყვილი, თვისება, ”სახელი - მნიშვნელობა” ტიპის კონსტრუქციაა. რაც ნიშნავს იმას, რომ ყოველ თვისებას გააჩნია სახელი და მნიშვნელობა. თვისებას შეიძლება ჰქონდეს მარტივი, სტრიქონული მნიშვნელობა, ან შესაძლოა მისი მნიშვნელობა თავად იყოს თვისებათა სტრუქტურა. თვისებათა სტრუქტურების ერთერთი ძირითადი უპირატესობა, სხვა სახის მონაცემთა სტრუქტურებთან შედარებით, მათი უნივერსალურობაა. თვისებათა სტრუქტურის საშუალებით შესაძლებელია მონაცემთა საკმარისად დიდი სპექტრის წარმოდგენა კომპიუტერში. ჩვენს შემთხვევაში თვისებათა სტრუქტურები ძალიან ფართოდ არის გამოყენებული. მაგალითად: ლექსიკონის ფაილში, სიტყვის ძირების შესახებ ინფორმაციის წარმოსადგენად, ვიყენებთ თვისებათა სტრუქტურებს; გრამატიკის ფაილში ჩაწერილი წესების ყოველ შემადგენელ სიმბოლოს შეესაბამება თავისი თვისებათა სტრუქტურა; ასევე, შეზღუდვებში თვისებათა სტრუქტურებს და მათზე განსაზღვრულ ოპერაციებს ვიყენებთ გარკვეული პირობების შესამოწმებლად. თვისებათა სტრუქტურების პროგრამული რეალიზაცია საკმაოდ მარტივად და ეფექტურად ხორციელდება ასოციატიური მასივების გამოყენებით. თვისებათა სტრუქტურებთან ერთად, ფართოდ ვიყენებთ მათზე განსაზღვრულ ოპერაციებს. ასეთი ოპერაციებია: ტოლობაზე შემოწმება, უნიფიკაცია, მინიჭება და ა.შ.. განსაკუთრებით აღსანიშნავია უნიფიკაციის ოპერაცია. მისი მუშაობის პრინციპი

შემდეგია: ორი თვისებათა სტრუქტურა ერთმანეთთან უნიფიცირდება, თუკი მათში ერთიდაიგივე სახელის მქონე თვისებების მნიშვნელობები არ ეწინააღმდეგებიან ერთმანეთს. თვისებების მნიშვნელობები არ ეწინააღმდეგებიან ერთმანეთს მაშინ, როდესაც:

- მარტივი მნიშვნელობები ერთმანეთის ტოლია
- ერთერთი მნიშვნელობა განუსაზღვრელია
- თუ ორივე მნიშვნელობა თავადაა თვისებათა სტრუქტურა, მაშინ ისნი უნდა უნიფიცირდებოდნენ ერთმანეთთან

უნიფიკაციის შედეგად, თუკი იგი წარმატებული აღმოჩნდა, ის თვისებები, რომლებიც გვხვდებოდა მეორე თვისებათა სტრუქტურაში და არ გვხვდებოდა პირველში, ავტომატურად გადმოიტანებიან პირველ თვისებათა სტრუქტურაში. თვისებათა სტრუქტურებს, მათზე განსაზღვრულ ოპერაციებს, და ჩვენს ფორმალიზმში მათი ჩაწერის სახეს, დეტალურად განვიხილავთ ცალკე პარაგრაფში.

სინტაქსური ანალიზის ერთერთი ცნობილი მეთოდია ზოგადი, ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმი. ბუნებრივი ენის სინტაქსის წესებს ვწერთ კონტექსტისაგან თავისუფალი გრამატიკის წესების სახით, მაგრამ, რადგანაც კონტექსტისაგან თავისუფალი გრამატიკის წესები არ არის საკმარისი ბუნებრივი ენის გრამატიკის სრულფასოვნად და მარტივი სახით ჩასაწერად, ამიტომ წესებზე შეზღუდვების დადების შესაძლებლობაც გათვალისწინებულია. ამგვარად ჩაწერილი გრამატიკის ფაილის მიხედვით, ტექსტების სინტაქსური ანალიზისათვის, გამოიყენება კომბინირებული ალგორითმი, რომელშიც გაერთიანებულია სინტაქსური გარჩევის ალგორითმი და შეზღუდვების დინამიურად შემოწმების მექანიზმი. სინტაქსური გარჩევის ალგორითმად არჩეულ იქნა ზოგადი, ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმი. მისი საშუალებით შესაძლებელია ნებისმიერი კონტექსტისაგან თავისუფალი გრამატიკის წესების მეშვეობით ჩაწერილი ენის წინადადებების გარჩევა. ამ ალგორითმის ერთერთი დამახასიათებელი თვისება ბუნებრივი თანამიმდევრულობაა, რაც საშუალებას გვაძლევს შეზღუდვების შემოწმება გარჩევის პარალელურად მოხდეს და არასწორი განშტოებები დროულად იქნას მოშორებული გადასარჩევი ვარიანტების სიმრავლიდან. მართალია, თავად ზემოდან-ქვემოთ სინტაქსური გარჩევის ალგორითმი არ არის ძალზედ სწრაფი, მაგრამ წესების დინამიური შემოწმება

გარკვეულწილად ახდენს ამ ნაკლის კომპენსირებას და ასწრაფებს ალგორითმს. ქვემოდან-ზემოთ სინტაქსური გარჩევის ალგორითმის რეალიზაცია ადვილად ხორციელდება სტეკების გამოყენებით. ალგორითმის მუშაობის შედეგად ვიღებთ სინტაქსური გარჩევის ხეს, რომელიც გვიჩვენებს, თუ რა ურთიერთდამოკიდებულებებში არიან წინადადების შემადგენელი წევრები.

შეზღუდვების შემოწმება, როგორც უკვე ავღნიშნეთ, ხდება სინტაქსური გარჩევის პარალელურად. შეზღუდვა წარმოადგენს ლოგიკურ გამოსახულებას, რომელიც გამოთვლის შედეგად იღებს ან ლოგიკურად ჭეშმარიტ ანდ ლოგიკურად მცდარ მნიშვნელობას. ეს გამოსახულება შედგება თვისებათა სტრუქტურებზე განსაზღვრული ოპერაციების და ძირითადი ლოგიკური კავშირებისაგან. ეს კავშირებია: ”&” - ლოგიკური ნამრავლი, ”|” - ლოგიკური ჯამი და ”~” - ლოგიკური უარყოფა. თუკი გრამატიკის წესს, გააჩნია შეზღუდვა, მაშინ სინტაქსური გარჩევის დროს, როდესაც შესაბამისი წესი ამოქმედდება და ჩაანაცვლებს სიმბოლოთა გარკვეულ მიმდევრობას ერთი არატერმინალური სიმბოლოთი, მოხდება ამ შეზღუდვის შემოწმება. ანუ გამოითვლება ლოგიკური გამოსახულება. თუკი შეზღუდვა დაკმაყოფილდა, რაც ნიშნავს, რომ ლოგიკური გამოსახულების გამოთვლის შედეგად მივიღეთ ლოგიკურად ჭეშმარიტი მნიშვნელობა, წესი ჩაითვლება დასაშვებად და სინტაქსური გარჩევა ჩვეულებრივად გაგრძელდება. იმ შემთხვევაში, თუ შეზღუდვა არ დაკმაყოფილდა, მოხდება წესის უკუგდება ამ ეტაპზე და განიხილება სხვა ალტერნატიული ვარიანტები. შეზღუდვები გამოიყენება აგრეთვე მორფოლოგიურ ანალიზში, სადაც ისინი კომბინირებულნი არიან არადეტერმინისტული ძიების ალგორითმთან. შეზღუდვების პროგრამული რეალიზაცია საკმაოდ კომპლექსურია და ორ ეტაპად ხორციელდება. ეს ეტაპებია:

- გრამატიკის ფაილის დამუშავების დროს შეზღუდვები (ლოგიკური გამოსახულებები) კომპილირდება საშუალედო კოდში, რომელიც ინახება ოპერატიულ მეხსიერებაში შემდგომი გამოყენებისთვის
- წესის აქტივაციის დროს, ხდება მიმართვა მის შესაბამის შეზღუდვაზე. ლოგიკური გამოსახულება გამოითვლება ინტერპრეტაციის რეჟიმში და მიღებული შედეგი უბრუნდება გარჩევის ალგორითმს

ლოგიკური გამოსახულების გამოთვლა წარმოებს ოპერაციათა პრიორიტეტების მიხედვით და აუცილებელია არ არის ბოლომდე შესრულდეს, თუკი გამოთვლის

პროცესში მიღებული იქნება შედეგი, რომელიც ცალსახად განსაზღვრავს მთელი გამოსახულების მნიშვნელობას, გამოსახულების დარჩენილი ნაწილი აღარ გამოითვლება. ეს მნიშვნელოვანი მომენტი საჭიროა გვახსოვდეს, რადგან ისეთი ოპერაციების გამოყენებისას რომლებიც ოპერანდების შიგთავსს ცვლიან, საჭიროა სწორად დაიწეროს ლოგიკური გამოსახულება და გათვალისწინებულ იქნას გამოთვლების არა მხოლოდ დეკლარაციული ბუნება, არამედ იმპერატიულიც.

მორფოლოგიური ანალიზისათვის გამოიყენება არადეტერმინისტული ძიების ალგორითმი (ძიება უკან დაბრუნებებით). პრაქტიკული თვალსაზრისით, ისეთი ძლიერად განვითარებული მორფოლოგიის მქონე ბუნებრივი ენებისათვის როგორიც ქართული ენაა, ამ ალგორითმზე უფრო მარტივი ალგორითმების გამოყენება არ გვაძლევს დამაკმაყოფილებლ შედეგს (მაგ: სპეციალიზებული წრფივი მეთოდები). არადეტერმინისტული ძიების ალგორითმი, არ მოითხოვს ამონახსნის ცალსახობას და საშუალებას გვაძლევს ვიპოვოთ სიტყვაფორმის მორფემებად დაშლის ყველა შესაძლო ვარიანტი. ის ასევე მოსახერხებელია შეზღუდვების შემოწმების დროსაც, ვინაიდან ეს ალგორითმი შეზღუდვების მორფემებს შორის გადანაწილების საშუალებას გვაძლევს, რაც მაქსიმალურად ამაღლებს მის სწრაფქმედებას. ამ ალგორითმის კომპიუტერული რეალიზაცია შესაძლებელია როგორც სტეკის მეშვეობით, ასევე რეკურსიული დაპროგრამებით. პრაქტიკულმა ექსპერიმენტებმა აჩვენა რომ ისეთი რთული მორფოლოგიური ბუნების მქონე მეტყველების ნაწილისათვისაც კი, როგორიც ქართული ენის ზმნაა, არადეტერმინისტული ძიების ალგორითმით და შეზღუდვების გამოყენებით ეფექტურად ხერხდება მისი მორფოლოგიური ანალიზი.

§4. თვისებათა სტრუქტურები და მათზე განსაზღვრული ოპერაციები

წარმოდგენილ ნაშრომში ფართოდ არის გამოყენებული თვისებათა სტრუქტურები. თვისებათა სტრუქტურა წარმოადგენს მონაცემთა სტრუქტურის ერთ-ერთ სახეს. იგი გახლავთ წყვილთა სიმრავლე. თითოეული წყვილი წარმოადგენს ”სახელი - მნიშვნელობა” ტიპის კონსტრუქციას, ანუ მოკლედ რომ ვთქვათ თვისებას. თვისებას აქვს სახელი, რომელიც ჩაიწერება ჩვეულებრივი

სტრიქონის (სიმბოლოთა მიმდევრობის) სახით და აქვს მნიშვნელობა, რომელიც მოიცემა ასევე სტრიქონის სახით, ან წარმოადგენს სხვა თვისებათა სტრუქტურას, ან ცარიელია. ეს გახლავთ რეკურსიული განმარტება. შესაბამისად დასაშვებია თვისებათა რამოდენიმე სტრუქტურა საკმარისად დიდ სიღრმეზე იყოს ერთიმეორეში ჩალაგებული. ჩვენს მიერ შემუშავებულ ფორმალიზმში თვისებათა სტრუქტურებს კვადრატულ ფრჩხილებში ვწერთ, ხოლო თვისების სახელს მისი მნიშვნელობისაგან ორწერტილით გამოვყოფთ. მაგალითად:

1.
[ფერი: წითელი]

2.
[რიცხვი: მხ
პირი: ’1’]

3.
[თვისება_1: მნიშვნელობა_1
თვისება_2:
[თვისება_3: მნიშვნელობა_3
თვისება_4: None]]

პირველ მაგალითში მოცემული თვისებათა სტრუქტურა შეიცავს მხოლოდ ერთ თვისებას, რომელის სახელია ”ფერი” და მნიშვნელობად აქვს ”წითელი”. ამ თვისებათა სტრუქტურით შესაძლებელია აღიწეროს მაგალითად შუქნიშნის მიმდინარე მდგომარეობა.

შემდეგ მაგალითში მოცემულ თვისებათა სტრუქტურას ორი თვისება გააჩნია. ”რიცხვი” და ”პირი”, რომელთა მნიშვნელობებია შესაბამისად ”მხ” (მხოლობითი) და ”1” (პირველი პირი).

მესამე მაგალითში ნაჩვენებია ერთმანეთში ჩალაგებული თვისებათა სტრუქტურები. მოცემული გვაქვს თვისებათა სტრუქტურა, რომელიც შეიცავს ორ

თვისებას. პირველი ”თვისება_1” რომლის მნიშვნელობაა ”მნიშვნელობა_1”, ხოლო მეორე (”თვისება_2”) თავადაა თვისებათა სტრუქტურა რომელიც ასევე ორელემენტიანია. ამავე მაგალითში მოცემული გვაქვს ცარიელი თვისება, ჩვენს ფორმალიზმში იგი აღინიშნება რეზერვირებული სიტყვა None -ით.

თვისებათა სტრუქტურები მეტად მოსახერხებელია სხვადასხვა ობიექტების ფორმალური აღწერისათვის, კომპიუტერში მათ თვისებათა წარმოსადგენად და დასამუშავებლად.

ამრიგად, ჩვენ გავნსაზღვრეთ თვისებათა სტრუქტურები და მათი მნიშვნელობათა სიმრავლე. ახლა გავნსაზღვროთ ის ოპერაციები, რომლებიც შეგვიძლია ჩავატაროთ თვისებათა სტრუქტურებზე. ეს ოპერაციებია:

- მინიჭება;
- ტოლობაზე შემოწმება;
- უნიფიკაცია;
- უნიფიკაციაზე შემოწმება (და მათი მრავალარგუმენტიანი ანალოგები. იხ. ქვემოთ).

ყოველი ოპერაცია შესრულების შემდეგ აბრუნებს ლოგიკურ მნიშვნელობას: მცდარს ან ჭეშმარიტს. განვიხილოთ თითოეული ოპერაცია ცალცალკე.

მოცემული გვაქვს ორი თვისებათა სტრუქტურა a და b. შემდეგი ოპერაცია:

a := b (მინიჭება)

წარმოადგენს მინიჭების ოპერაციას. მისი აზრობრივი მნიშვნელობა პროგრამირების ენაში მინიჭების ოპერატორის მნიშვნელობის ანალოგიურია. ოპერაციის შედეგად a თვისებათა სტრუქტურის შიგთავსი შეიცვლება b თვისებათა სტრუქტურის შიგთავსით და შედეგად ისინი გაუტოლდებიან ერთიმეორეს. მინიჭების ოპერაცია ყოველთვის ლოგიკურად ჭეშმარიტ მნიშვნელობას აბრუნებს.

a = b (ტოლობაზე შემოწმება)

ამ ოპერაციით ორი თვისებათა სტრუქტურა ერთმანეთთან ტოლობაზე მოწმდება. ორი თვისებათა სტრუქტურა ერთიმეორეს ტოლია, თუკი ისინი შეიცავენ

მხოლოდ ერთიდაიგივე თვისებებს, და ამ თვისებათა მნიშვნელობები ერთიმეორეს ემთხვევა. ტოლობის დადგენის შემთხვევაში ოპერაცია დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას, ხოლო თუ თვისებათა სტრუქტურები განსხვავდებიან ერთიმეორესგან დაბრუნდება ლოგიკურად მცდარი მნიშვნელობა.

a <== b (უნიფიკაცია)

ორი თვისებათა სტრუქტურა ერთიმეორესთან უნიფიცირდება თუკი მათი ერთნაირი სახელის მქონე თვისებების მნიშვნელობები არ ეწინააღმდეგებიან ერთიმეორეს. თვისებების მნიშვნელობები ერთიმეორეს არ ეწინააღმდეგებიან, მაშინ როდესაც:

1. ერთერთი მნიშვნელობა არის ცარიელი (None);
2. ორივე მნიშვნელობა მარტივია (არ არიან სხვა თვისებათა სტრუქტურები) და ისინი ერთიმეორეს ემთხვევა;
3. ორივე მნიშვნელობა რთულია (თავად არიან თვისებათა სტრუქტურები) და ისინი უნიფიცირდებიან ერთმანეთთან.

უნიფიკაციის შედეგად (თუკი იგი წარმატებით დასრულდა) ის ველები, რომლებიც არ გვხვდებიან a თვისებათა სტრუქტურაში ან გვხვდებიან და არიან ცარიელნი, მაგრამ გვხვდებიან b-ში, ავტომატურად გადმოიტანებიან a თვისებათა სტრუქტურაში თავის მნიშვნელობებთან ერთად. მაგალითად:

```
a = [f1: v1
     f2: [f3: None]]
b = [f4: v4
     f2: [f3: v3]]
```

a <== b უნიფიკაციის შედეგად (რომელიც იქნება წარმატებული) a თვისებათა სტრუქტურა მიიღებს შემდეგ მნიშვნელობას:

```
[f1: v1
```

f2: [f3: v3]

f4: v4]

b თვისებათა სტრუქტურის მნიშვნელობა, რომ ყოფილიყო მაგალითად:

[f4: v4

f2: v2]

უნიფიკაცია ვერ მოხდებოდა, და ოპერაცია დააბრუნებდა ლოგიკურად მცდარ მნიშვნელობას. უნიფიკაციის წარმატებით შესრულების შემთხვევაში ბრუნდება ლოგიკურად ჭეშმარიტი მნიშვნელობა, ხოლო როდესაც თვისებათა სტრუქტურები არ უნიფიცირდებიან ერთიმეორესთან ბრუნდება ლოგიკურად მცდარი მნიშვნელობა.

a == b (უნიფიკაციაზე შემოწმება)

უნიფიკაციაზე შემოწმების ოპერაცია. მუშაობს უნიფიკაციის ოპერაციის ანალოგიურად, თუმცა იგი აბრუნებს მხოლოდ და მხოლოდ უნიფიკაციის შედეგს, ლოგიკურად ჭეშმარიტ ან მცდარ მნიშვნელობას იმისდა მიხედვით ხერხდება, თუ არა მოცემულ მონაცემთა სტრუქტურების უნიფიცირება. თავად ოპერანდები ცვლილებას არ ექვემდებარებიან და ინარჩუნებენ იგივე მნიშვნელობას, რაც ოპერაციის ჩატარებამდე გააჩნდათ.

აქვე ავღნიშნოთ, რომ ყველა ამ ოპერაციის ჩაწერა ჩვენს ფორმალიზმში შესაძლებელია სხვა სახითაც, შესაბამისი ფუნქციის გამოძახების მეშვეობით. ყველა ჩვენს მიერ ჩამოთვლილ ოპერაციას აქვს თავისი სახელი, რომლითაც იგი რეგისტრირებულია სისტემაში. მაგალითად მინიჭების ოპერაციას (:=) შეესაბამება სახელი 'assign', ტოლობას (=) - სახელი 'equal', უნიფიკაციას (<==) - სახელი 'unify', უნიფიკაციის შემოწმებას კი - სახელი `unicheck'. ამ სახელების გამოყენებით ზემოთმოყვანილი ოპერაციები შეგვიძლია ასეთნაირად ჩავწეროთ:

a := b assign(a, b)

a = b equal(a, b)

a <== b unify(a, b)

a == b unicheck(a, b)

გარდა ამისა, არსებობს სპეციალური ფუნქცია nop() რომელიც გამოიძახება არგუმენტების გარეშე და ყოველთვის ჭეშმარიტ მნიშვნელობას აბრუნებს. მას რაიმე განსაკუთრებული დატვირთვა არ გააჩნია გარდა იმისა, რომ იგი შეგვიძლია გამოვიყენოთ როგორც კონსტანტა ლოგიკური გამოსახულებების შედგენისას ან სადიაგნოსტიკო მიზნებისთვის (გრამატიკის შედგენა/კორექტირების დროს).

ასევე გვაქვს ორი დამატებითი ფუნქცია. მათ შეგვიძლია ნებისმიერი რაოდენობის არგუმენტი გადავცეთ. ამიტომ, ისინი ჩაიწერებიან როგორც ფუნქციები და მათი გამოძახებისას ფრჩხილებში ხდება ყველა იმ პარამეტრის გადაცემა, რომლებიც მონაწილეობენ ოპერაციის შესრულებაში. ეს ფუნქციები გახლავთ: meq (multiple equation checking) და muc (multiple unification checking). მათი ჩაწერის ფორმატი შემდეგნაირია:

meq(e, a, b, c…)

muc(e, a, b, c…)

e - წარმოადგენს შესამოწმებელ თვისებათა სტრუქტურას, ეტალონს. ხოლო დანარჩენი პარამეტრები იმ თვისებათა სტრუქტურებს, რომლებთანაც უნდა შემოწმდეს პირველი პარამეტრი. meq - ნიშნავს მიმდევრობით რამოდენიმე ტოლობის შემოწმებას (equal-ის მრავალარგუმენტიანი ანალოგი), muc - მიმდევრობით რამოდენიმე უნიფიკაციის შემოწმებას (unicheck-ის მრავალარგუმენტიანი ანალოგი). meq (muc) ფუნქციის მუშაობის ალგორითმი შემდეგნაირია: მიმდევრობით ხდება პირველი არგუმენტის ყველა სხვა დანარჩენ არგუმენტთან შემოწმება, თუკი რომელიმე მათგანთან მოხდა დამთხვევა ტოლობაზე (უნიფიკაციაზე) ფუნქცია დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას, წინააღმდეგ შემთხვევაში ბრუნდება ლოგიკურად მცდარი მნიშვნელობა. ამ ფუქციების ძირითადი გამოყენება მდგომარეობს გამოსახულებების ჩაწერის შემოკლებაში, ვინაიდან იმ შემთხვევაში, თუ დაგვჭირდება რამოდენიმე

ალტერნატივის შემოწმება, არ არის საჭირო ყოველი მათგანის ცალკე დაწერა. შეგვიძლია ისინი გავაერთიანოთ რომელიმე ამ ფუნქციაში. ეს ფუნქციები მეტად მოხერხებულია და პრაქტიკაში ხშირად გვიხდება მათი გამოყენება. სიმარტივისათვის შემოღებულ იქნა მათი ჩაწერის ოპერატორული ფორმა:

meq(e, a, b, c,…) ---------- e = (a, b, c,…)

muc(e, a, b, c,…) ---------- e == (a, b, c,…)

ბოლოს შევნიშნოთ, რომ გარდა ჩვენს მიერ შემოღებული სტანდარტული ფუნქციების ნაკრებისა, შესაძლებელია პროგრამულად დავამატოთ ნებისმიერი სხვა ფუნქცია თუკი მომავალში ამის საჭიროება იქნება. ამისთვის, საკმარისია პროგრამის კოდში შესაბამისი დამატებების შეტანა.

თავი 2

მორფოლოგიური გამრჩევი

§1. მორფოლოგიური ანალიზის ამოცანა

ბუნებრივი ენის ანალიზის ერთერთი შემადგენელი ნაწილი გახლავთ მორფოლოგიური ანალიზი. რაც გულისხმობს სიტყვის მორფემებად დაშლას და ამ დაშლის საფუძველზე სიტყვის შესახებ ძირითადი გრამატიკული ინფორმაციის დადგენას. ასეთ ინფორმაციას განეკუთვნებიან: მეტყველების ნაწილი, რიცხვი, ბრუნვა, პირი... და სხვა. შემდგომში ამ ინფორმაციის გამოყენება შეიძლება სინტაქსური ანალიზის ეტაპზე. მორფოლოგიური ანალიზის ჩატარება, თავიდან გვარიდებს იმის აუცილებლობას, რომ ენაში არსებული ყველა სიტყვაფორმა

შევიტანოთ ლექსიკონში და მივუთითოთ მათი ძირითადი გრამატიკული კატეგორიები. ამის მაგივრად, ლექსიკონში შეგვიძლია შევიტანოთ სიტყვის უცვლელი ნაწილი, ხოლო ყველა სხვა სიტყვაფორმის შესახებ ინფორმაციის მიღება შესაძლებელი ხდება მორფოლოგიური ანალიზის მეშვეობით. ცხადია, რომ ამით პროგრამის სამუშაო ლექსიკონის მოცულობა მკვეთრად მცირდება. განსაკუთრებით ეს მნიშვნელოვანია ქართული ენისა და სხვა მსგავსი ენებისათვის, რომლებიც მორფოლოგიური თვალსაზრისით მდიდარი ენებია. მორფოლოგიური წესების გამოყენებით სიტყვის უცვლელი ნაწილისგან შესაძლებელია მრავალი სიტყვაფორმის წარმოქმნა. ამის ნათელი მაგალითი ქართულ ენაში გახლავთ ზმნები, სადაც ერთი ძირისგან ზმნის ათეულობით სხვადასხვა ფორმის მიღებაა შესაძლებელი.

ჩამოვაყალიბოთ მორფოლოგიური ანალიზის ამოცანა მათემატიკურად. პირველ რიგში განვიხილოთ ის როგორც წმინდა კომბინატორული ამოცანა, სადაც ძირითად მიზანს წარმოადგენს სიტყვის მორფემებად დაშლა:

მოცემული გვაქვს არაცარიელი ანბანი Σ.

ასევე მოცემული გვაქვს N რაოდენობა მორფემათა სიმრავლე M_1 M_2 M_3 ... M_N.

მორფემათა სიმრავლე M_i განისაზღვრება, როგორც Σ ანბანით ჩაწერილ სტრიქონთა სიმრავლე. $M_i = \{ m_1, m_2, ... m_k \}, k > 0$, სადაც მორფემა m_i ეკუთვნის Σ^* -ს (შესაძლებელია m_i ცარიელი სტრიქონი იყოს).

მოცემული გვაქვს Σ ანბანით განსაზღვრული შემავალი სიტყვა (სტრიქონი) α, რომელიც უნდა დაიშალოს მორფემებად.

საჭიროა ვიპოვოთ მორფემათა ყველა ისეთი კომბინაცია $m_{i1}, m_{i2}, ... m_{iN}$ (სადაც m_{ik} ეკუთვნის M_K -ს), რომლებიც კონკატენაციის შედეგად გვაძლევს α საწყის სიტყვას.

ცხადია რომ ზოგ შემთხვევაში შეიძლება ვერ მოხერხდეს სიტყვის დაშლა. ამ შემთხვევაში ამონახსნთა სიმრავლე ცარიელი იქნება. აგრეთვე შესაძლებელია მრავალი ამონახსნი მივიღოთ შედეგად. მოცემული ამოცანა წარმოადგენს ტიპიურ დისკრეტულ ამოცანას. სადაც განსაზღვრის არე სასრული სიმრავლეა. შესაბამისად

ჩვენ ყოველთვის შეგვიძლია ამ ამოცანის ამოხსნა თუკი გადავარჩევთ მორფემათა ყველა კომბინაციას. ვინაიდან დასმული ამოცანა არ არის წრფივი და არ გვაძლევს ცალსახა შედეგს, შესაბამისად ვერ მოხერხდება მისი ანალიზური ამოხსნა, ან თუნდაც წრფივი ალგორითმის შედგენა. ამიტომ აქ საჭირო გახდა გადარჩევის (ძიება უკან დაბრუნებით) ალგორითმის გამოყენება.

განვიხილოთ კონკრეტული მაგალითი და ვაჩვენოთ თუ როგორ მიმდინარეობს ამონახსნის ძიების პროცესი.

მოცემული გვაქვს ანბანი { a, b } და შემდეგი მორფემათა კლასები:

M1 = { “a”, “aab”, “” }

M2 = { “bb”, “ab”, “b”, “” }

M3 = { “aa“, “baa” }

შევნიშნოთ რომ M1 და M2 შეიცავენ ცარიელ მორფემას. მოცემულია შემავალი სიტყვა ”aabbbaa”

მივყვეთ ძიების ალგორითმს:

1. ვიღებთ M1 -დან პირველ მორფემა ”a” -ს. ვხედავთ რომ იგი ემთხვევა შემავალი სიტყვის პირველ სიმბოლოს: a-abbbaa. ვიმახსოვრებთ არჩეულ მორფემას. გადავდივართ M2 -ზე და ვცდილობთ დარჩენილი სიტყვის (abbbaa) დაშლას.
2. M2 -დან ვიღებთ პირველ მორფემას ”bb”-ს,მაგრამ იგი არ ემთხვევა დარჩენილი სიტყვის (abbbaa) საწყის სიმბოლოებს. მაშასადამე ამ ვარიანტს უკუვაგდებთ. ვიღებთ მეორე მორფემას M2 -დან, ეს გახლავთ ”ab”. იგი ზუსტად ემთხვევა განსახილველი სიტყვის პირველ ორ სიმბოლოს. შესაბამისად ვიმახსოვრებთ ამ ვარიანტს, გადავდივართ M3 -ზე და ვცდილობთ გავარჩიოთ სიტყვის დარჩენილი ნაწილი (bbaa).
3. სამწუხაროდ ორივე მორფემიდან, რომელიც M3 -ში გვხვდება არცერთი არ ემთხვევა დარჩენილ სიტყვას (bbaa). გამოგვაქვს დასკვნა, რომ წინა არჩევანი მცდარი იყო. ვბრუნდებით ერთი ნაბიჯით უკან M2 -ში და ვცდილობთ სხვა დასაშვები ვარიანტის მონახვას.
4. M2 -ში განსახლიველი დაგვრჩა ორი ვარიანტი, ”b” და ცარიელი მორფემა (მოცემული მომენტისათვის განსახილველი სიტყვა იყო - abbbaa). როგორც ვხედავთ ”b” არ ემთხვევა განსახილველი სიტყვის პირველ სიმბოლოს.

5. M3 -ში არსებული არცერთი მორფემა არ ემთხვევა სიტყვის დარჩენილ ნაწილს. შესაბამისად ისევ უკან ვბრუნდებით სხვა ვარიანტების საძებნად.
6. M2 -ში უკვე ამოვწურეთ ყველა ვარიანტი. ვბრუნდებით M1 -ში.
7. M1 -ში გადავდივართ შემდეგ ვარიანტზე: ”aab”. განსახილველი სიტყვაა aabbbaa. სიტყვის პირველი სიმბოლოები ემთხვევა არჩეულ მორფემას. ვიმახსოვრებთ ამ ვარიანტს. გადავდივართ M2 -ზე. განსახილველ სიტყვად გვაქვს bbaa.
8. M2 -დან პირველი მორფემა ”bb” ემთხვევა განსახილველი სიტყვის დასაწყისს. ვიმახსოვრებთ ამ ვარიანტს. გადავდივართ M3 -ზე. განსახილველი სიტყვაა ”aa”.
9. M3 -ში პირველი მორფემა ”aa” ემთხვევა დარჩენილ სიტყვას. ანუ ნაპოვნია ამონახსნი რომლის მიხედვითაც საწყისი სიტყვა aabbbaa დაიშალა როგორც aab-bb-aa. ვიმახსოვრებთ ამ ამონახსნს და ვაგრძელებთ სხვა ამონახსნების ძიებას. M3 -ში სხვა ვარიანტები არ ემთხვევა. ვბრუნდებით ერთი ნაბიჯით უკან.
10. განსახილველი სიტყვაა “bbaa”. შემდეგი მორფემა M2 -ში ”ab” არ ემთხვევა სიტყვის დასაწყისს, თუმცა მისი მომდევნო მორფემა ”b” ემთხვევა. ვიმახსოვრებთ ამ ვარიანტს და გადავდივართ M3 -ზე.
11. დარჩენილი სიტყვაა “baa”. იგი ზუსტად ემთხვევა მეორე მორფემას M3-დან. მაშასადამე გვაქვს უკვე მეორე ამონახსნი: aab-b-baa. ვიმახსოვრებთ ნაპოვნ ამონახსნს და ვბრუნდებით უკან ძიების გასაგრძელებლად.

12. M2 -ში შეგვიძლია ცარიელი მორფემის გამოყენება. თუმცა მას ჩიხისკენ მივყავართ რადგან მომდევნო ნაბიჯზე M3 -დან არცერთი ვარიანტი არ ესადაგება სიტყვის დარჩენილ ნაწილს. ამიტომ ვუბრუნდებით M1 -ს.
13. M1 -ში ასევე ცარიელი მორფემა დაგვრჩა განსახილველი. ადვილი შესამოწმებელია რომ მისი გამოყენება შედეგს არ გვაძლევს, ვინაიდან დარჩენილი ორი კლასიდან ნებისმიერი წყვილის კონკატენაცია 5-ზე მეტის სიგრძის სიტყვას არ გვაძლევს. შესაბამისად იგი ვერაფრით ვერ დაემთხვევა თავდაპირველ განსახილველ სიტყვას რომლის სიგრძეც 6-ის ტოლია. შესაბამისად დარჩენილ ძიების პროცესში ალგორითმი ახალ ამონახსნს ვერ მოგვცემს.

საბოლოოდ მივიღეთ საწყისი სიტყვის დაშლის ორი დასაშვები ვარიანტი:

1. aab-b-baa
2. aab-bb-aa

ჩვენს მიერ განხილული ძიების ალგორითმი ჩვეულებრივ არ გამოირჩევა დიდი ეფექტურობით. თუმცა სიტყვის მორფემებად დაშლის ამოცანაში ყველაფერი დამოკიდებულია განსახილველ სიტყვაზე და მორფემათა კლასების შემადგენლობაზე. ბუნებრივ ენებში, მათ შორის ქართულ ენაში, თუკი განვიხილავთ მორფემათა შესაძლო ვარიანტებს, შევამჩნევთ რომ ძიების პროცესში შესაძლო განშტოებათა ვარიანტები არც თუ ისე ბევრია. პრაქტიკულმა ექსპერიმენტებმაც გვანახა რომ მორფოლოგიური ანალიზის ალგორითმი საკმაოდ სწრაფია. აქ უნდა გავითვალისწინოთ ისიც, რომ გარდა წმინდა კომბინატორული ძიებისა, ჩვენს ალგორითმში დამატებულია აგრეთვე ანალიზის სხვადასხვა ეტაპზე შეზღუდვების დადების შესაძლებლობა. რაც კიდევ უფრო მეტად კვეცავს არასასურველი განშტოებებს ვინაიდან დროულად ხდება შეზღუდვების შემოწმება და იმ ვარიანტების ამოგდება, რომლებიც არ აკმაყოფილებენ დადებულ შეზღუდვებს.

განვიხილოთ, თუ როგორ მუშაობს შეზღუდვების მექანიზმი მორფოლოგიური ანალიზის პროცესში. სიტყვის მორფემებად დაშლის ჩვენს მიერ განხილული მათემატიკური მოდელი ბოლომდე ვერ ასახავს ბუნებრივი ენის მორფოლოგიის თავისებურობებს. საქმე იმაშია, რომ მხოლოდ სიტყვიდან მორფემების გამოყოფა არ არის საკმარისი სწორი მორფოლოგიური ანალიზისათვის. ხშირ შემთხვევაში უნდა მოხდეს შემოწმება, თუ როგორ კავშირში არიან ერთმანეთთან გამოყოფილი

მორფემები. მორფემათა ზოგიერთი კომბინაცია კონკრეტული ენის მორფოლოგიის წესებით შესაძლოა არ იყოს დასაშვები. შესაბამისად მორფოლოგიური ანალიზის ალგორითმს უნდა შეეძლოს ამგვარი ვარიანტების ამოგდება ამონახსნთა სიმრავლიდან. განვიხილოთ შემდეგი მაგალითი:

ავიღოთ სიტყვა ”ვაშლები” და შევეცადოთ მის მორფემებად დაშლას. შედეგად ვიღებთ ასეთ კომბინაციას: ”ვაშლ”-”ებ”-”ი”. სადაც ”ვაშლ” არის ფუძე, ”ებ” რიცხვის ნიშანი და ”ი” ბრუნვის ნიშანი. ახლა ავიღოთ სიტყვა ”ხალხები”. ჩვეულებრივი მორფოლოგიური ანალიზით ამ სიტყვასაც ანალოგიურად ვშლით: ”ხალხ” – ”ებ” – ”ი”. მაგრამ ქართულ ენაში ასეთი სიტყვა არ გამოიყენება. ანუ ეს დაშლა არის მცდარი, ვინაიდან ფუძე ”ხალხ” არ შეიძლება იხმარებოდეს ”ებ” რიცხვის ნიშანთან. სემანტიკურად ეს ცხადია ვინაიდან სიტყვა ხალხი თავად მრავლობითი დატვირთვის მატარებელია და მისი მრავლობით რიცხვში ხმარება არაკორექტულია (გამონაკლისია მრავლობითის ნართანიანი ფორმები: ხალხნი და ხალხთა). აქედან ჩანს რომ საჭიროა შემოვიტანოთ დამატებითი მექნიზმი, რომელიც ენის მორფოლოგიური წესების საუძველზე შეამოწმებს ეთანხმებიან თუ არა სიტყვაში გამოყოფილი მორფემები ერთმანეთს. სწორედ ამ მიზანს ემსახურება თვისებათა შეზღუდვების მექანიზმი. მისი პრინციპი შემდეგია, წესში ყოველ მორფემას შეესაბამება თავისი თვისება. ამ თვისებების საწყისი მნიშვნელობა მიეთითება მორფოლოგიის ფაილში მორფემების აღწერის დროს. ამასთანავე წესში შეგვიძლია ჩავწეროთ შეზღუდვები, რომელთა მეშვეობითაც ვამოწმებთ მორფემათა თვისებებს და ვადგენთ შეესაბამებიან თუ არა ისინი ერთმანეთს. შეზღუდვები წარმოადგენენ ლოგიკურ გამოსახულებას. თუკი ამ ლოგიკური გამოსახულების გამოთვლის შედეგად მივიღეთ ლოგიკურად მცდარი მნიშვნელობა მოხდება მიმდინარე ვარიანტის უკუდგება და ძიება გაგრძელდება. ფორმალურად მორფოლოგიური წესის ჩაწერა შემდეგი სახით ხდება:

$$W \rightarrow M_1 \{ C_1 \} M_2 \{C_2\} \dots M_N \{ C_N \} ;$$

სადაც C_i წარმოადგენს შეზღუდვებს. შესაძლებელია შეზღუდვების გადანაწილება მორფემებთან მიმართებაში სხვადასხვა ადგილას. რაც უფრო მარცხნივ არის

შეზღუდვა მით უფრო ადრეულ ეტაპზე ხდება მისი შემოწმება. თუმცა შეზღუდვას შეუძლია ოპერირება მოახდინოს მხოლოდ იმ მორფემათა თვისებებზე, რომლებიც მის აღწერამდე არიან მითითებულნი. ამ მეთოდში საინტერესო გახლავთ ის, რომ პრაქტიკულად ჩვენ ვინარჩუნებთ თავდაპირველად აღწერილ მათემატიკურ მოდელს. ძიების ალგორითმი მიმდინარეობს იგივენაირად, მაგრამ დამატებით ჩვენ წესში შეგვიძლია ჩავრთოთ შემოწმებები, რომელთა მეშვეობითაც ვზღუდავთ ამონახსნების სიმრავლეს და უკუვაგდებთ ძიების პროცესში არასასურველ განშტოებებს. ამასთანავე შეზღუდვების შემოწმება ხდება დინამიურად, სიტყვის დაშლის პროცესში. შესაბამისად უფრო მარცხნივ მდგარი შეზღუდვები უფრო ადრე შემოწმდებიან, რაც ალგორითმის ეფექტურობას მკვეთრად ამაღლებს.

§2. ფორმალიზმი

მორფოლოგიური გამრჩევისათვის შემოშავებული იქნა სპეციალური ფორმალიზმი, რომლის ფარგლებშიც ხდება ბუნებრივი ენის მორფოლოგიის აღწერა. მორფოლოგიური ანალიზის პროგრამას გაშვების ეტაპზე გადავცემთ მორფოლოგიის ფაილს, რომელშიც სწორედ ამ ფორმალიზმის მეშვეობით არის ჩაწერილი ბუნებრივი ენის მორფოლოგია. გამრჩევი ჩატვირთავს მორფოლოგიის ფაილს და მასში არსებულ მორფოლოგიურ წესებს გადაიყვანს პროგრამის შიდა წარმოდგენაში. ძირითადი მოთხოვნები, რომლებიც დასმული იყო ფორმალიზმის შემუშავების წინ გახლდათ მისი უნივერსალურობა (ენების რაც შეიძლება ფართო კლასისათვის გამოყენების შესაძლებლობა), ალგორითმული რეალიზაციის ეფექტურობა და მორფოლოგიის ფაილის კომპაქტურად ჩაწერის შესაძლებლობა.

მორფოლოგიის ფაილი პირობითად ორ ნაწილად შეგვიძლია გავყოთ: მორფემების კლასების აღწერად და წესების აღწერად.

მორფემების კლასის აღწერა იწყება ’@’ სიმბოლოთი, რომელსაც მოსდევს მორფემების კლასი სახელი, ტოლობის ნიშანი ’=’ და ფიგურულ ფრჩხილებში ჩამოწერილი მორფემების კლასის შესაძლო მნიშვნელობები. მორფემების კლასის შესაძლო წარმომადგენლები იწერება ბრჭყალეში. მას მოსდევს თვისებათა

სტრუქტურა (შესაძლოა ცარიელი). განიხილოთ შემდეგი მორფემების კლასის აღწერა:

```
@ბრუნვა =
  {
  ”ი”   [],
  ”მა”  [],
  ”ის”  [],
  ”ს”   [],
  ”ით” [],
  ”თ”   [],
  ”თი” [],
  ”ად” [],
  ”დ”   [],
  ”ო”   [],
  ”ვ”   [],
  ”მ”   [],
  ”ა”   [],
  ””     []
  }
```

ამ ჩანაწერით განისაზღვრება მორფემათა კლასი ”ბრუნვა”. მისი შესაძლო ლექსიკური მნიშვნელობებია ”ი”, ”მა”, ”ის” ... და ა.შ. ანუ ქართულ ენაში არსებული ყველა შესაძლო ბრუნვის ნიშანი. აგრეთვე ცარიელი სტრიქონი ””, რომელიც გამოიყენება მორფემის ცარიელი მნიშვნელობის აღსანიშნავად. მისი არსებობა მორფემის აღწერაში ნიშნავს, რომ ეს მორფემა შესაძლოა ცარიელი იყოს, ანუ იგი არ გვხვდებოდეს განსახილველ სიტყვაში. მაგალითად, სიტყვა ”სახლთან” (არსებითი სახელი) იშლება ორ მორფემად ”სახლ” (ფუძე) და ”თან” (თანდებული). მასში არ გვხვდება ბრუნვის ნიშანი, თუმცა ამის გამო ჩვენ არ გვაქვს უფლება არსებითი სახელის მწარმოებელი წესიდან ამოვიღოთ მორფემა ”ბრუნვა” ვინაიდან არსებით სახელთა უმეტესობას გააჩნია იგი. იმისათვის, რომ თავიდან აგვერიდებინა

ერთიდაიგივე წესების დუბლირება მათში მორფემების სხვადასხვა შესაძლო განლაგებებით, შემოტანილი იქნა ცარიელი მორფემის ცნება და შემუშავებულ ფორმალიზმში შეგვიძლია მისი გამოყენება მორფემათა კლასის ერთერთი სავარაუდო მნიშვნელობად. მორფემათა კლასის აღწერის შემდეგ ჩვენ შეგვიძლია ეს მორფემათა კლასი წესის აღწერაში გამოვიყენოთ. განვიხილოთ მაგალითი, სადაც მორფემის აღწერაში მის შესაძლო მნიშვნელობებს მიეთითებათ შესაბამისი თვისებები:

@რიცხვი =
{
”ებ” [რიცხვი: მრავლობითი],
”თ” [რიცხვი: მრავლობითი_ნართანიანი],
”ნ” [რიცხვი: მრავლობითი_ნართანიანი],
”” [რიცხვი: მხოლობითი]
}

მოცემულ მაგალითში თვისება ”რიცხვი”-ს ყველა შესაძლო მნიშვნელობას შეესაბამება მისთვის დამახასიათებელი თვისებათა სტრუქტურა. მაგალითად სიტყვაში არსებული ”ებ” რიცხვის ნიშანი გვიჩვენებს რომ სიტყვა მრავლობით რიცხვში დგას. შესაბამისად ჩვენ ამას ვუთითებთ მორფემის აღწერისას თვისებათა სტრუქტურის მეშვეობით. სიტყვის გარჩევის ეტაპზე მოხდება ამ ინფორმაციის გადატანა წესში არსებულ მორფემებში და მისი გამოყენება შესაძლებელი იქნება შეზღუდვებში.

განვიხილოთ წესის აღწერა მორფოლოგიის ფაილში. უმარტივეს შემთხვევაში თუკი წესს შეზღუდვები არ გააჩნია მისი აღწერა შემდეგნაირად გამოიყურება:

სიტყვა -> მორფემა_1 მორფემა_2 ... მორფემა_N ;

წესის მარცხენა მხარეში დგას ამ წესის სახელი, ხოლო მარჯვენა მხარეში ჩამოთვლილია მისი შემადგენელი მორფემები. ეს მორფემები მორფოლოგიის ფაილში უნდა განსაზღვრულნი იყვნენ ამ წესის აღწერამდე. მორფოლოგიის ფაილში

შეიძლება მოცემული იყოს მრავალი ამგვარი წესი, თითოეული მათგანი წარმოადგენს სიტყვის დაშლის შესაძლო ვარიანტს. ანალიზატორი სიტყვის გარჩევის დროს განიხილავს ყველა შესაძლო წესს და შედეგად მოგვცემს სიტყვის ყველა დასაშვებ დაშლას. შესაძლებელია ერთი სიტყვა რამოდენიმე სხვადასხვა წესის მიხედვით დაიშალოს, ასეთი ომონიმური შემთხვევები საკმაოდ ხშირია ბუნებრივ ენაში. მაგალითად, სიტყვა ”ველი” ქართული ენის გრამატიკის მიხედვით ორგვარად შეიძლება დაიშალოს, პირველ შემთხვევაში იგი წარმოადგენს არსებით სახელს, რომელის ფუძეა ”ველ” და ბრუნვის ნიშანი ”ი” (მაგალითად: მწვანე ველი), ხოლო მეორე შემთხვევაში იგივე სიტყვა წარმოადგენს ორპირიან ზმნას (მაგალითად: მე ველი მას). განვიხილოთ მარტივი მაგალითი:

@მ1 = { ”ა”, ”აბ”, ”” }
@მ2 = { ”ბბა”, ”ბა” }

სიტყვა -> მ1 მ2 ;

ამ მაგალითში თავდაპირველად განსაზღვრულია ორი მორფემა. პირველი მორფემა არის ”მ1” რომელის შესაძლო მნიშვნელობებია ”ა”, ”აბ” და ცარიელი მნიშვნელობა ””. მეორე მორფემა არის ”მ2”, მას ორი შესაძლო მნიშვნელობა აქვს ”ბბა” და ”ბა”. აგრეთვე გვაქვს ერთი მარტივი წესი, რომელიც სიტყვას აღწერს როგორც მ1 და მ2 მორფემების მიმდევრობას. მორფოლოგიურ ანალიზატორს გასარჩევად მივაწოდოთ სიტყვა ”აბბა”. მისი გარჩევის პროცესში მორფოლოგიური ანალიზატორი გადასინჯავს მორფემათა ყველა შესაძლო ვარიანტს, და ეკრანზე გამოიტანს სიტყვის დაშლის ყველა დასაშვებ ვარიანტს. ჩვენს შემთხვევაში სიტყვა ”აბბა” შეიძლება ორგვარად დაიშალოს:

1. ”აბ” ”ბა”
2. ”ა” ”ბბა”

მორფემათა მნიშვნელობების არც ერთი სხვა კომბინაცია ამ კონკრეტულ შემთხვევაში არ იძლევა ”აბბა” სიტყვის დაშლას. თუ გამრჩევს მივაწვდით სიტყვა ”ბბა”-ს, ვნახავთ რომ მოცემული მორფემების და წესების ფარგლებში ანალიზატორი გაარჩევს ამ სიტყვას და მოგვცემს ამონახსნს. ვინაიდან მორფემა მ1-ის მნიშვნელობა შეიძლება

იყოს ცარიელი, ხოლო მორფემა მ2-ის ერთერთი მნიშვნელობა არის ”ბბა”, შეტანილი სიტყვა დაიშლება შემდეგ კომბინაციად:

”” ”ბბა”

მორფემათა მიმდევრობის გარდა, წესში შესაძლოა გვხვდებოდეს შეზღუდვები. შემუშავებული ფორმალიზმის და მისი რეალიზაციის ერთერთ მნიშვნელოვან სიახლეს წარმოადგენს ის, რომ შესაძლებელია წესში შეზღუდვების გადანაწილება სიტყვის მორფემებად დაშლის სხვადასხვა ეტაპზე, რაც საშუალებას გვაძლევს დროულად მოხდეს აღმოჩენილ მორფემათა შემოწმება და იმ ალტერნატივების უარყოფა, რომლებიც არ აკმაყოფილებენ დადებულ შეზღუდვებს. ზოგადი სახით წესის აღწერა ასე გამოიყურება:

წესი -> მორფემა_1 { შეზღუდვები_1} მორფემა_2 { შეზღუდვები_2 } ... მორფემა_N { შეზღუდვები_N } ;

შეზღუდვები მოსდევს მორფემას და იწერება ფიგურულ ფრჩხილებში. ყოველი მორფემის შემდეგ შეგვიძლია ჩავურთოთ შეზღუდვა, და მისი შემოწმება მოხდება მაშინ, როდესაც გამრჩევი ამ კონკრეტულ მორფემას შეუსაბამებს გარკვეულ დასაშვებ მნიშვნელობას. სიტყვის მორფემებად დაშლა მიმდინარეობს მარცხნიდან მარჯვნივ, შესაბამისად უფრო მარცხნივ მდგომი მორფემები უფრო ადრე მიიღებენ მნიშვნელობებს და უფრო მარცხნივ მგომი შეზღუდვები უფრო ადრეულ ეტაპზე ამუშავდებიან, რაც საშუალებას გვაძლევს არ დაველოდოთ სიტყვის ბოლომდე დაშლას და დროულად შევამოწმოთ შეზღუდვები. ბუნებრივია, შეზღუდვებში შეგვიძლია მხოლოდ იმ მორფემების შემოწმება, რომლებიც შეზღუდვების მარცხენა მხარეს მდებარეობენ, ანუ განსაზღვრულნი იქნენ მანამდე, სანამ ამუშავდებოდა შეზღუდვა. მაგალითად, ”შეზღუდვა_2”-ში შეგვიძლია გამოვიყენოთ მხოლოდ ”მორფემა_1” და ”მორფემა_2”. თუკი შეზღუდვაში გვინდა ნებისმიერი მორფემის გამოყენება, მაშინ იგი უნდა დავწეროთ წესის ბოლოში (ბოლო მორფემის შემდეგ).

შეზღუდვები წარმოადგენს ერთი ან რამოდენიმე შეზღუდვისაგან შემდგარ ლოგიკურ გამოსახულებას. შეზღუდვა არის თვისებათა სტრუქტურაზე განსაზღვრული ნებისმიერი ოპერაცია (ტოლობაზე შემოწმება, უნიფიკაცია, და ა.შ.).

როგორც ვიცით ეს ოპერაციები აბრუნებენ ლოგიკურად ჭეშმარიტ ან ლოგიკურად მცდარ მნიშვნელობებს. შესაბამისად მათი გაერთიანება შეგვიძლია შეზღუდვებში ძირითადი ლოგიკური ოპერაციების გამოყენებით. ეს ოპერაციებია: ~ (უარყოფა), & (ლოგიკური ნამრავლი), | (ლოგიკური ჯამი). შეზღუდვების ამუშავების დროს ხდება ამ ლოგიკური გამოსახულების გამოთვლა. თუკი გამოსახულება შედეგად დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას, ჩაითვლება რომ წესი აკმაყოფილებს დადებულ შეზღუდვას და გაგრძელდება სიტყვის დამუშავება, წინააღმდეგ შემთხვევაში მოხდება მიმდინარე ვარიანტის უკუგდება და გარჩევის ალგორითმი შეეცდება სიტყვის დაშლის სხვა ვარიანტები მონახოს. მნიშვნელოვანია აღინიშნოს, რომ რაც უფრო ადრეულ ეტაპზე მოხდება შეზღუდვის დადება და შემოწმება მით უფრო გაიზრდება გარჩევის სისწრაფე, ვინაიდან არასაჭირო ალტერნატივების უკუგდება გარჩევის ადრეულ ეტაპებზე მოხდება.

იმის გარდა, რომ შეზღუდვები საშუალებას გვაძლევს შევამოწმოთ მორფემებს შორის დამოკიდებულებები და ამით დროულად უკუვაგდოთ სიტყვის დაშლის გრამატიკულად არასწორი ვარიანტები, შეზღუდვების მეშვეობით ჩვენ აგრეთვე შეგვიძლია ანალიზის შედეგად მიღებული ინფორმაცია გადავიტანოთ გამოსავალ შედეგში. მორფოლოგიური ანალიზის შედეგად ჩვენ ვიღებთ ამონახსნს, რომელიც გვეუბნება, თუ რომელი წესის მიხედვით მოხდა სიტყვის დაშლა. გარდა ამისა, ყოველ ამონახსნს აქვს თავისი თვისება, სადაც გარჩევის ეტაპზე შეგვიძლია ჩავწეროთ გარკვეული ინფორმაცია შედეგის შესახებ. ამის გაკეთება შესაძლებელია სწორედ შეზღუდვების მეშვეობით. მაგალითად, განვიხილოთ ქართული ენისათვის არსებითი სახელის მორფოლოგიური დაშლის წესის შემოკლებული ვარიანტი:

```
არს_სახ ->
        ფუძე { <არს_სახ ფუძე> := <ფუძე ლექს>}
        რიცხვი
                {
                ( <რიცხვი რიცხვი> = ”მრ” & <ფუძე რიცხვის_ტიპი> = (”1” ”3”) |
                <რიცხვი რიცხვი> = ”მხ” & <ფუძე რიცხვის_ტიპი> = (”1” ”2”) |
                <რიცხვი რიცხვი> = ”მრ_ნართ”
                ) &
```

<არს_სახ რიცხვი> := <რიცხვი რიცხვი>

}

ბრუნვა

თანდებული

ნაწილაკი

გავრცობის_ნიშანი

;

პირველ შეზღუდვაში, რომელიც მორფემა ”ფუძის” შემდგომ დგას, არსებითად გვაქვს ინფორმაციის გადატანა მარცხენა მხარეში. მორფემა ”ფუძის” ლექსიკური მნიშვნელობა გადადის არსებითი სახელის თვისება ”ფუძეში”. აქ შევნიშნოთ, რომ მორფემა ”ფუძის” თვისება ”ლექს” ანალიზატორის მიერ ავტომატურად არის გენერირებული. სიტყვის გარჩევის დროს, როდესაც წესის ამუშავებისას შესაბამისი მორფემები მიიღებენ მნიშვნელობას, მათ შესაბამის თვისებათა სტრუქტურებს ავტომატურად უჩნდებათ თვისება ”ლექს”, ანუ მორფემის ლექსიკური მნიშვნელობა (”ლექს” სახელს პროგრამა გაჩუმებით ანიჭებს შესაბამის თვისებას, თუმცა ჩვენ სპეციალური დირექტივის მეშვეობით შეგვიძლია ამ სახელის შეცვლა ნებისმიერ სხვა სახელზე). მოყვანილ მაგალითში გამრჩევი როგორც კი დაიწყებს სიტყვის ანალიზს და პირველ მორფემას (”ფუძეს”) მიანიჭებს დასაშვებ მნიშვნელობას, მაშინვე ამუშავდება ამ მორფემის შემდგომ მდგარი შეზღუდვები. ჩვენს შემთხვევაში მოხდება ინფორმაციის გადატანა მარცხენა მხარეში. ვინაიდან თვისებათა სტრუქტურებზე განსაზღვრული მინიჭების ოპერაცია ყოველთვის აბრუნებს ლოგიკურად ჭემარიტ მნიშვნელობას, პროგრამა ჩათვლის, რომ შეზღუდვები დაკმაყოფილებულია და გადავა მეორე მორფემის ამოცნობაზე. როდესაც, მეორე მორფემას (”რიცხვს”) შეუსაბამებს დასაშვებ მნიშვნელობას, მოხდება ამ მორფემის შემდგომ მდგარი შეზღუდვების ამოქმედება. კერძოდ, მოხდება შემოწმება აკმაყოფილებს თუ არა ფუძის ბრუნვის ტიპი და რიცხვის ნიშანი ერთიმეორეს. ამის შემოწმება საჭირო ხდება, რადგან ზოგიერთი სიტყვა მოითხოვს მხოლოდ მხოლობით ან მხოლოდ მრავლობით რიცხვს. თუკი რიცხვში შეთანხმება მოხდა, მაშინ მოხდება მარცხენა მხარეში ინფორმაციის გადმოტანა სიტყვის რიცხვის შესახებ. შევნიშნოთ, რომ

ლოგიკური გამოსახულების სახით ჩაწერილი შეზღუდვები ძალზედ მოხერხებულად გამოიყენება შემოწმებების და ინფორმაციის გადმოტანის კომბინირებისთვის.

როგორც უკვე ავღნიშნეთ, წესში არსებულ ყველა მორფემას მორფოლოგიური ანალიზატორი სიტყვის გარჩევის ეტაპზე შეუსაბამებს გარკვეულ თვისებათა სტრუქტურას. ამ თვისებათა სტრუქტურის წამოღება ხდება მორფემის აღწერისას მითითებული შესაბამისი თვისებათა სტრუქტურიდან. შესაძლებელია აღწერის დროს თვისებათა სტრუქტურა ცარიელი იყოს. თუმცა მეტწილად მასში იწერება ის მორფოლოგიური ანალიზისათვის საჭირო ინფორმაცია, რომელიც გააჩნია მორფემის კონკრეტულ მნიშვნელობას. წესში მორფემასთან ასოცირებულ თვისებას აღვნიშნავთ სამკუთხა ფრჩხილების გამოყენებით. მაგალითად ჩანაწერი <მორფემა_1> მიუთითებს იმ თვისებათა სტრუქტურაზე, რომელიც გააჩნია ”მორფემა_1”-ს. თუ გვაინტერესებს არა მთელი თვისებათა სტრუქტურა არამედ მისი რომელიმე კონკრეტული ქვეთვისება, მაშინ სამკუთხა ფრჩხილებში მივუთითებთ ასევე გზას, რომელიც მიუთითებს კონკრეტულ ქვეთვისებაზე. მაგალითად: <მორფემა_1 ქვეთვისება_1 ქვეთვისება_2>. ამგვარად ჩაწერილი მორფემათა თვისებები და ქვეთვისებები შეგვიძლია გამოვიყენოთ შეზღუდვებში ოპერანდების სახით. წესის მარცხენა მხარეში მყოფ სიმბოლოსაც შეესაბამება თავისი თვისებათა სტრუქტურა. სიტყვის გარჩევის საწყის ეტაპზე იგი ცარიელია. შეზღუდვების საშუალებით სიტყვის გარჩევის პროცესში მიღებული საჭირო ინფორმაცია , მაგალითად: რიცხვი, ფუძე, პირიანობა, დ ა.შ. შეგვიძლია გადავიტანოთ ამ თვისებათა სტრუქტურაში. ანალიზატორი შედეგად გამოიტანს სწორედ ამ შეგროვებულ ინფორმაციას სიტყვის შესახებ.

მორფოლოგიის ფაილში ასევე შეგვიძლია განვსაზღვროთ კონსტანტები და ცვლადები. ისინი ძირითადად შემოკლებებში გამოიყენება. კონსტანტის განსაზღვრა ხდება შემდეგი სახით:

კონსტანტა = [თვისება_1: მნიშვნელობა_1
თვისება_2: მნიშვნელობა_2
....]

ანალოგიურად განისაზღვრება ცვლადიც, მხოლოდ მას წინ დოლარის ნიშანი ’$’ ეწერება:

```
$ცვლადი = [თვისება_1: მნიშვნელობა_1
           თვისება_2: მნიშვნელობა_2
          ....]
```

ჯერ ვწერთ კონსტანტის (ცვლადის) სახელს, შემდეგ ტოლობის ნიშანს რომელსაც მოსდევს თვისებათა სტრუქტურა. ამის შემდეგ კონსტანტა (ცვლადი) მნიშვნელობად მიიღებს ამ თვისებათა სტრუქტურას. იმის მაგივრად, რომ მრავალ ადგილას დავწეროთ ერთიდაიგივე თვისებათა სტრუქტურა შეგვიძლია პირდაპირ კონსტანტა (ცვლადი) გამოვიყენოთ. მაგალითად:

```
ბრ_სახ = [ბრუნვა: სახელობითი]

@თანდებული =
        {
        ”დან” [(ბრ_სახ)]
        ...
        }
```

განსხვავება კონსტანტას და ცვალდს შორის არის ის, რომ ცვლადის გამოყენება შესაძლებელია როგორც წაკითხვისათვის ასევე ჩაწერისათვის, ხოლო კონსტანტას გამოყენება მისი განსაზღვრის შემდეგ შესაძლებელია მხოლოდ წაკითხვის მიზნით. ცვლადის ეს თავისებურება უმეტესად მორფოლოგიური ანალიზის ეტაპზე გამოიყენება.

ამ მაგალითში, თავდაპირველად განვსაზღვრეთ კონსტანტა ”ბრ_სახ”, რომელმაც მნიშვნელობად მიიღო თვისებათა სტრუქტურა [ბრუნვა: სახელობითი]. შემდეგ განვსაზღვრეთ მორფემა ”თანდებული” და მის ერთერთ შესაძლო მნიშვნელობას ”დან” თვისებად შევუსაბამეთ [ბრუნვა: სახელობითი]. თუმცა იმის მაგივრად, რომ პირდაპირ ჩაგვეწერა ეს თვისებათა სტრუქტურა გამოვიყენეთ უკვე განსაზღვრული

კონსტანტა. იგი მოვათავსეთ თვისებათა კონსტრუქტორის ინიციალიზაციის ნაწილში (მრგვალ ფრჩხილებში) და ამის შედეგად მთელი მისი შიგთავსი გადავიდა შესაბამის თვისებათა სტრუქტურაში. პრაქტიკაში ძალზედ ხშირია შემთხვევები, როდესაც მრავალი თვისებისაგან შემდგარი ერთიდაიგივე თვისებათა სტრუქტურის დაწერა მრავალ სხვადასხვა ადგილას გვიხდება. ამ შემთხვევაში კონსტანტების გამოყენება ძალზედ ეფექტურია და მკვეთრად ამცირებს ჩანაწერის მოცულობას.

ცვლადების გამოყენება დასაშვებია ასევე შეზღუდვებში ოპერანდებად. მათი ჩაწერა ხდება სამკუთხა ფრჩხილებში (ისევე როგორც მორფემის შესაბამისი თვისებების შემთხვევაში), ისინი მორფემის თვისებისაგან დოლარის ნიშნით განსხვავდებიან ასე, რომ მათი ერთმანეთში არევა შეუძლებელია. როგორც აღვნიშნეთ ცვლადს სახელის წინ ვუთითებთ დოლარის ნიშანს ”$”. მაგალითად, <$ცვლადი>. თუ გვსურს გამოვიყენოთ ცვლადის რომელიმე კონკრეტული ქვეთვისება, მაშინ მისი გზა უნდა ჩავწეროთ ამავე სამკუთხა ფრჩხილებში. მაგალითად, <$ცვლადი ქვეთვისება_1 ქვეთვისება_2>.

ფორმალიზმში აგრეთვე შემოტანილი გვაქვს ფაილის ჩართვის დირექტივა. მისი გამოყენებით შეგვიძლია მორფოლიგიის ფაილი დავანაწილოთ რამოდენიმე სხვადასხვა ფაილად. ეს საშუალებას გვაძლევს უფრო ნათელი და ლოგიკურად დალაგებული გავხადოთ მორფოლოგიის ფაილის სტრუქტურა. ფაილის ჩართვის დირექტივას შემდეგი სახე აქვს:

<”filename”>

ჩასასმელი ფაილის სახელი სამკუთხა ფრჩხილებში და ბრჭყალებშია ჩაწერილი. როდესაც გამრჩევი აღმოაჩენს მსგავს ჩანაწერს იგი მოძებნის შესაბამის ფაილს, დაამუშავებს მას, ხოლო შემდგომ გააგრძელებს საწყისი ფალის დამუშავებას. თუ ფაილი სრული სახითაა მოცემული გამრჩევი პირდაპირ შეეცდება მის წაკითხვას, წინააღმდეგ შემთხვევაში იგი შეეცდება მოძებნოს ფაილი მიმდინარე კატალოგში. თუ ფაილი არ არსებობს ან ვერ მოხერხდა მისი წაკითხვა, გამრჩევი გამოიტანს შესაბამის შეტყობინებას.

ფორმალიზმში საშუალება გვაქვს აგრეთვე გამოვიყენოთ კომენტარები. გვაქვს კომენტარების აღნიშვნის ორი ვარიანტი.

1. კომენტარი იწყება ’#’ სიმბოლოთი და ვრცელდება სტრიქონის ბოლომდე. ნებისმიერ ადგილას სადაც გამრჩევი აღმოაჩენს ’#’ სიმბოლოს, იგი

2. კომენტარები თავსდება /* და */ მარკერებს შორის (ისევე როგორც დაპროგრამების ენა C-ში). მთელი ტექსტი, რომელიც ამ მარკერებს შორისაა მოთავსებული უგულვებელყოფილი იქნება გამრჩევის მიერ. შესაბამისად შეგვიძლია საკმაოდ ვრცელი, მრავალსტრიქონიანი კომენტარები. აღვნიშნოთ, რომ პროგრამა საშუალებას გვაძლევს გვქონდეს ჩადგმული (nested) კომენტარები.

§3. პროგრამის აღწერა

ჩვენს მიერ შემუშავებული ფორმალიზმის და მორფოლოგიური ანალიზის საფუძველზე მოხდა პროგრამული უზრუნველყოფის რეალიზაცია. შედეგად მივიღეთ კომპიუტერული პროგრამა, რომელის მეშვეობითაც შესაძლებელია მორფოლოგიური ანალიზის ჩატარება. ამისათვის საჭიროა, რომ პროგრამას გადავცეთ მორფოლოგიის ფაილი, რომელშიც ჩვენი ფორმალიზმის გამოყენებით ჩაწერილი იქნება ბუნებრივი ენის მორფოლოგიის წესები.

პროგრამა მუშაობს დიალოგურ რეჟიმში. გაშვებისას მას გადავცემთ ერთ პარამეტრს, მორფოლოგიის ფაილი სახელს. პროგრამა ჩატვირთავს ამ ფაილს, გადაიყვანს ენის მორფოლოგიურ წესებს თავის შინაგან წარმოდგენაში. შემდეგ დიალოგურ რეჟიმში პროგრამას მივაწვდით იმ სიტყვებს, რომელთა მორფოლოგიური ანალიზიც გვსურს. თუკი ანალიზი მოხერხდა, პროგრამა გამოგვიტანს ანალიზის შედეგს (ერთ ან რამოდენიმე ამონახსნს). წინააღმდეგ შემთხვევაში მივიღებთ შეტყობინებას რომ ვერ მოხერხდა საწყისი სიტყვის მორფემებად დაშლა. პროგრამა მუშაობს ტექსტურ რეჟიმში რაც მაქსიმალურად ამარტივებს მასთან მომხმარებლის ურთიერთობას.

პროგრამის გამშვები ფაილია ‘mparse’ (ან ‘mparse.exe’ Windows-ში). მისი გაშვების შემდეგ ვიღებთ პროგრამასთან მუშაობის მიწვევას:

Morphological Analyzer (Version 1.0)

Enter words (use '\q' to quit)

#

სიმბოლო ’#’ ნიშნავს რომ პროგრამა დიალოგურ რეჟიმში იმყოფება და შეგვიძლია გასარჩევი სიტყვის აკრეფვა და მორფოლოგიურ ანალიზზე გადაცემა. პროგრამიდან გამოსვლისათვის უნდა გამოვიყენოთ ’\q’ ბრძანება, ან Ctrl + C კლავიშების კომბინაცია.

მივაწოდოთ ანალიზატორს რამოდენიმე სიტყვა გასარჩევად. (შევნიშნოთ, რომ ამ მაგალითების გაშვებისას პროგრამაში ჩატვირთულია ქართული ენის მორფოლოგიის ფრაგმენტი, რომელიც დაიწერა ნაშრომის დამუშავების პროცესში)

წვიმს

Solutions:

1. ზმნა

[დრო: 1

პირი_სუბ: 3

რიცხვი: მხ]

პროგრამამ წარმატებით გაარჩია სიტყვა ”წვიმს” და გამოგვიტანა მორფოლოგიური ანალიზის შედეგი. კერძოდ ”წვიმს” დაიშალა მორფემებად როგორც ზმნა, და დადგინდა მისი გრამატიკული კატეგორიები: დრო (აწმყო, ამ შემთხვევაში გრამატიკაში აღნიშნული გვაქვს რიცხობრივად 1-ით), სუბიექტის პირი (მესამე) და რიცხვი (მხოლობითი).

სახლებიდანაც

Solutions:

1. არს_სახ

[თანდ: დან

ბრუნვა: სახ

ბრუნვა_ლექს: ი

ფუძე: სახლ

ნაწ: აც

რიცხვი: მრ

რიცხვი_ლექს: ებ]

პროგრამამ ასევე წარმატებით გაარჩია სიტყვა ”სახლებიდანაც”. კერძოდ, დადგინდა, რომ იგი არის არსებითი სახელი, დგას სახელობით ბრუნვაში, არის მრავლობით რიცხვში, სიტყვის ფუძე არის ”სახლ”, დაირთავს თანდებულს ”დან”, შეიცავს ნაწილაკს ”აც”, რიცხვის ნიშანია ”ებ”, ბრუნვის ნიშანია ”ი”. და მთელი ეს ინფორმაცია დადგინდა მხოლოდ ამ სიტყვის ფუძის (”სახლ”) მეშვეობით, რომელიც ლექსიკონში იყო შეტანილი. ასევე დადგინდებოდა ნებისმიერი ამ ფუძიდან წარმოებული სხვა სიტყვაფორმის გრამატიკული კატეგორიები. მაგალითად: ”სახლმა”, ”სახლებიდან”, ”სახლებადაც”, ”სახლთათვის”, ”სახლებისთვისაც”, ”სახლნიღა”... და ა.შ. ასეთი სიტყვაფორმები ძალზედ ბევრია, და სწორედ მორფოლოგიური ანალიზატორი გვაძლევს იმის საშუალებას რომ არ მოხდეს ყველა ამ სიტყვაფორმის ლექსიკონში შეტანა, რაც დაკავშირებულია ლექსიკონის მოცულობის მკვეთრ ზრდასთან და მის შედგენაზე დახარჯულ დროსთან. ლექსიკონში გვაქვს მხოლოდ ერთი ჩანაწერი: "სახლ" [(ბრტ1 რიცტ1 წოდტ1)] სადაც მითითებულია ფუძე ”სახლ”-ის ელემენტარული გრამატიკული თვისებები (ბრუნვის ტიპი, რიცხვის ტიპი, წოდებითის წარმოების ტიპი). ყველა სხვა სიტყვაფორმის გარჩევა ხდება ამ ინფორმაციაზე დაყრდნობით და გრამატიკის ფაილში არსებული მორფოლოგიური წესებით. ამასთანავე, მორფოლოგიური ანალიზატორი გვაძლევს საშუალებას მაქსიმალურად მოქნილად ვიმოქმედოთ. თანდათან დავხვეწოთ მორფოლოგიის ფაილი. დროთა განმავლობაში ენაში მიმდინარე ცვლილებები შეიძლება მარტივად აისახოს მორფოლოგიის ფაილში წესების კორექტირებით. შესაძლებელია სხვადასხვა მიდგომების გამოყენებით შეიქმნას სხვადასხვა მორფოლოგიური ფაილები, გამოიცადოს სხვადასხვა ლინგვისტური თეორიები. პროგრამა მოქმედების სრულ თავისუფლებას გვაძლევს. იგი გახლავთ მეტად მოხერხებული ინსტრუმენტი ლინგვისტისათვის.

სტრუქტურულად პროგრამა შემდეგი ნაწილებისაგან შედგება:

1. მორფოლოგიის ფაილის დამამუშავებელი მოდული
2. პროგრამის ბირთვი (გარჩევის ალგორითმი)
3. მომხმარებლის ინტერფეისი.

მორფოლოგიის ფაილის დამამუშავებელი მოდული ახორციელებს ჩვენს მიერ შემუშავებული ფორმალიზმის მეშვეობით ჩაწერილი მორფოლოგიის ფაილის გადაყვანას პროგრამის შიდა წარმოდგენაში. ამ მოდულის საშუალებით ხდება ფაილში აღწერილი ყველა ელემენტის (წესის, მორეფმათა კლასის აღწერის, ცვლადის, და ა.შ.) გადაყვანა პროგრამისთვის სპეციფიურ მონაცემთა სტრუქტურებში, რის შემდეგაც გამრჩევის ბირთვი (ძირითადი ალგორითმი) მზად იქნება მომხმარებლის მიერ შეტანილი სიტყვების მორფოლოგიური ანალიზისათვის. თავად შემუშავებული ფორმალიზმი წარმოადგენს LL(1) კლასის გრამატიკას, რომლის გარჩევაც ამ მოდულის მეშვეობით ხდება. LL(1) გამრჩევის გენერირებისათვის გამოყენებული იქნა Coco/R პროგრამული ინსტრუმენტარი.

პროგრამის ბირთვი, ანუ გარჩევის ალგორითმის მოდული, წარმოადგენს ჩვენს მიერ ზემოთ ჩამოყალიბებული ალგორითმის (ძიება უკან დაბრუნებით და შეზღუდვების შემოწმებით) პროგრამულ რეალიზაციას. ალგორითმი იყენებს იმ ინფორმაციას, რომელიც წაიკითხა მორფოლოგიის ფაილის დამმუშავებელმა მოდულმა.

მომხმარებლის ინტერფეისი უზრუნველყოფს მომხმარებლის ურთიერთობას მორფოლოგიურ ანალიზატორთან. მისი მეშვეობით მომხმარებელს შეაქვს გასარჩევად სიტყვები და ღებულობს შედეგს. გასარჩევად გადაცემულ სიტყვას მომხმარებლის ინტერფეისი მიაწვდის გამრჩევის ბირთვს, რომელიც ჩაატარებს მორფოლოგიურ ანალიზს და ისევ მომხმარებლის ინტერფეისის მეშვეობით დააბრუნებს გარჩევის შედეგს. მომხმარებლის ინტერფეისი რეალიზებულია ტექსტურ რეჟიმში. ძირთადი მიზეზი, რის გამოც არჩევანი გაკეთდა ტექსტურ რეჟიმზე, გახლავთ მისი სიმარტივე და უნივერსალურობა. იგი გვაძლევს საშუალებას მოვახდინოთ პროგრამის გადატანა პრაქტიკულად ნებისმიერ პლატფორმაზე,

ვინაიდან C++ -ის რეალიზაციათა აბსოლიტურ უმრავლესობას გააჩნია ტექსტურ რეჟიმთან მუშაობის სტანდარტული ბიბლიოთეკები.

მორფოლოგიური ანალიზატორის პროგრამა დაწერილია დაპროგრამების ენა C++ -ის ANSI სტანდარტზე, ობიექტზე ორიენტირებული მეთოდოლოგიის გამოყენებით. პროგრამა იყენებს STL სტანდარტულ ბიბლიოთეკას. პროგრამა კომპილირებულია Windows და Linux ოპერაციული სისტემებისათვის. შესაძლოა მისი გადატანა ნებისმიერ სხვა პლატფორმაზე, სადაც არის C++ -ის თანამედროვე კომპილატორი.

§4. პრეპროცესორი

პრაქტიკული ექსპერიმენტებიდან გამომდინარე საჭირო გახდა მორფოლოგიური ანალიზატორის პროგრამაში შემოგვეღო მაკრო-ჩასმების მექანიზმი. ისინი საშუალებას გვაძლევენ მრავალჯერადად გამეორებადი ტექსტები მოკლე აღნიშვნების მეშვეობით ჩავწეროთ. მორფოლოგიის ფაილის დამუშავება ორეტაპიანი გახდა. პირველ ეტაპზე აქტიურდება პრეპროცესორი, რომელიც ამუშავებს მაკროჩასმებს, მეორე ეტაპზე მორფოლოგიის ფაილის გამრჩევი მოდული იწყებს მუშაობას და კითხულობს მისთვის გამზადებულ ფაილს. მაკრო-ჩასმები შემდეგი სახით ჩაიწერებიან:

```
$$სახელი
ტექსტი....
...
$$.
```

მიყოლებით მდგარი ორი დოლარის ნიშნის შემდეგ მოდის მაკრო-ჩასმის სახელი. შემდეგი ხაზიდან იწყება ტექსტი, რომელიც ამ მაკრო-ჩასმით აღინიშნება. მაკრო-ჩასმა მთავრდება მიყოლებით მდგარი ორი დოლარის ნიშნით და წერტილით. ამგვარი ჩანაწერი ნიშნავს, რომ მთელი ტექსტი, რომელიც მაკრო-ჩასმაშია ჩაწერილი

ფაილის მომდევნო ნაწილში შეგვიძლია მოკლედ ჩავწეროთ მაკრო-ჩასმის გამოძახებით. მაკრო-ჩასმის გამოძახება შემდგნაირად ხდება:

${სახელი}$

როდესაც პრეპროცესორი აღმოაჩენს ამგვარ ჩანაწერს იგი მონახავს მის შესაბამის ტექსტს და ჩაანაცვლებს მას ამ ჩანაწერის ადგილას. ანუ მოხდება ტექსტური ჩასმა. განვიხილოთ მარტივი მაგალითი.
მოცემულია საწყისი ფაილი:

```
A A A A A A A
B B B B B B B
$$MyMacro
Macro text goes here....
one more line.
$$.
CCCCCCCCCCC
${MyMacro}$
DDDDDDDDDDD
${MyMacro}$
EEEEEEEEEEEE
```

მას შემდეგ, რაც პრეპროცესორი დაამუშავებს ამ ფაილს, იგი მიიღებს შემდეგ სახეს:

```
A A A A A A A
B B B B B B B
CCCCCCCCCCC
Macro text goes here....
one more line.
DDDDDDDDDDD
Macro text goes here....
```

one more line.

EEEEEEEEEEEEE

როგორც ვხედავთ მარტივად მოხერხდა ერთიდაიგივე ტექსტის ორ სხვადასხვა ადგილას ჩასმა. აქ შევნიშნოთ, რომ პრეპროცესორი საწყისი ფაილის შიგთავსს არ ცვლის, იგი შექმნის დროებით ფაილს და მასში ჩაწერს დამუშავებულ ფაილს. სწორედ ეს დროებითი ფაილი გადაეცემა მორფოლოგიურ ანალიზატორს დასამუშავებლად.

ჩვენს მიერ განხილული მაკრო-ჩასმა გახლავთ მაკრო-ჩასმის უმარტივესი შემთხვევა. რადგანაც ჩვენს შემთხვევაში მოხდა უბრალოდ სტატიკური ტექსტის დუბლირება. ხშირად საჭირო ხდება არა უბრალოდ სტატიკური ტექსტის ჩასმა, არამედ მისი გარკვეული ვარიაციების მიღება სხვადასხვა პარამეტრების მეშვეობით. ჩვენ პრეპროცესორში საშუალება გვაქვს გამოვიყენოთ პარამეტრიანი მაკრო-ჩასმები. ისინი შემდეგნაირად აღიწერებიან:

$$მაკრო

ტექსტი... პირველი პარამეტრი: $0

მეორე პარამეტრი: $1

კვლავ ტექსტი...

$$.

როგორც ვხედავთ ამჯერად მაკრო-ჩასმის აღწერაში ტექსტის გარდა გვაქვს $0 და $1 აღნიშვნები. ისინი წარმოადგენენ პარამეტრულ ცვლადებს, ანუ მათ ადგილას ჩაენაცვლება ის ტექსტი რომელსაც მაკრო-ჩასმას გადავცემთ გამოძახებისას. პარამეტრი აღინიშნება დოლარის ნიშნით ’$’ და რიგითი ნომრით. შევნიშნოთ, რომ ნუმერაცია იწყება ნულიდან, ანუ $0 წარმოადგენს მაკრო-ჩასმისათვის გადაცემულ პირველ პარამეტრს, $1 - მეორეს და ა.შ. პარამეტრიანი მაკრო-ჩასმის გამოძახება შემდეგნაირად ხდება:

${მაკრო: AAA, BBB}$

მაკრო-ჩასმის სახელის შემდეგ იწერება ორწერტილი და ხდება პარამეტრების ჩამოთვლა. პარამეტრები ერთმანეთისაგან მძიმით გამოიყოფიან. ამ მაკრო-ჩასმის დამუშავების შემდეგ საწყისი ტექსტი მიიღებს ასეთ სახეს:

ტექსტი... პირველი პარამეტრი: AAA
მეორე პარამეტრი: BBB
კვლავ ტექსტი...

პარამეტრიანი მაკრო-ჩასმები მეტად მოსახერხებელია. მათი მეშვეობით შესაძლებელია მრავალჯერ გამეორებადი ტექსტის შეკვეცა, რაც ძალზედ ამოკლებს მორფოლოგიის ფაილს და მის სტრუქტურას უფრო ნათელს და ადვილად გასაგებს ხდის.

აქვე ავღნიშნოთ, რომ შესაძლებელია მაკრო-ჩასმების გამოყენება თავად მაკრო-ჩასმების შიგნით. ამ მხრივ პრეპროცესორი სრულ თავისუფლებას გვაძლევს. თუმცა უნდა გვახსოვდეს ერთმანეთში ჩადგმულ მაკრო-ჩასმებთან დაკავშირებული საშიშროება, რაც გამოიხატება იმაში, რომ შეიძლება უსასრულო რეკურსიული ჯაჭვი წარმოიქმნას. ამის თავიდან ასაცილებლად პრეპროცესორი თვალყურს ადევნებს მაკრო-ჩასმების დამუშავებისას მათი ჩალაგების სიღრმეს და როცა ეს სიღმე გარკვეულ ზღვარს გადააჭარბებს პრეპროცესორი წყვეტს მუშაობას. ამ შემთხვევაში საჭიროა მოვნახოთ ის ადგილი, სადაც წარმოიქმნა გამოძახებების რეკურსიული ჯაჭვი და გამოვასწოროთ ეს შეცდომა.

ასევე არა აქვს მნიშვნელობა თუ ფაილის რომელ ნაწილშია აღწერილი მაკრო-ჩასმა, მისი გამოყენება შეიძლება ნებისმიერ ადგილას. ეს შესაძლებელი გახდა იმის გამო, რომ პრეპროცესორი თავდაპირველად კითხულობს ყველა მაკრო-ჩასმის აღწერას და ადგენს მათ სიას. ხოლო შემდეგ იგი ფაილს თავიდან გაივლის და მასში გამოძახებულ მაკრო-ჩასმებს დაამუშავებს.

ვინაიდან დოლარის ნიშანს ’$’ მაკრო-ჩასმების აღწერის შიგნით სპეციალური დატვირთვა გააჩნია, მისი გამოყენება ამ ადგილებში პირდაპირ არ შეიძლება რადგანაც შეიძლება პრეპროცესორმა მისი არასწორი ინტერპრეტაცია მოახდინოს. მაგალითად, აღიქვას იგი როგორც პარამეტრი. ამის თავიდან ასარიდებლად, იმ ადგილას სადაც დოლარის ნიშნის გამოყენებაა საჭირო, ერთი დოლარის ნიშნის

მაგივრად ვწერთ გვერდიგვერდ მდგომ ორ დოლარის ნიშანს ’$$’. პრეპროცესორი ფაილის დამუშავების პროცესში მათ გარდაქმნის ერთ დოლარის ნიშნად. ამ მეთოდით მთლიანად აღმოიფხვრება არასწორი ინტერპრეტაციის პრობლემა.

§5. მაგალითები

განვიხილოთ რამოდენიმე მაგალითი. მათი მეშვეობით ჩვენ ნათლად დავინახავთ თუ როგორ მუშაობს მორფოლოგიური ანალიზატორი პრაქტიკულ ამოცანებზე. შევისწავლით მისი გამოყენების თავისებურებებს. თავიდან განვიხილავთ შედარებით მარტივ მაგალითს.

მაგალითი 1.
მოცემული გვაქვს შემდეგი მორფემათა კლასები:

M1 = { “a”, “aab”, “” }
M2 = { “bb”, “ab”, “b”, “” }
M3 = { “aa“, “baa” }

განვსაზღვროთ სიტყვა W როგორც ამ სამი მორფემის მიმდევრობა.

W -> M1 M2 M3 ;

შევადგინოთ მორფოლოგიის ფაილი გამრჩევისათვის და მოვახდინოთ შემდეგი სიტყვების ანალიზი:
aabbbbaa
aabbbaab
aabbbaa

ჩვენი ფორმალიზმის გამოყენებით შექმნილ მორფოლოგიის ფაილს შემდეგი სახე ექნება:

ფაილი sample1.rul

@M1 = { “a” [], “aab” [], “” []}
@M2 = { “bb” [], “ab” [], “b” [], “” [] }
@M3 = { “aa“ [], “baa” []}

W -> M1 M2 M3 ;
.

ფორმალიზმის თანახმად მორფემების აღწერა იწყება ’@’ ნიშნით. თითოეულ მორფემას აღწერის დროს შეესაბამება გარკვეული თვისება (ჩვენს შემთხვევაში ცარიელი თვისება: []). ფაილი ბოლოვდება წერტილით ”.”, ეს ფორმალიზმის მიერ მოითხოვება. ფაილის ბოლოში წერტილის არარსებობის შემთხვევაში პროგრამა გამოიტანს შესაბამის შეტყობინებას შეცდომის შესახებ.

გავუშვათ მორფოლოგიური გამრჩევის პროგრამა და გადავცეთ მას პარამეტრად ჩვენს მიერ შექმნილი ფაილის სახელი. Windows-ში ბრძანებათა სტრიქონში პროგრამის გამოძახებას შემდეგი სახე ექნება:

C:\m-parse-bin> mparse.exe sample1.rul

გაშვების შემდეგ გადავცეთ მას გასარჩევად სიტყვა ’aabbbbaa’. შედეგად მივიღებთ:

aabbbbaa
Solutions:
1. W
(None)

რაც ნიშნავს რომ გარჩევამ წარმატებით ჩაიარა და მიღებულია გარჩევის ერთი ვარიანტი W წესის მიხედვით. ქვედა სტრიქონში (None) მიუთითებს რომ გარჩეული სიტყვის შესაბამისი თვისებათა სტრუქტურა ცარიელია, ეს ბუნებრივიცაა ვინაიდან

ჩვენ წესში არ მიგვითითებია ამ ინფორმაციის ჩაწერა. სიტყვა დაიშალა “aab”-“bb”-“baa” ვარიანტად. თუმცა გამრჩევის მიერ ცხადად ამის მითითება არ ხდება. ქვემოთ ჩვენ ვნახავთ, თუ როგორ შეიძლება გამოვიტანოთ ინფორმაცია სიტყვის შემადგენელი მორფემების შესახებ.

ახლა, იგივე გრამატიკის მეშვეობით გავარჩიოთ სიტყვა aabbbaab. გამრჩევი შედეგად ეკრანზე შემდეგ შეტყობინებას გამოიტანს:

```
# aabbbaab
Unable to analyze word
```

რაც ნიშნავს, რომ სიტყვის გარჩევა ვერ მოხერხდა. ჩვენს შემთხვევაში სიტყვა ვერ დაიშალა მორფემებად.

ბოლოს გადავცეთ გამრჩევს სიტყვა aabbbaa. შედეგად მივიღებთ შემდეგ შეტყობინებას:

```
# aabbbaa
Solutions:
1. W
(None)
2. W
(None)
```

იგი გვეუბნება რომ მოხერხდა სიტყვის წარმატებით გარჩევა და ნაპოვნი იქნა გარჩევის ორი ვარიანტი. ორივე ვარიანტი გაირჩა W წესის მიხედვით. ორივე ვარიანტის შესაბამისი თვისებათა სტრუქტურები ცარიელია.

ახლა მოვახდინოთ მორფოლოგიის ფაილის ისეთნაირად შეცვლა, რომ პროგრამამ შედეგად გამოგვიტანოს ინფორმაცია სიტყვის შემადგენელ მორფემებზე. ამისათვის, გამოვიყენოთ თვისებათა შეზღუდვების მექანიზმი. თუმცა ჩვენს შემთხვევაში ამ მექანიზმს ვიყენებთ არა გრამატიკული მიზნით, ანუ ამონახსნთა სიმრავლის შეზღუდვის მიზნით, არამედ ინფორმაციის გადასატანად გარჩეული სიტყვის შესაბამის თვისებათა სტრუქტურაში. ამ თვისებათა სტრუქტურაში ჩვენ

გადავიტანთ ინფორმაციას სიტყვის შემადგენელი მორფემების შესახებ. მორფოლოგიის ფაილს ამჯერად შემდეგი სახე ექნება:

ფაილი sample2.rul

```
$LEX = “lex”

@M1 = { “a” [], “aab” [], “” []}
@M2 = { “bb” [], “ab” [], “b” [], “” [] }
@M3 = { “aa“ [], “baa” []}

W -> M1 M2 M3
        { <W m1> := <M1 lex> &
          <W m2>  := <M2 lex> &
          <W m3> := <M3 lex> } ;
.
```

ფაილის დასაწყისში ხდება სპეციალური ცვლადის $LEX -ის განსაზღვრა. მას მნიშვნელობად ვანიჭებთ “lex” -ს. ამის შედეგად სისტემა სიტყვის გარჩევის პროცესში ყველა მორფემას ავტომატურად შეუსაბამებს lex თვისებას, რომლის მნიშვნელობაც იქნება მორფემის ლექსიკური მნიშვნელობა, ანუ ის სტრიქონი, რომელსაც იგი წარმოადგენს. მაგალითად, ამ განსაზღვრის შემდეგ მორფემა ”ab” -ს შესაბამისი თვისებათა სტრუქტურა იქნება [lex: ab]. გავაგრძელოთ მორფოლოგიის ფაილში შემოტანილი ახალი ჩანაწერების განხილვა. მორფემების აღწერა იგივე დარჩა. შეცვალა მხოლოდ წესის აღწერა. მას ბოლოში დაემატა შეზღუდვები:

```
{ <W m1> := <M1 lex> &
  <W m2>  := <M2 lex> &
  <W m3> := <M3 lex> }
```

შეზღუდვები როგორც ვთქვით წარმოადგენენ ლოგიკურ გამოსახულებას. ამ შემთხვევაში გვაქვს თვისებათა სტრუქტურაზე განსაზღვრული სამი მინიჭების

ოპერაცია, რომლებიც ერთმანეთთან დაკავშირებულნი არიან ლოგიკური ”და” ოპერაციის მეშვეობით. გამრჩევი ამ ჩანაწერის ინტერპრეტაციას შემდეგი სახით ახდენს. ჯერ გამოითვლება პირველი ოპერაცია <W m1> := <M1 lex>, შემდეგ მოდის ლოგიკური ”და”-ს ნიშანი, ვინაიდან თვისებათა სტრუქტურებზე მინიჭების ოპერაცია ყოველთვის აბრუნებს ლოგიკურად დადებით მნიშვნელობას გამრჩევი გააგრძელებს გამოსახულების გამოთვლას და გადავა მეორე მინიჭებაზე. ანალოგიურად შესრულდება ეს მინიჭების ოპერაციაც და მისი მომდევნო მინიჭების ოპერაციაც. ანუ ყველა ვარიანტში მოხდება სამივე ოპერაციის ჩატარება. ამის შედეგად კი სიტყვის შესაბამის თვისებათა სტრუქტურაში გადაიწერება ინფორმაცია. კერძოდ, მისი თვისება m1 გახდება M1 -დან ამორჩეული მორფემის ლექსიკური მნიშვნელობის ტოლი. ანალოგიურად M2 და M3 -დან ამორჩეული მორფემების ლექსიკური მნიშვნელობებით შეივსებიან m2 და m3 თვისებები. ახლა გამრჩევს კვლავ გადავცეთ სიტყვა aabbbaa გასარჩევად. ეკრანზე მივიღებთ შემდეგ რეზულტატს:

```
# aabbbaa
Solutions:
1. W
[m1: aab
 m2: bb
 m3: aa]
2. W
[m1: aab
 m2: b
 m3: baa]
```

გამოტანილ შედეგში ჩვენ ვხედავთ სიტყვის დაშლის ორივე ვარიანტის შესაბამის თვისებათა სტრუქტურებს. მათში ჩანს ინფორმაცია მორფემების შესახებ. ამჯერად ნათლად ვხედავთ რომ პირველი ვარიანტში სიტყვა მორფემებად დაიშალა როგორც “aab”–“bb”–“aa”, ხოლო მეორე ვარიანტში იგი დაიშალა როგორც “aab”-“b”-“baa”.

მაგალითი 2.

მოცემული გვაქვს ჰიპოთეტური ენა, რომელის ანბანიც შედგება 4 სიმბოლოსგან: D, B, A და O. აქედან D და B არიან თანხმოვნები, A და O კი ხმოვნები. ენაში სიტყვათა წარმოების მკაცრი წესია. გრამატიკულად სწორ სიტყვად ითვლება მხოლოდ ის სიტყვა, რომელიც შემდეგი პირობებიდან ყველას აკმაყოფილებს:

1. სიტყვა მთავრდება ხმოვანზე
2. არ შეიძლება ზედიზედ ორი თანხმოვანის გამოყენება
3. სიტყვის სიგძე უნდა იყოს 4-ის ტოლი
4. არ შეიძლება ზედიზედ ორი ერთიდაიგივე ხმოვნის გამოყენება

შევადგინოთ მორფოლოგიის ფაილი ამ ენისათვის და გავარჩიოთ შემდეგი სიტყვები:

AABO

DAOB

DOBA

BOOA

BABA

ამ ჰიპოთეტური ენის გასარჩევ მორფოლოგიის ფაილს შემდეგი სახე ექნება:

ფაილი sample3.rul

```
$LEX = “lex”

@M =
    {
    “A” [ cons: + ],
    “O” [ cons: + ],
    “B” [ cons: - ],
    “D” [ cons: - ]
```

```
        }

W -> M.1 M.2
        {
        ~( (<M.1 cons> = “-“ & <M.2 cons> = “-“) |
            (<M.1 cons> = “+” & <M.1 lex> = <M.2 lex>)
          )
        }
        M.3
        {
        ~( (<M.2 cons> = “-“ & <M.3 cons> = “-“) |
            (<M.2 cons> = “+” & <M.2 lex> = <M.3 lex>)
          )
        }
        M.4
        {
        <M.4 cons> = “+” &
        ~ <M.3 lex> = <M.4 lex> &
        <W m1> := <M.1 lex> &
        <W m2> := <M.2 lex> &
        <W m3> := <M.3 lex> &
        <W m4> := <M.4 lex>
        } ;
```

.

ფაილის დასაწყისში ხდება ლექსიკური ველის სახელის განსაზღვრა. მას მოსდევს M მორფემის აღწერა. მაში მორფემების სახით განსაზღვრულია ენის ანბანში მყოფი ოთხივე სიმბოლო. წინა მაგალითებისგან განსხვავებით ამ მაგალითში მორფემების შესაბამისი თვისება სტრუქტურა ცარიელი არ გვაქვს. ყოველ მორფემას (ჩვენს შემთხვევაში სიმბოლოს) შეესაბამება თვისებათა სტრუქტურა, რომელისაც გააჩნია ერთი თვისება “cons” (ხმოვანი), ხოლო მისი მნიშვნელობება არის ”+” ან ”-”, რაც

ნიშნავს რომ მოცემული სიმბოლო შესაბამისად არის ხმოვანი ან თანხმოვანი. მოცემულობის თანახმად 'A' და 'O' სიმბოლოების შემთხვევაში თვისება "cons" -ს მნიშვნელობად აქვს "+", ხოლო 'B' და 'D' სიმბოლოების შემთხვევაში "-".

M მორფემის შემდეგ მოდის წესის განსაზღვრა, მისი მიხედვით სიტყვა განისაზღვრება როგორც M.1 M.2 M.3 M.4 მორფემათა მიმდევრობა. აქ შევნიშნოთ რომ ოთხივე ადგილას გამოყენებულია ერთი მორფემა M, ისინი ერთმანეთისაგან მხოლოდ ინდექსებით განსხვავდებიან. მორფემების ინდექსაცია გამოიყენება მაშინ, როდესაც წესში ერთიდაიგივე მორფემა რამოდენიმეჯერ მეორდება. შეზღუდვებში ამ მორფემების ერთიმეორესგან გარჩევა ინდექსების საშუალებით ხდება. გაჩუმებით მორფემის ინდექსი 1 -ის ტოლია. ანუ M და M.1 ერთიდაიგივე მორფემას აღნიშნავს. ინდექსის მისათითებლად მორფემის შემდეგ იწერება წერტილი და მას მოსდევს მთელი, დადებითი რიცხვი, რომელიც ჩვეულებრივ ტოლია გამეორებადი მორფემის რიგითი ნომრის (ერთიდაგიგვე მორფემებს შორის). ვინაიდან ამ მაგალითში ვიყენებთ შეზღუდვებს და გვჭირდება ერთიმეორესაგან ოთხივე მორფემის გარჩევა, ამიტომაც ყველა M მორფემას თავის ინდექსს ვუწერთ. როგორც ვხედავთ, წესში მორფემებს შორის ფიგურულ ფრჩხილებში ჩაწერილია შეზღუდვები. მათი მეშვეობით ხდება ენის გრამატიკული წესების ჩაწერა. განვიხილოთ შეზღუდვა, რომელიც მოსდევს M.1 და M.2 მორფემებს:

~((<M.1 cons> = "-" & <M.2 cons> = "-") |
(<M.1 cons> = "+" & <M.1 lex> = <M.2 lex>))

ამ ლოგიკური გამოსახულებით ხდება ენის გრამატიკის 2 და 4 წესებს შემოწმება M.1 და M.2 მორფემებისთვის. შეზღუდვა მოითხოვს, რომ შემდეგი ორი პირობიდან ამ მორფემებისთვის არცერთი არ სრულდებოდეს:

<M.1 cons> = "-" & <M.2 cons> = "-" (ზედიზედ ორი თანხმოვანი)
<M.1 cons> = "+" & <M.1 lex> = <M.2 lex> (ზედიზედ ორი ერთიდაიგივე ხმოვანი)

შეიძლებოდა ამ შეზღუდვის წესის ბოლოში ჩაწერა, მაგრამ ვინაიდან მასში მხოლოდ M.1 და M.2 მორფემები მონაწილეობენ, სრული უფლება გვაქვს ჩავწეროთ იგი

პირდაპირ მეორე მორფემის შემდეგ. რაც ცხადია გარჩევის სისწრაფეს აამაღლებს, რადგანაც გამრჩევი პირველი ორი მორფემის ამოცნობისთანავე შეამოწმებს შეზღუდვას და მისი დაუკმაყოფილებლობის შემთხვევაში აღარ გააგრძელებს ამ ვარიანტისთვის მესამე და მეოთხე მორფემების გარჩევას. ანალოგიური შეზღუდვები მეორდება 2 და 3 მორფემებისთვის. იგი მესამე მორფემის შემდეგ არის ჩაწერილი. ხოლო წესის ბოლოში ჩაწერილია შემდეგი შეზღუდვები:

<M.4 cons> = “+” &
~ <M.3 lex> = <M.4 lex> &
<W m1> := <M.1 lex> &
<W m2> := <M.2 lex> &
<W m3> := <M.3 lex> &
<W m4> := <M.4 lex>

რომელიც გვეუბნება რომ ბოლო მეოთხე მორფემა უნდა იყოს ხმოვანი (რაც მოთხოვილია ენის გრამატიკის 1 წესის მიერ, სიტყვა უნდა მთავრდებოდეს ხმოვანზე) და იგი უნდა განსხვავდებოდეს წინა მორფემისაგან (რაც უზრუნველყოფს 4 წესის დაცვას). შევნიშნოთ, რომ 2 წესი ავტომატურად იქნება დაცული, ვინაიდან მოთხოვნა რომ ბოლო მორფემა იყოს ხმოვანი გამორიცხავს სიტყვის ბოლოში ზედიზედ ორი თანხმოვნის არსებობას. ხოლო რაც შეეხება მესამე წესს, რომლის მიხედვითაც სიტყვის სიგრძე ტოლი უნდა იყოს 4 -ის, იგი თავისთავად დაცულია, რადგანაც W სიტყვის წარმოების წესი ჩვენ განვსაზღვრეთ როგორც ოთხი მორფემისგან შემდგარი მიმდევრობა. ამ შეზღუდვებში ბოლო ოთხი მინიჭების ოპერაცია გამოიყენება სიტყვის შესაბამის თვისებათა სტრუქტურაში ინფორმაციის გადასატანად. კერძოდ, აქ ხდება მორფემების შესახებ ინფორმაციის ჩაწერა, რათა ეკრანზე გამოტანილ შედეგში ნათლად დავინახოთ, თუ რა მორფემებად დაიშალა სიტყვა.

ამგვარად, გამზადებული მორფოლოგიის ფაილი გადავცეთ გამრჩევს. რის შემდეგაც იგი მზად იქნება გაარჩიოს ამოცანაში მოცემული სიტყვები.

გადავცეთ გამრჩევს სიტყვა AABO.

შედეგად მივიღებთ:

AABO

Unable to analyze word

სიტყვის გარჩევა ვერ მოხერხდა ვინაიდან დარღვეულია ენის გრამატიკის 4 წესი, ანუ ორი ერთიდაიგივე ხმოვანი ერთიმეორეს მიყოლებით დგას (AA).

გადავცეთ გამრჩევს სიტყვა DAOB.
შედეგად მივიღებთ:

DAOB

Unable to analyze word

როგორც ვხედავთ სიტყვა ხმოვანზე მთავრდება. აქედან ნათლად ჩანს, თუ რატომ ვერ მოხერხდა სიტყვის გარჩევა.

გადავცეთ გამრჩევს სიტყვა DOBA.
შედეგად მივიღებთ:

DOBA

Solutions:

1. W

[m1: D

m2: O

m3: B

m4: A]

სიტყვა სრულიად აკმაყოფილებს გრამატიკის წესების მოთხოვნებს და იგი წარმატებით იქნა გარჩეული.

ანალოგიურად, გამრჩევი წარმატებით გაარჩევს სიტყვა BABA -ს. ხოლო სიტყვა BOOA -ს გარჩევა წარუმატებელი იქნება, ვინაიდან იგი არღვევს გრამატიკის მეოთხე წესს, ზედიზედ ორ ერთიდაიგივე ხმოვანს შეიცავს.

§6. მორფოლოგიის ფაილის სტრუქტურა

მორფოლოგიური გამრჩევის უმთავრეს შემავალ ფაილს წარმოადგენს ენის მორფოლოგიის აღმწერი ფაილი. მასში იწერება იმ წესთა ერთობლიობა, რომლებიც ადგენენ ენას. ფაილის ჩატვრითვისას გამრჩევი ანალიზს უკეთებს მას და მასში არსებული წესები გადაყავს თავის შინაგან წარმოდგენაში. დაწვრილებით აღვწეროთ ის კონსტრუქციები, რომლებისაგანც შედგება მორფოლოგიის ფაილი. ამისათვის, გამოვიყენოთ ბეკუს-ნაურის გაფართოებული ფორმალიზმი. ქვემოთ მოცემულია მორფოლოგიის ფაილის სტრუქტურის სრული აღწერა კომენტარებთან ერთად.

საწყისი ტერმინალური სიმბოლო განისაზღვრება შემდეგი სახით:

<მორფოლოგიის_ფაილი> ::= { <წესის_განსაზღვრა> | <მუდმივას_განსაზღვრა> | <ცვლადის_განსაზღვრა> | <მორფემის_განსაზღვრა> | <ფაილის_ჩართვა> | <დიაგნოსტიკური_შეტყობინება> } ”.”

ამ წესის მიხედვით მორფოლოგიის ფაილი შედგება სხვადასხვა განსაზღვრებისა და დირექტივებისაგან. კერძოდ: წესის, მუდმივას, ცვლადის ან მორფემის განსაზღვრა. აგრეთვე ფაილის ჩართვის დირექტივა, რომლის მეშვეობითაც შესაძლებელია მორფოლოგიის ფაილი დაიყოს ცალცალკე ფაილებად. დიაგნოსტიკური შეტყობინების ოპერატორი, მისი მეშვეობით შესაძლებელია მორფოლოგიის ფაილის ჩატვირთვის ეტაპზე სხვადასხვა ინფორმაციის გამოტანა ეკრანზე. მისი გამოყენებით შესაძლებელია ადვილად დავადგინოთ მორფოლოგიის ფაილში სემანტიკური შეცდომები და გავასწოროთ ისინი.

<წესის_განსაზღვრა> ::= <იდენტიფიკატორი> ”->” <წესის_მარჯვენა_მხარე>

წესის განსაზღვრისას მის მარცხენა მხარეში მდგომი იდენტიფიკატორი წარმოადგენს წესის სახელს. წესის მარჯვენა მხარეს გვაქვს სიტყვის შემადგენელი მორფემების მიმდევრობა და მათზე დადებული შეზღუდვები.

<წესის_მარჯვენა_მხარე> ::= <წესის_შემადგენელი> { <წესის_შემადგენელი> }

<წესის_შემადგენელი> ::= <წესის_სიმბოლო> [”{” <წესის_შეზღუდვა> ”}”]

წესის შეზღუდვა წარმოადგენს ლოგიკურ გამოსახულებას. მისი სინტაქსური კონსტრუქცია ანალოგიურია ჩვეულებრივი მათემატიკური გამოსახულების სინტაქსური კონსტრუქციისა. განსხვავება მხოლოდ გამოყენებულ ოპერატორებშია. ჩვენს შემთხვევაში ვიყენებთ 3 ძირითად ლოგიკურ ოპერაციას: ”|” (ლოგიკური ჯამი), ”&” (ლოგიკური ნამრავლი) და ”~” (ლოგიკური უარყოფა). გამოსახულების სინტაქსური კონსტრუქცია აგებულია ისეთნაირად, რომ მოხდეს ოპერაციათა პრიორიტეტების დიფერენციაცია. ლოგიკური ნამრავლი უფრო პრიორიტეტულია ვიდრე ლოგიკური ჯამი, ხოლო ლოგიკურ უარყოფას ორივე მათგანზე უფრო მეტი პრიორიტეტი აქვს. გამოსახულებაში შეგვიძლია აგრეთვე ფრჩხილების გამოყენება პრიორიტეტების გადაფარვის მიზნით.

<წესის_შეზღუდვა> ::=
<წესის_შეზღუდვის_თერმი> ”|” <წესის_შეზღუდვის_თერმი>

<წესის_შეზღუდვის_თერმი> ::=
<წესის_შეზღუდვის_ფაქტორი> ”&” <წესის_შეზღუდვის_ფაქტორი>

<წესის_შეზღუდვის_ფაქტორი> ::= [”~”] (<ლოგიკური_კონსტანტა> | [”+” | ”-”]

<წესის_შეზღუდვის_ოპერაცია> | ”(” <წესის_შეზღუდვის_ფაქტორი> ”)”)

<ლოგიკური_კონსტანტა> ::= ”0” | ”1”

შევნიშნოთ, რომ შეზღუდვებში შეგვიძლია ლოგიკური კონსტანტების გამოყენება. ისინი აღინიშნება ”0” და ”1” -ის მეშვეობით. ”0” წარმოადგენს ლოგიკურად მცდარ მნიშვნელობას (False) ხოლო ”1” ლოგიკურად ჭეშმარიტ მნიშვნელობას (True).

<წესის_შეზღუდვის_ოპერაცია> ::= <წესის_შეზღუდვის_ოპერატორი> |
<წესის_შეზღუდვის_ფუნქცია>

როგორც მორფოლოგიური გამრჩევის ფორმალიზმის განხილვისას აღვნიშნეთ წესის შეზღუდვის ოპერაციის ჩაწერის ორგვარი სახე არსებობს - ოპერატორული და ფუნქციონალური. ფუნქციონალური ჩაწერის დროს ვუთითებთ ფუნქციის სახელს და ფრჩხილებში ვწერთ მისთვის გადასაცემ არგუმენტებს. სტანდარტული ოპერაციების გამოყენებისას უფრო მოსახერხებელია ოპერატორული ჩანაწერის გამოყენება. ფუნქციონალური ჩაწერის აუცილებლობა შესაძლოა გამოწვეული იყოს მომავალში ახალი ფუნქციების დამატებით, რომლებსაც ოპერატორული ანალოგები არ ექნებათ.

<წესის_შეზღუდვის_ოპერატორი> ::= <წესის_შეზღუდვის_არგუმენტი> [”:=” | ”==” |
”<==” | ”=”] [<წესის_შეზღუდვის_არგუმენტი> | <წესის_შეზღუდვის_არგუმენტები>]

<წესის_შეზღუდვის_არგუმენტები> ::= ”(” { <წესის_შეზღუდვის_არგუმენტი> } ”)”

<წესის_შეზღუდვის_არგუმენტი> ::= <ცვლადზე_მითითება> |
<თვისებათა_სტრუქტურა> | <სტრიქონი> | <გზა>

<წესის_შეზღუდვის_ფუნქცია> ::= <იდენტიფიკატორი> <
წესის_შეზღუდვის_არგუმენტები >

<გზა> ::= ”<” <წესის_სიმბოლო> <გზის_კომპონენტები> | ”$” <იდენტიფიკატორი>
<გზის_კომპონენტები> ”>”

<გზის_კომპონენტები> ::= { <იდენტიფიკატორი> }

შევნიშნოთ, რომ კონსტრუქცია <გზა> საშუალებას გვაძლევს მივმართოთ არამარტო მთლიანად თვისებათა სტრუქტურას, არამედ მის კონკრეტულ ქვეთვისებასაც. თუ ასეთი ქვეთვისება არ არსებობს, მაშინ იგი ავტომატურად შეიქმნება და მას მიენიჭება ცარიელი თვისება None.

<წესის_სიმბოლო> ::= <იდენტიფიკატორი> [”.” <ნომერი>]

წესში გამოყენებული სიმბოლო წარმოადგენს მორფემათა კლასს რომელიც უნდა აღწერილი იყოს წესში მის გამოყენებამდე. ერთიდაიგივე მორფემათა კლასის წესში მრავალჯერადი გამეორების შემთხვევაში აუცილებელია ისინი გავარჩიოთ ინდექსების მიხედვით.

<მორფემის_განსაზღვრა> ::= ”@” <იდენტიფიკატორი> ”=” ”{”
<მორფემის_მნიშვნელობები> ”}”

<მორფემის_მნიშვნელობები> ::= <მორფემის_მნიშვნელობა> { ”,”
<მორფემის_მნიშვნელობა>}

<მორფემის_მნიშვნელობა> ::= <სტრიქონი> <თვისება>

მორფემის განსაზღვრისას მას მიეთითება მისი შესაბამისი თვისებათა სტრუქტურა, რომელშიც იწერება ინფორმაცია ამ კონკრეტული მორფემის თვისებების შესახებ. შესაძლებელია ეს თვისებათა სტრუქტურა ცარიელიც იყოს.

<ცვლადის_განსაზღვრა> ::= ”$” <იდენტიფიკატორი> ”=” <თვისება>

<ცვლადზე_მითითება> ::= ”$” <იდენტიფიკატორი>

<მუდმივას_განსაზღვრა> ::= <იდენტიფიკატორი> ”=” <თვისება>

<მუდმივაზე_მითითება> ::= <იდენტიფიკატორი>

ცვლადი და მუდმივა პრაქტიკულად ერთი პრინციპით განისაზღვრებიან. განსხვავება მხოლოდ იმაშია, რომ ცვლადს წინ ეწერება დოლარის ნიშანი ”$” და მისი გამოყენება შეიძლება როგორც წასაკითხად ასევე ჩასაწერად.

<თვისებათა_სტრუქტურა> ::= ”[” [<ინიციალიზაციის_ნაწილი>] [
<წყვილების_მიმდევრობა>] ”]”

<ინიციალიზაციის_ნაწილი> ::= ”(” <ინიციალიზატორთა_მიმდევრობა> ”)”

<ინიციალიზატორთა_მიმდევრობა> ::= <ინიციალიზატორი> [<ინიციალიზატორი>]

<ინიციალიზატორი> ::= <ცვლადზე_მითითება> | <მუდმივაზე_მითითება>

<წყვილების_მიმდევრობა> ::= <წყვილი> { <წყვილი> }

<წყვილი> ::= <სახელი> ”:” <თვისება>

<სახელი> ::= <ინდენტიფიკატორი>

<თვისება> ::= ”+” | ”-” | <რიცხვი> | <იდენტიფიკატორი> | <სტრიქონი> |
<თვისებათა_სტრუქტურა>

§7. ქართული ენის მორფოლოგიის ფრაგმენტი

პროგრამული სისტემა გამოიცადა ქართული ენისათვის. შემუშავებული ფორმალიზმის ფარგლებში ჩაიწერა ქართული ენის მორფოლოგიის წესების გარკვეული ნაწილი. მიღებული გრამატიკის ფაილის საფუძველზე მოხდა ექსპერიმენტული შედეგების მიღება.

მორფოლოგიური წესები 4 ფაილად არის დანაწილებული. ეს ფაილებია:

- geo.mor – მორფემების დეკლარაცია არსებითი და ზედსართავი სახელებისათვის
- geo.rul - მორფოლოგიური წესები არსებითი და ზედსართავი სახელებისათვის
- geo_zmna.mor - მორფემების დეკლარაცია ზმნებისათვის
- geo_zmna.rul - მორფოლოგიური წესები ზმნებისათვის; ეს ფაილი ასევე განსაკუთრებულია იმით რომ მასში ფართოდაა გამოყენებული პარამეტრიანი მაკრო-ჩასმები

ფაილი **geo.mor**

```
ТЙЭ_Мб = [ТЙЭбЕЙ: [Мб: +]]
ТЙЭ_МТ = [ТЙЭбЕЙ: [МТ: +]]
ТЙЭ_Мб_МТ = [ТЙЭбЕЙ: [Мб: + МТ: +]]

БТФ1  = [(ТЙЭ_Мб_МТ) БТФ: 1] # УАбЛЙ
БТФ2  = [(ТЙЭ_Мб)    БТФ: 2] # ЭА
БТФ3  = [(ТЙЭ_Мб_МТ) БТФ: 3] # Э
БТФ4  = [(ТЙЭ_Мб)    БТФ: 4] # УФЦМАТЙ
БТФ5  = [(ТЙЭ_Мб_МТ) БТФ: 5] # УФЦМТ
БТФ6  = [(ТЙЭ_Мб)    БТФ: 6] # МПЪЕАТД
БТФ7  = [(ТЙЭ_Мб)    БТФ: 7] # КПГАЛА
БТФ8  = [(ТЙЭ_Мб_МТ) БТФ: 8] # бД
БТФ9  = [(ТЙЭ_Мб)    БТФ: 9] # б
БТФ10 = [(ТЙЭ_Мб) БТФ: 10] # ВЙПТВЙ
БТФ11 = [(ТЙЭ_МТ)    БТФ: 11] # КПГАЛ
БТФ12 = [(ТЙЭ_Мб_МТ) БТФ: 12] # ЦИП


БТ_УАб  = [БТЦНЕА: УАб]
БТ_МПИб = [БТЦНЕА: МПИб]
БТ_МЙЭ  = [БТЦНЕА: МЙЭ]
БТ_НАИ  = [БТЦНЕА: НАИ]
БТ_МПШМ = [БТЦНЕА: МПШМ]
БТ_ЕЙИ  = [БТЦНЕА: ЕЙИ]
БТ_ЯПГ  = [БТЦНЕА: ЯПГ]
БТ_НАИ_ЭАТ = [БТЦНЕА: НАИ_ЭАТ]

НАЯЙЛАКЙ = [НАЯЙЛАКЙ: +]
ВАЕТЭ_НЙЫ = [ВАЕТЭПБА: +]

ТЙЭФ1 = [ТЙЭФ: 1] # ПТЙЕД
ТЙЭФ2 = [ТЙЭФ: 2] # МбПЛПБЙИЙ
ТЙЭФ3 = [ТЙЭФ: 3] # МТАЕЛПБЙИЙ
ЯПГФ0 = [ЯПГФ: 0] # АТ ЙбМАТДБА ЯПГДБЙИЫІ
```

```
ЯПГФ1 = [ЯПГФ: 1] # П - УАбЛЙ
ЯПГФ2 = [ЯПГФ: 2] # Е, "" - ЦИП
ЯПГФ3 = [ЯПГФ: 3] # "" - ВЙПТВЙ
ЯПГФ4 = [ЯПГФ: 4] # Е, П - ЭА
ЯПГФ5 = [ЯПГФ: 5] # П, "" - МЦТМАНЙ

@ЧЦЮД =
   {
   "УАбЛ" [(БТФ1 ТЙЭФ1 ЯПГФ1)],
   "ЭА"   [(БТФ2 ТЙЭФ2 ЯПГФ4)],
   "Э"    [(БТФ3 ТЙЭФ2 ЯПГФ0)],
   "УФЦМАТ"  [(БТФ4 ТЙЭФ2 ЯПГФ1)],
   "УФЦМТ"   [(БТФ5 ТЙЭФ1 ЯПГФ0)],
   "МПЪЕАТД" [(БТФ6 ТЙЭФ2 ЯПГФ2)],
   "КПГАЛА" [(БТФ7 ТЙЭФ2 ЯПГФ2)],
   "бД" [(БТФ8 ТЙЭФ1 ЯПГФ4)],
   "б" [(БТФ9 ТЙЭФ2 ЯПГФ0)],
   "ВЙПТВЙ" [(БТФ10 ТЙЭФ2 ЯПГФ3)],
   "КПГАЛ" [(БТФ11 ТЙЭФ3 ЯПГФ0)],
   "ЦИП" [(БТФ12 ТЙЭФ1 ЯПГФ2)],
   "УПЧДЛ" [(БТФ1 ТЙЭФ1 ЯПГФ1)],
   "УПЧЛ" [(БТФ5 ТЙЭФ1 ЯПГФ0)]
   }

БТФУ0 = [БТФУ: 0] # АТУДБЙИ УАбДЛИАН АТ ЙБТЦНЕЙУ
БТФУ1 = [БТФУ: 1] # ФЙРЙ 1
БТФУ2 = [БТФУ: 2] # ФЙРЙ 2

@ЧЦЮД_ЖДГУ =
   {
   "ГЙГ" [(БТФ1 БТФУ1 ТЙЭФ1)],
   "РАФАТА" [(БТФ2 БТФУ2 ТЙЭФ2)],
   "РАФАТ" [(БТФ3 БТФУ0 ТЙЭФ1)],
   "ЦДНП" [(БТФ4 БТФУ2 ТЙЭФ1)]
   }

@БТЦНЕА_ЖДГУ =
   {
   "Й"  [], # 1
   "МА" [], # 2
   "У"  [], # 1 2 4
   "ЙУ" [], # 1 3
   "ЙИ" [], # 1 3
   "ИЙ" [], # 4
   "АГ" [], # 1
   "Г"  [], # 2 4
   "П"  [], # 1
   "Е"  [], # 2 4
   "М"  [], # 2 4
   "А"  [], # 1 2 3 4
   ""   []  # 2 4
   }
```

```
@ТЙЭбЕЙ =
   {
   "ДБ" [ТЙЭбЕЙ: МТ],
   "И" [ТЙЭбЕЙ: МТ_НИ],
   "Н" [ТЙЭбЕЙ: МТ_НИ],
   "" [ТЙЭбЕЙ: Мб]
   }

@БТЦНЕА =
   {
   "Й"  [], #[(БТФ1 БТФ4)],
   "МА" [], #[(БТФ1 БТФ4)],
   "ЙУ" [], #[(БТФ1 БТФ3 БТФ5 БТФ9)],
   "У"  [], #[(БТФ1 БТФ2 БТФ4 БТФ6 БТФ7 БТФ8 БТФ10 БТФ12)],
   "ЙИ" [], #[(БТФ1 БТФ3 БТФ5 БТФ9)],
   "И"  [], #[(БТФ10)],
   "ИЙ" [], #[(БТФ7 БТФ12)],
   "АГ" [], #[(БТФ1 БТФ5)],
   "Г"  [], #[(БТФ2 БТФ7 БТФ8 БТФ10 БТФ12)],
   "П"  [], #[(БТФ1 БТФ2 БТФ4 БТФ6 БТФ7 БТФ8 БТФ10 БТФ12)],
   "Е"  [], #[(БТФ1 БТФ2 БТФ4 БТФ6 БТФ7 БТФ8 БТФ10 БТФ12)],
   "М"  [], #[(БТФ2 БТФ6 БТФ7 БТФ8 БТФ10 БТФ12)],
   "А"  [], #[],
   ""   []  #[(БТФ1 БТФ2 БТФ4 БТФ6 БТФ7 БТФ8 БТФ10 БТФ12)]
   }

@ИАНГ =
   {
   "ЕЙИ"   [(БТ_УАб)],
   "ГАН"   [(БТ_УАб)],
   "АЕЙИ"  [(БТ_МЙЭ)],
   "ИАН"   [(БТ_МЙЭ)],
   "АИЕЙУ" [(БТ_НАИ)],
   "ИЕЙУ"  [(БТ_НАИ)],
   "АВАН"  [(БТ_НАИ)],
   "ВАН"   [(БТ_НАИ)],
   "АКДН"  [(БТ_НАИ)],
   "КДН"   [(БТ_НАИ)],
   "ЫЙ"    [(БТ_НАИ_ЭАТ)],
   "ЖД"    [(БТ_НАИ_ЭАТ)],
   "ИАН"   [(БТ_НАИ_ЭАТ)],
   "АМГД"  [(БТ_НАИ_ЭАТ)],
   "АМГЙУ" [(БТ_НАИ_ЭАТ)],
   "ЦТИ"   [(БТ_МПШМ)],
   ""      []
   }
@НАЯ =
   {
   "Э" [(НАЯЙЛАКЙ)],
   "АЭ" [(НАЯЙЛАКЙ)],
   "ЕД" [(НАЯЙЛАКЙ)],
```

```
   "ЩА" [(НАЯЙЛАКЙ)],
   "" []
   }

@ВАЕТЭ =
   {
   "А" [(ВАЕТЭ_НЙЫ)],
   "" []
   }

.
```

ფაილი **geo.rul**

```
$LEX = "ЛДШУ"

<”geo.mor”>

АТУУАб ->
   ЧЦЮД   { <АТУУАб ЧЦЮД> := <ЧЦЮД ЛДШУ> }
   ТЙЭбЕЙ {
      <АТУУАб ТЙЭбЕЙ_ЛДШУ> := <ТЙЭбЕЙ ЛДШУ> &
      ( <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" & <ЧЦЮД ТЙЭФ> = ("1" "3") |
         <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" & <ЧЦЮД ТЙЭФ> = ("1" "2") |
         <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" & <ЧЦЮД БТФ> = ("1" "2" "4" "6" "7" "8" "11"
"12")
       ) &
      <АТУУАб ТЙЭбЕЙ> := <ТЙЭбЕЙ ТЙЭбЕЙ>
       }
   БТЦНЕА {
      <АТУУАб БТЦНЕА_ЛДШУ> := <БТЦНЕА ЛДШУ> &
       ( <БТЦНЕА ЛДШУ> = "Й"  &
         ( <ЧЦЮД БТФ> = ("1" "4") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" |
           <ТЙЭбЕЙ ЛДШУ> = "Н" ) &
         <АТУУАб БТЦНЕА> := "УАб" )  |
        ( <БТЦНЕА ЛДШУ> = "МА" &
         ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
         ( <ЧЦЮД БТФ> = ("1" "4") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
         <АТУУАб БТЦНЕА> := "МПИб" ) |
        ( <БТЦНЕА ЛДШУ> = "ЙУ" &
         ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД БТФ> = ("1" "3" "5" "9") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
         <АТУУАб БТЦНЕА> := "НАИ" ) |
        ( <БТЦНЕА ЛДШУ> = "У" &
         ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД БТФ> = ("1" "2" "4" "6" "7" "8" "10") |
           <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
           ( <ЧЦЮД БТФ> = ("7" "10" "12") &
             <АТУУАб БТЦНЕА> := "НАИ_МЙЭ" |
             <АТУУАб БТЦНЕА> := "МЙЭ") ) |
        ( <БТЦНЕА ЛДШУ> = "ЙИ" &
```

```
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
      ( <ЧЦЮД БТФ> = ("1" "3" "5" "9") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ") &
        <АТУУАб БТЦНЕА> := "МПШМ" ) |
        <БТЦНЕА ЛДШУ> = "ИЙ" & ~<ТЙЭбЕЙ ТЙЭбЕЙ> = ("МТ_НИ" "МТ")  &
        <ЧЦЮД БТФ> = ("7" "12") & <АТУУАб БТЦНЕА> := "МПШМ" |
      ( <БТЦНЕА ЛДШУ> = "АГ" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД БТФ> = ("1" "5") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
        <АТУУАб БТЦНЕА> := "ЕЙИ" ) |
      ( <БТЦНЕА ЛДШУ> = ("П") &
        ( <ЧЦЮД ЯПГФ> = ("1" "4" "5") |
          <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ") &
        <АТУУАб БТЦНЕА> := "ЯПГ" ) |
      ( <БТЦНЕА ЛДШУ> = ("Е") &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> =("МТ" "МТ_НИ") &
        <ЧЦЮД ЯПГФ> = ("2" "4") &
          <АТУУАб БТЦНЕА> := "ЯПГ" ) |
      ( <БТЦНЕА ЛДШУ> = "М" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
      <ЧЦЮД БТФ> = ("2" "6" "7" "8" "10" "12") &
        <АТУУАб БТЦНЕА> := "МПИб" ) |
      ( <БТЦНЕА ЛДШУ> = "А" &
      <ТЙЭбЕЙ ЛДШУ> = "И" &
        <АТУУАб БТЦНЕА> := "НАИ_МПИб_МЙЭ" ) |
      ( <БТЦНЕА ЛДШУ> = "И" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        <ЧЦЮД БТФ> = "10" &
        <АТУУАб БТЦНЕА> := "МПШМ") |
      ( <БТЦНЕА ЛДШУ> = "Г" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        <ЧЦЮД БТФ> = ("2" "7" "8" "10" "12") &
        <АТУУАб БТЦНЕА> := "ЕЙИ" ) |
      ( <БТЦНЕА ЛДШУ> = ""  & ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" )
  }
  ИАНГ {
     (
     ( <АТУУАб ТЙЭбЕЙ> = ("Мб" "МТ") &
      ( <АТУУАб БТЦНЕА> =
            ("УАб" "МПИб" "МЙЭ" "НАИ" "ЕЙИ" "МПШМ" "ЯПГ" "НАИ_МЙЭ"
"НАИ_МПИб_МЙЭ") &
          <АТУУАб БТЦНЕА> = <ИАНГ БТЦНЕА> |
          <АТУУАб БТЦНЕА> = "НАИ_МЙЭ" & <ИАНГ БТЦНЕА> = ("НАИ" "МЙЭ") |
          <АТУУАб БТЦНЕА> = "НАИ_МПИб_МЙЭ" & <ИАНГ БТЦНЕА> = ("НАИ"
"МЙЭ") |
          ( <БТЦНЕА ЛДШУ> = "" &
            <ИАНГ БТЦНЕА> = "НАИ_ЭАТ" &
            <АТУУАб БТЦНЕА> := "НАИ"
          ) ) ) |
      ( <ИАНГ ЛДШУ> = ("ИЕЙУ" "") &
        <БТЦНЕА ЛДШУ> = "А" &
        <ТЙЭбЕЙ ЛДШУ> = "И"
      ) |
```

```
        ( <ИАНГ ЛДШУ> = "" &
          ( <БТЦНЕА ЛДШУ> = "" &
            <ЧЦЮД БТФ> = ("2" "6" "7" "8" "10" "12") &
            ( <ЧЦЮД ЯПГФ> = ("2" "3" "5") & <АТУУАб БТЦНЕА> := "УАб_ЯПГ" |
              <ЧЦЮД ЯПГФ> = ("0" "1" "4") & <АТУУАб БТЦНЕА> := "УАб"  ) |
          <АТУУАб БТЦНЕА> = ("УАб" "МПИб" "МЙЭ" "НАИ" "ЕЙИ" "МПШМ" "ЯПГ"
"НАИ_МЙЭ" "НАИ_МПИб_МЙЭ") |
          <ТЙЭбЕЙ ЛДШУ> = "Н") )
        ) &
        <АТУУАб ИАНГ> := <ИАНГ ЛДШУ>
    }
    НАЯ {
        ( <АТУУАб БТЦНЕА> = "ЯПГ" & <НАЯ ЛДШУ> = "" |
          ( ~(<АТУУАб БТЦНЕА> = "ЯПГ") &
            (~<НАЯ ЛДШУ> = "") & <АТУУАб НАЯ> := <НАЯ ЛДШУ> | <НАЯ ЛДШУ> =
"" )
        )
    }
    ВАЕТЭ {
       (~<ВАЕТЭ ЛДШУ> = "") &
        ( ~(<АТУУАб БТЦНЕА> = "ЭПГ" | <БТЦНЕА ЛДШУ> = "МА") ) & <АТУУАб
ВАЕТЭ> := <ВАЕТЭ ЛДШУ>  |
        <ВАЕТЭ ЛДШУ> = ""
    } ;

ЖДГУ_УАб ->
    ЧЦЮД_ЖДГУ
    ТЙЭбЕЙ {
       ( <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" & <ЧЦЮД_ЖДГУ ТЙЭФ> = ("1" "3") |
          <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" & <ЧЦЮД_ЖДГУ ТЙЭФ> = ("1" "2") |
          <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ"
        ) &
       <ЖДГУ_УАб ТЙЭбЕЙ> := <ТЙЭбЕЙ ТЙЭбЕЙ> &
        <ЖДГУ_УАб ТЙЭбЕЙ_ЛДШУ> := <ТЙЭбЕЙ ЛДШУ>
    }
    БТЦНЕА {
       (
        ( <БТЦНЕА ЛДШУ> = "Й" &
          ( <ЧЦЮД_ЖДГУ БТФ> = "1" | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" |
            <ТЙЭбЕЙ ЛДШУ> = "Н") &
          <ЖДГУ_УАб БТЦНЕА> := "УАб" &
          ( <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" &
            <ЖДГУ_УАб УАб_БТЦНЕА> := "УАб_НАИ_МПШМ" | 1 ) ) |
        ( <БТЦНЕА ЛДШУ> = "МА" &
          ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
          ( <ЧЦЮД_ЖДГУ БТФ> = "1" | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
          <ЖДГУ_УАб БТЦНЕА> := "МПИб" &
          ( <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" &
            <ЧЦЮД_ЖДГУ БТФУ> = "1" & <ЖДГУ_УАб УАб_БТЦНЕА> := "МПИб" | 1 ) )
|
        ( <БТЦНЕА ЛДШУ> = "У" &
          ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
```

```
          ( <ЧЦЮД_ЖДГУ БТФ> = ("1" "2" "4") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
            ( <ЧЦЮД_ЖДГУ БТФ> = "4" & <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" &
              <ЖДГУ_УАб БТЦНЕА> := "НАИ_МЙЭ" | <ЖДГУ_УАб БТЦНЕА> := "МЙЭ" )
) |
      ( <БТЦНЕА ЛДШУ> = "ЙУ" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("1" "3") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
        <ЖДГУ_УАб БТЦНЕА> := "НАИ" ) |
       ( <БТЦНЕА ЛДШУ> = "ЙИ" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("1" "3") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
          <ЖДГУ_УАб БТЦНЕА> := "МПШМ" ) |
       ( <БТЦНЕА ЛДШУ> = "АГ" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("1") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
          <ЖДГУ_УАб БТЦНЕА> := "ЕЙИ" ) |
       ( <БТЦНЕА ЛДШУ> = "П" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("1") | <ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ" ) &
        ~<ТЙЭбЕЙ ЛДШУ> = "И" &
        <ЖДГУ_УАб БТЦНЕА> := "ЯПГ" &
        ( <ТЙЭбЕЙ ТЙЭбЕЙ> = "Мб" &
          <ЧЦЮД_ЖДГУ БТФУ> = "1" & <ЖДГУ_УАб УАб_БТЦНЕА> := "ЯПГ" | 1 ) ) |
       ( <БТЦНЕА ЛДШУ> = "М" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("2" "4") ) &
          <ЖДГУ_УАб БТЦНЕА> := "МПИб" ) |
       ( <БТЦНЕА ЛДШУ> = "Г" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("2" "4") ) &
          <ЖДГУ_УАб БТЦНЕА> := "ЕЙИ" ) |
       ( <БТЦНЕА ЛДШУ> = "Е" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = ("2" "4") ) &
          <ЖДГУ_УАб БТЦНЕА> := "ЯПГ" ) |
       ( <БТЦНЕА ЛДШУ> = "ИЙ" &
        ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        ( <ЧЦЮД_ЖДГУ БТФ> = "4" ) &
          <ЖДГУ_УАб БТЦНЕА> := "МПШМ" ) |
       ( <БТЦНЕА ЛДШУ> = "А" &
      <ТЙЭбЕЙ ЛДШУ> = "И" &
        <ЖДГУ_УАб БТЦНЕА> := "НАИ_МПИб_МЙЭ" ) |
       ( <БТЦНЕА ЛДШУ> = "" & ~<ТЙЭбЕЙ ТЙЭбЕЙ> = "МТ_НИ" &
        <ЧЦЮД_ЖДГУ БТФ> = ("1" "2" "4") &
        ( <ЧЦЮД_ЖДГУ БТФУ> = "2" & <ЖДГУ_УАб УАб_БТЦНЕА> := "ЪЕДЛА" | 1
)
        )
   ) &
   <ЖДГУ_УАб БТЦНЕА_ЛДШУ> := <БТЦНЕА ЛДШУ>
   }
   ИАНГ {
      (
      ( <ЖДГУ_УАб ТЙЭбЕЙ> = ("Мб" "МТ") &
```

```
        ( <ЖДГУ_УАб БТЦНЕА> =
              ("УАб" "МПИб" "МЙЭ" "НАИ" "ЕЙИ" "МПШМ" "ЯПГ" "НАИ_МЙЭ"
"НАИ_МПИб_МЙЭ") &
            <ЖДГУ_УАб БТЦНЕА> = <ИАНГ БТЦНЕА> |
            <ЖДГУ_УАб БТЦНЕА> = "НАИ_МЙЭ" & <ИАНГ БТЦНЕА> = ("НАИ" "МЙЭ")
|
            <ЖДГУ_УАб БТЦНЕА> = "НАИ_МПИб_МЙЭ" & <ИАНГ БТЦНЕА> = ("НАИ"
"МЙЭ") |
            ( <БТЦНЕА ЛДШУ> = "" &
              <ИАНГ БТЦНЕА> = "НАИ_ЭАТ" &
              <ЖДГУ_УАб БТЦНЕА> := "НАИ"
            ) ) ) |
          ( <ИАНГ ЛДШУ> = ("ИЕЙУ" "") &
            <БТЦНЕА ЛДШУ> = "А" &
            <ТЙЭбЕЙ ЛДШУ> = "И"
          ) |
          ( <ИАНГ ЛДШУ> = "" &
            ( <БТЦНЕА ЛДШУ> = "" &
              <ЧЦЮД_ЖДГУ БТФ> = ("2" "4") &
              ( <ЖДГУ_УАб БТЦНЕА> := "УАб" &
                ( <ЧЦЮД_ЖДГУ БТФУ> = "1" & <ЖДГУ_УАб УАб_БТЦНЕА> :=
"МЙЭ_ЕЙИ" | 1 ) &
                ( <ЧЦЮД_ЖДГУ БТФУ> = "2" & <ЖДГУ_УАб УАб_БТЦНЕА> := "ЪЕДЛА" |
1 )
        ) |
            <ЖДГУ_УАб БТЦНЕА> = ("УАб" "МПИб" "МЙЭ" "НАИ" "ЕЙИ" "МПШМ" "ЯПГ"
"НАИ_МЙЭ" "НАИ_МПИб_МЙЭ") |
            <ТЙЭбЕЙ ЛДШУ> = "Н") )
          ) &
          <ЖДГУ_УАб ИАНГ> := <ИАНГ ЛДШУ>
    }
    НАЯ {
          ( <ЖДГУ_УАб БТЦНЕА> = "ЯПГ" & <НАЯ ЛДШУ> = "" |
            ( ~(<ЖДГУ_УАб БТЦНЕА> = "ЯПГ") &
              (~<НАЯ ЛДШУ> = "") & <ЖДГУ_УАб НАЯ> := <НАЯ ЛДШУ> | <НАЯ
ЛДШУ> = "" )
        )
    }
    ВАЕТЭ {
        (~<ВАЕТЭ ЛДШУ> = "") &
          ( ~(<ЖДГУ_УАб БТЦНЕА> = "ЭПГ" | <БТЦНЕА ЛДШУ> = "МА") ) |
          <ВАЕТЭ ЛДШУ> = ""
    } ;

<”geo_zmna.mor”>
<”geo_zmna.rul”>

.
```

ფა ი ლი **geo_zmna.mor**

```
@ЖЯ =
   {
   "МЙ"    [],
   "МП"    [],
   "АМП"   [],
   "А"     [],
   "ГА"    [],
   "ЬАМП"  [],
   "ЬА"    [],
   "ЫДМП"  [],
   "ЫД"    [],
   "ВАГА"  [],
   "ВАГМП" [],
   "ВАМП"  [],
   "ВА"    [],
   "ЯАМП"  [],
   "ЯА"    [],
   ""      []
   }

@РНРТДЧ =
   {
   "Е"  [],
   "У"  [],
   "б"  [],
   "д"  [],
   "М"  [],
   "ВЕ" [],
   "В"  [],
   ""   []
   }

@бРТДЧ =
   {
   "А" [],
   "Й" [],
   "Ц" [],
   "Д" [],
   ""  []
   }

@ГЕНДБ =
   {
   "Г" [],
   ""  []
   }
@КФ =
   {
   "ДБЙН" [],
   ""     []
```

```
    }

@МЯН =
    {
    "ГЙ"   [],
    "Г"    [],
    "ГД"   [],
    "Д"    [],
    "П"    [],
    "А"    [],
    "ПУ"   [],
    "Й"    [],
    "ПГЙ"  [],
    "ПГД"  [],
    "ПГ"   [],
    "ДБЦЛ" [],
    "ЙА"   [],
    ""     []
    }

@ИН =
    {
    "ДБ" [],
    "ПБ" [],
    "АМ" [],
    "АЕ" [],
    "Й"  [],
    "ДМ" [],
    "ПЕ" [],
    ""   []
    }

@РНУЦЧ =
    {
    "У"   [],
    "П"   [],
    "АН"  [],
    "А"   [],
    "ДН"  [],
    "НДН" [],
    "Н"   [],
    "ДУ"  [],
    "ЕАТ" [],
    "бАТ" [],
    ""    []
    }

@ТЭб =
    {
    "И" [],
    ""  []
    }
```

```
ЭАТ    = [ЖЯУ: ""]
ГА     = [ЖЯУ: "ГА"]
ВА     = [ЖЯУ: "ВА"]
МП     = [ЖЯУ: "МП"]
А      = [ЖЯУ: "А"]
ЫД     = [ЖЯУ: "ЫД"]

ГЙАИ11 =  [ГЙАИ: 11]
ГЙАИ1  =  [ГЙАИ: 1]

КЛАУЙ1 = [КЛАУЙ: 1]
КЛАУЙ5 = [КЛАУЙ: 5]

ФЙРЙ1  = [ФЙРЙ: 1]
ФЙРЙ2  = [ФЙРЙ: 2]
ФЙРЙ3  = [ФЙРЙ: 3]

бРА1   = [бРА: 1 ]
бРА0   = [бРА: 0]
бРЙ1   = [бРЙ: 1]
бРЦ1   = [бРЦ: 1]
бРЭ1   = [бРЭ: 1]
бРД1   = [бРД: 1]
бРЦ0   = [бРЦ: 0]
бРД0   = [бРД: 0]
@ЮЙТЙ =
   {
   "ИАЕЙУЦЧЛ"[(ГЙАИ11 КЛАУЙ5 ВА  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД1 ФЙРЙ3
ТЙЭФ1)],
   "ИАЕУ"    [(ГЙАИ11 КЛАУЙ5 МП  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД1 ФЙРЙ3 ТЙЭФ1)],
   "КДТ"     [(ГЙАИ11 КЛАУЙ5 ГА  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД1 ФЙРЙ3 ТЙЭФ1)],
   "ЙМДГ"    [(ГЙАИ11 КЛАУЙ5 ГА  бРА1 бРЙ1 бРЦ0 бРЭ1 бРД1 ФЙРЙ3 ТЙЭФ1)],
   "ЛАМАЖ"   [(ГЙАИ11 КЛАУЙ5 ВА  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД0 ФЙРЙ3 ТЙЭФ1)],
   "УАЬЦШТ"  [(ГЙАИ11 КЛАУЙ5 ГА  бРА1 бРЙ1 бРЦ0 бРЭ1 бРД0 ФЙРЙ3 ТЙЭФ1)],
   "МПЯМ"    [(ГЙАИ11 КЛАУЙ5 ЫД  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД1 ФЙРЙ3 ТЙЭФ1)],
   "ЯЕЙМ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ1 ТЙЭФ2)],
   "ШЦб"  [(ГЙАИ1 КЛАУЙ1 МП  бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ШЦб"  [(ГЙАИ1 КЛАУЙ1 ГА  бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ЫДН"  [(ГЙАИ11 КЛАУЙ5 А  бРА1 бРЙ1 бРЦ1 бРЭ1 бРД1 ФЙРЙ3 ТЙЭФ1)],
   #"ЫДН"  [(ГЙАИ11 КЛАУЙ5 ГА бРА1 ФЙРЙ3 ТЙЭФ1)],
   "аДШ"  [(ГЙАИ1 КЛАУЙ1 ГА  бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ЯЦб"  [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ2 ТЙЭФ1)],
   "СЩДТ" [(ГЙАИ1 КЛАУЙ1 ВА  бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ЪДЧ"  [(ГЙАИ1 КЛАУЙ1 ГА  ФЙРЙ3 бРА0 ТЙЭФ1)],
   "аАГ"  [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ЫбДЧ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ЬШДЧ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ2 ТЙЭФ1)],
   "ЮВДТ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ3 ТЙЭФ1)],
   "БВДТ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ВЛПЕ" [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ3 ТЙЭФ1)],
   "ГЦМ"  [(ГЙАИ1 КЛАУЙ1 ГА  бРА0 ФЙРЙ2 ТЙЭФ1)],
```

```
   "ГЦЩ"  [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ2 ТЙЭФ1)],
   "ИПЕ"  [(ГЙАИ1 КЛАУЙ1 ЭАТ бРА0 ФЙРЙ1 ТЙЭФ2)]
   }
```

.

ფაილი **geo_zmna.rul**

```
$$МЯКТЙЕЙ
(<ЖЯ ЛДШУ> = "" | (~<ЖЯ ЛДШУ> = "" &
<ЮЙТЙ ЖЯУ> = <ЖЯ ЛДШУ> & ~$3 = ("1" "2" "3"))) &
<ИН ЛДШУ> = $0 &
<бРТДЧ ЛДШУ> = $1 &
<МЯН ЛДШУ> = ($2 $13) &
<ЖМНА МЯК> := $3 &
(
 (
 <РНРТДЧ ЛДШУ> = $4 &
 <РНУЦЧ ЛДШУ> = $5 &
 <ЮЙТЙ ФЙРЙ> = "2" &
 <ЖМНА РЙТЙ> := "1"
 ) |
 (
 <РНРТДЧ ЛДШУ> = $6 &
 <РНУЦЧ ЛДШУ> = $7 &
 <ЮЙТЙ ФЙРЙ> = "2" &
 <ЖМНА РЙТЙ> := "2"
 ) |
 (
 <РНРТДЧ ЛДШУ> = $8 &
 (
  (
  <РНУЦЧ ЛДШУ> = $9 & <МЯН ЛДШУ> = $13 &
  <ЖМНА РЙТЙ> := "3"
  ) |
  (
  <РНУЦЧ ЛДШУ> = $10 & <МЯН ЛДШУ> = $13 &
  <ТЭб ЛДШУ> = "" &
  <ЖМНА РЙТЙ> := "3" &
  <ЖМНА ТЙЭбЕЙ> := "МТ"
  )
 )
) |
(
<РНРТДЧ ЛДШУ> = $4 &
<РНУЦЧ ЛДШУ> = $11 &
<ЮЙТЙ ФЙРЙ> = "3" &
<ЖМНА РЙТЙ> := "1"
 ) |
(
```

```
<РНРТДЧ ЛДШУ> = $6 &
<РНУЦЧ ЛДШУ> = $12 &
<ЮЙТЙ ФЙРЙ> = "3" &
<ЖМНА РЙТЙ> := "2"
)
)
 &
(
<ЖМНА ТЙЭбЕЙ> = "МТ" & <ЮЙТЙ ТЙЭФ> = ("1" "3") |
<ЖМНА ТЙЭбЕЙ> = "Мб" & <ЮЙТЙ ТЙЭФ> = ("1" "2")
)
$$.
$$МДУАМДУ
((<ЖЯ ЛДШУ> = "" | (~<ЖЯ ЛДШУ> = "" &
<ЮЙТЙ ЖЯУ> = <ЖЯ ЛДШУ>)) &
<ИН ЛДШУ> = $0 & <бРТДЧ ЛДШУ> = $1 &
(<МЯН ЛДШУ> = $2) &
((
 <РНУЦЧ ЛДШУ> = "" & ((<РНРТДЧ ЛДШУ> = "М" & <ЮЙТЙ ТЙЭФ> = ("1" "2")
  & ~<ЮЙТЙ ФЙРЙ> = "1"
  & <ЖМНА ТЙЭбЕЙ> := "Мб")  | (<РНРТДЧ ЛДШУ> = "ВЕ" & <ЮЙТЙ ТЙЭФ> =
  ("1" "3") & <ЖМНА ТЙЭбЕЙ> := "МТ") )
  & <ЖМНА РЙТЙ > := "1"
 ) |
 (<РНРТДЧ ЛДШУ> = "В" & (<ТЭб ЛДШУ> = ""  | <МЯН ЛДШУ> = "П" &
  <ТЭб ЛДШУ> = "И" & <РНУЦЧ ЛДШУ> = "" | <ТЭб ЛДШУ> = "И"
  & ~<ЮЙТЙ ФЙРЙ> = "1") &<РНУЦЧ ЛДШУ> = "" &  <ЖМНА РЙТЙ> := "2"
 ) |
 (<РНРТДЧ ЛДШУ> = "" & (<ТЭб ЛДШУ> = "" | <МЯН ЛДШУ> = "П" &
  <ТЭб ЛДШУ> = "И" & <РНУЦЧ ЛДШУ> = "" | <ТЭб ЛДШУ> = "И"
  & ~<ЮЙТЙ ФЙРЙ> = "1") & <РНУЦЧ ЛДШУ> = "" & <ЖМНА РЙТЙ> := "3" &
  <бРТДЧ ЛДШУ> = $3
 )
)
& <ЖМНА МЯК> := $4)
 $$.
 $$ТЙЭбЕЙ
 (<ТЭб ЛДШУ> = "" & <ЖМНА ТУЦБ> := "Мб"
  | <ТЭб ЛДШУ> = "И" & <ЖМНА ТУЦБ> := "МТ")
  $$.
 $$РЙТЙТ
((<РНРТДЧ ЛДШУ> = "" & ((<РНУЦЧ ЛДШУ> = "" & <ЖМНА РУЦБ> := "2" &
                      ${ТЙЭбЕЙ:}$)
                     | (<РНУЦЧ ЛДШУ> = $0 & <ЖМНА РУЦБ> := "3"
                       & <ЖМНА ТУЦБ> := "Мб")
                          | (<РНУЦЧ ЛДШУ> = $1 & <ЖМНА РУЦБ> := "3"
                       & <ЖМНА ТУЦБ> := "МТ")) &
                             <ЖМНА РПБ> := "3" & <ЖМНА ТПБ> := "")
  |  (<РНРТДЧ ЛДШУ> = "М" & ((<РНУЦЧ ЛДШУ> = "" & <ЖМНА РУЦБ> := "2" &
                      ${ТЙЭбЕЙ:}$)
                     | (<РНУЦЧ ЛДШУ> = $0 & <ЖМНА РУЦБ> := "3"
                       & <ЖМНА ТУЦБ> := "Мб")
```

```
                        | (<РНУЦЧ ЛДШУ> = $1 & <ЖМНА РУЦБ> := "3"
                    & <ЖМНА ТУЦБ> := "МТ")) &
                          <ЖМНА РПБ> := "1" & <ЖМНА ТПБ> := "Мб")
  | (<РНРТДЧ ЛДШУ> = "Е" & <РНУЦЧ ЛДШУ> = "" & <ЖМНА РУЦБ> := "1" &
                    ${ТЙЭбЕЙ:}$ & <ЖМНА РПБ> := "3" & <ЖМНА ТПБ> := "")
  | (<РНРТДЧ ЛДШУ> = "ВЕ" & ((<РНУЦЧ ЛДШУ> = "" & <ЖМНА РУЦБ> := "2" &
                    ${ТЙЭбЕЙ:}$)
                   | (<РНУЦЧ ЛДШУ> = $0 & <ЖМНА РУЦБ> := "3"
                    & <ЖМНА ТУЦБ> := "Мб")
                        | (<РНУЦЧ ЛДШУ> = $1 & <ЖМНА РУЦБ> := "3"
                    & <ЖМНА ТУЦБ> := "МТ")) &
                          <ЖМНА РПБ> := "1" & <ЖМНА ТПБ> := "МТ")
 | (<РНРТДЧ ЛДШУ> = "В" & ((<РНУЦЧ ЛДШУ> = "" & <ЖМНА РУЦБ> := "2" &
                    ${ТЙЭбЕЙ:}$)
                   | (<РНУЦЧ ЛДШУ> = $0 & <ЖМНА РУЦБ> := "3"
                    & <ЖМНА ТУЦБ> := "Мб")
                        | (<РНУЦЧ ЛДШУ> = $1 & <ЖМНА РУЦБ> := "3"
                    & <ЖМНА ТУЦБ> := "МТ")) &
                          <ЖМНА РПБ> := "2" & <ЖМНА ТПБ> := "Мб"))
$$.

$$МЯК1
(<ИН ЛДШУ> = $2 &  <МЯН ЛДШУ> = $3 & <ЖЯ ЛДШУ> = $8
 & (${$7:$0,$1}$) &<ЖМНА ЖЯ> := <ЖЯ ЛДШУ> & (<ЖМНА РУЦБ> = "3" &
 <МЯН ЛДШУ> = $4 | <МЯН ЛДШУ> = $5)
  & <ЖМНА МЯК> := $6 & ~<бРТДЧ ЛДШУ> = "")
  $$.


ЖМНА ->
   ЖЯ
   РНРТДЧ
   бРТДЧ
   ЮЙТЙ
    {
    (<ЮЙТЙ КЛАУЙ> = "1" &
       (
       (<ЖЯ ЛДШУ> = "" |  <ЖЯ ЛДШУ> = <ЮЙТЙ ЖЯУ> & <ЖМНА ЖЯУ> := <ЮЙТЙ
ЖЯУ>) &
       <бРТДЧ ЛДШУ> = ("" "Й" "Ц" "Д") &
       <ЖМНА ЛДШУ> := <ЮЙТЙ ЛДШУ>
       ))
       | (<ЮЙТЙ ГЙАИ> = "11" & <ЮЙТЙ КЛАУЙ> = "5" &
       (<ЖЯ ЛДШУ> = "" | <ЖЯ ЛДШУ> = <ЮЙТЙ ЖЯУ> ))
       }
     ГЕНДБ
     КФ
     ИН
       {(
       <ЮЙТЙ КЛАУЙ> = "1" &
           <ИН ЛДШУ> = ("" "ДБ")
           )
```

```
          | (<ЮЙТЙ ГЙАИ> = "11" & <ЮЙТЙ КЛАУЙ> = "5" &
      (<ЖЯ ЛДШУ> = "" | <ЖЯ ЛДШУ> = <ЮЙТЙ ЖЯУ> ))
         }
     МЯН
    РНУЦЧ
    ТЭб
     {
     (<ЮЙТЙ КЛАУЙ> = "1" &

            ((
            <ТЭб ЛДШУ> = "" & <ЮЙТЙ ТЙЭФ> = ("1" "2") & <ЖМНА ТЙЭбЕЙ> := "Мб"
|
            <ТЭб ЛДШУ> = "И" & <ЮЙТЙ ТЙЭФ> = ("1" "3") & <ЖМНА ТЙЭбЕЙ> :=
"МТ"
            ) &
 ((${МЯКТЙЕЙ: "", "","","1","Е","ЕАТ","",
 "бАТ","","У",("АН" "ДН"), "", "", ""}$)

  | (${МЯКТЙЕЙ: "", "", "ГЙ", "2", "Е", "", "", "","","А","НДН", "", "", "Г"}$)
  | (${МЯКТЙЕЙ: "", "", "ГД", "3", "Е", "", "", "","","ДУ","НДН", "", "", "Г"}$)
  | (${МЯКТЙЕЙ: "ДБ", "Й", "", "4", "Е", "", "", "","","У","ДН", "", "", ""}$)
  | (${МЯКТЙЕЙ: "ДБ", "Й", "ГЙ", "5", "Е", "", "",
   "","","А","НДН", "", "", "Г"}$)
  | (${МЯКТЙЕЙ: "ДБ", "Й", "ГД", "6", "Е", "",
   "", "","","ДУ","НДН", "", "", "Г"}$)
  | (${МЯКТЙЕЙ: "", "Й", "Д", "7", "Е", "", "", "","","А","ДУ", "", "", ""}$)
  | (${МЯКТЙЕЙ: "", "Й", "П", "8", "Е", "", "", "","","У","Н", "", "", "П"}$)
  | (${МДУАМДУ: "", ("Й" "Ц"), "ЙА",       "Ц", "9"}$)
  | (${МДУАМДУ: "", "Д",       "А",        "Д", "10"}$)
  | (${МДУАМДУ: "", "Д",       ("ПУ" "П"), "Д", "11"}$)
  ))
   | (<ЮЙТЙ ГЙАИ> = "11" & <ЮЙТЙ КЛАУЙ> = "5" &
      (<ЖЯ ЛДШУ> = "" | <ЖЯ ЛДШУ> = <ЮЙТЙ ЖЯУ> ) &
      (<ИН ЛДШУ> = ""  | <ИН ЛДШУ> = "ДБ") &
      ((<бРТДЧ ЛДШУ> = "А" &
       <ЮЙТЙ бРА> = "1") |(<бРТДЧ ЛДШУ> = "Й" &
       <ЮЙТЙ бРЙ> = "1") |(<бРТДЧ ЛДШУ> = "Ц" &
       <ЮЙТЙ бРЦ> = "1") |(<бРТДЧ ЛДШУ> = "" &
       <ЮЙТЙ бРЭ> = "1") |(<бРТДЧ ЛДШУ> = "Д" &
       <ЮЙТЙ бРД> = "1"))  &
      (${МЯК1: "У", "ДН", "ДБ", ("" ""),"", "","1", РЙТЙТ, ""}$)
      | (${МЯК1: "А", "НДН", "ДБ", ("ГЙ" "Г"),"Г", "ГЙ","2", РЙТЙТ, ""}$)
      | (${МЯК1: "ДУ", "НДН", "ДБ", ("ГД" "Г"),"Г", "ГД","3", РЙТЙТ, ""}$)
      | (${МЯК1: "У", "ДН", "ДБ", ("" ""),"", "","4", РЙТЙТ, <ЮЙТЙ ЖЯУ>}$)
      | (${МЯК1: "А", "НДН", "ДБ", ("ГЙ" "Г"),"Г", "ГЙ","5", РЙТЙТ, <ЮЙТЙ ЖЯУ>}$)
      | (${МЯК1: "А", "НДН", "ДБ", ("ГД" "Г"),"Г", "ГД","6", РЙТЙТ, <ЮЙТЙ ЖЯУ>}$)
      | (${МЯК1: "А", "ДУ", "", ("Д" ""),"", "Д","7", РЙТЙТ, <ЮЙТЙ ЖЯУ>}$)
      | (${МЯК1: "У", "Н", "", ("П" "П"),"П", "П","8", РЙТЙТ, <ЮЙТЙ ЖЯУ>}$)
      ))
 }
 ;
 .
```

თავი 2

სინტაქსური გამრჩევი

§1. ფორმალიზმი

იმისათვის, რომ სინტაქსურმა გამრჩევმა მოახერხოს ბუნებრივ ენაზე ჩაწერილი წინადადებების გარჩევა, საჭიროა მივაწოდოთ მას ამ ენის გრამატიკის წესები. ამ წესების საფუძველზე, ანალიზატორი ცდილობს ააგოს სინტაქსური გარჩევის ხე, ანუ დაადგინოს თუ როგორ ურთიერთკავშირში არიან შემავალი ლექსიკური ერთეულები. იმისათვის, რომ გავნსაზღვროთ რომელიმე ენის გრამატიკა, და ჩავწეროთ იგი ადამიანისა და მანქანისათვის გასაგები სახით, საჭიროა გარკვეული ფორმალიზმის შემოღება, რომლის ფარგლებშიც მოხდება ამგვარი ჩაწერები. სწორედ, ამგვარი ფორმალიზმის შემუშავება გახდა საჭირო პირველ ეტაპზე.

შემუშავებული ფორმალიზმი ემყარება კონტექსტისაგან თავისუფალი გრამატიკის ცნებას და კონტექსტისაგან თავისუფალი გრამატიკის წესებს. ყოველი ასეთი წესი არის აღნიშვნა იმისა, თუ როგორ შეიძლება სიმბოლოთა მიმდევრობა ჩაენაცვლოს ერთი არატერმინალური სიმბოლოთი.

მაგალითად:

S -> A S B
S -> c
A -> a
B -> b

ეს გრამატიკა აღწერს ფრაზათა ისეთ სიმრავლეს რომელთაც ცენტრში აქვთ 'c'-სიმბოლო ხოლო თავში და ბოლოში თანაბარი, არანულოვანი რაოდენობის `a' და შესაბამისად 'b' სიმბოლოები. ეს გრამატიკა დაუშვებს შემდეგი ტიპის სიმბოლოთა მიმდევრობებს: acb, aacbb, aaacbbb,… და ა.შ.. მოცემული წესები აღწერენ იმ დასაშვებ

ურთიერთდამოკიდებულებებს, რომლებშიც შეიძლება იყვნენ შემავალი ლექსიკური ერთეულები (უფრო დაწვრილებით კონტექსტისაგან თავისუფალი გრამატიკების შესახებ იხილეთ [8]). ამავე წესების გამოყენებით ჩვენ შეგვიძლია ვცადოთ ბუნებრივი ენის გრამატიკის ჩაწერა. მაგალითად, შეგვიძლია წინადადება მარტივად შემდეგი სახით განვსაზღვროთ (ქართულში ეს შეესაბამება ერთპირიანი ზმნების შემთხვევას):

S -> NP VP;

NP არის სახელის ჯგუფი, VP - ზმნის ჯგუფი. ავიღოთ წინადადება: `The boy is running'. აქ სახელის ჯგუფს წარმოადგენს სიტყვათა წყობა `The boy' ხოლო ზმნის ჯგუფს - `is running'. ქართულად უფრო მარტივია: 'ბიჭი მირბის', აქ სახელის ჯგუფი ერთი სიტყვით (არსებითი სახელით) არის წარმოდგენილი, ისევე როგორც ზმნის ჯგუფი. თუმცა, ქართლისათვის დასაშვებია აგრეთვე შებრუნებული წყობა 'მირბის ბიჭი'. ქართულისაგან განსხვავებით ინგლისური ენაში სიტყვათა წყობას მნიშვნელოვანი აზრობრივი დატვირთვა აქვს და იგი არ არის თავისუფალი, ასე რომ შემდეგი წინადადება 'Is runnng the boy' არასწორი იქნება, ხოლო შემდეგნაირი 'Is the boy running' - იცვლის აზრს და გვევლინება როგორც კითხვითი წინადადება. ქართულში შეგვიძლია ნებისმიერი კომბინაციით დავალაგოთ ერთიმეორესთან ზმნა და არსებითი სახელი, ყველა შემთხვევაში ჩვენ მივიღებთ სწორად ჩაწერილ წინადადებას (თუმცაღა სტილისტური თვალსაზრისით შეიძლება ყველა მათგანი მისაღები არ იყოს). ამ მიზეზის გამო, ქართული ენის შემთხვევაში ზემოთმოყვანილი გრამატიკა უნდა შეიცავდეს აგრეთვე მეორე წესსაც:

S -> VP NP;

რათა სიტყვათა ერთი წყობის "ჩავარდნისას" განხილულ იქნას მეორენაირი წყობა. ერთპირიანი ზმნების შემთხვევაში წინადადებაში სიტყვების წყობის სულ 2 ვარიანტი გვაქვს, ორპირიანი ზმნების შემთხვევაში ვარიანტების რაოდენობა იზრდება 6-მდე, ხოლო სამპირიანი ზმნების დროს იგი აღწევს 24-ს (!) რაც ნიშნავს იმას, რომ ქართული ენისათვის ასეთნაირად დაწერილი გრამატიკა უნდა შეიცავდეს ძალიან დიდი რაოდენობის წესებს, ეს კი არასასურველია, რადგანაც გრამატიკის ფაილი მეტისმეტად დიდი გამოდის, იგი ხდება ნაკლებად გასაგები და ამ ფაილის საწყის ეტაპზე ჩატვირთვა-დამუშავებას დიდი დრო სჭირდება. ეს გახლავთ

ერთერთი ყველაზე თვალშისაცემი ნაკლოვანება უკვე არსებულ სტანდარტულ ფორმალიზმებში, რის გამოც ქართული ენისათვის ისინი არ იყვნენ ბოლომდე მისაღები. ამ ნაკლოვანების აღმოფხვრა მოხდა სტანდარტული ფორმალიზმის გაფართოვების და მასში სიმბოლოთა ურთიერთმდებარეობის მარეგულირებელი კონსტრუქციების შემოტანის ხარჯზე. ვაჩვენოთ ეს მარტივი მაგალითის საშუალებით (ჯერჯერობით ამ ახალი ფორმალიზმის ახსნა ხდება ზოგად დონეზე, ილუსტრაციების მეშვეობით, უფრო დეტალური აღწერა იხ. ქვემოთ):

S -> A B C D : D < A B - C;

შევნიშნოთ, რომ მოცემულ კონტექსტისაგან თავისუფალი გრამატიკის წესში ორწერტილის შემდეგ გვაქვს სიმბოლოთა ურთიერთმდებარეობის მარეგულირებელი კონსტრუქცია. თავად ორწერტილის არსებობა უკვე მიუთითებს ანალიზატორს იმის თაობაზე რომ ამ წესში სიმბოლოთა ყველანაირი კომბინაციის განხილვა უნდა მოხდეს, ხოლო დამატებითი დირექტივები განსაზღვრავენ თუ რა მდებარეობით დამოკიდებულებებშია ერთიმეორესთან ეს სიმბოლოები. გაფილტვრის შედეგად დარჩება სიტყვათა მხოლოდ ის წყობები, რომლებიც ემორჩილებიან ამ ურთიერთდამოკიდებულებებს. ასეთნაირად ჩაწერილი წესი აღნიშნავს, რომ ანალიზატორის შიგნით ამ ერთი წესისაგან მოხდება სამი სხვადასხვა წესის წარმოქმნა, რომლებიც ერთიმეორესგან მხოლოდ სიტყვათა წყობით განსხვავდებიან, ეს წესებია:

S -> B C D A;
S -> D A B C;
S -> D B C A;

ისინი ემორჩილებიან იმ მარეგულირებელ კონსტრუქციებს, რომლებიც დგანან ორწერტილის შემდეგ. ფორმალიზმში შემოღებული გვაქვს ორი ტიპის მარეგულირებელი კონსტრუქცია. პირველი არის დირექტივა "წინმსწრები" და იგი აღინიშნება '<' ნიშნით, ხოლო მეორე - "უშუალოდ წინმდგომი" და მას ავღნიშნავთ '-' დეფისით. ზემოთმოყვანილი მაგალითი უფრო დაწვრილებით განვიხილოთ. 'D < A' კონსტრუქცია ნიშნავს, რომ სიმბოლო 'D' წინ უსწრებს სიმბოლო `A'-ს, იგი შეიძლება გვხვდებოდეს ნებისმიერ ადგილას ’A’-ს წინ, თუნდაც უშუალოდ მის წინ მდგომი იყოს. ჩვენს შემთხვევაში, შემდეგ კომბინაციებთან გვექნება საქმე:

D A * * * D A *
D * A * * D * A
D * * A * * D A

* -ის ადგილას იგულისხმება B ან C სიმბოლო.

მომდევნო დირექტივა წარმოადგენს "უშუალოდ წინმდგომს": `B - C' რაც ნიშნავს იმას, რომ B სიმბოლო უშუალოდ C სიმბოლოს წინ უნდა გვხვდებოდეს, ანუ C სიმბოლო უშუალოდ უნდა მოსდევდეს B სიმბოლოს. შესაბამისად, ჩვენ მივიღებთ შემდეგ შესაძლო დასაშვებ კომბინაციებს:

B C * *
* B C *
* * B C

* -ის ადგილას იგულისხმება A ან D სიმბოლო.
ერთად აღებული ორივე შეზღუდვა ამ ვარიანტებიდან მოგვცემს იმ სამ ვარიანტს რომლებიც ზემოთ მოვიყვანეთ.

ამ მაგალითმა თვალსაჩინოდ დაგვანახა თუ როგორ გვეხმარება ჩვენს მიერ შემოღებული ფორმალიზმი ენის გრამატიკის კომპაქტურად და თვალსაჩინოდ ჩაწერაში. ცხადია, ჩვენ შეგვიძლია ჩავწეროთ წესი, რომელსაც ორწერტილის შემდეგ არ მოსდევს არც ერთი ურთიერთმდებარეობათა მარეგულირებელი კონსტრუქცია, ამ შემთხვევაში მოხდება ყველა იმ კომბინაციის გათვალისწინება, რომელიც შეიძლება მივიღოთ ამ წესში სიმბოლოთა თავისუფალი გადანაცვლებით. მაგალითად, შემდეგი წესი:

S -> A B C : ;

უშვებს კომბინაციებს:

A B C B C A
A C B C A B
B A C C B A

შესაბამისად, ეს წესი იმუშავებს ყველა შემთხვევაში როცა, ერთიმეორეს მიყოლებით ნებისმიერი მიმდევრობით შეხვდება A B C სიმბოლოები, ამ შემთხვევაში მოხდება მათი ჩანაცვლება S არატერმინალით.

განვიხილოთ ამ ფორმალიზმის კიდევ ერთი თავისებურება. თუკი წესის შიგნით რომელიმე სიმბოლო მეორდება, საჭირო ხდება მათი ერთმანეთისაგან გამოყოფა ინდექსების მეშვეობით. ინდექსი სიმბოლოს გვერდით ეწერება და მისგან წერტილით ('.') გამოიყოფა. მოვიყვანოთ ინდექსირებული სიმბოლოების მაგალითები და მათ მიერ შედგენილი წესი:

A.1 B.2
S -> A A.2 S.2 B.1 B.2;

აქ 'A' განსხვავდება `A.2'-ისგან, `S' განსხვავდება `S.2'-ისგან და `B.1' - `B.2'-ისგან. გაჩუმებით სიმბოლოს ენიჭება 1-ის ტოლი ინდექსი, შესაბამისად `B' და `B.1' ერთიდაიგივე სიმბოლოებს აღნიშნავენ. მაცხენა მხარეში მდგომ არატერმინალს ყოველთვის 1-ის ტოლი ინდექსი აქვს და მას ჩვეულებრივ არ ავღნიშნავთ ცხადი სახით. ინდექსირებული აღნიშვნები გვხმარება როგორც ურთიერთმდებარეობის მარეგულირებელ კონსტრუქციებში, აგრეთვე თვისებათა შეზღუდვებში (იხ. ქვემოთ).

შემდეგ საფუძველს, რომელსაც ემყარება ჩვენი ფორმალიზმი, წარმოადგენს ე.წ. თვისებათა შეზღუდვების მექანიზმი (დაწვრილებითი ინფორმაციისათვის იხ. [9]). ავხსნათ ამ მექნიზმის არსი. დავუბრუნდეთ საწყის მაგალითს: S -> NP VP საკმარისია თუ არა ეს წესი იმისათვის, რომ კორექტულად მოხდეს წინადადების გარჩევა? ბუნებრივი ენის თავისებურობებიდან გამომდინარე აშკარაა, რომ მხოლოდ ამ წესზე დაყრდნობით ვერ მოხერხდება წინადადების სწორი სინტაქსური ანალიზი. მოცემული წესი არის აუცილებელი, მაგრამ სამწუხაროდ არასაკმარისი წინადადების სწორად გარჩევისათვის. ამ წესის მიერ განსაზღვრულ წინადადებათა სიმრავლე გაცილებით უფრო ფართოა, ვიდრე ამას მოითხოვს ბუნებრივი ენა. ყოველივე ამის მიზეზი გახლავთ ის, რომ მოცემული წესი არ ითვალისწინებს ბუნებრივი ენის ისეთ თავისებურობებს როგორიცაა: პირი, რიცხვი, ბრუნვა, ზმნის ფორმა. იგი მხოლოდ მიგვითითებს წინადადების შემადგენელ კომპონენტების ურთიერთგანლაგებაზე, მის სტრუქტურაზე. შედეგად ჩვენ ვიღებთ ისეთ გრამატიკას, რომელშიც დაშვებულია და კორექტულად ითვლება არა მარტო "ბიჭი მირბის" წინადადება, არამედ ამ წინადადების ქართული ენისათვის მიუღებელი ტრანსფორმაციები, მაგალითად: "ბიჭმა მირბის", "ბიჭი მირბიან", "მე მირბის" და ა.შ. რაც ნათლად გვიჩვენებს თუ რამდენად შემწყნარებლურია ეს მარტივი გრამატიკა მსგავსი

არასწორი წინადადებებისადმი და რაოდენ დიდია პროცენტულად იმ წინადადებების რიცხვი, რომლებიც ამ გრამატიკის წყალობით მოხვდნენ დასაშვებ წინადადებათა სიმრავლეში. ზემოთმოყვანილი მაგალითი გვიჩვენებს, თუ რაოდენ გამოუსადეგარია ასეთი მარტივი მიდგომა ბუნებრივ ენასთან მიმართებაში. სრულყოფილი ანალიზის მისაღებად საჭიროა რომ გრამატიკის წესები რამოდენადმე გავართულოდ. დავადოთ მათ გარკვეული შეზღუდვები, რათა გავცხრილოთ და ამოვყაროთ ის უსარგებლო კონსტრუქციები, რომლებიც ხელს უშლიან ქართული (და ზოგადად ბუნებრივი) ენისთვის მისაღებ წინადადებათა სიმრავლის ჩამოყალიბებას. სწორედ ამ მიზანს ემსახურება ჩვენს მიერ ნახსენები თვისებათა შეზღუდვები. მათი საშუალებით შეგვიძლია უფრო გავამკაცროთ გრამატიკა, გავხადოთ იგი მოხერხებული და მოქნილი, რაც საშუალებას მოგვცემს დავხვეწოთ წესები, რათა მათ დიდი სიზუსტით მოიცვან ბუნებრივი ენის გრამატიკა.

თვისებათა შეზღუდვების მექნიზმი ემყარება ჩვენს მიერ უკვე განხილულ თვისებათა სტრუქტურებს. განვიხილოთ რაიმე წესი:

S -> A B C;

წესში შემავალ ნებისმიერ სიმბოლოს (ტერმინალურს ან არატერმინალურს) ჩვენს სისტემაში შეესაბამება თვისება, რომელის მნიშვნელობაც შეიძლება იყოს როგორც მარტივი (ვთქვათ რაიმე სტრიქონი), ასევე რთული (ანუ თვისებათა სტრუქტურა, რაც უფრო ხშირი შემთხვევაა). შესაბამისად S, A, B და C სიმბოლოებს გააჩნიათ ამგვარი თვისებები. ისინი წესის ამოქმედების მომენტში ცარიელია, ან შეიცავენ რაიმე მნიშვნელობას მარჯვენა მხარეში მდგომ იმ სიმბოლოთათვის, რომლებიც მიღებულნი იქნენ წინა ეტაპზე რომელიმე წესის გამოყენებით. ავღნიშნოთ ასევე, რომ მარჯვენა მხარეში გამოყენებული სიმბოლოების შესაბამისი თვისების გამოყენება შეიძლება მხოლოდ წასაკითხად. მარცხენა მხარეში მდგარი ერთადერთი არატერმინალური სიმბოლოსთვის კი მისი შესაბამისი თვისების გამოყენება შესაძლებელია როგორც წასაკითხად ასევე (უფრო ხშირად) ჩასაწერად. ბუნებრივია ისმის კითხვა, თუ როგორ მოვახდინოთ ამ თვისებებით მანიპულირება, მათი შემოწმება, შეცვლა და ა.შ.. ამ მიზნისთვის გამოყენება თვისებათა შეზღუდვები. მათ

ვწერთ წესის ბოლოში, წერტილმძიმის (';') მაგივრად ფიგურულ ფრჩხილებში. მაგალითად:

A -> B { <A> := [name: value] }

ფიგურულ ფრჩხილებში იწერება ლოგიკური გამოსახულება, რომელიც თვისებებზე განსაზღვრული ოპერაციების გამოყენებით არის მიღებული. ვინაიდან თვითოეული ასეთი ოპერაცია აბრუნებს ლოგიკურ მნიშვნელობას, ისინი შეგვიძლია გავაერთიანოთ შემდეგი უმარტივესი ლოგიკური ოპერაციებით:

- '~' - ლოგიკური უარყოფა, ერთადგილიანი (უნარული) ოპერაცია, გვაძლევს მისი არგუმენტის საწინააღმდეგო მნიშვნელობას;
- '&' - ლოგიკური ნამრავლი, "და" ორადგილიანი (ბინარული) ოპერაცია, გვაძლევს ჭეშმარიტ მნიშვნელობას თუკი მისი ორივე ოპერანდი ჭეშმარიტია, წინააღმდეგ შემთხვევაში გვაძლევს მცდარ მნიშვნელობას;
- '|' - ლოგიკური ჯამი, "ან" ორადგილიანი (ბინარული) ოპერაცია, გვაძლევს ჭეშმარიტ მნიშვნელობას თუკი ერთი რომელიმე ოპერანდი მაინც არის ჭეშმარიტი, წინააღმდეგ შემთხვევაში გვაძლევს მცდარ მნიშვნელობას.

ასეთნაირად, მიღებული ლოგიკური გამოსახულება გამოთვლება (შესრულდება) ამ ლოგიკური შეზღუდვის შესაბამისი წესის ნებისმიერი ამოქმედების შემთხვევაში. თუკი ეს ლოგიკური გამოსახულება დააბრუნებს ჭეშმარიტ მნიშვნელობას, სისტემა ჩათვლის, რომ მოცემული გრამატიკული წესის არგუმენტები აკმაყოფილებენ მათზე დადებულ შეზღუდვებს. ამ შემთხვევაში წესი უნდა ჩაითვალოს როგორც დასაშვები და წინადადების გარჩევის პროცესი გაგრძელდეს ჩვეულებრივად. ხოლო თუკი ლოგიკური გამოსახულების მიერ დაბრუნებული იქნა ლოგიკურად მცდარი მნიშვნელობა, სისტემა ჩათვლის, რომ ამ შემთხვევაში ეს წესი არ არის დასაშვები და წინადადების გარჩევა არ გაგრძელდება მოცემულ მომენტში კონკრეტულ წესის გამოყენებით, არამედ განიხილება სხვა დასაშვები წესი, ან თუ ასეთი არ მოიძებნა, მოხდება ანალიზატორის დაბრუნება ერთი ნაბიჯით უკან და გაგრძელდება გარჩევის შემდგომი მცდელობები.

თვისებათა შეზღუდვების ძირითადი მიზანია, რომ გავამკაცროთ გრამატიკა, დავადოთ მას უფრო მეტი შეზღუდვა ვიდრე ამის საშუალებას ჩვეულებრივი კონტექსტისაგან დამოუკიდებელი გრამატიკის წესები იძლევა. ამის შედეგად,

მოხდება უფრო დახვეწილი გრამატიკის ჩამოყალიბება და ენის მიერ დასაშვებ წინადადებების სიმრავლეში აღარ მოხვდება მრავალი არასაჭირო და ბუნებრივი ენისათვის მიუღებელი წინადადება. მეორე მიზანი, რომელსაც ემსახურება თვისებათა შეზღუდვები არის ინფორმაციის გადატანა წესის მარცხენა მხარეში მოთავსებული არატემინალური სიმბოლოს შესაბამის თვისებაში. ეს საშუალებას გვაძლევს გარჩევის მომდევნო ეტაპებზე მოხდეს ამ ინფორმაციის გამოყენება სხვადასხვა შემოწმებებისთვის და დამუშავებისთვის. განვიხილოთ ისევ ძველი მაგალითი, სადაც წინადადების განმსაზღვრავი წესია მოცემული და დავამატოთ მას რამოდენიმე შეზღუდვა:

```
S -> NP VP {
        <NP რიცხვი> == <VP რიცხვი> &
        <NP პირი>    == <VP პირი> &
        <S სუბიექტი> := <NP>
}
```

ამ მაგალითში მოცემულ წესზე დადებულია სამი შეზღუდვა. პირველი მათგანი გვეუბნება, რომ სუბიექტის რიცხვი უნდა ემთხვეოდეს ზმნის რიცხვს (თუკი ეს თვისებები წარმოდგენილია მათში), მეორე იგივე შეზღუდვას ადებს თვისება "პირს". ბოლო შეზღუდვა (სინამდვილეში ეს უფრო მოქმედებაა ვიდრე შეზღუდვა) ახდენს ინფორმაციის (თვისების) გადმოტანას მარცხენა მხარეს მდგომ სიმბოლოში. მარტივად ავხსნათ თუ როგორ იმუშავებს ეს წესი. იგი მოქმედებს როგორც კონტექსტისაგან თავისუფალი გრამატიკის ჩვეულებრივი წესი, აღმოაჩენს სიმბოლოთა დამთხვევას, ანუ გვერდიგვერდ მდგომ NP და VP სიმბოლოებს, ამის შემდგომ შესრულებას დაიწყებს ფიგურულ ფრჩხილებში არსებული გამოსახულების გამოთვლა. ჯერ შემოწმდება უნიფიცირდება თუ არა NP სიმბოლოს ქვეთვისება "რიცხვი" VP სიმბოლოს იგივე ქვეთვისებასთან. თუკი უნიფიცირდება, გამოსახულების გამოთვლა გაგრძელდება, წინააღმდეგ შემთხვევაში წესის დამუშავება მომენტალურად შეწყდება და მოხდება მისი უკუგდება. დავუშვათ პასუხი დადებითია, მაშინ გამოსახულების გამოთვლა გაგრძელდება და ამჯერად იგივე პრინციპით მოხდება ქვეთვისება "პირის" შემოწმება. თუკი უნიფიცირება კვლავაც წარმატებით დასრულდა მოხდება ბოლო, მინიჭების ოპერაციის გამოთვლა (შესრულება), რომელიც ყოველთვის ჭეშმარიტ მნიშვნელობას აბრუნებს და

შესაბამისად წესს უკუდგების საფრთხე უკვე აღარ ემუქრება. შედეგად წესი დაშვებული იქნება, და მის მარცხენა მხარეს მდგომ არატერმინალურ სიმბოლოში გადავა საჭირო ინფორმაცია ქვეთვისება "სუბიექტის" სახით.

მაგალითებიდან ჩვენ ვხედავთ, რომ ზოგიერთი თვისება (თვისებათა სტრუქტურები) მოცემულია პირდაპირ კონსტანტების სახით, ხოლო სხვები ჩაწერილია კუთხოვანი ფრჩხილების მეშვეობით. განვიხილოთ ეს ჩანაწერი უფრო დაწვრილებით. როგორც უკვე ვთქვით, ყოველ სიმბოლოს წესის შიგნით აქვს შესაბამისი თვისება. მაგალითად, სიმბოლო A-ს თვისება შეზღუდვებში შეგვიძლია ჩავწეროთ <A> სახით. ასევე, სამკუთხა ფრჩხილებში შეგვიძლია მივუთითოთ გზა რომელიმე ჩვენთვის საინტერესო ქვეთვისებისათვის. მაგალითად, <A head num> ჩანაწერი აღნიშნავს head-ის თვისება num-ს, ხოლო head თავისთავად A სიმბოლოს თვისებაა. თუკი ასეთი ქვეთვისება უკვე არსებობდა A სიმბოლოს შესაბამის თვისებაში, მაშინ მოხდება მასზე წვდომა და მისი გამოყენება შესაბამის ოპერაციებში. თუკი იგი არ შეიცავს ამ ქვეთვისებას ამ შემთხვევაში ავტომატურად შეიქმნება ყველა ის საშუალედო და საბოლოო თვისება, რომელიც არ გვხვდება ამ სიმბოლოს შესაბამის თვისებაში, მაგრამ მოცემულია კუთხოვან ფრჩხილებში ჩაწერილ გზაში, საბოლოო თვისება კი მიიღებს ცარიელ მნიშვნელობას (None-ს).

ზემოთ ნაჩვენები იყო, თუ როგორ შეიძლება შევცვალოთ მარცხენა მხარეში არსებული სიმბოლოს შესაბამისი თვისება, მაგრამ არ გვითქვამს, თუ საიდან აიღება ტერმინალურ სიმბოლოთა საწყისი თვისებები (თუკი ასეთი მოცემულია). გავიხსენოთ რომ ტერმინალურ სიმბოლოებს არ გააჩნიათ წარმომქმნელი წესები, და ისინი გვხვდებიან პირდაპირ შემავალ ტექსტში ლექსემების სახით. იმისათვის, რომ ჩვენ განვსაზღვროთ ლექსემების შესაბამისი თვისებები ისინი უნდა მივუთითოთ ლექსიკონში ან მიღებულ იქნეს სიტყვის მორფოლოგიური ანალიზის შედეგად. მაგალითად:

```
სახლს [
   cat: N
   ბრუნვა: მიც
   რიცხვი: მხ]
```

ეს ჩანაწერი (იგი მოთავსებულია ლექსიკონის ფაილში) აღნიშნავს ლექსიკურ ერთეულს, სიტყვა "სახლს", ხოლო კვადრატულ ფრჩხილებში კი, რომლებიც მოსდევს

ამ სიტყვას, ჩაწერილია თვისებათა სტრუქტურა. კერძოდ, აღნიშნულია, რომ სიტყვა "სახლს" წარმოადგენს არსებით სახელს, დგას მიცემით ბრუნვაში და არის მხოლობით რიცხვში.

ამის შემდეგ თუკი სადმე შეგვხვდება წესი, რომელშიც მარჯვენა მხარეში მდგომი ერთერთი სიმბოლოა N და ამ წესის ამუშავება მოხდა სიტყვა "სახლს"-ის დამთხვევის შედეგად. ამ სიმბოლოს შესაბამისი თვისება მნიშვნელობად მიიღებს იმ თვისებათა სტრუქტურას, რომელიც მოცემული იყო ლექსიკონში. მაგალითად, მოცემულია შემდეგი წესი:

```
S -> N {
        <S ბრუნვა> := <N ბრუნვა>
}
```

და შესავალი წინადადება ერთი სიტყვისგან შესდგება: "სახლს". მოხდება ამ წინადადების დაშვება, როგორც კორექტულის და S-ის თვისება "ბრუნვა" მნიშვნელობად მიიღებს "მიც"-ს.

ამგვარად, ჩვენ მოკლედ მიმოვიხილეთ თვისებათა შეზღუდვების მექანიზმი, როგორც ამ სისტემაში გამოყენებული ერთერთი ფუნდამენტური მეთოდი. მისი უფრო დეტალური აღწერა, სინტაქსი და სემანტიკა მოცემული იქნება ქვემოთ, როდესაც ვისაუბრებთ გრამატიკის ფაილის ფორმატზე.

განვიხილოთ კიდევ ერთი მოხერხებული საშუალება, რომლის გამოყენებაც შესაძლებელია ამ ახალ ფორმალიზმში. ეს გახლავთ ცვლადები. ყველა ცვლადს გააჩნია სახელი და მის შიგთავსს წარმოადგენს თვისება, რომლის მნიშვნელობაც შეიძლება იყოს როგორც მარტივი ასევე რთული (თვისებათა სტრუქტურა). ცვლადებს აქვთ სახელი და პრეფიქსად დოლარის ნიშანი უდგათ წინ '$'. მაგალითად, შემდგომი აღნიშვნები წარმოადგენენ ცვლადებს: $a, $myvar, $test_123. ცვლადებს შეგვიძლია მივანიჭოთ მნიშვნელობა, მოვახდინოთ მათი ინიციალიზაცია. ყოველივე ეს გრამატიკის ფაილში შემდეგნაირად ჩაიწერება:

```
$a = [f1: v1]
```

მოყვანილი გამოსახულება ნიშნავს, რომ ცვლად a-ს მნიშვნელობად მიენიჭა თვისებათა სტრუქტურა [f1: v1]. ეს ცვლადი შეგვიძლია გამოვიყენოთ როგორც ოპერანდი თვისებათა შეზღუდვებში შემდეგი სახით: <$a> ან თუ გვინდა კონკრეტული გზის მითითება <$a some path>. ცვლადები ამარტივებენ წესების ჩაწერას. მაგალითად, როდესაც რამოდენიმე ადგილას საჭიროა დიდი მოცულობის მქონე თვისებათა სტრუქტურის ჩაწერა, შეგვიძლია ეს თვისებათა სტრუქტურა ავღნიშნოთ რაიმე ცვლადით და შესაბამის ადგილებში გამოვიყენოთ ეს ცვლადი. გარდა ამისა, ცვლადები შეგვიძლია გამოვიყენოთ წესების მუშაობის პროცესში როგორც ინფორმაციის გადაცემის კიდევ ერთი საშუალება ან გამოვიყენოთ დიაგნოსტიკის მიზნით. გრამატიკის ფაილის გარდა, ცვლადები გვაქვს აგრეთვე ლექსიკონის ფაილშიც. ორივე ფაილს შეესაბამება ერთიდაიგივე ცვლადების ცხრილი. ანუ ლექსიკონის ფაილში აღწერილი ცვლადი შეგვიძლია გამოვიყენოთ გრამატიკის ფაილში. ერთადერთ განსხვავებას წარმოადგენს ის, რომ ლექსიკონის ფაილში ცვლადებისათვის საჭირო არ არის დოლარის ნიშნის მითითება. მაგალითად, a = [f1: v1] ასეთი ჩანაწერი ლექსიკონის ფაილში ახდენს a ცვლადის ინიციალიზაციას შესაბამისი თვისებათა სტრუქტურით.

§2. პროგრამის აღწერა

ჩვენს მიერ შემუშავებული ფორმალიზმის პრაქტიკული გამოყენებისათვის და მისი რეალიზაციისათვის საჭიროა პროგრამული უზრუნველყოფა. ეს პროგრამული უზრუნველყოფა გახლავთ მოცემული ნაშრომის შემდგენელი ნაწილი. იგი წარმოადგენს სინტაქსურ ანალიზატორს, რომელიც გამოიყენება ბუნებრივ ენაზე ჩაწერილ წინადადებათა გარჩევისათვის. პროგრამისთვის ძირითად შემავალ ინფორმაციას წარმოადგენს ჩვენს მიერ განხილულ ფორმალიზმში ჩაწერილი გრამატიკის ფაილი, რომელიც გრამატიკული წესების მეშვეობით ასახავს დასაშვებ წინადადებათა სიმრავლეს. პროგრამა საჭიროებს აგრეთვე ლექსიკონის ფაილს, რომელშიც მოცემულია ენის ლექსიკური მარაგი, ანუ მარტივად რომ ვთქვათ ენაში

შემავალი სიტყვები თავისი დამახასიათებელი თვისებებით და მათი მნიშვნელობებით (ისეთებით როგორიცაა: ბრუნვა, რიცხვი, პირი და ა.შ.).

ორივე ფაილის არსებობის შემთხვევაში გამრჩევი მზადაა თავისი ფუნქციების შესასრულებლად. ამისათვის უნდა ავკრიფოთ ბრძანება "parse" და ის წინადადება, რომლის გარჩევაც გვსურს. გამრჩევი მონახავს წინადადების შემადგენელ სიტყვებს ლექსიკონში და ლექსემებად დაყოფილ შემავალ წინადადებას გადასცემს დამუშავების მომდევნო ეტაპს, სადაც გრამატიკული წესების საფუძველზე მოხდება წინადადების ანალიზი, აიგება გარჩევის ხე (ან ხეები, თუკი გარჩევის ვარიანტების რაოდენობა ერთზე მეტია) და მოხდება შედეგების გამოტანა ეკრანზე. თუკი წინადადების გარჩევა ვერ ხერხდება, გამრჩევი ამის შესახებაც შეგვატყობინებს და მოთხოვნის შემთხვევაში გამოგვიტანს მაქსიმალურ "ბუჩქს", რომლის აგებაც მოხერხდა სინტაქსური ანალიზის შედეგად. გარჩევის ხის გარდა, ანალიზატორი ეკრანზე გამოგვიტანს იმ თვისებებს (და მათ მნიშვნელობებს), რომლებიც მიიღეს ხის შემადგენელმა სიმბოლოებმა გარჩევის პროცესში. ეს დამატებითი საშუალებაც გვაძლევს საკმაოდ მნიშვნელოვან ინფორმაციას წინადადების აგებულებაზე და გრამატიკული წესების მუშაობის პროცესზე.

კომპილირებული გამრჩევის პროგრამა წარმოადგენს გამშვებ ფაილს სახელად 'eparser' (eparser.exe - Windows-ში). იგი გახლავთ კონსოლური ტიპის პროგრამა. მუშაობს დიალოგურ, ტექსტურ რეჟიმში. პროგრამის გაშვების შემდეგ კომპიუტერის ეკრანზე ჩნდება შემდეგი შეტყობინება:

```
Welcome to Enhanced Parser 0.1
  For help type: \h

&>
```

დიალოგურ რეჟიმში მუშაობისათვის მიწევევის ნიშანი '&>' მიგვითითებს სისტემის მზადყოფნაზე შეასრულოს ჩვენს მიერ შეტანილი ბრძანებები. დახმარების მისაღებად შეგვიძლია გამოვიყენოთ '\h' ბრძანება. მისი შესრულების შედეგად პროგრამა გამოიტანს სისტემაში არსებული ყველა ბრძანების სიას მოკლე აღწერითურთ:

```
Help:
    `\h': Display Help
    `\q': Quit this program
   `\lc': Clear current lexicon
```

`\ll': Load lexicon from file
`\lf': Find lexical entry
`\li': Information about lexicon
`\gl': Load grammar from file
`\sl': List symbols from current grammar
`\rl': List rules from current grammar
`parse': Parse sentence
`\df': Display feature option
`\vl': Show variable list
`\vc': Clear variable list
`\dp': Parser debugging
`\ms': Set maximum number of displayed parse trees
`\lb': Display the largest parse bush if parsing failed
`\mc': Set match control
`\fc': Set default 'cat' field
`\fl': Set default 'lex' field

პირველ ადგილზე დგას ბრძანება, ორწერტილის შემდეგ კი მოსდევს მოკლე განმარტება იმისა, თუ რა მოქმედებებს ახორციელებს ეს ბრძანება. განვიხილოთ თითოეული ბრძანება ცალცალკე. თვალსაჩინოებისათვის ჩავთვალოთ, რომ მოცემულია ლექსიკონის და გრამატიკის შემდეგი ფაილები.

გრამატიკა: sample.grm

S -> A S.2 B;
S -> C;

ლექსიკონი: sample.lex

a [cat: A]
b [cat: B]
c [cat: C]

\h - ეს ბრძანება როგორც უკვე ვნახეთ გვაძლევს დახმარებას. თუკი ამ ბრძანებას პარამეტრების გარეშე შევასრულებინებთ სისტემას, იგი მოგვცემს სისტემაში არსებულ ყველა ბრძანების სიას. გვაქვს ასევე მეორე ვარიანტი, როდესაც ამ ბრძანებას ეძლევა პარამეტრად ის ბრძანება, რომლის შესახებაც გვინდა დახმარების მიღება. ამ შემთხვევაში გამოვა დახმარება ამ კონკრეტული ბრძანებისათვის. მაგალითად:

&>\h parse
Command `parse': Parse sentence.

Usage: parse word1 [word2, ... , wordN]

დახმარების ტექსტში აგრეთვე აღნიშნულია ის, თუ რა პარამეტრები უნდა გადავცეთ ჩვენთვის საინტერესო ბრძანებას შესრულებაზე გაშვებისას.

\q - პროგრამიდან გამოსვლა.

\lc - მიმდინარე ლექსიკონის გასუფთავება. ამ ბრძანების შედეგად მოხდება უკვე არსებული ლექსიკური ერთეულების წაშლა ანალიზატორის ლექსიკონიდან და საჭირო ხდება მათი თავიდან ჩატვირთვა.

\ll - ლექსიკონის ჩატვირთვა ფაილიდან. ბრძანება მოითხოვს ერთ პარამეტრს, ფაილის სახელს. მისი შესრულების შედეგად მოხდება ლექსიკონის ჩატვირთვა მითითებული ფაილიდან. ცხადია ასეთი ფაილი წინასწარ უნდა იყოს მომზადებული და მასში შესაბამისი ფორმატით უნდა ჩაიწეროს ლექსიკური ერთეულები (ანუ ბუნებრივი ენის სიტყვები). რაც უფრო დიდია ლექსიკონი, მით უფრო ვრცელია იმ წინადადებათა სიმრავლე რომელთა დამუშავებაც შეგვიძლია გამრჩევის მეშვეობით. მაგალითად: \ll sample ან \ll sample.lex (ორივე ვარიანტი თითქმის ანალოგიურია) პირველ შემთხვევაში, სისტემა თუკი მან ვერ აღმოაჩინა sample ფაილი მიმდინარე კატალოგში, ფაილის სახელს ავტომატურად დაუმატებს `.lex' გაფართოებას და ამჯერად უკვე ამ სახელით შეეცდება მოძებნოს იგი.

\lf - სიტყვის მოძებნა ლექსიკონში. ბრძანება მოითხოვს ერთ პარამეტრს, საძიებელ სიტყვას. მისი შესრულებისას სისტემა შეეცდება მოძებნოს ეს სიტყვა მიმდინარე ლექსიკონში. ძიების წარმატებით დასრულების შემთხვევაში, ეს სიტყვა, თავისი ყველა იმ თვისებით, რომლებიც ლექსიკონშია შეტანილი, გამოვა კომპიუტერის ეკრანზე. თუკი ვერ მოხერხდა ასეთი სიტყვის მოძებნა გამოვა შესაბამისი შეტყობინება. მაგალითად,

```
&> \lf a
[cat: A
 lex: a]
```

1 entry(ies) was(were) found.

\li - ინფორმაცია ლექსიკონის შესახებ. ამ ბრძანებას გამოაქვს ინფორმაცია ჩატვირთული ლექსიკონის შესახებ. ეს შეიძლება იყოს სიტყვათა რაოდენობა მიმდინარე ლექსიკონში. მაგალითად:

```
&> \li
Current Lexicon Information:
Word Count: 3
```

\gl - გრამატიკის ჩატვირთვა ფაილიდან. ბრძანება მოითხოვს ერთ პარამეტრს, ფაილის სახელს. მისი შესრულების შედეგად მოხდება გრამატიკის ჩატვირთვა მითითებული ფაილიდან. ამ ფაილში იწერება ის წესები რომლებიც აღწერენ შესაბამის ენას. გრამატიკის ფაილი ცხადია წინასწარ უნდა იქნეს გამზადებული ჩვენს მიერ და მასში წესები უნდა ჩაიწეროს შესაბამის ფორმალიზმში. მაგალითი: \gl sample ან \gl sample.grm პირველ შემთხვევაში, სისტემა თავად დაუმატებს sample ფაილის სახელს '.grm' გაფართოებას თუკი ეს ფაილი ვერ აღმოაჩინა მიმდინარე კატალოგში.

\sl - ენის გრამატიკის შემადგენელი სიმბოლოების სიის გამოტანა. იგულისხმება, რომ გამრჩევში უკვე ჩატვირთულია რაიმე გრამატიკა. ამ ბრძანების შედეგად კომპიუტერის ეკრანზე გამოვა ყველა ის ტერმინალური და არატერმინალური სიმბოლო, რომლებიც ერთხელ მაინც არიან მოხსენიებულნი გრამატიკის შემადგენელ წესებში. ყოველ სიმბოლოს მანქანურ წარმოდგენაში შეესაბამება უნიკალური იდენტიფიკატორი, ნომერი, რომელიც აგრეთვე გამოტანილი იქნება ეკრანზე. მაგალითად:

```
&> \sl
1    = A
2    = B
3    = C
0    = S

4 symbol(s) total.
```

\rl - გრამატიკის შემადგენელი წესების გამოტანა. იგულისხმება, რომ გამრჩევში უკვე ჩატვირთულია რაიმე გრამატიკა. ამ ბრძანების შედეგად კომპიუტერის ეკრანზე

გამოტანილი იქნება ყველა ის წესი, რომლებიც ამ გრამატიკის შემადგენლობაში შედიან. საიმბლოები წესის შიგნით გადანომრილია. ისინი აგრეთვე გადანომრილნი არიან გრამატიკის შიგნით. მაგალითად:

```
&> \rl
1   : Rule (1) 0<1> -> 1<2> 0<3> 2<4>
2   : Rule (2) 0<5> -> 3<6>
2 rule(s) total.
```

parse - წინადადების გარჩევა. ბრძანებას მოსდევს ის წინადადება, რომლის გარჩევაც უნდა სცადოს სინტაქსურმა ანალიზატორმა. წარმატებული გარჩევის შემთხვევაში კომპიუტერის ეკრანზე გამოვა გარჩევის ხე. წინააღმდეგ შემთხვევაში, სისტემისგან მივიღებთ შესაბამის შეტყობინებას. გარჩევის ხის გარდა, ჩვენ აგრეთვე ვიღებთ შედეგად იმ თვისებას, რომელიც შეესაბამება გრამატიკის საწყის სიმბოლოს. ჩვენს შემთხვევაში იგი ცარიელია (შეესაბამება None მნიშვნელობა).

მაგალითი 1:

```
&> parse a a c b b
Parsing: a(A) a(A) c(C) b(B) b(B)
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|--------|--------------------------|
A:2  (a) S:3                    B:4  (b)
       |--------|--------|
       A:5  (a) S:6      B:7  (b)
              |
              C:8  (c)

1: S
(None)
```

მაგალითი 2:

```
&> parse a c b a
Parsing: a(A) c(C) b(B) a(A)
Parsing failed.
```

\df - თვისებათა გამოტანის რეჟიმის გადართვა. ეს ბრძანება მოითხოვს ერთ პარამეტრს, რომლის შესაძლო მნიშვნელობებია: 'none', `root', 'all'. none მიუთითებს, რომ გარჩევის ხის სიმბოლოთა თვისებები არ უნდა იქნეს გამოტანილი კომპიუტერის ეკრანზე. root მიუთითებს, რომ ნაჩვენები უნდა იქნეს მხოლოდ საწყისი სიმბოლოს თვისებები. ხოლო all გარჩევის ხის შემადგენელი ყოველი სიმბოლოს თვისებების გამოტანას აღნიშნავს. განვიხილოთ ყოველი შემთხვევის შესაბამისი მაგალითი.

მაგალითი 1.

```
&> \df none
&> parse a c b
Parsing: a(A) c(C) b(B)
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|--------|--------|
A:2  (a) S:3      B:4  (b)
        |
        C:5  (c)
```

მაგალითი 2.

```
&> \df root
&> parse a c b
Parsing: a(A) c(C) b(B)
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|--------|--------|
A:2  (a) S:3      B:4  (b)
        |
        C:5  (c)

1: S
(None)
```

მაგალითი 3:

```
&> \df all
&> parse a c b
Parsing: a(A) c(C) b(B)
```

```
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|--------|--------|
A:2  (a) S:3      B:4  (b)
        |
        C:5  (c)

1: S
(None)
2: A
[cat: A
 lex: a]
3: S
(None)
4: B
[cat: B
 lex: b]
5: C
[cat: C
 lex: c]
```

\vl - გამოაქვს სისტემაში მიმდინარე მომენტისათვის არსებული ცვლადების სია და მათი შესაბამისი მნიშვნელობები. თვითოეული ცვლადის მნიშვნელობას წარმოადგენს თვისება (მარტივი, ან რთული). მაგალითად:

```
&> \vl
1: v1 = [f: v]
2: v2 = [f: [f: v]]
```

\vc - ცვლადების ცხრილის გასუფთავება. აცარიელებს სისტემაში არსებულ ცვლადების ცხრილს.

\dp - გამრჩევის გამართვის რეჯიმი. ბრძანება მოითხოვს ერთ პარამეტრს რომლის მნიშვნელბა შეიძლება იყოს `on' ან `off'. პირველ შემთხვევაში გამართვის რეჟიმი აქტიურდება. ხოლო მეორე შემთხვევაში ხდება მისი გამორთვა. გაჩუმებით გამრჩევის გამართვის რეჟიმი გამორთულია. რეჟიმის ჩართვის შემთხვევაში, ყოველი წინადადების გარჩევისას, ანალიზატორს გამოაქვს დიაგნოსტიკური ინფორმაცია

მიმდინარე მოქმედებებზე (მაგ: სიმბოლოს სტეკში ჩაგდება, უკან დაბრუნება, წესის ამუშავება და ა.შ.). მაგალითი:

```
&> \dp on
parser debugging is turned on.
&> parse a c b
Parsing: a(A) c(C) b(B)
shifting 1.
reduce using rule 2.
shifting 0<5>.
reduce using rule 1.
new solution found.
return back.
shifting 2.
return back.
shifting 3.
shifting 2.
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|--------|--------|
A:2  (a) S:3      B:4  (b)
       |
       C:5  (c)
```

\ms - ამონახსნთა რაოდენობის შეზღუდვა. მოითხოვს ერთ პარამეტრს, ამონახსნთა მაქსიმალურ რაოდენობას. თუკი ეს პარამეტრი ნულისაგან განსხვავებულია მოხდება ლიმიტის დაყენება და როგორც კი გამრჩევი იპოვის საკმარისი რაოდენობის გარჩევის ხეებს, მოხდება გარჩევის პროცესის დროზე ადრე შეწყვეტა და მომხმარებლისთვის გამოტანილი იქნება მხოლოდ უკვე ნაპოვნი ამონახსნები. თუკი ეს პარამეტრი ნულის ტოლია, ყოველგვარი ლიმიტი გამოსატანი გარჩევის ხეების რაოდენობაზე მოიხსნება.

\lb - წარუმატებელი გარჩევის შემთხვევაში მაქსიმალური ბუჩქის გამოტანა. ეს ბრძანება მოითხოვს ერთ პარამეტრს რომლის მნიშვნელობაც შეიძლება იყოს 'on' ან 'off' და იგი შესაბამისად ააქტიურებს ან გამორთავს მაქსიმალური ბუჩქის გამოტანის რეჟიმს. ეს რეჟიმი თავს იჩენს მხოლოდ წარუმატებელი გარჩევის შემთხვევაში. თუკი ამ დროს იგი ჩართულია, მოხდება იმ მაქსიმალური ბუჩქის გამოტანა, რომლის

აგებაც მოხერხდა გარჩევის მცდელობის პროცესში. მაქსიმალური ბუჩქი არის ისეთი ბუჩქი, რომელიც შეიცავს მისი შემადგენელი ხეების ძირთა მინიმალურ რაოდენობას. ეს ბუჩქი ჩვენ გვაძლევს დამატებით ინფორმაციას გარჩევის პროცესზე, მისი წარუმატებლობის მიზეზებზე და ძირითადად გამოიყენება სადიაგნოსტიკო პროცესისათვის. გაჩუმებით ეს რეჟიმი გამორთულია. მაგალითი:

```
&> \lb on
Largest bush is turned on.
&> parse a c b a
Parsing: a(A) c(C) b(B) a(A)
Parsing failed.
Largest bush:
|-------------------------|
S:1                   A:2  (a)
|--------|--------|
A:3  (a) S:4      B:5  (b)
       |
       C:6  (c)
```

\mc - თვისებათა შეზღუდვების მექანიზმის კონტროლი. ბრძანება მოითხოვს ერთ პარამეტრს, რომლის მნიშვნელობაც შეიძლება იყოს 'on' ან 'off' და შესაბამისად ააქტიურებს ან ბლოკავს თვისებათა შეზღუდვების მექანიზმის მუშაობას. თუკი ეს რეჟიმი ჩართულია (გაჩუმებით იგი ყოველთვის ჩართულია) ხდება წესებთან დაკავშირებული თვისებათა შეზღუდვების (თუკი ასეთები არსებობენ) შემოწმება და მიღებული შედეგის გათვალისწინება გარჩევის პროცესში. თუკი ეს რეჟიმი გამორთულია წინადადების ანალიზი მოხდება მხოლოდ კონტექსტისაგან თავისუფალი წესების საფუძველზე და ნებისმიერი თვისებათა შეზღუდვა იგნორირებული იქნება.

\fc - კატეგორიის ველის დაყენება. გაჩუმებით კატეგორიის ველს, ანუ ველს რომლის მიხედვითაც განისაზღვრება ლექსიკური ერთეულის კუთვნილება გრამატიკის რომელიმე ტერმინალური სიმბოლოსადმი, აქვს `cat' მნიშვნელობა. მოცემული ბრძანების საშუალებით შეგვიძლია შევცვალოთ ამ ველის სახელი თუკი ჩვენი მიზნებისათვის ეს საჭიროებას წარმოადგენს (მაგალითად ქართული სახელის მიცემა

ამ ველისთვის). ბრძანება მოითხოვს ერთ პარამეტრს რომელიც წარმოადგენს ამ ველის ახალ სახელს.

\fl - ლექსიკური ველის დაყენება. მოცემული ბრძანებით ყენდება ლექსიკური მნიშვნელობის აღმნიშვნელი ველის სახელი. ამ ბრძანების ერთადერთ პარამეტრს წარმოადგენს ლექსიკური ველის ახალი სახელი.

ფუნქციონალურად გამრჩევის პროგრამა შეგვიძლია პირობითად შემდეგ სამ ნაწილად დავყოთ:

1.გრამატიკის და ლექსიკონის ფაილების დამამუშავებელი ნაწილი
2.ანალიზატორის ბირთვი. კონტექსტისაგან თავისუფალის გრამატიკის ქვემოდან ზემოთ გარჩევის სტანდარტული მეთოდი, ამას დამატებული თვისებათა შეზღუდვების მექანიზმი.
3.მომხმარებლის ინტერფეისი.

სქემატურად ეს ნაწილები შემდეგნაირ კავშირებში არიან ერთმანეთთან:

{ მომხმარებელი } <-> { 3 } <-> { 2 } <-> { 1 } <-> { სპეციალისტი }

განვიხილოთ თვითოეული ნაწილი ცალცალკე.

1.გრამატიკის და ლექსიკონის ფაილების დამამუშავებელი ნაწილი წარმოადგენს პროგრამულ მოდულებს, რომელთა მეშვეობითაც ხორციელდება საკმარისად რთულ ფორმალიზმში ჩაწერილი გრამატიკის და ლექსიკონის ფაილის სინტაქსური ანალიზი, გარჩევა, დამუშავება და მისი მანქანისათვის მოსახერხებელ შინაგან ფორმაში გადაყვანა. მაგალითისათვის ავიღოთ ნებისმიერი გრამატიკული წესი, რომელიც ჩაწერილია ჩვენს მიერ შემოღებულ ფორმალიზმში. თუკი არ მოხდება ამ წესის გადაყვანა მანქანისათვის მისაღებ ფორმაში და მასში არსებული ყველა სიმბოლო, სტრიქონული სახით მოცემული, არ გადავა შესაბამის რიცხვით იდენტიფიკატორებში, არ მოხდება წესის მარჯვენა და მარცხენა მხარის მოთავსება შესაბამის სტრუქტურებში (მაგ. სიებში), თვისებათა შეზღუდვების გარდაქმნა ხეების

სახით ჩაწერილ ლოგიკურ გამოსახულებად, რომელიც დინამიურად შესრულდება ნებისმიერ დროს, ასეთ შემთხვევაში ჩვენთვის სრულიად უსარგებლო იქნება ამ ფაილში მოცემული ტექსტი, რაოდენ კარგ და მოხერხებულ ფორმალიზმშიც არ უნდა იქნეს იგი ჩაწერილი. სწორედ ამ მიზანს ემსახურება პროგრამული უზრუნველყოფის ეს ნაწილი. იგი ნალიზს უკეთებს LALR(1) ტიპის გრამატიკით მოცემულ ფორმალიზმს, და საშუალებას გვაძლევს ჩვენთვის სასურველი წესით დავამუშავოთ მისი სინტაქსური კონსტრუქციები. აღსანიშნავია, რომ ამ მიზნის მიღწევაში დაგვეხმარა Lex და Yacc ინსტრუმენტები, რომლებიც ამარტივებენ სხვადასხვა ფორმალიზმებისათვის ანალიზატორების გაკეთებას. მათი მეშვეობით საკმაოდ რთული დაპროგრამების ენების კომპილატორების გაკეთებაც კი მნიშვნელოვნად გამარტივებულია.

2. გამრჩევის ბირთვი. ძირითადი, აქტიური ლოგიკა სწორედ პროგრამის ამ ნაწილშია თავმოყრილი. გამრჩევის ბირთვს ევალება მოახდინოს კონტექსტისაგან თავისუფალ გრამატიკაში ჩაწერილი ფრაზის სინტაქსური ანალიზი და გზადაგზა შეამოწმოს აკმაყოფილებს თუ არა თითოეული წესი მასში არსებულ შეზღუდვებს. კონტექსტისაგან თავისუფალი გრამატიკის გარჩევისათვის გამოიყენება სტანდარტული ქვემოდან ზემოთ გარჩევის ალგორითმი (იხ. [8]). მოკლედ ავღწეროთ ამ ალგორითმის მუშაობის პრინციპი. გამრჩევი სათითაოდ განიხილავს სიმბოლოებს ერთიმეორეს მიყოლებით და ათავსებს მათ სტეკში ვიდრე სტეკის თავში მოთავსებული სიმბოლოთა მიმდევრობა არ დაემთხვევა რომელიმე წესის მარჯვენა მხარეში არსებულ სიმბოლოთა მიმდევრობას. დამთხვევის შემთხვევაში იგი ანაცვლებს სტეკში ამ სიმბოლოთა მიმდევრობას წესის მარცხენა მხარეში მდგარ არატერმინალურ სიმბოლოთი. თუ საბოლოო ჯამში სტეკში მივიღეთ ერთადერთი სიმბოლო და იგი დაემთხვევა საწყის სიმბოლოს, ანალიზატორი ჩათვლის რომ მიღებულია გარჩევა და ამონახსნების სიას უმატებს იმ გარჩევის ხეს, რომლის აგებაც მოხდა გარჩევის პროცესში. თუკი ეს ვერ მოხერხდა და გამრჩევი ვეღარ ახერხებს წინ წასვლას, მოხდება ერთი ნაბიჯით უკან დაბრუნება და შემდგომი დასაშვები წესის ძიება. უკან დაბრუნება ხდება იმ შემთხვევაშიც თუ გამრჩევმა მოახერხა ამონახსნის პოვნა, რათა მოხდეს ყველა ვარიანტის გადასინჯვა და მოიძებნოს სხვა ამონახსნებიც. გარჩევის პროცესის პარალელურად, ყოველი ამოქმედებული წესისათვის ხდება თვისებათა შეზღუდვების შემოწმება. ეს ხდება დინამიურად, ინტერპრეტირების

რეჟიმში. თუკი თვისებათა შეზღუდვებისგან შემდგარი გამოსახულება დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას, მაშინ წესი განხილული იქნება, თუ არა და მოხდება მისი უარყოფა გარჩევის ამ ეტაპზე.

3. მომხმარებლის ინტერფეისი. გამრჩევის პროგრამას აქვს ტექსტური ინტერფეისი. იგი მუშაობს დიალოგურ რეჟიმში. რაც მდგომარეობს იმაში, რომ ჩვენ მიმდევრობით ვაწვდით მას ბრძანებებს, გამრჩევი მათ ამუშავებს და გვიბრუნებს მიღებულ შედეგს. ტექსტურ რეჟიმში ხდება აგრეთვე გარჩევის ხის აგება. აქ საჭიროა აღინიშნოს, თუ რატომ მოხდა მომხმარებლის ინტერფეისის შემთხვევაში არჩევანის გაკეთება ტექსტურ რეჟიმზე. უპირველეს ყოვლისა იმიტომ, რომ იგი არის უფრო უნივერსალური და ადვილად გადატანადი სხვადასხვა პლატფორმებზე. ვინაიდან დაპროგრამების ენა C++-ის სტანდარტით არ არის გათვალისწინებული გრაფიკულ ბიბლიოთეკებთან მუშაობა, საჭირო ხდებოდა სხვა მწარმოებლების მიერ შემოთავაზებული ბიბლიოთეკების გამოყენება, რომლებიც როგორც წესი არ არიან ერთმანეთთან თავსებადი და ყველა პლატფორმისათვის არ გამოდგებიან.

პროგრამული უზრუველყოფა დაწერილია დაპროგრამების ენა C++-ზე ANSI სტანდარტით და იყენებს მხოლოდ სტანდარტულ ბიბლიოთეკებს, ისეთებს როგორიცაა STL (Standart Template Library) სტანდარტული ბიბლიოთეკა, რომელშიც რეალიზებულია მონაცემთა მრავალფეროვანი სტრუქტურები (სტეკები, ვექტორები, სიები, ასოციატიური მასივები) კლასების სახით. იგი ძირითადად ორიენტირებულია UNIX და Microsoft Windows-ის ტიპის ოპერაციული სისტემებისთვის, თუმცა შესაძლებელია მისი გადატანა ნებისმიერ აპარატურულ თუ პროგრამულ პლატფორმაზე, სადაც მოიძებნება სტანდარტული C++-ის თანამედროვე კომპილატორი (UNIX-ისთვის GCC - Gnu Project Compiler, Windows-თვის VC++ - Microsoft Visual Studio).

§3. მაგალითები

განვიხილოთ რამოდენიმე მარტივი მაგალითი, რომლებიც ნათლად გვაჩვენებენ გამრჩევის მუშაობის პრინციპს, მის თავისებურობებს და შესაძლებლობას მოგვცემს გამოვიყენოთ იგი პრაქტიკული ამოცანების გადასაჭრელად. მაგალითები მოცემულია სირთულის მიხედვით დალაგებული მიმდევრობით. თავიდან განვიხილავთ ელემენტარულ კონტექსტისაგან თავისუფალი გრამატიკის რეალიზაციას. შემდგომ უფრო გავართულებთ მას და დავამატებთ გარკვეულ შეზღუდვებს.

მაგალითი 1.

მოცემული გვაქვს შემდეგი კონტექსტისაგან თავისუფალი გრამატიკა:

S -> A B;
A -> A a;
A -> b B;
B -> a;
B -> S b;

(მაგალითი აღებულია ა. აჰოს და ჯ. ულმანის წიგნიდან [8])

ჩავწეროთ ეს გრამატიკა ფაილში ex1.grm .

შევქმნათ კიდევ ერთი ფაილი ლექსიკონისათვის - ex1.lex და მასში შევიტანოთ შემდეგი ლექსემები:

a [cat: a]
b [cat: b]

განვიხილოთ დაწვრილებით ეს ჩანაწერები. a [cat: a] მიუთითებს იმაზე, რომ ლექსემა a-ს გააჩნია თვისება cat და მისი მნიშვნელობა უდრის a-ს. ეს ინფორმაცია ესაჭიროება გამრჩევას, რათა მან ყოველ ლექსემას შეუსაბამოს ის ტერმინალური სიმბოლო, რომლითაც იგი წარმოდგენილია გრამატიკაში. ჩვენს შემთხვევაში ლექსემის სახელი და მისი კატეგორია ერთმანეთს ემთხვევა. სტანდარტულად კატეგორიის განმსაზღვრელ თვისებას ეწოდება: cat. მაგრამ ჩვენ შეგვიძლია შევცვალოთ ეს სახელი, თუ გამოვიყენებთ გამრჩევის ბრძანებას `\fc'.

ორივე ფაილის გამზადების შემდეგ, გავუშვათ გამრჩევი და მივცეთ მას ბრძანება:

\ll ex1

ამ ბრძანებით მოხდება სისტემაში ლექსიკონის ჩატვირთვა. იმისათვის, რომ დავრწმუნდეთ ლექსიკონის წარმატებით ჩატვირთვაში შეგვიძლია გამოვიყენოთ '\li' და '\lf' ბრძანებები:

```
&> \li
Current Lexicon Information:
Word Count: 2

&> \lf a
[cat: a
 lex: a]

1 entry(ies) was(were) found.
```

პირველ შემთხვევაში, სისტემამ გამოგვიტანა ინფორმაცია მასში არსებულ ლექსიკურ ერთეულთა რაოდენობაზე (სულ 2 ლექსიკური ერთეული, რომლებიც ჩვენ ლექსკონის ფაილში განვსაზღვრეთ: a და b). მეორე ბრძანება კი მოძებნის კონკრეტულ სიტყვას ლექსიკონში. ამ შემთხევაში a-ს, და გამოგვიტანს მის შესაბამის თვისებებს.
ახლა, ჩავტვირთოთ გრამატიკის ფაილი შემდეგი ბრძანებით:

\gl ex1

გრამატიკის ფაილში შეცდომების არარსებობის შემთხვევაში ბრძანება ჩვეულებრივ შესრულდება. წინააღმდეგ შემთხვევაში, პროგრამა გამოიტანს შეტყობინებას შეცდომის აღმოჩენის შესახებ. იმისათვის, რომ გავიგოთ წარმატებით ჩაიტვირთა თუ არა გრამატიკის ფაილი შეგვიძლია გამოვიყენოთ '\sl' და ` \rl' ბრძანებები:

```
&> \sl
1   = A
2   = B
0   = S
3   = a
4   = b

5 symbol(s) total.
```

```
&> \rl
1   : Rule (1) 0<1> -> 1<2> 2<3>
2   : Rule (2) 1<4> -> 1<5> 3<6>
3   : Rule (3) 1<7> -> 4<8> 2<9>
4   : Rule (4) 2<10> -> 3<11>
5   : Rule (5) 2<12> -> 0<13> 4<14>
5 rule(s) total.
```

პირველ შემთხვევაში ეკრანზე გამოვა გრამატიკაში მოცემულ სიმბოლოთა სია. მეორე შემთხვევაში კი გრამატიკის წესები, ჩაწერილნი გამრჩევის შიდა მანქანურ წარმოდგენაში. მას შემდეგ, რაც ჩატვირთება ორივე ფაილი (ლექსიკონის და გრამატიკის), უკვე შეგვიძლია ჩავატაროთ ფრაზის სინტაქსური ანალიზი. ამისათვის, გამოვიყენოთ გამრჩევს შემდეგი ბრძანება:

```
&> parse b a a b a a b
Parsing: b(b) a(a) a(a) b(b) a(a) a(a) b(b)
1 solution(s) was(were) found.
Parse Tree 1:
|
S:1
|-------------------------|
A:2                   B:3
|-----------------|       |-------------------------|
A:4             a:5  (a) S:6                   b:7  (b)
|--------|              |-----------------|
b:8  (b) B:9             A:10            B:11
        |             |--------|       |
        a:12 (a)        b:13 (b) B:14     a:15 (a)
                                |
                                a:16 (a)

1: S
(None)
```

შედეგად ჩვენ მივიღებთ გარჩევის ხეს და გრამატიკის საწყისი სიმბოლოს (S) თვისებას (ამ შემთხვევაში იგი ცარიელია). ჩვენს მიერ მოცემული ფრაზის გარჩევა წარმატებით დასრულდა. თუკი გამრჩევს მივაწვდით ისეთ ფრაზას, რომელიც მოცემული გრამატიკის ფარგლებში არ მიეკუთვნება ამ გრამატიკით განსაზღვრულ ენას, გამრჩევი გამოიტანს შესაბამის შეტყობინებას:

```
&> parse a b b a
Parsing: a(a) b(b) b(b) a(a)
Parsing failed.
```

რაც ნიშნავს, რომ ჩვენს მიერ შეტანილი ფრაზის გარჩევა ვერ მოხერხდა. პროგრამასთან მუშაობის დამთავრების შემდეგ ავკრიფოთ სისტემიდან გამოსვლის ბრძანება '\q':

```
&> \q
Quit.
```

რის შემდეგაც გამრჩევის პროგრამა შეწყვეტს შესრულებას და მართვას დაუბრუნებს ოპერაციულ სისტემის გარსს, საიდანაც მოხდა მისი გაშვება.

მაგალითი 2.

განვიხილოთ ბუნებრივი ენის წინადადების გამარტივებული წესი:

წინადადება -> არსებითი_სახელი ზმნა;

მოცემული წესით განსაზღვრული გრამატიკა აღწერს წინადადებას, როგორც არსებითი სახელისა და ზმნის მიმდევრობას. შევნიშნოთ რომ გამრჩევის პროგრამა მუშაობს როგორც ინგლისურ ასევე ქართულ იდენტიფიკატორებთან. შესაბამისად გრამატიკის და ლექსიკონის ფაილი შეგვიძლია მთლიანად ქართულ ენაზე დავწეროთ. განვიხილოთ შემდეგი ლექსიკონი:

```
ას = [კატ: არსებითი_სახელი]
ზ = [კატ: ზმნა]
ბიჭი [(ას) რიცხვი: მხ]
ბიჭები [(ას) რიცხვი: მრ]
მირბის[(ზ) რიცხვი: მხ]
მირბიან [(ზ) რიცხვი: მრ]
```

აქ ვამჩნევთ რამოდენიმე სიახლეს. ჩაწერების შესამოკლებლად შემოვიტანეთ ცვლადები. ჩანაწერი: ას = [კატ: არსებითი_სახელი] ნიშნავს რომ განვსაზღვრავთ ცვლადს "ას" როგორც თვისებათა სტრუქტურას, რომელსაც აქვს ერთი ველი სახელად "კატ" და მისი მნიშვნელობაა: "არსებითი_სახელი". შემდგომში ეს ცვლადი შეგვიძლია გამოვიყენოთ ლექსიკონის შემადგენელი სიტყვების ჩაწერებში. თვისებათა

კონსტრუქტორი ('[' და ']' ფრჩხილებს შორის მოთავსებული გამოსახულება) შესაძლებელია შეიცავდეს ინიციალიზაციის ნაწილს. იგი აღიწერება თვისებათა კონსტრუქტორის დასაწყისში, მრგვალ ფრჩხილებში ჩამოთვლილი ცვლადების სახით. მაგ: a [(v1, v2,…) f1: val1 f2: val2] ინიციალიზაციის ნაწილში მოხდება ჩამოთვლილი ცვლადების (ამ შემთხვევაში: v1, v2, ...) მნიშვნელობების გაერთიანება, შეჯამება და ისინი დაემატებიან მთლიანად თვისებათა სტრუქტურას. ჩვენ შეგვეძლო მოცემული ლექსიკონის ფაილი შემდეგი სახით ჩაგვეწერა:

```
ბიჭი [კატ: არსებითი_სახელი რიცხვი: მხ]
ბიჭები [კატ: არსებითი_სახელი რიცხვი: მრ]
მირბის [კატ: ზმნა რიცხვი: მხ]
მირბიან [კატ: ზმნა რიცხვი: მრ]
```

ან კიდევ:

```
ას = [კატ: არსებითი_სახელი]
ზ = [კატ: ზმნა]
მხ = [რიცხვი: მხ]
მრ = [რიცხვი: მრ]
ბიჭი [(ას, მხ)]
ბიჭები [(ას, მრ)]
მირბის[(ზ, მხ)]
მირბიან [(ზ, მრ)]
```

ყველა ეს ჩანაწერი ერთმანეთის ეკვივალენტურია. თუმცა ბოლო ვარიანტი აშკარად ჯობნის დანარჩენებს კომპაქტურობით.
მოცემული ლექსიკონის და გრამატიკის ფაილებს დავარქვათ შესაბამისად ex2.lex და ex2.grm სახელები. ჩავტვირთოთ ისინი სისტემაში შემდეგი ბრძანებებით:

```
&> \fc კატ
&> \fl ლექს
&>\ll ex2
&>\gl ex2
```

პირველი ორი ბრძანება ახალ მნიშვნელობებს აძლევს სისტემაში არსებულ კატეგორიის და ლექსიკური ერთეულის აღმნიშვნელ სახელებს (გაჩუმებით მათ მნიშვნელობებს cat და lex წარმოადგენენ). მივცეთ გამრჩევს შემდეგი ბრძანება:

```
&> parse ბიჭი მირბის
```

```
Parsing: ბიჭი(არსებითი_სახელი) მირბის(ზმნა)
Parse Tree 1:
|
წინადადება:1
|------------------------|
არსებითი_სახელი:2  (ბიჭი) ზმნა:3  (მირბის)

1: წინადადება
(None)
```

როგორც ვხედავთ გამრჩევმა წარმატებით გაართვა თავი დასმულ ამოცანას. ახლა ვცადოთ ასეთი წინადადების გარჩევა: "ბიჭები მირბის". მივცეთ შესაბამისი ბრძანება და ვნახოთ თუ რა შედეგს გამოგვიტანს სინტაქსური ანალიზატორი:

```
&> parse ბიჭები მირბის
Parsing: ბიჭები(არსებითი_სახელი) მირბის(ზმნა)
1 solution(s) was(were) found.
Parse Tree 1:
|
წინადადება:1
|--------------------------|
არსებითი_სახელი:2  (ბიჭები) ზმნა:3  (მირბის)

1: წინადადება
(None)
```

მოცემული წინადადება ამ მარტივი გრამატიკისათვის დასაშვები აღმოჩნდა, თუმცა ბუნებრივი ენისათვის იგი არ არის მისაღები. აქედან ჩანს, რომ საჭიროა ამ გრამატიკის სრულყოფა, რათა გამრჩევმა შესძლოს სუბიექტის და პრედიკატის რიცხვში შეთანხმება. სწორედ ამ მიზანს ემსახურება ჩვენს მიერ უკვე ნახსენები თვისებათა შეზღუდვები. ლექსიკონში სიტყვების თვისება "რიცხვი" მოცემულია, რაც იმას ნიშნავს, რომ სინტაქსურ ანალიზატორს გარჩევის ეტაპზე ექნება ის ინფორმაცია, რომელიც საჭიროა ჩვენს მიერ დასმული ამოცანის გადასაწყვეტად. გასაკეთებელი არის ის, რომ გრამატიკის წესში მივუთითოთ ზმნის და არსებითი სახელის რიცხვში შეთანხმების პირობა. იგი ასეთნაირად შეგვიძლია ჩავწეროთ:

```
წინადადება -> არსებითი_სახელი ზმნა {
        <არსებითი_სახელი რიცხვი> == <ზმნა რიცხვი>
}
```

წერტილმძიმის მაგივრად ამ წესში გვაქვს ფიგურულ ფრჩხილებში მოთავსებული გამოსახულება - შეზღუდვა. კონკრეტულად იგი მიუთითებს იმას, რომ სიმბოლო "არსებითი_სახელი"-ს და სიმბოლო "ზმნა"-ს თვისება რიცხვი ერთმანეთთან უნდა უნიფიცირდებოდნენ. რაც იმას ნიშნავს, რომ თუკი ორივე სიმბოლოს აქვს ეს თვისება, მაშინ მათი მნიშვნელობები ერთმანეთის ტოლი უნდა იყოს. თუკი ახლა გამრჩევს მივაწვდით იგივე წინადადებას: "ბიჭები მირბის". ის უკვე აღარ მიიჩნევს მას კორექტულად. სამაგიეროდ "ბიჭები მირბიან" წინადადებას დაუშვებს როგორც სწორს:

```
&> parse ბიჭები მირბის
Parsing: ბიჭები(არსებითი_სახელი) მირბის(ზმნა)
Parsing failed.

&> parse ბიჭები მირბიან
Parsing: ბიჭები(არსებითი_სახელი) მირბიან(ზმნა)
1 solution(s) was(were) found.
Parse Tree 1:
|
წინადადება:1
|--------------------------|
არსებითი_სახელი:2  (ბიჭები) ზმნა:3  (მირბიან)

1: წინადადება
(None)
```

როგორც ვხედავთ თვისებათა შეზღუდვების საშუალებით მოხერხდა განხილული პრობლემის გადაწყვეტა. ანალოგიურად შეგვიძლია გავაუმჯობესოთ გრამატიკა დავუმატებთ რა მას მსგავს შეთანხმებებს ბრუნვაში, პირში და ა.შ.. შეზღუდვები შეგვიძლია გავაერთიანოთ ერთმანეთთან ლოგიკური ოპერაციების მეშვეობით. თუკი ეს ლოგიკური გამოსახულება გამოთვლის შედეგად იძლევა ლოგიკურად ჭეშმარიტ მნიშვნელობას, მაშინ წესი ჩაითვლება როგორც დასაშვები, წინააღმდეგ შემთხვევაში მოხდება წესის უკუგდება მიმდინარე ეტაპზე. გარდა შეზღუდვების დადებისა, თვისებათა შეზღუდვების მექანიზმს აქვს კიდევ სხვა დანიშნულება. კერძოდ, თვისებათა გადატანა გარჩევის უფრო მაღალ დონეზე. ჩვენს შემთხვევაში, შეგვიძლია გრამატიკის საწყის სიმბოლოს (წინადადებას) გარჩევის პროცესში მივანიჭოთ რაიმე თვისება. მაგალითად:

```
წინადადება -> არსებითი_სახელი ზმნა {
        <არსებითი_სახელი რიცხვი> == <ზმნა რიცხვი> &
        <წინადადება სუბიექტი რიცხვი> :=  <არსებითი_სახელი რიცხვი> &
        <წინადადება პრედიკატი რიცხვი> := <ზმნა რიცხვი>
}
```

თუკი სისტემაში ჩავტვირთავთ ამ გრამატიკას და გარჩევაზე გავუშვებთ წინადადებას ”ბიჭები მირბიან” მივიღებთ შემდეგ შედეგს:

```
&> parse ბიჭები მირბიან
Parsing: ბიჭები(არსებითი_სახელი) მირბიან(ზმნა)
1 solution(s) was(were) found.
Parse Tree 1:
|
წინადადება:1
|--------------------------|
არსებითი_სახელი:2  (ბიჭები) ზმნა:3  (მირბიან)

1: წინადადება
[პრედიკატი: [რიცხვი: მრ]
 სუბიექტი: [რიცხვი: მრ]]
```

როგორც ვხედავთ, გრამატიკის საწყის სიმბოლო "წინადადება"-ს დაემატა ორი ახალი თვისება. ეს თვისებები შეივსნენ პრედიკატში და სუბიექტში არსებული ინფორმაციით რიცხვის შესახებ. თვისებათა გადატანა შესაძლებელია მინიჭების ოპერატორის (':= ') გამოყენებით. აგრეთვე უნიფიკაციის ოპერატორის ('<==') გამოყენებით. ეს ოპერატორი ახდენს თვისებათა უნიფიკაციას და მიღებული შედეგი გადააქვს მარცხენა მხარეში.

§4. ლექსიკონის ფაილის სტრუქტურა

გამრჩევის ერთ-ერთ მნიშვნელოვან ფაილს წარმოადგენს ლექსიკონის ფაილი. მასში მოცემულია საწყისი ინფორმაცია ლექსიკური ერთეულების შესახებ. ბუნებრივი ენისათვის ისინი წარმოადგენენ ენაში არსებულ სალაპარაკო სიტყვებს. ლექსიკური ერთეულების გარდა ლექსიკონში აგრეთვე მოცემულია მათი დამახასიათებელი თვისებები. ბუნებრივი ენისათვის ეს თვისებებია: პირი, რიცხვი,

ბრუნვა და ა.შ.. ლექსიკონის ფაილის სტრუქტურა საკმაოდ მარტივია, იგი წარმოადგენს ლექსიკური ერთეულის ან ცვლადის განსაზღვრებების მიმდევრობას. ცვლადები ძირითადად შემოკლებებისთვის გამოიყენება. განვიხილოთ ლექსიკური ერთეულის განსაზღვრის მაგალითი:

```
სახლი [
  კატ: ას
  ბრუნვა: სახ
  რიცხვი: მხ
  პირი: 3
]
```

ეს ჩანაწერი განსაზღვრავს ლექსიკურ ერთეულს "სახლი", რომელსაც მოსდევს კვადრატულ ფრჩხილებში ჩაწერილი თვისებათა სტრუქტურა. ამ თვისებათა სტრუქტურაში მოთავსებულია სიტყვის ძირითადი თვისებები, რომლებიც გამოიყენებიან გრამატიკული ანალიზის დროს. ასეთი სახით ჩაწერილ ლექსიკურ ერთეულთა მიმდევრობა ადგენს მთლიანად ლექსიკონის ფაილს. მეორე ტიპის ჩანაწერები, რომლებიც შეიძლება გვხვდებოდნენ ლექსიკონის ფაილში გახლავთ ცვლადების განსაზღვრებები. მათი ჩაწერის ფორმა შემდეგია:

```
სახ = [ბრუნვა: სახ]
```

ამ ჩანაწერით ჩვენ განვსაზღვრავთ ცვლადს სახელად "სახ" (სახელობითი), რომლის მნიშვნელობას წარმოადგენს თვისებათა სტრუქტურა და ამ თვისებათა სტრუქტურის ერთადერთი ველი არის "ბრუნვა", რომელსაც მნიშვნელობად აქვს "სახ". შემდგომ ჩვენ უკვე შეგვიძლია ნებისმიერ ადგილას, სადაც საჭიროა იმის მითითება, რომ სიტყვაფორმა დგას სახელობით ბრუნვაში, გამოვიყენოთ ეს ცვლადი:

```
სახლი [ (სახ)
  კატ: ას
  რიცხვი: მხ
  პირი: 3
]
```

ცვლადის მითითება ხდება თვისებათა კონსტრუქტორის დასაწყისში და იწერება მრგვალ ფრჩხილებში. მრგვალ ფრჩხილებში მოთავსებულ გამოსახულებას ეწოდება კონსტრუქტორის საინიციალიზაციო ნაწილი. თუკი საჭიროა ინიციალიზაციისათვის

რამოდენიმე ცვლადის გამოყენება, ისინი უნდა ჩამოვთვალოთ და ერთიმეორისაგან მძიმით გამოვყოთ. მაგალითად:

სახლი [(ას, სახ, მხ)]

აქ იგულისხმება, რომ განსაზღვრული გვაქვს შემდეგი ცვლადები:

ას = [კატ: არსებითი_სახელი]
სახ = [ბრუნვა: სახ]
მხ = [რიცხვი: მხ]

ეს მაგალითი გვიჩვენებს, თუ როგორ ამოკლებს ლექსიკური ერთეულების ჩაწერას ცვლადების შემოღება. პრაქტიკულად, სულ რამოდენიმე, ყველაზე გავრცელებული შემოკლების (ცვლადის) შემოღებით, ლექსიკური ერთეულების უმრავლესობა შეგვიძლია მხოლოდ ამ შემოკლებების მეშვეობით ჩავწეროთ.

ლექსიკონის ფაილში შესაძლებელია გვქონდეს ერთიდაიგივე ლექსიკური მნიშვნელობის მქონე სიტყვები. ანუ სიტყვები რომლებიც ერთნაირად იწერება და მხოლოდ თვისებებით განსხვავდებიან. ბუნებრივ ენაში ეს გახლავთ ომონიმური შემთხვევები. მაგალითად: "ვაშლი" როგორც არსებითი სახელი (მაგ: წითელი ვაშლი) და "ვაშლი" როგოც ზმნა (მაგ: მე ვაშლი მას). ჩვენი ლექსიკონის ფაილში ისინი შემდეგნაირად ჩაიწერებიან:

```
ვაშლი [
  კატ: არსებითი_სახელი
  ბრუნვა: სახ
  რიცხვი: მხ
  პირი: 3
]

ვაშლი [
  კატ: ზმნა
  პირი: 1
  სერია: 1
  რიცხვი:მხ
  პირდაპირი_ობიექტის_პირი: 3
  ირიბი_ობიექტის_პირი: 3
  ირიბი_ობიექტის_რიცხვი: მხ
  პირიანობა: 3
  დრო: აწმყო
]
```

თუკი სისტემაში ჩავტვირთავთ ამგვარად განსაზღვრულ ლექსიკონს და გამრჩევს მივცემთ ბრძანებას '\lf ვაშლი', შედეგად მივიღებთ ორივე ამ სიტყვას - თითოეულს თავისი თვისებებით. როდესაც გამრჩევს ვაძლევთ ისეთ წინადადებას, რომელიც რამოდენიმე სხვადასხვა ალტერნატივას შეიცავს იგი განიხილავს ყველა შესაძლო ვარიანტს და ყოველი მათგანისათვის, თუკი მოხერხდება გარჩევის ხის აგება, გამოიტანს ამონახსნს კომპიუტერის ეკრანზე.

დანართში მოყვანილია ლექსიკონის ფაილის ფორმალიზმის მკაცრი აღწერა. მასში დაწვრილებითაა აღწერილი ის სინტაქსური კონსტრუქციები, რომლებიც შეადგენენ ლექსიკონის ფაილის სტრუქტურას. ეს ჩაწერები მოცემულია Bison (დაწვრილებით იხ. [11]) პროგრამის შემავალი ფაილის ფორმალიზმში. Bison წარმოადგენს მეტად მოხერხებულ პროგრამას, რომელიც საშუალებს გვაძლევს მარტივად შევადგინოთ სხვადასხვა ტიპის ფორმალიზმების გამრჩევები, მაგ: კონფიგურაციის ფაილების გამრჩევი, დაპროგრამების ენის კომპილატორი და ა.შ.. ჩვენ შემთხვევაში ასეთ ფორმალიზმს წარმოადგენს ლექსიკონის ფაილის ფორმალიზმი (აგრეთვე გრამატიკის ფორმალიზმიც, რომელსაც შემდგომ განივიხილავთ).

§5. გრამატიკის ფაილის სტრუქტურა

გამრჩევის უმთავრეს შემავალ ფაილს წარმოადგენს გრამატიკის ფაილი. მასში იწერება იმ წესთა ერთობლიობა, რომლებიც განსაზღვრავენ ენას. ფაილის ჩატვრითვისას გამრჩევი ანალიზს უკეთებს მას და მასში არსებული წესები გადაყავს თავის შინაგან წარმოდგენაში. ავღწეროთ ის კონსტრუქციები, რომლებისაგანც შედგება გრამატიკის ფაილი. ამისათვის, გამოვიყენოთ ბეკუს-ნაურის ფორმალიზმი. ქვემოთ მოცემულია გრამატიკის ფაილის სრული აღწერა კომენტარებთან ერთად:

<წესი> ::= <თავი> <ტანი> <კუდი>
<წესი> ::= <თავი> <ტანი> ':' <მდებარეობათა_მარეგულირებლები> <კუდი>

აქ ვხედავთ რომ წესი ორი სახით შეგვიძლია ჩავწეროთ. პირველი გახლავთ მარტივი ჩაწერა, მეორე ჩაწერაში კი ორწერტილის შემდეგ შემოტანილია მდებარეობათა მარეგულირებელი კონსტრუქციები.

<თავი> ::= <არატერმინალური_სიმბოლო> '->'
<არატერმინალური_სიმბოლო> ::= <იდენტიფიკატორი>

წესის "თავი" გახლავთ მარცხენა მხარეში მდგომი არატერმინალური სიმბოლო რომელსაც მოსდევს ისარი ('->'). ხოლო თავად არატერმინალური სიმბოლო გახლავთ იდენტიფიკატორი.

<ტანი> ::= <სიმბოლო>
<ტანი> ::= <ტანი> <სიმბოლო>

წესის "ტანი" წარმოადგენს მარჯვენა მხარეში მდგომ სიმბოლოთა მიმდევრობას.

<მდებარეობათა_მარეგულირებლები> ::=
<მდებარეობათა_მარეგულირებლები> ::= <მდებარეობათა_მარეგულირებლები>
<სიმბოლო> '<' <სიმბოლო>
<მდებარეობათა_მარეგულირებლები> ::= <მდებარეობათა_მარეგულირებლები>
<სიმბოლო> '-' <სიმბოლო>

მდებარეობათა მარეგულირებელი კონსტრუქციები გამოიყენება იმის აღსაწერად თუ რა დამოკიდებულებაში არიან ერთმანეთთან წესში შემავალი სიმბოლოები. არსებობს ორი ტიპის ურთიერთდამოკიდებულება. პირველი მათგანია "წინმსწრები" - ანუ შემთხვევა, როდესაც ერთი სიმბოლო დგას მეორეს წინ ნებისმიერ ადგილას და იგი ასე ჩაიწერება: A < B, სადაც A და B წესის მარჯვენა მხარეში შემავალი სიმბოლოებია. მეორე მათგანი გახლავთ "უშუალოდ წინმდგომი" - როდესაც ერთი სიმბოლო უშუალოდ მეორეს წინ დგას რაც შემდეგნაირად ჩაიწერება: A - B. მდებარეობათა მარეგულირებელი კონსტრუქციების სიმრავლე შესაძლოა ცარიელიც იყოს, ამ შემთხვევაში განიხილება წესში სიმბოლოთა ყველანაირი მიმდევრობა, ყველა კომბინაცია, რომელიც სიმბოლოების გადანაცვლებებით მიიღება.

<კუდი> ::= ';'
<კუდი> ::= '{' <შეზღუდვები> '}'

წესის "კუდი" პირობითი ცნებაა. იგი შეიძლება იყოს წერტილმძიმე, რაც ნიშნავს იმას, რომ წესს არ გააჩნია მასთან ასოცირებული შეზღუდვები. ხოლო თუ გააჩნია შეზღუდვები, მაშინ ისინი ჩაიწერებიან ფიგურულ ფრჩხილებში.

<შეზღუდვები> ::= <შეზღუდვის_ოპერაცია>
<შეზღუდვები> ::= '+' <შეზღუდვის_ოპერაცია>
<შეზღუდვები> ::= '-' <შეზღუდვის_ოპერაცია>
<შეზღუდვები> ::= <შეზღუდვები> '&' <შეზღუდვები>
<შეზღუდვები> ::= <შეზღუდვები> '|' <შეზღუდვები>
<შეზღუდვები> ::= '~' <შეზღუდვები>
<შეზღუდვები> ::= '(' <შეზღუდვები> ')'

თვისებათა შეზღუდვები წარმოადგენს ლოგიკურ გამოსახულებას. იგი შედგება შეზღუდვის ოპერაციებისაგან, რომელთაც წინ შესაძლოა ჰქონდეთ '+' ან '-' ნიშანი, რაც ნიშნავს იმას, რომ გაჩუმებით ამ ოპერაციას შეესაბამება ლოგიკურად ჭეშმარიტი ან ლოგიკურად მცდარი მნიშვნელობა (როგორც წესი ეს საშუალება არ გამოიყენება, გაჩუმებით კი ოპერაციას შეესაბამება ლოგიკურად ჭეშმარიტი მნიშვნელობა). ლოგიკური გამოსახულების ასაგებად შეგვიძლია გამოვიყენოთ სამი ძირითადი ლოგიკური ოპერაცია. ესენია: ~ - ლოგიკური უარყოფა, & - ლოგიკური ნამრავლი, | - ლოგიკური ჯამი. ოპერაციები დალაგებულია პრიორიტეტების მიხედვით. ამ პრიორიტეტების გადაფარვისთვის შეგვიძლია გამოვიყენოთ მრგვალი ფრჩხილები.

<შეზღუდვის_ოპერაცია> ::= <ოპერაცია>
<შეზღუდვის_ოპერაცია> ::= <ფუნქციის_გამოძახება>

შეზღუდვის ოპერაცია წარმოიდგინება ოპერაციის ან ფუნქციის გამოძახების სახით. ისინი ერთიმეორეს ექვივალენტურებია.

<ოპერაცია> ::= <არგუმენტი> ':=' <არგუმენტი>
<ოპერაცია> ::= <არგუმენტი> '=' <არგუმენტი>
<ოპერაცია> ::= <არგუმენტი> '==' <არგუმენტი>
<ოპერაცია> ::= <არგუმენტი> '<==' <არგუმენტი>
<ოპერაცია> ::= <არგუმენტი> '=' '(' <არგუმენტები> ')'
<ოპერაცია> ::= <არგუმენტი> '==' '(' <არგუმენტები> ')'

სულ გვაქვს ექვსი ტიპის ოპერაცია: მინიჭება, ტოლობა, უნიფიკაციის შემოწმება, უნიფიკაცია, ტოლობის მრავალარგუმენტიანი შემოწმება, უნიფიკაციის მრავალარგუმენტიანი შემოწმება.

<ფუნქციის_გამოძახება> ::= <იდენტიფიკატორი> '(' <არგუმენტები> ')'
<ფუნქციის_გამოძახება> ::= <იდენტიფიკატორი> '(' ')'

ფუნქციის გამოძახება ხდება სტანდარტული სახით. ფუნქციის სახელს მოსდევს მრგვალ ფრჩხილებში ჩაწერილი არგუმენტების მიმდევრობა (შესაძლოა ცარიელი).

<არგუმენტი> ::= <ცვლადი>
<არგუმენტი> ::= <გზა>
<არგუმენტი> ::= <თვისების_მნიშვნელობა>

ოპერაციის ან ფუნქციის არგუმენტი შეიძლება იყოს: ცვლადი, გზა ან თვისების მნიშვნელობა. ყოველი მათგანი საბოლოო ჯამში გვაძლევს რაიმე თვისებას, რომელზეც შეგვიძლია ოპერაციის ჩატარება.

<არგუმენტები> ::= <არგუმენტი>
<არგუმენტები> ::= <არგუმენტები> ',' <არგუმენტი>

ეს გახლავთ არგუმენტების სია, რომელიც განისაზღვრება როგორც არგუმენტების ჩვეულებრივი მიმდევრობა ერთიმეორისგან მძიმეებით გამოყოფილი.

<გზა> ::= '<' <სიმბოლო> <თვისებათა_სახელების_მიმდევრობა> '>'
<გზა> ::= '<' '$' <ცვლადის_სახელი> <თვისებათა_სახელების_მიმდევრობა> '>'

"გზა" წარმოადგენს კუთხოვან ფრჩხილებში ჩაწერილ სიმბოლოს (ან ცვლადს), რომელსაც მოსდევს თვისებათა სახელების (შესაძლოა ცარიელი) მიმდევრობა. ეს მიმდევრობა გამოიყენება იმ შემთხვევაში, თუკი გვინდა რაიმე კონკრეტულ ქვეთვისებაზე წვდომა.

<თვისებათა_სახელების_მიმდევრობა> ::=
<თვისებათა_სახელების_მიმდევრობა> ::= <თვისებათა_სახელების_მიმდევრობა>
<თვისების_სახელი>

თვისებათა სახელების მიმდევრობა წარმოადგენს ერთიმეორეს მიყოლებით ჩაწერილ თვისების სახელებს.

<თვისებათა_კონსტრუქტორი> ::= '[' <თვისებები> ']'
<თვისებათა_კონსტრუქტორი> ::= '[' '(' <ინიციალიზაციის_ნაწილი> ')' <თვისებები> ']'

თვისებათა კონსტრუქტორი გამოიყენება თვისებათა სტრუქტურის ასაგებად. იგი წარმოადგენს კვადრატულ ფრჩხილებში ჩაწერილ თვისებებს. ამის გარდა მას შესაძლოა ჰქონდეს ინიციალიზაციის ნაწილი, რომელშიც ჩაიწერებიან ცვლადები. ამ ცვლადებიდან მოხდება თვისებების გადმოტანა და დაგროვება მთავარ თვისებათა სტრუქტურაში.

<ინიციალიზაციის_ნაწილი> ::=
<ინიციალიზაციის_ნაწილი> ::= <ინიციალიზაციის_ნაწილი> <ცვლადის_სახელი>

თვისებათა კონსტრუქტორის ინიციალიზაციის ნაწილი წარმოადგენს ერთიმეორის მიყოლებით ჩაწერილ ცვლადების სახელებს.

<თვისებები> ::=
<თვისებები> ::= <თვისებები> <თვისების_სახელი> ':' <თვისების_მნიშვნელობა>

თვისებები გახლავთ წყვილების მიმდევრობა. წყვილები შემდეგნაირია: სახელი: მნიშვნელობა, ერთად ისინი წარმოადგენენ თვისებას.

<თვისების_სახელი> ::= <იდენტიფიკატორი>

თვისების სახელი გახლავთ ჩვეულებრივი იდენტიფიკატორი.

<თვისების_მნიშვნელობა> ::= None
<თვისების_მნიშვნელობა> ::= <იდენტიფიკატორი>
<თვისების_მნიშვნელობა> ::= <თვისებათა_კონსტრუქტორი>

თვისების მნიშვნელობა შეიძლება იყოს რეზერვირებული სიტყვა None (რაც ნიშნავს ცარიელ მნიშვნელობას), იდენტიფიკატორი (სტრიქონული მნიშვნელობა), თვისებათა კონსტრუქტორი (რაც რეკურსიის საშუალებას გვაძლევს).

<ცვლადი> ::= '$' <ცვლადის_სახელი>
<ცვლადის_სახელი> ::= <იდენტიფიკატორი>

ცვლადებს წინ უძღვით დოლარის ნიშანი. ცვლადის სახელი გახლავთ იდენტიფიკატორი.

<სიმბოლო> ::= <იდენტიფიკატორი>
<სიმბოლო> ::= <იდენტიფიკატორი> '.' <მთელი_დადებითი_რიცხვი>

წესის სიმბოლო წარმოადგენს ან იდენტიფიკატორს, ან ინდექსირებულ იდენტიფიკატორს. ინდექსი ჩაიწერება სიმბოლოს შემდეგ და მისგან წერტილით გამოიყოფა. ინდექსი გახლავთ მთელი, დადებითი რიცხვი.
გრამატიკის ფაილის აღწერა Bison პროგრამის შემავალი ფაილის სახით მოცემულია დანართში.

§6. ქართული ენის სინტაქსის ფრაგმენტი

განვიხილოთ ქართული ენის გრამატიკის ფრაგმენტი, რომელიც ამუშავებს სამპირიანი ზმნების გამოყენებით აგებულ წინადადებებს და ენაში არსებულ სხვა კონსტრუქციებს. აგრეთვე მოვიყვანოთ ლექსიკონის ფაილი მცირეოდენი ლექსიკური მარაგით, რომელშიც შედიან ენაში არსებული ძირითადი მეტყველების ნაწილები

გრამატიკის ფაილი:

```
# ÆãÂ3Ð - ÆÌÍÉÓ ãÂÖ×É, ÌÄÓÀÌÄ ÐÉÒÉ
ÆãÂ3Ð -> ÓÐÍÓ.1 ÓÐÍÓ.2 ÆãÂ ÓÐÍÓ.3 : {
   <ÆãÂ ÐÉÒÉÀÍÏÁÀ>   == '3'   &
   <ÆãÂ ÓÄÒÉÀ>       == '1'   &
   <ÓÐÍÓ.1 ÁÒÖÍÅÀ>   == 'ÓÀá' &
   <ÓÐÍÓ.1 ÐÉÒÉ>     == <ÆãÂ ÐÉÒÉ>   &
   <ÓÐÍÓ.1 ÒÉÝáÅÉ>   == <ÆãÂ ÒÉÝáÅÉ> &
   <ÓÐÍÓ.2 ÁÒÖÍÅÀ>   == 'ÌÉÝ' &
   <ÓÐÍÓ.2 ÐÉÒÉ>     == <ÆãÂ ÐÉÒÃÀÐÉÒÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ> &
   <ÓÐÍÓ.3 ÁÒÖÍÅÀ>   == 'ÌÉÝ' &
   <ÓÐÍÓ.3 ÐÉÒÉ>     == <ÆãÂ ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ>   &
   <ÓÐÍÓ.3 ÒÉÝáÅÉ>   == <ÆãÂ ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÒÉÝáÅÉ> &
   <ÆãÂ3Ð ÓÖÁÉÄØÔÉ>  := <ÓÐÍÓ.1> &
   <ÆãÂ3Ð ÐÒÄÃÉÊÀÔÉ> := <ÆãÂ>    &
   <ÆãÂ3Ð ÏÁÉÄØÔÉ_1> := <ÓÐÍÓ.2> &
   <ÆãÂ3Ð ÏÁÉÄØÔÉ_2> := <ÓÐÍÓ.3>
}
ÆãÂ3Ð -> ÓÐÍÓ.1 ÓÐÍÓ.2 ÆãÂ ÓÐÍÓ.3 : {
   <ÆãÂ ÐÉÒÉÀÍÏÁÀ>   == '3' &
   <ÆãÂ ÓÄÒÉÀ>       == '2' &
   <ÓÐÍÓ.1 ÁÒÖÍÅÀ>   == 'ÌÏÈáÒ' &
   <ÓÐÍÓ.1 ÐÉÒÉ>     == <ÆãÂ ÐÉÒÉ> &
   <ÓÐÍÓ.1 ÒÉÝáÅÉ>   == <ÆãÂ ÒÉÝáÅÉ> &
```

```
   <ÓÐÍÓ.2 ÁÒÖÍÅÀ>   == 'ÓÀá' &
   <ÓÐÍÓ.2 ÐÉÒÉ>     == <ÆãÂ ÐÉÒÃÀÐÉÒÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ> &
   <ÓÐÍÓ.3 ÁÒÖÍÅÀ>   == 'ÌÉÝ' &
   <ÓÐÍÓ.3 ÐÉÒÉ>     == <ÆãÂ ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ> &
   <ÓÐÍÓ.3 ÒÉÝáÅÉ>   == <ÆãÂ ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÒÉÝáÅÉ> &
   <ÆãÂ3Ð ÓÖÁÉÄØÔÉ>  := <ÓÐÍÓ.1> &
   <ÆãÂ3Ð ÐÒÄÃÉÊÀÔÉ> := <ÆãÂ> &
   <ÆãÂ3Ð ÏÁÉÄØÔÉ_1> := <ÓÐÍÓ.2> &
   <ÆãÂ3Ð ÏÁÉÄØÔÉ_2> := <ÓÐÍÓ.3>
}

# ÓÐÍÓ - ÓÀáÄËÉ ÀÍ ÐÉÒÉÓ ÍÀÝÅÀËÓÀáÄËÉ
ÓÐÍÓ -> ÓãÂÌ { <ÓÐÍÓ> := <ÓãÂÌ> }
ÓÐÍÓ -> ÐÍÓ { <ÓÐÍÓ> := <ÐÍÓ> }

# ÆãÂ - ÆÌÍÉÓ ãÂÖ×É
ÆãÂ -> Æ { <ÆãÂ> := <Æ> }
ÆãÂ -> ÆÆ ÆãÂ.2 : { <ÆãÂ> := <ÆãÂ.2> }
ÆãÂ -> ÆÓ ÆãÂ.2 : { <ÆÓ ÁÒÖÍÅÀ> = 'ÅÉÈ' & <ÆãÂ> := <ÆãÂ.2> }

# ÓãÂÌ - ÓÀáÄËÉ ãÂÖ×É ÌÀÒÈÅÀ
ÓãÂÌ -> ÓãÂÌ.2 ÓãÂÌ.3 { <ÓãÂÌ.2 ÁÒÖÍÅÀ> == 'ÍÀÈ' & <ÓãÂÌ> := <ÓãÂÌ.3> }
ÓãÂÌ -> ÓãÂ { <ÓãÂÌ> := <ÓãÂ> }

# ÓãÂ - ÓÀáÄËÉÓ ãÂÖ×É
ÓãÂ -> ÀÔ ÓãÂ.2
{
   (
      <ÓãÂ.2 ÁÒÖÍÅÀ> = ('ÓÀá', 'ÍÀÈ', 'ÌÏØÌ') & <ÀÔ ÁÒÖÍÅÀ> = 'ÓÀá_ÍÀÈ_ÌÏØÌ' |
      <ÓãÂ.2 ÁÒÖÍÅÀ> = ('ÌÉÝ', 'ÅÉÈ')        & <ÀÔ ÁÒÖÍÅÀ> = 'ÌÉÝ_ÅÉÈ'     |
      <ÓãÂ.2 ÁÒÖÍÅÀ> == <ÀÔ ÁÒÖÍÅÀ>
   ) &
   <ÓãÂ> := <ÓãÂ.2>
}
ÓãÂ -> ÀÓ { <ÓãÂ> := <ÀÓ> }

# ÀÔ - ÀÔÒÉÁÖÔÉ
ÀÔ -> ÆÓ  { <ÀÔ ÁÒÖÍÅÀ> := <ÆÓ ÁÒÖÍÅÀ>  }
ÀÔ -> ÒÓ  { <ÀÔ ÁÒÖÍÅÀ> := <ÒÓ ÁÒÖÍÅÀ>  }
ÀÔ -> ÍÓ  { <ÀÔ ÁÒÖÍÅÀ> := <ÍÓ ÁÒÖÍÅÀ>  }
ÀÔ -> ÌÉÌ { <ÀÔ ÁÒÖÍÅÀ> := <ÌÉÌ ÁÒÖÍÅÀ> }
```

ლექსიკონის ფაილი:

```
ÆÓ     = [ÊÀÔ: ÆÓ]
ÀÓ     = [ÊÀÔ: ÀÓ]
ÌÉÌ     = [ÊÀÔ: ÌÉÌ]
```

ÆÆ = [ÊÀÔ: ÆÆ]
Æ = [ÊÀÔ: Æ]
ÐÍÓ = [ÊÀÔ: ÐÍÓ]

ÌÒ = [ÒÉÝáÅÉ: ÌÒ]
Ìá = [ÒÉÝáÅÉ: Ìá]

ÓÀá = [ÁÒÖÍÅÀ: ÓÀá]
ÌÏÈáÒ = [ÁÒÖÍÅÀ: ÌÏÈáÒ]
ÌÉÝ = [ÁÒÖÍÅÀ: ÌÉÝ]
ÍÀÈ = [ÁÒÖÍÅÀ: ÍÀÈ]
ÅÉÈ = [ÁÒÖÍÅÀ: ÅÉÈ]
ÌÏØÌ = [ÁÒÖÍÅÀ: ÌÏØÌ]
ßÏÃ = [ÁÒÖÍÅÀ: ßÏÃ]

ÌÉÝ_ÅÉÈ = [ÁÒÖÍÅÀ: ÌÉÝ_ÅÉÈ]
ÓÀá_ÍÀÈ_ÌÏØÌ = [ÁÒÖÍÅÀ: ÓÀá_ÍÀÈ_ÌÏØÌ]

Ð1 = [ÐÉÒÉ: 1]
Ð2 = [ÐÉÒÉ: 2]
Ð3 = [ÐÉÒÉ: 3]

ÐÍ1 = [ÐÉÒÉÀÍÏÁÀ: 1]
ÐÍ2 = [ÐÉÒÉÀÍÏÁÀ: 2]
ÐÍ3 = [ÐÉÒÉÀÍÏÁÀ: 3]

Ó1 = [ÓÄÒÉÀ: 1]
Ó2 = [ÓÄÒÉÀ: 2]
Ó3 = [ÓÄÒÉÀ: 3]

ÐÏÐ1 = [ÐÉÒÃÀÐÉÒÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 1]
ÐÏÐ2 = [ÐÉÒÃÀÐÉÒÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 2]
ÐÏÐ3 = [ÐÉÒÃÀÐÉÒÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 3]
ÉÏÐ1 = [ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 1]
ÉÏÐ2 = [ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 2]
ÉÏÐ3 = [ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÐÉÒÉ: 3]
ÉÏÒÌá = [ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÒÉÝáÅÉ: Ìá]
ÉÏÒÌÒ = [ÉÒÉÁÉ_ÏÁÉÄØÔÉÓ_ÒÉÝáÅÉ: ÌÒ]

ÃÒÀ = [ÃÒÏ: ÀßÌÚÏ]
ÃÒß = [ÃÒÏ: ßÀÒÓÖËÉ]
ÃÒÌ = [ÃÒÏ: ÌÏÌÀÅÀËÉ]

ÝÍÏÁÉË [(ÆÓ, ÌÉÝ_ÅÉÈ)]
ÝÍÏÁÉËÉ [(ÆÓ, ÓÀá_ÍÀÈ_ÌÏØÌ)]
ÝÍÏÁÉËÀÃ [(ÆÓ, ÅÉÈ)]
ØÀÒÈÅÄË [(ÆÓ, ÌÉÝ_ÅÉÈ)]
ØÀÒÈÅÄËÉ [(ÆÓ, ÓÀá_ÍÀÈ_ÌÏØÌ)]
ÌÛÄÍÄÁËÄÁÓ [(ÀÓ, ÌÉÝ, ÌÒ, Ð3)]
ÌÛÄÍÄÁËÄÁÉ [(ÀÓ, ÓÀá, ÌÒ, Ð3)]

ÌÛÄÍÄÁÄËÉ [(ÀÓ, ÓÀá, Ìá, Ð3)]
ÌÛÄÍÄÁÄËÌÀ [(ÀÓ, ÌÏÈáÒ, Ìá, Ð3)]
ÂÀÊÄÈÄÁÖËÉ [(ÌÉÌ, ÓÀá_ÍÀÈ_ÌÏØÌ)]
ÌÀÈÄÌÀÔÉÊÖÒÉ [(ÆÓ, ÓÀá_ÍÀÈ_ÌÏØÌ)]
ÀÍÀËÉÆÉÓ [(ÀÓ, ÍÀÈ, Ìá)]
ÌÊÀÝÒÉ [(ÆÓ, ÓÀá_ÍÀÈ_ÌÏØÌ)]
ÊÖÒÓÉÓ [(ÀÓ, ÍÀÈ, Ìá, Ð3)]
ÀÂÄÁÀ [(ÀÓ, ÓÀá, Ìá, Ð3)]
ßÉÍ [(ÆÆ)]
ÖÊÀÍ [(ÆÆ)]
ÓÀáËÉ [(ÀÓ, ÓÀá, Ìá, Ð3)]
ÓÀáËÓ [(ÀÓ, ÌÉÝ, Ìá, Ð3)]
ÌÄÂÏÁÀÒÓ [(ÀÓ, ÌÉÝ, Ìá, Ð3)]
ÌÄ [(ÐÍÓ, Ìá, Ð1)]
ÜÅÄÍ [(ÐÍÓ, ÌÒ, Ð1)]
ÛÄÍ [(ÐÍÓ, Ìá, Ð2)]
ÈØÅÄÍ [(ÐÍÓ, ÌÒ, Ð2)]
ÉÓ [(ÐÍÓ, ÓÀá, Ìá, Ð3)]
ÉÓÉÍÉ [(ÐÍÓ, ÓÀá, ÌÒ, Ð3)]
ÌÀÓ [(ÐÍÓ, ÌÉÝ, Ìá, Ð3)]
ÌÀÈ [(ÐÍÓ, ÌÉÝ, ÌÒ, Ð3)]
ÖÛÄÍÄÁÓ [(Æ, Ð3, Ó1, Ìá, ÐÏÐ3, ÉÏÐ3, ÉÏÒÌá, ÐÍ3, ÃÒÀ)]
ÂÅÉÛÄÍÄÁÓ [(Æ, Ð3, Ó1, Ìá, ÐÏÐ3, ÉÏÐ1, ÉÏÒÌÒ, ÐÍ3, ÃÒÀ)]
ÅÖÛÄÍÄÁ [(Æ, Ð1, Ó1, Ìá, ÐÏÐ3, ÉÏÐ3, ÉÏÒÌá, ÐÍ3, ÃÒÀ)]
ÀÖÛÄÍÀ [(Æ, Ð3, Ó2, Ìá, ÐÏÐ3, ÉÏÐ3, ÉÏÒÌá, ÐÍ3, ÃÒß)]
ÀÅÖÛÄÍÄ [(Æ, Ð1, Ó2, Ìá, ÐÏÐ3, ÉÏÐ3, ÉÏÒÌá, ÐÍ3, ÃÒß)]

დასკვნა და ძირითადი შედეგები

ნაშრომში განხილულია ბუნებრივ ენათა კომპიუტერული დამუშავების საკითხები. ზოგიერთი არსებული პრობლემის გადასაჭრელად შემუშავებულია ახალი მიდგომები. შექმნილია ახალი ფორმალიზმები და მათი რეალიზაციის საფუძველზე მიღებულია პროგრამული სისტემა, რომლითაც შეგვიძლია ჩავატაროთ ბუნებრივ ენაზე ჩაწერილი ტექსტების სინტაქსური და მორფოლოგიური ანალიზი. ამისათვის საჭიროა, სპეციალისტის (ლინგვისტის) მიერ გრამატიკის ფაილებში ჩაწერილ იქნეს კონკრეტული ბუნებრივი ენის სინტაქსის და მორფოლოგიის შემადგენელი წესები. შექმნილ პროგრამულ სისტემას, სხვა სისტემებთან შედარებით, გააჩნია შემდეგი უპირატესობები:

- ფორმალიზმები რომლებსაც იყენებენ სინტაქსური და მორფოლოგიური ანალიზატორები არის გაცილებით უნივერსალური და მოქნილი. მასში შესაძლებელია რთული გრამატიკული წესების ჩაწერა კომპაქტური სახით. გათვალისწინებულია მრავალი მოსახერხებელი კონსტრუქცია რომელიც უადვილებს სამუშაოს სპეციალისტს (ლინგვისტს).
- მორფოლოგიური ანალიზისათვის შემუშავებული ალგორითმი საშუალებას იძლევა ჩავატაროთ სრულყოფილი მორფოლოგიური ანალიზი ისეთი მორფოლოგიურად მდიდარი ენებისათვის როგორიცაა: ქართული, რუსული, უნგრული, არაბული და ა.შ.. მორფოლოგიური წესის შიგნით შეზღუდვების გადანაწილების შესაძლებლობა მაქსიმალურად ამაღლებს ალგორითმის სწრაფქმედებას.
- სტანდარტული კონტექსტისაგან თავისუფალი გრამატიკის წესებისგან განსხვავებით, სინტაქსის წესების ჩაწერისას საშუალება გვეძლევა მივუთითოთ წესში შემავალ სიმბოლოთა თავისუფალი ან ნაწილობრივ თავისუფალი წყობა, რაც დამახასიათებელია ზოგიერთი ბუნებრივი ენისათვის, მაგალითად ქართული ენისათვის.
- მორფოლოგიურ ანალიზატორში ჩადგმულია ტექსტური პრეპროცესორი, რომელიც საშუალებას გვაძლევს, რომ მორფოლოგიის ფაილში გამოვიყენოთ პარამეტრიანი მაკრო-ჩასმები. ამით თავს

გ ა მ ო ყ ე ნ ე ბ უ ლ ი ლ ი ტ ე რ ა ტ უ რ ა

1. ჯ. ანთიძე, ნ. გულუა. მასწავლებელი სისტემის აგებისადმი ერთი მიდგომის შესახებ, თბილისის სახელმწიფო უნივერსიტეტის სოხუმის ფილიალის მოამბე, §1, 1998.
2. ჯ. ანთიძე, დ. მიშელაშვილი. ინსტრუმენტული საშუალებები ქართული ტექსტების კომპიუტერული დამუშავებისათვის, კონფერენცია - ბუნებრივ ენათა დამუშავება, ქართული ენა და კომპიუტერული ტექნოლოგიები, თბილისი, 2003.
3. ჯ. ანთიძე, ს. შენგელია. ქართული წინადადების სინტაქური და სემანტიკური სტრუქტურის წარმოდგენის შესახებ, კონფერენცია - ბუნებრივ ენათა დამუშავება, ქართული ენა და კომპიუტერული ტექნოლოგიები, თბილისი, 2003.
4. ჯ. ანთიძე, დ. მელიქიშვილი, დ. მიშელაშვილი. ქართული ენის კომპიუტერული მორფოლოგია. კონფერენცია - ბუნებრივ ენათა დამუშავება, ქართული ენა და კომპიუტერული ტექნოლოგიები, თბილისი, 2004.
5. ჯ. ანთიძე, დ. მიშელაშვილი. ბუნებრივი ენის მორფოლოგიური ანალიზატორი. კონფერენცია - ბუნებრივ ენათა დამუშავება, ქართული ენა და კომპიუტერული ტექნოლოგიები, თბილისი, 2004.
6. ქ. დათუკიშვილი, ნ. ლოლაძე, მ. ზაკალაშვილი, ქართული ენის კორპუსი. ქართული ენის მანქანური დამუშავება, მორფოლოგიის დონე.კონფერენცია - ბუნებრივ ენათა დამუშავება, ქართული ენა და კომპიუტერული ტექნოლოგიები, თბილისი, 2003.
7. ჯ. ანთიძე. ქართული ენიდან რუსულად საცდელი მანქანური თარგმანის შესახებ. საკანდიდატო დისერტაცია ფ.მ.მ.კ. სამეცნიერო ხარისხის მოსაპოვებლად, თბილისი, 1966.
8. A. V. Aho and J. D. Ullman. The Theory of Parsing, Translation, and Compiling. Prentice-Hall, Englewood Cliffs, NJ, 1972
9. S. McConnel, PC-PATR Reference, 2000, Summer Institute of Linguists, http://www.sil.org

10. V. Paxson, Flex - A fast scanner generator, Edition 2.5, March 1995, http://www.cs.princeton.edu/~appel/modern/c/software/flex/flex_toc.html
11. Ch. Donnelly, R. Stallman. Bison – The Yacc-compatible Parser Generator, Version 2.0, 2004, http://www.gnu.org/software/bison/manual/
12. J. Antidze. The structure of dictionary for machine translation from Georgian language, Communications of The Georgian Academy of Fciences, XXXI:2, 1963 (in Georgian).
13. J. Antidze. About syntactic analysis of phrases for machine translation from Georgian language, Proceedings of the Tbilisi State University, volume 98, 1964 (in Russian, coauthor).
14. J. Antidze. Program for Georgian verbs decomposition in morphemes, Report of I Georgian mathematicians congress, Tbilisi, 1994 (in Georgian).
15. J. Antidze, N. Gulua. Analysis of Georgian texts for construction of a teaching system, Reports of enlarged session of the seminar of IAM TSU, vol.13, N4, 1998
16. J. Antidze, N. Gulua. On selection of Georgian texts computer analysis formalism, Communications of The Georgian Academy of Sciences, 162, N2, 2000.
17. J. Antidze, N. Gulua, D. Mishelashvili. Syntactic analysis of Georgian texts, Fourth Tbilisi symposium Language, Logic and Computation, TSU, 2001
18. J. Antidze, N. Gulua, D. Mishelashvili. Syntactic and semantic analysis of georgian texts, Georgian mathematicians congress, Tbilisi, 2001.
19. J. Antidze, D. Mishelashvili. Instrumental Tool for Morphological Analysis of Some Natural Languages, Reports of Enlarged Session of the Seminar of IAM TSU, vol.19, 2004.
20. J. Antidze, D.Mishelashvili. Recognition of Wordforms and Their Morphological Categories by Computer, Reports of Enlarged Session of the Seminar of IAM TSU, vol.19, 2004.
21. D. Mishelashvili. Software Tools for Morphological and Syntactic Analysis of Georgian Texts. Reports of Enlarged Session of the Seminar of IAM TSU 2005, to be appear.
22. J. Antidze, D. Mishelashvili. Software Tools for Morphological and Syntactic Analysis of Natural Language Texts. Conference on Natural Language Processing, The

Georgian Language and Computer Technologies, Institute of Linguistics of Georgian Academy of Sciences, Tbilisi, 2005.

23. M. Gross. Grammaire transformationnelle elu français, Paris, 1986.