ჯემალ ანთიძე

ფორმალურ ენათა და გრამატიკათა თეორია, ბუნებრივი ენების კომპიუტერული მოდელირება

თბილისი
2009

რედაქტორი:

ასოც. პროფესორი **ნ. გულუა**

რეცენზენტი:

მეცნიერებათა დოქტორი **თ. საჟენიუკი**

დაიბეჭდა გამომცემლობაში „ნეკერი“, 2009

ISBN 978-9941-404-92-4

წინათქმა

ეს ნაშრომი წარმოადგენს ავტორის ცდას გადმოეცა ზოგიერთი ძირითადი შედეგები, რომლებიც გამოიყენებიან ბუნებრივი ენის ტექსტების კომპიუტერული მოდელირებისათვის. მათ გასაცნობად აუცილებელია პირველ რიგში ფორმალურ ენათა და გრამატიკათა თეორიის საფუძვლების ცოდნა. ამიტომაც ნაშრომის პირველი ორი თავი ეძღვნება მათ გადმოცემას და საკითხები აღებულია [1]-დან.

მესამე თავი ეძღვნება ბუნებრივი ენების კომპიუტერულ დამუშავებას. ზოგადად, მასში გადმოცემულია როგორც სინტაქსისა და სემანტიკის, ასევე მორფოლოგიის საკითხები კომპიუტერული დამუშავების თვალსაზრისით.

მეოთხე თავში აღწერილია კომპიუტერული მასწავლებელი სისტემები და ის შედეგები, რომლებიც მიღებულია ნანა გულუასა და ჩემს მიერ 2000 წლამდე. ჩვენი ახალი შედეგები აქ არაა შეტანილი.

მეხუთე თავში გადმოცემულია დავით მიშელაშვილისა და ჩემს მიერ მიღებული შედეგები ბუნებრივი ენის კომპიუტერული დამუშავების ავტომატიზაციის საკითხებში 2005 წლამდე.

ეს, ამდენი საკითხების ერთად თავმოყრის და გადმოცემის პირველი ცდაა და ამიტომაც არ იქნება უნაკლო. ავტორი მადლიერებით მიიღებს ყველა საქმიან შენიშვნასა თუ რჩევას.

1 თავი
ფორმალური ენები და გრამატიკები

1.1 ფორმალური ენის განსაზღვრა. განვიხილოთ რაიმე სიმბოლოების სასრული სიმრავლე Σ, რომელსაც ვუწოდოთ ალფაბეტი. ალფაბეტი გამოიყენება ნებისმიერ ბუნებრივ თუ ფორმალურ ენაში. სიმბოლოს ქვეშ იგულისხმება ნებისმიერი ნიშანი, რომელიც შეიძლება შეგვხვდეს ტექსტში. სიმბოლოთა ნებისმიერ სასრულ მიმდევრობას ვუწოდოთ ჯაჭვი (ზოგჯერ მას უწოდებენ სიტყვას ან სტრიქონს). ისეთ ჯაჭვს, რომელიც არ შეიცავს არცერთ სიმბოლოს ვუწოდოთ ცარიელი ჯაჭვი და ავღნიშნოთ e სიმბოლოთი. ჯაჭვების ნებისმიერ სასრულ მიმდევრობას ვუწოდოთ ტექსტი. იგულისხმება, რომ ჯაჭვები ტექსტში ერთიმეორისაგან გამოყოფილია სპეციალური ხარვეზის ნიშნით. ხარვეზის ნიშნად ფორმალურ ენებში გამოიყენება სიმბოლო # , ხოლო ტექსტში მის აღსანიშნავად გამოიყენება ცარიელი ადგილი. ტექსტში გამოყენებულ ცარიელ ადგილს ჩვენ ჩავთვლით სიმბოლოდ. ხარვეზის ნიშანი არ შეიძლება შედიოდეს ჯაჭვში, რადგანაც იგი გამოიყენება ჯაჭვების განმაცალკევებლად, მაგრამ იგი ყოველთვის შედის ალფაბეტში. ცალკეულ სიმბოლოებს ზოგადად a,b,c,d ასოებით აღვნიშნავთ, ხოლო ჯაჭვებს t,u,v,w,x,y, და z ასოებით აღვნიშნავთ. c^i-თი აღვნიშნავთ ჯაჭვს, რომელიც i რაოდენობა c სიმბოლოსაგან შედგება. თუ გვაქვს ორი x და y ჯაჭვი, მაშინ xy ჯაჭვს x და y ჯაჭვების კონკატენაცია ეწოდება და y-ის x-ის მარჯვნივ მიწერით მიიღება. ჩვენ არ გამოვიყენებთ სპეციალურ სიმბოლოს კონკატენაციის ოპერაციის აღსანიშნავად.

შემოტანილი შეთანხმებებიდან გამომდინარეობს, რომ $a^0=e$ და $a^2=aa$. რეკურსიულად, რაიმე ჯაჭვი Σ ალფაბეტში

განისაზღვრება შემდეგნაირად:

1. e არის ჯაჭვი Σ ალფაბეტში;

2. თუ x არის ჯაჭვი Σ ალფაბეტში და $a \in \Sigma$, მაშინ xa არის ჯაჭვი Σ-ში;

3. რაიმე z ჯაჭვი არის ჯაჭვი Σ-ში მხოლოდ და მხოლოდ მაშინ, თუ იგი 1 და 2 წესებით მიიღება.

იგულისხმება, რომ $ex=xe=x$. თუ $x=c_1...c_n$, მაშინ $c_nc_{n-1}...c_1$ ჯაჭვს ეწოდება x-ის შებრუნებული ჯაჭვი და x^r-ით აღვნიშნავთ. თუ გვაქვს yxz ჯაჭვი, მაშინ y-ს ეწოდება yxz-ის პრეფიქსი, ხოლო z-ს ეწოდება yxz-ის სუფიქსი, ხოლო x-ს ინფიქსი ეწოდება. x ჯაჭვის სიგრძე Vx-ით აღვნიშნოთ და განვმარტოთ შემდეგნაირად: თუ $x=e$, მაშინ $V_x=0$; თუ $x=a_1a_2...a_n$, მაშინ $V_x=n$. მაგალითად, $V_{aabcd}=5$. ჯაჭვების L რაიმე სიმრავლეს Σ ალფაბეტში ვუწოდოთ L ენა Σ ალფაბეტში. ეს განმარტება მოიცავს როგორც ფორმალურ ენებს ასევე ბუნებრივ ენებსაც. Σ^*-ით აღვნიშნოთ ენა, რომელიც მოიცავს ყველა ჯაჭვს Σ ალფავიტიდან e ჯაჭვის ჩათვლით, ხოლო Σ^+-ით აღვნიშნოთ სიმრავლე $\Sigma^*-\{e\}$, ე.ი. Σ^* სიმრავლე ცარიელი ჯაჭვის გარდა. Σ^*-ს უწოდებენ უნივერსალურ ენას Σ ალფაბეტში, რადგან ნებისმიერი L ენა Σ ალფაბეტში არის მისი ქვესიმრავლე. დაპროგრამების FORTRAN ენა არის ენა ჩვენი განმარტებით FORTRAN-ის ალფაბეტში, თუ ჯაჭვებად ავიღებთ სინტაქსურად სწორ პროგრამებს. ასევე ქართული ენა არის ენა ჩვენი განმარტებით ქართულ ალფაბეტში, თუ სწორ ქართულ წინადადებებს ავიღებთ ჯაჭვებად.

1.2 ფორმალური გრამატიკები. როგორც წინა პარაგრაფიდან უკვე ვიცით ნებისმიერი L ფორმალური ენა შეიძლება ჩაიწეროს ასე $L \subseteq \Sigma^*$, სადაც Σ რაიმე ალფაბეტია. ასეთი ენა ყოველთვის უსასრულოა, თუ ჩვენ არ

შევზღუდავთ ჯაჭვის სიგრძეს. რადგან ჩვენ შემდგომში დაგვაინტერესებს ისეთი ენები, რომელთა ჯაჭვებს აქვთ სასრული სიგრძე, ამიტომ ჩვენ ყოველთვის ვიგულისხმებთ, რომ $V_x \leq n$, სადაც n არაუარყოფითი მთელი რიცხვია და $x \in L$. რაიმე L ენის მოცემა შეიძლება მისი ჯაჭვების ჩამოთვლით, თუ ენა სასრულია. მაგალითად, ასეთ შემთხვევაშიც კი მოუხერხებელია ენის მოცემა მისი ჯაჭვების ჩამოწერით. ამიტომ, ბუნებრივია მოიძებნოს ისეთი ხერხი, რომელიც სრულად აღწერდა ენას და მარტივი იქნებოდა იმის შესამოწმებლად, ეკუთვნის თუ არა რაიმე x ჯაჭვი ამ ენას. ზოგჯერ შესაძლებელია L ენა ჩაიწეროს რაიმე ფორმულით, მაგალითად $\{a^n b^n c^n | n \geq 0\}$. მაგრამ ფორმულით ენის წარმოდგენა ხერხდება იშვიათ შემთხვევებში. ამიტომ, უნდა მოიძებნოს სხვა ხერხი. ერთ-ერთ ასეთ ხერხს წარმოადგენს ენის მოცემა წარმომქმნელი გრამატიკის საშუალებით. მეორე გზას წარმოადგენს ენის მოცემა კერძო ალგორითმით, რომელიც ამთავრებს მუშაობას თუ ჯაჭვი ეკუთვნის ენას. ჩვენ შევჩერდებით პირველ ხერხზე, კერძოდ ხომსკის გრამატიკებზე. ახლა, ჩვენ შევუდგებით ასეთი გრამატიკების აღწერას ჯერ არაფორმალურად, შემდეგ კი ზუსტად განვმარტავთ მათ. რაიმე L ენის აღსაწერად განვიხილოთ ორი სიმრავლე. ერთი სიმრავლეა ამ ენის ალფაბეტი $\sum$, რომელსაც ვუწოდებთ ტერმინალურ სიმბოლოთა სიმრავლეს და მეორე სიმრავლეა სიმბოლოთა N სიმრავლე, რომელიც ასევე სასრულია და ვუწოდებთ არატერმინალურ სიმბოლოთა სიმრავლეს. განვიხილოთ წესების P სასრული სიმრავლე, რომელიც წარმოადგენს (α,β) წყვილთა სიმრავლეს, სადაც $(\alpha,\beta) \in (N \cup \sum)^* N (N \cup \sum)^* \times (N \cup \sum)^*$ სიმრავლის რაიმე ელემენტია. ე. ი. წესის პირველი ელემენტი აუცილებლად შეიცავს ერთ არატერმინალურ სიმბოლოს მაინც, მაშინ როცა მეორე ელემენტი შეიძლება იყოს ცარიელი ჯაჭვი. წესი შეიძლება ჩაიწეროს ასეც $\alpha \rightarrow \beta$.

ჩვენ შემდგომში გამოვიყენებთ წესის ჩაწერის მეორე სახეს. N სიმრავლეში არსებობს სპეციალურად გამოყოფილი სიმბოლო, რომელსაც S-ით აღვნიშნავთ და ვუწოდებთ საწყის სიმბოლოს. ყოველთვის ვიწყებთ L ენის რაიმე ჯაჭვის მიღებას P სიმრავლის რაიმე წესით, როლმელსაც აქვს სახე $S \to \alpha$ და შემდეგ α-ში($\alpha=\alpha_1\beta\alpha_2$) მის რაიმე ქვეჯაჭვს β-ს ვცვლით γ-თი, სადაც $\beta \to \gamma$ P–ს რაიმე წესია. მივიღებთ ჯაჭვს $\alpha_1\gamma\alpha_2$ ასევე ვიქცევით ამ ჯაჭვის მიმართაც, სანამ არ დასრულდება ეს პროცესი. თუ შედეგად მივიღეთ ჯაჭვი, რომელიც მხოლოდ ტერმინალურ სიმბოლოებს შეიცავს, მაშინ მას ვაცხადებთ L ენის x ჯაჭვად. ყველა ასეთი x ჯაჭვების ერთობლიობა განსაზღვრავს L ენას და მას ეწოდება ზემოთ აღწერილი გრამატიკით განსაზღვრული ენა. ზემოთ აღწერილი გრამატიკა ფორმალურად განისაზღვრება შემდეგნაირად: G=(N,Σ,P,S) ოთხეულს ვუწოდებთ გრამატიკას, სადაც

1. N არატერმინალური სიმბოლოების სასრული სიმრავლეა;

2. Σ - ტერმინალურ სიმბოლოთა სასრული სიმრავლეა და $N \cap \Sigma = \emptyset$ (N-ის გადაკვეთა Σ-სთან ცარიელი სიმრავლეა);

3. P არის $(N \cup \Sigma)^* N (N \cup \Sigma)^* X (N \cup \Sigma)^*$ სიმრავლის სასრული ქვესიმრავლე;

4. $S \in N$ გამოყოფილი სიმბოლოა, რომელსაც საწყისი სიმბოლო ეწოდება.

G გრამატიკა განსაზღვრავს L(G) ენას, რომელსაც G გრამატიკით განსაზღვრული ენა ეწოდება. შემოვიტანოთ რეკურსიულად სპეციალური სახის ჯაჭვები, რომელთაც G გრამატიკის გამოყვანადი ჯაჭვები ეწოდებათ:

1. S - გამოყვანადი ჯაჭვია;

2. თუ $\alpha\beta\gamma$ - გამოყვანადი ჯაჭვია და $\beta \to \delta$ ელემენტია P-სი, მაშინ $\alpha\delta\gamma$ გამოყვანადი ჯაჭვია აგრეთვე.

გამოყვანადი ჯაჭვი, რომელიც არ შეიცავს არატერმინალურ სიმბოლოებს ეწოდებათ G გრამატიკით წარმოქმნილი ტერმინალური ჯაჭვები. ტერმინალურ ჯაჭვთა ერთობლიობა ქმნის G გრამატიკით წარმოქმნილ L ენას და აღინიშნება L(G)-ით. ახლა შემოვიტანოთ $\Rightarrow$ დამოკიდებულება იმისათვის, რომ მარტივად ჩავწეროთ L(G) ენა. თუ $\alpha_1\beta\alpha_2$ გამოყვანადი ჯაჭვია G გრამატიკაში და $\beta\rightarrow\gamma \in P$, მაშინ $\alpha_1\beta\alpha_2 \Rightarrow\alpha_1\gamma\alpha_2$.

$\Rightarrow$ დამოკიდებულების ტრანზიტული ჩაკეტვა ავღნიშნოთ $\Rightarrow^+$-ით, ხოლო ტრანზიტული და რეფლექსური ჩაკეტვა $\Rightarrow^*$-თი. ამის შემდეგ L(G) ენა შეიძლება ჩაიწეროს შემდეგნაირად: $L(G)=\{w|w\in\sum^*, S\Rightarrow^*w\}$

განვიხილოთ გრამატიკის მაგალითი, რომელიც განსაზღვრავს არითმეტიკულ გამოსახულებათა ენას, შეიცავს არითმეტიკულ ოპერაციებს შეკრებას და გამრავლებას, იდენტიფიკატორს a-ს და მრგვალ ფრჩხილებს. ვთქვათ,

$G_1= (\{E,T,F\},\{a,+,*,(,)\},P,E)$,

სადაც P შეიცავს წესებს:

$E\rightarrow E+T|T$

$T\rightarrow T*E|F$

$F\rightarrow (E)|a$

მაგალითად, ჯაჭვი (a+a)*a ეკუთვნის $L(G_1)$-ს, რის ჩვენებაც მარტივია შემდეგი გამოყვანით: $E\Rightarrow T\Rightarrow T*F \Rightarrow F*F\Rightarrow (E)*F\Rightarrow (E+T)*F\Rightarrow (T+T)*F\Rightarrow (F+T)*F\Rightarrow (a+T)*F\Rightarrow (a+F)*F \Rightarrow (a+a)+F\Rightarrow (a+a)*a$.

შევნიშნოთ, რომ არსებობს სხვა გამოყვანებიც, რომლებიც მოგვცემენ იგივე შედეგს. ისინი მიიღებიან გამოყენებული წესების რიგის შეცვლით, როცა არსებობს რამოდენიმე ალტერნატივა. ეს გრამატიკა საინტერესოა იმით, რომ თუ ჩვენ მას დავუმატებთ ახალ წესებს სხვა

არითმეტიკული ოპერაციებისათვის მათი პრიორიტეტების გათვალისწინებით, შეიძლება მივიღოთ გრამატიკა პროგრამირების ფართოდ ცნობილ ენებში არითმეტიკული გამოსახულების განსასაზღვრავად. ამ მაგალითში ჩვენ გამოვიყენეთ წესების შემოკლებული ჩაწერა. საზოგადოდ, თუ ჩვენ გვაქვს წესები:

$\alpha \rightarrow \beta_1$

$\alpha \rightarrow \beta_2$

$\alpha \rightarrow \beta_n$

შემოკლებით იგი ასე ჩაიწერება:

$\alpha \rightarrow \beta_1|\beta_2|\ldots|\beta_n$.

1.3 გრამატიკათა ხომსკის კლასიფიკაცია. ვთქვათ, მოცემულია გრამატიკა $G=(N,\Sigma,P,S)$. თუ ჩვენ P-ს წესებს დავადებთ გარკვეულ შეზღუდვებს, მივიღებთ სხვადასხვა ტიპის გრამატიკებს, რომლებიც ფართოდ ცნობილია ფორმალურ გრამატიკათა თეორიაში და შემოტანილია ხომსკის მიერ. გრამატიკას ეწოდება მარჯვნივ წრფივი, თუ მის ყოველ წესს აქვს სახე:

$A \rightarrow XB$ ან $A \rightarrow X$, სადაც $A,B \in N$ და $X \in \Sigma^*$

გრამატიკას ეწოდება კონტექსტისაგან თავისუფალი ან კონტექსტისადმი არამგრძნობიარე თუ მის ყოველ წესს აქვს სახე: $A \rightarrow \alpha$, სადაც $A \in N$, $\alpha \in (N \cup \Sigma)^*$. გრამატიკას ეწოდება კონტექსტზე დამოკიდებული ან კონტექსტისადმი მგრძნობიარე, ზოგჯერ არადამოკლებადი თუ მის ყოველ წესს აქვს სახე $\alpha \rightarrow \beta$, სადაც $V\alpha V \leq V\beta V$.

თვითეული ტიპის გრამატიკა განსაზღვრავს გარკვეული კლასის ენებს, რომელთაც გარკვეული თვისებები გააჩნიათ. საზოგადოდ შევნიშნოთ, რომ კონკრეტული L ენა განისაზღვრება ცალსახად რაიმე გრამატიკით, ე. ი. თუ L ენა განისაზღვრება G გრამატიკით ეს იმას არ

ნიშნავს, რომ არ არსებობს რაიმე G_1 გრამატიკა G გრამატიკისაგან განსხვავებული, რომელიც იგივე L ენას განსაზღვრავს. მიუხედავად ამისა თუ L ენა განისაზღვრება რაიმე ტიპის გრამატიკით, ჩვენ შეგვიძლია ვილაპარაკოთ L ენის გარკვეულ თვისებებზე. დავიწყოთ უმარტივესი გრამატიკებით მარჯვენივ წრფივი გრამატიკებით და ვაჩვენოთ, რომ მათ მიერ განსაზღვრული ენები არიან რეგულარული სიმრავლეები და პირიქით ნებისმიერი რეგულარული სიმრავლე განისაზღვრება რაიმე მარჯვნივ წრფივი გრამატიკით.

1.4 **რეგულარული სიმრავლეები.** ვთქვათ მოცემულია Σ სასრული ალფაბეტი. რეგულარული სიმრავლე Σ ალფაბეტში რეკურსიულად განისაზღვრება შემდეგნაირად:

1. ცარიელი სიმრავლე $\emptyset$ არის რეგულარული სიმრავლე Σ ალფაბეტში;
2. {e} არის რეგულარული სიმრავლე Σ ალფაბეტში;
3. {a} არის რეგულარული სიმრავლე Σ ალფაბეტში, სადაც $a \in \Sigma$;
4. თუ P და Q რეგულარული სიმრავლეებია Σ ალფაბეტში, მაშინ რეგულარული სიმრავლეებია აგრეთვე $P \cup Q$, PQ და P^*
5. რეგულარული სიმრავლეებია, მხოლოდ ის სიმრავლეები, რომლებიც მიიღებიან 1 - 4 წესებით.

რეგულარული სიმრავლეები გამოისახებიან რეგულარული გამოსახულებებით. რეგულარული გამოსახულება Σ ალფაბეტში მოიცემა შემდეგი რეკურსიული განსაზღვრებებით:

1. $\emptyset$ აღნიშნავს რეგულარულ სიმრავლეს $\emptyset$
2. e აღნიშნავს რეგულარულ სიმრავლეს {e}.
3. a აღნიშნავს რეგულარულ სიმრავლეს {a}, სადაც

$a \in \Sigma$.

4. თუ p და q რეგულარული გამოსახულებებია, რომლებიც აღნიშნავენ რეგულარულ სიმრავლეებს P და Q შესაბამისად, მაშინ (p+q), (pq) და (p)* აგრეთვე რეგულარული გამოსახულებებია, რომლებიც აღნიშნავენ რეგულარულ სიმრავლეებს PUQ, PQ და P* შესაბამისად.

5. რეგულარული გამოსახულებებია მხოლოდ ის გამოსახულებები, რომლებიც მიიღებიან 1 – 4 წესებით.

რეგულარულ გამოსახულებათათვის α, β და γ სამართლიანია შემდეგი ტოლობები, რომელთა დამტკიცება ადვილია:

$\alpha+\beta=\beta+\alpha$

$\alpha+(\beta+\gamma)=(\alpha+\beta)+\gamma$

$\alpha(\beta+\gamma)=\alpha\beta+\alpha\gamma$

$\alpha e=e\alpha=\alpha$

$\emptyset^{\bullet}=e$

$\alpha^{\bullet}=\alpha+\alpha^{\bullet}$

$\alpha(\beta\gamma)=(\alpha\beta)\gamma$

$\alpha+\alpha=\alpha$

$(\alpha+\beta)\gamma=\alpha\gamma+\beta\gamma$

$\emptyset\alpha=\alpha\emptyset=\emptyset$

$(\alpha^{\bullet})^{\bullet}=\alpha^{\bullet}$

$\alpha+\emptyset=\alpha$

1.5 კავშირი რეგულარულ სიმრავლეებსა და მარჯვნივწრფ გრამატიკებს შორის. არსებობს მჭიდრო კავშირი რაგულარულ სიმრავლეებსა და მარჯვნივწრფიე გრამატიკებს შორის. სამართლიანია დებულება, რომ ნებისმიერი P რეგულარული სიმრავლისათვის შეიძლება აიგოს შესატყვისი მარჯვნივწრფივი გრამატიკა G ისეთი, რომ L(G)=P და პირიქით, ნებისმიერი G მარჯვნივწრფივი

გრამატიკით განსაზღვრული ენა L(G) არის რეგულარული სიმრავლე. ამ დებულების სრული დამტკიცება იხილეთ [1]-ში. ჩვენ აქ მხოლოდ ამ დებულების პირველ ნაწილს განვიხილავთ, რომელიც ხშირად გამოიყენება პრაქტიკაში მოცემული P რეგულარული სიმრავლით G მარჯვნივწრფივი გრამატიკის ასაგებად, რომლისთვისაც L(G)=P. განვიხილოთ რეგულარული სიმრავლეები: Ø, {e} და {a} a∈Σ და ავაგოთ მათთვის მარჯვნივწრფივი გრამატიკები, რომლებიც ამ სიმრავლეებს განსაზღვრავენ.

1. განვიხილოთ მარჯვნივწრფივი გრამატიკა G=({S},Σ,Ø,S) ცხადია L(G)= Ø.

2. განვიხილოთ G=({S,Σ,{S→e},S) ეს გრამატიკა მარჯვნივწრფივია და ცხადია L(G)={e}.

3. განვიხილოთ G=({S}, Σ ,{S→ a},S). იგულისხმება a∈ Σ ცხადია L(G)={a} და G მარჯვნივწრფივი გრამატიკაა.

ახლა ვაჩვენოთ, თუ L_1 და L_2 მარჯვნივწრფივი ენებია, მაშინ:

1. $L_1 \cup L_2$;
2. $L_1 L_2$;
3. L_1^*

მარჯვნივწრფივი ენებია.

1. ვთქვათ, L_1 მარჯვნივ წრფივ ენას განსაზღვრავს. მარჯვნივ წრფივი გრამატიკა $G_1=(N_1,\Sigma,P_1,S_1)$ და L_2 მარჯვნივწრფივ ენას განსაზღვრავს $G_2=(N_2,\Sigma,P_2,S_2)$. ავაგოთ ახალი მარჯვნივწრფივი გრამატიკა $G_3=(N_1 \cup N_2 \cup\{S_3\}, P_1 \cup P_2 \{S_3 \rightarrow S_1 | S_2\},S_3)$. ვაჩვენოთ, რომ $L(G_3)=L(G_1) \cup L(G_2)$. ყოველი $S_3 \Rightarrow^* W$-სათვის გრამატიკაში

G_3 არსებობს გამოყვანა $S_1 \Rightarrow {}^*W$ G_1-ში და $S_2 \Rightarrow {}^*W$ G_2–ში ე. ი. $W \in L_1 \cup L_2$.. რადგანაც $S_3 \Rightarrow {}^*W$ S_3-ში ნიშნავს $S_3 \Rightarrow S_1 \Rightarrow {}^*W$ ან $S_3 \Rightarrow S_2 \Rightarrow {}^*W$ G_3-ში.

2. ვთქვათ $G_3=(N_1 \cup N_2, \Sigma, P_3, S_1)$ მარჯვნივწრფივი გრამატიკაა, სადაც P_3 განსაზღვრულია შემდეგნაირად:

a). თუ $A \to xB$ ეკუთვნის P_1-ს, მაშინ იგი ეკუთვნის ასევე P_3-ს. b).თუ $A \to X$ ეკუთვნის P_1-ს, მაშინ $A \to xS_2$ ეკუთვნის P_3-ს. c) ყველა წესები P_2 –დან ეკუთვნის P_3-ს.

ცხადია, რომ თუ $S_1 \Rightarrow {}^+W$ G_1–ში, მაშინ $S_1 \Rightarrow {}^+WS_2$ G_3–ში. თუ $S_2 \Rightarrow {}^+V$ G_2-ში, მაშინ $S_1 \Rightarrow {}^+WV$ G_3-ში. ე.ი. $L(G_1)L(G_2) \subseteq L(G_3)$.

ახლა ვთქვათ $S_1 \Rightarrow {}^+W$ G_3-ში. რადგან G_3-ში არა გვაქვს $A \to X$ წესი, ამიტომ $S_1 \Rightarrow {}^+x$ $S_2 \Rightarrow {}^+xy=W$, სადაც $x \in L_1$ და $y \in L_2$. ამგვარად, $L(G_3)$ არის $L(G_1)L(G_2)$ და საბოლოოდ $L(G_3)=L(G_1)L(G_2)$.

3. ვთქვათ $G_a=(N_1 \cup S_a, \Sigma, P_a, S_a)$, სადაც S_a არ ეკუთვნის N_1-ს და P_a აგებულია შემდეგნაირად:

თუ $A \to xB$ ეკუთვნის P_1-ს, მაშინ იგი ეკუთვნის ასევე P_a -ს.

თუ $A \to x$ ეკუთვნის P_1-ს მაშინ $A \to X$ S_a და $A \to X$ ეკუთვნიან P_a–ს.

ცხადია, $S_a \Rightarrow {}^+x_1$ $S_a \Rightarrow {}^+x_1x_2$ $S_a \Rightarrow {}^+ \ldots \Rightarrow {}^+ x_1x_2 \ldots x_n$ $S_a \Rightarrow x_1x_2 \ldots x_n$ მიიღება G_a–ში, მაშინ და მხოლოდ მაშინ, როცა $S_1 \Rightarrow x_1$, $S_1 \Rightarrow {}^+x_2, \ldots,$ $S_1 \Rightarrow {}^+ x_n$ G_1-ში. ეს კი ნიშნავს, რომ $L(G_a)=(L(G_1))^*$.

1.6 სასრული ავტომატები. სასრული ავტომატი არის რეგულარული სიმრავლის წარმოდგენის ერთ-ერთი საშუალება. მისი საშუალებით ჩვენ შეგვიძლია იოლად

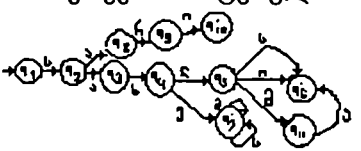
წარმოვადგინოთ ნებისმიერი რეგულარული სიმრავლე კომპიუტერზე. სასრული ავტომატი ჩვენ შეგვიძლია ავაგოთ როგორც მოწყობილობა, რომელსაც აქვს შესასვლელი ფირი და მმართველი მოწყობილობა სასრული მეხსიერებით. ფირზე მოდებულია თავაკი, რომელსაც შეუძლია წაიკითხოს ფირიდან მოცემულ მომენტში ერთი უჯრედი(ის უჯრედი, რომელსაც უთითებს თავაკი) და თავაკს შეუძლია გადაადგილება ერთი უჯრედიდან მის მეზობელ მეორე უჯრედზე. შესასვლელი ფირი დაყოფილია უჯრედებად და თვითეულ უჯრედში შეიძლება იყოს ჩაწერილი ალფაბეტის ერთი რომელიმე სიმბოლო. სიმარტივისათვის ჩვენ ვიგულისხმებთ, რომ თავაკს შეუძლია მოძრაობა ფირზე მარცხნიდან მარჯვნივ მიმართულებით. ასევე ვიგულისხმოთ, რომ ფირს აქვს უსასრულო სიგრძე, სადაც სიგრძის ქვეშ იგულისხმება უჯრედების რაოდენობა ფირზე. მმართველი მოწყობილობა შედგება მდგომარეობათა სასრული რაოდენობისაგან. მდგომარეობას ჩვენ ვუწოდებთ კვანძს და ჩვენი მოწყობილობა მოცემულ მომენტში შეიძლება იმყოფებოდეს ერთ რომელიმე კვანძში და ამ მოწყობილობას შეუძლია გადაადგილდეს ერთი კვანძიდან მეორეში იმისდამიხედვით თუ რა სიმბოლოა მოდებული თავაკზე. თავდაპირველად ვიგულისხმოთ, რომ მოწყობილობა იმყოფება რაიმე კვანძში და მას ვუწოდოთ საწყისი კვანძი (მდგომარება).

ასევე მოწყობილობაში მონიშნულია რაიმე კვანძების ქვესიმრავლე, რომლებსაც ეწოდება საბოლოო მდგომარეობები. ასეთ მოწყობილობას ეწოდება სასრული ავტომატი. უჯრედში შეიძლება ჩაწერილი იყოს რაიმე სიმბოლო, წინააღმდეგ შემთხვევაში ასეთ უჯრედს

ეწოდება ცარიელი უჯრედი. ცარიელი უჯრედის შემთხვევაში ჩვენ ვამბობთ, რომ მასში ჩაწერილია ცარიელი სიმბოლო. სიმბოლოების სიმრავლეს, რომლებიც შეიძლება შეგვხვდნენ ფირის უჯრედებში ვუწოდებთ ალფაბეტს. ცარიელი სიმბოლო არ ეკუთვნის ალფაბეტს.სიმბოლოების სასრულ მიმდევრობას ვუწო-დოთ სიტყვა. ჩვენ ვიტყვით, რომ რაიმე სიტყვა მოდებულია სასრულ ავტომატზე, თუ ავტომატი იმყოფება საწყის მდგომარეობაში და თავაკი უთითებს სიტყვის პირველ სიმბოლოს ფირზე. სასრული ავტომატი გრაფიკულად ჩვენ შეგვიძლია წარმოვიდგინოთ შემდეგნაირად: სასრული ავტომატის მდგომარეობები აღვნიშნოთ წერტილებით, რომლებიც მონიშნულია $q_1,\ldots,q_n$ ნიშნებით. ეს ნიშნები არ ეკუთვნიან ალფაბეტს. q_1 მდგომარეობა შევაერთოთ q_j მდგომარეობასთან რკალით, რომელიც მონიშნულია რაიმე a_k სიმბოლოთი $a_k \in \Sigma$ (Σ აღნიშნავს ალფაბეტს) და რკალი მთავრდება ისრით, რომელიც მიმართულია q_j მდგომარეობისაკენ. თუ ავტომატი იმყოფება q_i მდგომარეობაში და თავაკი უთითებს a_k სიმბოლოს, მაშინ სასრულ ავტომატს შეუძლია გადავიდეს q_j მდგომარეობაში და თავაკი გადაადგილდება ერთი უჯრედით მარჯვნივ. წყვილს (q_j,W) ვუწოდოთ სასრული ავტომატის კონფიგურაცია, სადაც q_j მიმდინარე მდგომარეობაა და W არის სტრიქონი, რომელიც სასრულ ავტომატს ჯერ არ წაუკითხავს ფირიდან, ე. ი. ფირზე ჩაწერილია $W=a_{i1}\ldots a_{ik}$ სტრიქონი და თავაკი უთითებს a_{i1}–ს. ვთქვათ q_1 სასრული ავტომატის საწყისი მდგომარეობაა, $W_1=a_1\ldots a_n$ ფირზე ჩაწერილი სტრიქონია და სასრული ავტომატი საწყის მდგომარეობაში უთითებს a_1 სიმბოლოს,

მაშინ საწყისი კონფიგურაცია იქნება (q_1,W_1).

განვიხილოთ კონფიგურაციათა შემდეგი მიმდევრობა (q_1, W_1), $(q_2,W_2),\ldots,(q_n,W_n)$, სადაც $W_i=a_i\ldots a_n$ $(i=1,\ldots,n)$. თუ q_n საბოლოო მდგომარეობაა, მაშინ ჩვენ ვიტყვით, რომ სასრული ავტომატი უშვებს W_1 სტრიქონს. განვიხილოთ დამოკიდებულება $(q_i,W_i) \vdash (q_{i+1}, W_{i+1})$, სადაც $(i=1,2,\ldots,n-1)$. ჩანაწერი $(q_i,W_i) \vdash (q_{i+1},W_{i+1})$ აღნიშნავს სასრული ავტომატის q_i მდგომარეობიდან q_{i+1} მდგომარეობაში გადასვლას a_i სიმბოლოთი. ეს იმას ნიშნავს, რომ სასრული ავტომატის დიაგრამაზე q_i კვანძი შეერთებულია q_{i+1} კვანძთან რკალით ისრით q_{i+1}–კენ და ეს რკალი მონიშნულია a_i სიმბოლოთი. ჩანაწერი $(q_1,W_1) \vdash^*(q_n,W_n)$ აღნიშნავს შემოტანილი დამოკიდებულების ტრანზიტულ-რეფლექსურ ჩაკეტვას. გაშლილი სახით იგი ასე ჩაიწერება $(q_1,W_1) \vdash (q_2,W_2) \vdash \ldots \vdash (q_n,W_n)$ თუ q_1 საწყისი მდგომარეობაა და q_n საბოლოო მდგომარეობა, მაშინ ვიტყვით რომ $W_1=a_1\ldots a_n$ სტრიქონს უშვებს მოცემული სასრული ავტომატი, ან რაც იგივეა $W_1 \in L(A)$, სადაც A მოცემული ავტომატია და L(A) არის ამ ავტომატით განსაზღვრული ენა. ყველა ასეთი W_1-ები ქმნიან L(A) ენას $L(A)=\{ W_1 | (q_1,W_1) \vdash^*(q_n,W_n)\}$, სადაც q_n ნებისმიერი საბოლოო მდგომარეობაა. განვიხილოთ მაგალითი. ვთქვათ მოცემულია სასრული დიაგრამა:



1 დიაგრამა

ისრით მითითებულია საწყისი მდგომარეობა, ხოლო კვანძები, რომლებიც აღნიშნულია O წრეებით, არიან საბოლოო მდგომარეობები. მაშასადამე q_1 საწყისი მდგომარეობაა, ხოლო q_6, q_7 და q_{10} საბოლოო მდგომარეობები. თუ ამ დიაგრამით განსაზღვრულ ავტომატს აღვნიშნავთ A-თი, მაშინ L(A)={სახლი, სახლმა, სახლს, სახე, სახემ^n, სახეს^n, სარი}. აქ n არის ნებისმიერი ნატურალური რიცხვი. შევნიშნოთ, რომ q_7 მდგომარეობიდან ს და მ სიმბოლოებით ავტომატი ისევ q_7 მდგომარეობაში რჩება, ხოლო q_2 მდგომარეობიდან ს სიმბოლოთი სასრულ ავტომატს შეუძლია გადავიდეს q_3 მდგომარეობაში ან q_8 მდგომარეობაში. ამიტომ ეს ავტომატი უშვებს სახემ, სახემმ და ა. შ. შემოვიტანოთ განმარტება: თუ სასრულ ავტომატს რაიმე მდგომარეობიდან ერთი და იგივე სიმბოლოთი შეუძლია ერთზე მეტ მდგომარეობაში გადასვლა, მაშინ ასეთ სასრულ ავტომატს ეწოდება არადეტერმინისტული, წინააღმდეგ შემთხვევაში დეტერმინისტული. დიაგრამაზე გამოსული სასრული ავტომატი არადეტერმინისტულია.

ამის შემდეგ, შემოვიტანოთ სასრული ავტომატის მკაცრი განმარტება. $A=(Q,\Sigma,\delta,q_0,F)$ არის სასრული ავტომატი, სადაც

Q არის მდგომარეობათა სასრული სიმრავლე,

Σ - შესასვლელ სიმბოლოთა სასრული სიმრავლეა,

q_0 – საწყისი მდგომარეობაა და $q_0 \in Q$,

F – საბოლოო მდგომარეობათა სიმრავლეა და $F \subseteq Q$,

δ არის $Q \times \Sigma$ დეკარტული ნამრავლის გადასახვა (Q)–ში. ე.ი. სიმრავლეში, რომლის ელემენტებია Q–ს ქვესიმრავლეები.

მოცემული განმარტებით წინა დიაგრამა წარმოიდგინება ასე:

$A=(\{q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_{10},q_{11}\},\{ს,ა,რ,ხ,ი,მ,ლ,ე\},\delta,$
$q_1,\{q_6,q_7,q_{10}\})$,

სადაც $\delta(q_1,ს)=\{q_2\}$, $\delta(q_2,ა)=\{q_3,q_8\}$, $\delta(q_3,ხ)=\{q_4\}$, $\delta(q_4,ლ)=\{q_5\}$, $\delta(q_5,ი)=\{q_6\}$ $\delta(q_5,მ)=\{q_{11}\}$, $\delta(q_{11},ა)=\{q_6\}$, $\delta(q_4,ე)=\{q_7\}$, $\delta(q_7,ს)=\{q_7\}$, $\delta(q_8,რ)=\{q_9\}$ და $\delta(q_9,ი)=\{q_{10}\}$. ამ სასრული ავტომატის არადეტერმინისტულობაზე მიუთითებს ის, რომ $\delta(q_2,a)$ სიმრავლე შეიცავს ორ ელემენტს.

ახლა ჩვენ შეგვიძლია სასრული ავტომა-ტის კონფიგურაცია განვმარტოთ ასე: $(q,W)\in Q\times\Sigma^*$, საწყისი კონფიგურაცია იქნება (q_0,W) და დამამთავრებელი კონფიგურაცია იქნება (q,e), სადაც $q\in F$. ბინარული დამოკიდებულება $\vdash$ განისაზღვრება ასე: თუ $\delta(q,a)$ სიმრავლე შეიცავს q_1-ს, მაშინ (q,aW_1). ეს ნიშნავს იმას, რომ თუ A სასრული ავტომატი იმყოფება q მდგომარეობაში და თავაკი მიუთითებს უჯრედს, რომელ-შიც ჩაწერილია a სიმბოლო, მაშინ სასრული ავტომატი გადადის q_1 მდგომარეობაში და თავაკი გადაადგილდება ერთი უჯრედით მარჯვნივ. ჩვენ ვიტყვით, რომ A სასრული ავტომატი უშვებს W სტრიქონს, როცა $(q_0,W)\vdash^*(q.e)$ და $q\in F$. ასეთ შემთხვევაში $W\in L(A)$.

1.7 სასრული ავტომატის ცხრილური წარმოდგენა. ჩვენ შეგვიძლია წინა მაგალითიდან აღებული სასრული ავტომატი გამოვსახოთ შემდეგი ცხრილის სახით:

δ	შესასვლელი							
	ს	ა	რ	ხ	ი	მ	ლ	ე
q_1	$\{q_2\}$							
q_2		$\{q_3,q_8\}$						
q_3				$\{q_4\}$				
q_4							$\{q_5\}$	$\{q_7\}$
q_5					$\{q_6\}$	$\{q_{11}\}$		
q_6						$\{q_7\}$		
q_7	$\{q_7\}$							
q_8			$\{q_9\}$					
q_9					$\{q_{10}\}$			
q_{10}								
q_{11}		$\{q_6\}$						

დიაგრამა 1

ეს არის A სასრული ავტომატის ცხრილური წარმოდგენა.

I დიაგრამა-ზე წარმოდგენილი A სასრული არადეტერმინისტული ავტომატი გადავაქციოთ სასრულ დეტერმინისტულ ავტომატად A_1 ისე, რომ $L(A)=L(A_1)$. ამისათვის საჭიროა, რომ q_8 დავამთხვიოთ q_3-ს და q_3-დან უნდა გამოდიოდეს ის რკალები, რომლებიც გამოდიოდა q_8-დან. ამის შემდეგ q_8 საერთოდ ამოვარდება მდგომარეობათა სიმრავლიდან. ასეთნაირად მიღებული ახალი სასრული ავტომატი A_1 იქნება დეტერმინისტული და $L(A)=L(A_1)$.საზოგადოდ, მტკიცდება თეორემა, რომელიც ამბობს, რომ ნებისმიერი არატერმინსტული A სასრული ავტომატი შეიძლება გარდაიქმნას ახალ დეტერმინისტულ სასრულ ავტომატად ისე, რომ $L(A)=L(A_1)$. დამტკიცება იხილეთ [1]-ში. რადგანაც დეტერმინისტული სასრული ავტომატის შემთხვევაში (q,a) სიმრავლე, ყოველი $a\in\Sigma$ და $q\in Q$-სთვის, არ შეიცავს ერთზე მეტ ელემენტს, ამიტომ ნაცვლად $(q,a)=\{p\}$ შეგვიძლია შემოკლებით დავწეროთ $(q,a)=p$ და როცა $(q,a)=\varnothing$ (ცარიელი სიმრავლეა), მაშინ ვიტყვით, რომ (q,a) განუსაზღვრელია.

შეიძლება მარტივად დამტკიცდეს შემდეგი დებულება: თუ რაიმე დეტერმინისტული სასრული ავტომატი M უშვებს L(M) ენას, მაშინ არსებობს ისეთი მარჯვნივწრფივი გრამატიკა G, რომ $L(G)=L(M)$. ვთქვათ $M=(Q,\Sigma,\delta,q_0,F)$ და $G=(Q,\Sigma,P,q_0)$ სადაც P განისაზღვრება შემდეგნაირად: თუ $(q,a)=r$, მაშინ P შეიცავს წესს $q\to ar$ და თუ $q_1\in F$, მაშინ $q\to e$ ეკუთვნის P-ს. ასეთნაირად აგებული G გრამატიკა უშვებს იგივე ენას, რასაც M დეტერმინისტული სასრული ავტომატი, რის დამტკიცებაც ადვილია. პირიქით ზემოთ მოყვანილი წესების შებრუნებით გამოყენებით მარჯვნივწრფივი გრამატიკისათვის

შეიძლება აიგოს დეტერმინისტული სასრული ავტომატი, რომელიც განსაზღვრავს იგივე ენას რასაც მოცემული მარჯვნივწრფივი გრამატიკა. ასევე შეიძლება ნებისმიერი რეგულარული სიმრავლისათვის აიგოს ისეთი დეტერმინისტული სასრული ავტომატი, რომელიც უშვებს მოცემულ რეგულარულ სიმრავლეს. აქედან გამომდინარეობს, რომ სასრული ავტომატებითა და მარჯვნივწრფივი გრამატიკებით განსაზღვრული ენები არიან რეგულარული სიმრავლეები. ვისაც აინტერესებს სასრული ავტომატების მინიმიზაციის საკითხი (გარკვეული აზრით) და რეგულარულ სიმრავლეებთან დაკავშირებული გადაწყვეტადი პრობლემები ეურჩევთ [1]-ს.

1.8 რეგულარული გამოსახულება "Perl"-ში. რეგულარული გამოსახულებები ფართოდ გამოიყენებიან Perl ენაზე დაწერილ პროგრამებში. ამ თავში ჩვენ აღვწერთ რეგულარულ გამოსახულებებს Perl–ის მიხედვით. რეგულარული გამოსახულებების შემოტანის წყალობით Perl არის ბუნებრივი ტექსტების დამუშავების მძლავრი საშუალება. აქამდე აღწერილი რეგულარული გამოსახულებები მნიშვნელოვნად არის გაფართოებული Perl-ში. Perl-ის რეგულარული გამოსახულება არის რაიმე სტრიქონი, რომელიც აღწერს რაიმე ნიმუშს. ნიმუშები გამოიყენებიან ტექსტში კონკრეტული სტრიქონის მოსაძებნად, მის შესაცვლელად სხვა სტრიქონით, ტექსტიდან მისი ამოსაგდებისათვის, ან ტექსტის უფრო რთული გარდაქმნისათვის.

ჩვენ დავიწყებთ უმარტივესი რეგულარული გამოსახულების განსაზღვრით, ვაჩვენებთ მის გამოყენებას და შემდეგში თანდათანობით გავაფართოებთ ამ

ცნებას. უმარტივესი რეგულარული გამოსახულებაა ნიშნების რაიმე მიმდევრობა რაიმე წინასწარ მოცემულ ალფაბეტში. რეგულარული გამოსახულება შემოსაზღვრული დახრილი ხაზებით(/). მაგალითად, /ab1/ არის ნიმუში, რომელიც შეიცავს ab1 რეგულარულ გამოსახულებას ASCII სტანდარტით განსაზღვრულ ალფაბეტში. იმისათვის, რომ განვსაზღვროთ რაიმე ნიმუში ეთანადება თუ არა მოცემულ ტექსტს, ამისათვის გამოიყენება ოპერატორი = ~, რომლის მარჯვენა მხარეში იწერება ნიმუში, ხოლო მარცხენა მხარეში იწერება ტექსტი ბრჭყალებში ან ცვლადი, რომელიც შეიცავს ტექსტს. =~ ოპერატორი წარმატებით სრულდება ან რაც იგივეა ეთანადება შაბლონს თუ შაბლონით განსაზღვრული რეგულარული გამოსახულება გვხვდება ამ ოპერატორის მარცხენა მხარით განსაზღვრულ ტექსტში, წინააღმდეგ შემთხვევაში ოპერატორი განიცდის მარცხს (შაბლონი არ ეთანადება ტექსტს). მაგალითად, ინსტრუქცია " this is a text " = ~ /is/;

წარმატებით სრულდება, რადგანაც is რეგულარული გამოსახულება პირველად გვხვდება this სიტყვაში. ოპერატორი !~ წარმატებით სრულდება თუ მის მარჯვენა მხარეში მოთავსებული გამოსახულება არ გვხვდება მარცხენა მხარეში.

მაგალითად, " this is a text " = ~ /at / არ ეთანადება;

" this is atext " = ~ /at / ეთანადება.

ალფაბეტის ნიშნები რეგულარულ გამოსახულებაში იწერება ისე, როგორც ისინი გვხვდება ალფაბეტში, გამონაკლისს წარმოადგენენ ზოგიერთი ნიშნები, რომლებიც გამოიყენებიან რეგულარულ გამოსახულებაში სპეციალური დანიშნულებით. ასეთ ნიშნებს ეწოდებათ

მეტანიშნები. მეტანიშნებია: [,],(,),{,},^,+,*,?. იმისათვის რომ რაიმე მეტანიშანი გაგებული იყოს როგორც ჩვეულებრივი ნიშანი, მას წინ უნდა დავუწეროთ \(შებრუნებული დახრილი ხაზი). მაგალითად \[გაიგება როგორც გახსნილი კვადრატული ფრჩხილი ან \+ გაიგება როგორც + ნიშანი და ა.შ..

მაგალითები: "3+5=8"=~/3+5/ არ ეთანადება.

"3+5=8"=~/3\+5/ ეთანადება.

"a.b*c"=~/a.b*c/ არ ეთანადება.

"a.b*c"=~/a\.b*c/ ეთანადება.

\\ რეგულარულ გამოსახულებაში აღნიშნავს \(შებრუნებულ დახრილ ხაზს). ASCII ნიშანთა სიმრავლის ნიშნები, რომელთა კოდებია 0-31, აქვთ სპეციალური აღნიშვნები. მაგალითად, \n აღნიშნავს ახალ სტრიქონზე გადასვლას, \t აღნიშნავს ტაბულაციას და სხვა. ეს აღნიშნები შეიძლება გამოყენებულ იქნეს ასევე რეგულარულ გამოსახულებებშიც. ნიშნის ნაცვლად შესაძლებელია გამოყენებულ იქნეს მისი კოდი თექვსმეტობით ან რვაობით სისტემაში. თექვსმეტობითი კოდი წარმოიდგინება \oxn$_1$n$_2$, სადაც n$_1$ და n$_2$ თექვსმეტობითი ციფრებია, ან \o n$_1$ n$_2$ n$_3$ როგორც რვაობითი კოდი, სადაც n$_1$ n$_2$ და n$_3$ რვაობითი ციფრებია, სადაც o არის ო ასო და არა ნული. მაგალითად \ ox f 5 ან \o125. თვით \(შებრუნებული დახრილი ხაზი) არის მეტანიშანი და მის წარმოსადგენად, როგორც ჩვეულებრივი ნიშანი საჭიროებს ნიშანს \. ასე, რომ \\ არის დახრილი ხაზის ჩვეულებრივ ნიშნად წარმოდგენა. მაგალითად, 'a\b'=~/a\\b/ ეთანადება, ე.ი. წარმატებით შესრულდება. მაგალითად, მივიღოთ, რომ თარიღს წარმოვადგენთ

ტექსტში, როგორც თვე:რიცხვი:წელი, სადაც თვისთვის და რიცხვისთვის გამოყოფილია ორი ათობითი ციფრი, ხოლო წლისათვის ოთხი ათობითი რიცხვი: მაგალითად, 09:28:1973 ნიშნავს 28 სექტემბერი 1973 წელი. ასეთ შემთხვევაში ტექსტში რომ ვიპოვოთ რაიმე თარიღი საჭიროა მისთვის შევადგინოთ შემდეგი ნიმუში: /dd:dd:dddd/ სადაც d ნიშნავს ნებისმიერ ათობით ციფრს. იგულისხმება, რომ თვე მოთავსებულია დიაპაზონში 01-12 და თვე 01-31. ეს, რომ გავითვალისწინოთ, მაშინ დაგვჭირდება უფრო რთული ნიმუშის შედგენა.

რეგულარულ გამოსახულებაში შეიძლება გექონდეს ცვლადებიც. Perl-ში სკალარული ცვლადის სახელი იწყება $ ნიშნით. მაგალითად, $name და $day არიან ცვლადის სახელები. ცვლადისთვის მნიშვნელობის მინიჭება ხდება = ოპერატორით. მაგალითად, $day='15`; ტექსტური კონსტანტების წარმოსადგენად გვაქვს ორი საშუალება: ტექსტის მოთავსება ერთმაგ ბრჭყალებში ან ორმაგ ბრჭყალებში. თუ ტექსტი მოთავსებულია ერთმაგ ბრჭყალებში, მაშინ იგი გაიგება, როგორც წარმოდგენილი ნიშნების უბრალო მიმდევრობა. მაგალითად, 'a\.+5`, ხოლო თუ იგი მოთავსებულია ორმაგ ბრჭყალებში, მაშინ მხედველობაში მიიღება მეტასიმბოლოების მნიშვნელობები და ცვლადების მნიშვნელობები. ასეთ შემთხვევაში ამბობენ, რომ ხდება ტექსტის ინტერპოლირება. მაგალითად ავიღოთ "a\nb$day" მისი ინტერპოლების შედეგად მიიღება "a\n5" თუ $day='15`; და \n გაიგება როგორც ერთი ნიშანი, რომელიც ნიშნავს ახალ სტრიქონზე გადასვლას, ხოლო 'a\nb$day` ნიშნავს იმ ნიშნების მიმდევრობას, რომელიც მოცემულია ბრჭყალებში. თუ ჩვენ გვინდა რომ ორმაგ ბრჭყალებში

მოცემული იგივე იყოს რაც ერთმაგ ბრჭყალებში მოცემული ტექსტი, მაშინ უნდა ჩავწეროთ ასე: "a\\tb\$day''. ვთქვათ,

$day='Sunday';

მაშინ

'today is Sunday'=~/$day/

ეთანადება.

'today is Sunday'=~/is Sunday/

ეთანადება.

'today is Sunday'=~/${day}

ეთანადება.

მესამე შემთხვევა არის Perl–ის ხეშმასივის ელემენტის სახით წარმოდგენა - ${day}-ის მნიშვნელობაა `Sunday'.თუ ჩვენ გვინდა, რომ რეგულარული გამოსახულება ემთხვეოდეს მოცემული ტექსტის ნაწილს დასაწყისიდანვე, მაშინ რეგულარული გამოსახულება უნდა დავიწყოთ ნიშნით ^, ხოლო თუ გვინდა, რომ მოცემული რეგულარული გამოსახულებით დამთავრდეს ტექსტი, მაშინ რეგულარულ გამოსახულებას ბოლოში უნდა დავუწეროთ $ ნიშანი. მაგალითად,

'regular day'=~/^regular / ეთანადება

'regular day'=~/^day/ არ ეთანადება

'regular day'=~/day$ / ეთანადება

'regular day'=~/^day/ არ ეთანადება

რეგულარულ გამოსახულებაში შეიძლება ნიშანთა კლასების გამოყენება, რაც მთელ რიგ შემთხვევებში საშუალებას გვაძლევს უფრო მოკლედ დაიწეროს რეგულარული გამოსახულება. კვადრატულ ფრჩხილებში მოთავსებული ნიშნების სიმრავლეს ეწოდება ნიშანთა კლასი. მაგალითად, [a b c] და იგი ნიშნავს ამ

სიმრავლიდან აღებულ ნებისმიერ ნიშნის შეხვედრას ტექსტის მიმდინარე პოზიციაში. მაგალითად, /r[a b c]d/ ეთანადება rad-ს, rbd ან rcd. თუ ჩვენ ავიღებთ /r[c b a]d/-ს, მაშინ იგი ეთანადება rcd-ს, rbd ან rad. ე. ი. იგივეს, მაგრამ განსხვავდება შედარების რიგით. ასევე, /[D d A a Y y]/ ეთანადება day–ს რეგისტრის მიუხედავად: Day, dAy, DAY და ა.შ., ე.ი. მათ ნებისმიერ კომბინაციას. [\$ a] კლასი შედგება ორი ნიშნისაგან $ და a. [a b c d e] კლასი შეგვიძლია ჩავწეროთ შემოკლებით [a-e]. ეს იმ შემთხვევაში თუ გვაქვს მიმდევრობით ყველა ნიშნები კლასში. მაგალითად, [a-e g u-x] წარმოადგენს კლასს [a b c d e g u v x]. თუ კლასის ელემენტია '–' ნიშანი, მაშინ იგი უნდა ჩაიწეროს ბოლო ნიშნად. მაგალითად, [a – c] ნიშნავს [a b c] კლასს. კლასი ლათინური ალფაბეტისათვის ჩაიწერება ასე: [A–Z a–z]. ნიშანთა ზოგიერთი კლასებისათვის გამოიყენება შემოკლებები: \d აღნიშნავს ნებისმიერ ციფრს და წარმოადგენს [0–9] კლასს.

\ S არის ხარვეზის ნიშანთა კლასი და წარმოადგენს [\ \t\ r\ f]

\w არის ალფაციფრულ ნიშანთა კლასი და წარმოიდგინება [0-9a-zA-Z]. \D არის არა \d. ე. ი. ნებისმიერი ნიშანი გარდა ციფრისა, რაც ნიშნავს \d\ ნამრავლის დამატებას უნივერსალურ სიმრავლემდე.

\ S არის არა \s. ე.ი. ნებისმიერი ნიშანი გარდა s-სა.

\ W არის არა \w.

.(წერტილი) ნიშნავს ნებისმიერ ნიშანს გარდა \n. ეს შემოკლებები შეიძლება გამოვიყენოთ როგორც კლასის შიგნით ასევე მის გარეთ რეგულარულ გამოსახულებაში. მაგალითად, [\s \d] ნიშნავს ხარვეზის ნიშანთა კლასს ან

ციფრს.

'Λ' ნიშანი კლასის შიგნით დასაწყისში ნიშნავს ამ კლასით წარმოდგენილი ნიშნების დამატებას უნივერსალურ სიმრავლემდე. ე. ი. ნებისმიერ ნიშანს, რომელიც განსხვავდება კლასში წარმოდგენილი ნიშნებისაგან. მაგალითად, [Λ a b c] არის ნებისმიერი ნიშანი, გარდა 'a' ,'b' ან 'c' [a b c Λ] არის 'a' , 'b' , 'c' ან 'Λ'. ე.ი. 'Λ' კლასში დასაწყისის გარდა ნებისმიერ ადგილას ნიშნავს თავისთავს. მაგალითები:

/ \w\w\w/-ეთანადება სამ ასოციფრულ ნიშანს.

/../ - ეთანადება ორ ნებისმიერ ნიშანს.

Λ./ - ეთანადება წერტილს '.'

/[Λ a b]/ - ეთანადება ნებისმიერ ნიშანს გარდა ' a' და b' -სი.

ზოგიერთი ნიშნები აღნიშნავენ ტექსტში გარკვეულ ადგილს და არ მოითხოვენ რაიმე ნიშანთან დამთხვევას. ასეთებს ჩვენ უკვე გაეცანით როგორიცაა 'Λ' და '$', რომლებიც მიუთითებენ სტრიქონის დასაწყისს და დამთავრებას შესაბამისად. კიდევ არის 'b' ნიშანი, რომელიც მიუთითებს ასოციფრული სიტყვის საზღვარს (დაწყებას ან დამთავრებას), ე.ი. W კლასის დაწყებას ან დამთავრებას. მაგალითად, Λb\w/ მიუთითებს ტექსტში ადგილს, რომლის შემდეგ გვაქვს ასოციფრული ნიშანი, რომლის უშუალოდ წინ არაა ასო-ციფრული ნიშანი. მაგალითები:

':-ab' =~/Λd\w/ ეთანადება 'a'-ს.

''''=~/Λ$/ ეთანადება.

''''=~/./ არ ეთანადება.

''\n''=~/Λ.$/ არ ეთანადება.

"a"=~/^.$/ ეთანადება.

"a\n"=~/^.$/ ეთანადება.

მოდიფიკატორები. როცა რეგულარული გამოსახულება მოთავსებულია დახრილ ხაზებს შორის (//) და ბოლო დახრილ ხაზს არ მოსდევს რაიმე ნიშანი ჩვენ ვამბობთ, რომ ეს არის სტანდარტული ნიმუში და მას არ გააჩნია რაიმე მოდიფიკატორი. თუ // ხაზებს მოსდევს სპეციალური ნიშნები, რომლებსაც ჩვენ ქვემოთ ჩამოვთვლით, მაშინ ვამბობთ, რომ რეგულარული გამოსახულება მოდიფიცირებულია ე.ი. იგი იქცევა შეთანხმებისას სხვანაირად იმისდამიხედვით თუ რა ნიშნები მოსდევს.

//S ნიშნავს, რომ შესათანადებელი სტრიქონი განიხილება, როგორც ერთი გრძელი სტრიქონი და '.' ეთანადება ნებისმიერ ნიშანს გარდა "\n"-ის წინ. //m ნიშნავს, რომ გვაქვს რამოდენიმე გრძელი სტრიქონი, რომელიც შეიცავს მრავალ სტრიქონს. ასეთ შემთხვევაში '.' ეთანადება ნებისმიერ ნიშანს გარდა "\n"-ისა. '^' და '$' ნიშნებს შეუძლიათ შეუთანადდნენ ნებისმიერი სტრიქონის დასაწყისსა და დაბოლოებას შესაბამისად მთლიან გრძელ სტრიქონში.

\\sm ნიშნავს, რომ ამუშავებს ერთ გრძელ სტრიქონს, მაგრამ მის შიგნით აღმოაჩენს მრავალ სტრიქონს. '.' ეთანადება ნებისმიერ ნიშანს "\n"-საც. '^' და '$' შეუძლიათ შეეთანადონ ნებისმიერი სტრიქონის დასაწყისსა და დაბოლოებას სტრიქონის შიგნით.

ალტერნატივები. რეგულარულ გამოსახულებაში შეიძლება გამოვიყენოთ ალტერნატივები ე.ი. რეგულარული გამოსახულების შიგნით შეიძლება გვქონდეს რამოდენიმე რეგულარული გამოსახულება და სავალდებულოა

მხოლოდ რომელიმე მათგანის შეთანადება ტექსტში.

ალტერნატივის მაჩვენებელია '|' (ვერტიკალური ნიშანი). მაგალითად, თუ ჩვენ გვინდა ვიპოვოთ ტექსტში 'სავარძელი' ან 'სკამი' ჩვენ უნდა შევადგინოთ ნიმუში: /სავარძელი|სკამი/.

მაგალითად, /[d e f] / იგივეა რაც /d|e|f/.

'სკამი'=~/სკამი|ს|სკ/ ეთანადება 'სკამი'

'სკამი'=~/ს|სკ|სკამი/ ეთანადება 'ს'-ს:

შეჯგუფება. ჩვენ შეგვიძლია რეგულარული გამოსახულების ნაწილი შევაჯგუფოთ და განვიხილოთ როგორც ერთი ელემენტარული ერთეული ან რაც იგივეა განვიხილოთ იგი როგორც ქვეგამოსახულება. ს საშუალებას გვაძლევს უფრო კომპაქტურად ჩავწეროთ რეგულარული გამოსახულება და თვითეული შეჯგუფებული ნაწილი დავამუშაოთ როგორც დამოუკიდებელი რეგულარული გამოსახულება. შეჯგუფება ხდება მრგვალი ფრჩხილების გამოყენებით. მაგალითად თუ გვაქვს რეგულარული გამოსახულება /დედაენა|დედაკაცი/ ჩვენ შეგვიძლია უფრო კომპაქტურად ჩავწეროთ შეჯგუფებით: /დედა(ენა|კაცი)/.

მაგალითები:

/(ა|გა|ჩა)(მო\)/ ეთანადება ამო, ა, გამო, გა, ჩამო, ჩა ან არა აქვს ზმნისწინი.

/სახლ(ის|ს|ით|ი|მა|ად|ო)/ ეთანადება ნებისმიერ ბრუნვაში მხოლობით რიცხვში დასმულ სიტყვას "სახლი".

/სახლ((ებ)|(ს|ის|ით|ი|მა|ად|ო))|(ნი|თა)/ ეთანადება მხოლობითში ან მრავლობითში ნებისმიერ ბრუნვაში დასმულ სიტყვას "სახლი".

/აცა|აც|ცა|ც|ლა|აე|ვე/ ეთანადება რომელიმე ნაწილაკს

(გავრცობის ნიშან ა-სთან ერთად) ან არა აქვს ნაწილაკი.

Perl-ში თეითეულ მრგვალ ფრჩხილებში მოთავსებულ გამოსახულებას შეესაბამება $n ცვლადი, სადაც n ნატურალური რიცხვია. $n-ში ინახება მე-n-ე მრგვალ ფრჩხილებში მოთავსებული გამოსახულების შესატყვისი ტექსტი. მაგალითად, $1 იქნება პირველ მრგვალ ფრჩხილებში მოთავსებულ რეგულარულ გამოსახულებასთან შეთანადებული ნუმერაცია. იწყება ერთიდან და გადაინომრება მარცხნიდან მარჯვნივ გახსნილი ფრჩხილები. მაგალითად, შეთანადების ინსტრუქციაში

"სახლისათვის"=~m/^((W+)(ის)(თვის\)(აც\))$/

ფრჩხილები გადაინომრება ასე: /^(1(2W+)2(ის)3(4თვის)4 (5აც)5)1$/ და შეთანადების ოპერატორის შესრულების შემდეგ:

$1 = "სახლისთვისაც"
$2 = "სახლის"
$3 = "ის"
$4 = "თვის"
$5 = "აც"

$n ცვლადების მნიშვნელობები შეიძლება გამოვიყენოთ სხვა ინსტრუქციებში რეგულარული გამოსახულების გარეთ და მათ ჩვენ არ შეგვიძლია მინიჭების ინსტრუქციით მივანიჭოთ ახალი მნიშვნელობები. თუ ჩვენ გვინდა $n-ის მნიშვნელობა გამოვიყენოთ იგივე რეგულარული გამოსახულების შიგნით ჩვენ უნდა მივუთითოთ იგი \n სახით, მხოლოდ მას შემდეგ რაც $n განისაზღვრება რეგულარულ გამოსახულებაში. მაგალითად:

"თავთავი"=~m/^((თავ)\2ი)$/ ეთანადება და

$1 იქნება თავთავი,
$2 იქნება თავ.

2 თავი

CF გრამატიკის გარდაქმნები და ნორმალური ფორმები

2.1 სასრული გარდამქმნელები. ერთი ფორმალური ენიდან მეორეზე თარგმანი შეიძლება შევასრულოთ მოწყობილობით, რომელიც წარმოადგენს სასრული ავტომატის განზოგადოებას. კერძოდ, სასრულ ავტომატს თუ დაეუმატებთ გამოსასვლელ ფირს, რომელზედაც სასრული ავტომატი ყოველი ტაქტის შესრულებისას გამოიტანს რაიმე გამოსასვლელ სიმბოლოს, ჩვენ მივიღებთ მოწყობილობას, რომლის გამოტანა იქნება თარგმანი იმ სიმბოლოთა მიმდევრობისა, რომლის შეტანა მოხდა შესასვლელი ფირიდან სასრული ავტომატის მუშაობის დროს. ამ შემთხვევაში გამოტანა ეს არის გამოსასვლელ ფირზე გამოტანილი სიმბოლოების მიმდევრობა. ასეთ მოწყობილობას ეწოდება სასრული გარდამქმნელი.

ფორმალურად სასრული გარდამქმნელი განისაზღვრება როგორც ექვსეული

$M = (Q, \Sigma, \Delta, \delta, q_0, F)$, სადაც

Q – არის მდგომარეობათა სიმრავლე,

Σ - შესასვლელ სიმბოლოთა სიმრავლე,

Δ - გამოსასვლელ სიმბოლოთა სიმრავლე,

q_0 საწყისი მდგომარეობაა,

δ არის $Q \times (\Sigma \cup \{e\})$ სიმრავლის გადასახვა $Q \times \Delta^*$ სიმრავლის ქვესიმრავლეთა სიმრავლეში. სასრული

ავტომატის ანალოგიურად განიმარტება სასრული გარდამქმნელის კონფიგურაცია. მხოლოდ აქ კონფიგურაციას დაემატება ახალი ელემენტი – გამოსასვლელი სტრიქონი. ამგვარად, სასრული გარდამქმნელის კონფიგურაციაა სამეული (q, x, y), სადაც q აღნიშნავს გარდამქმნელის მდგომარეობას, x – არის თავდაპირველი შესასვლელი სტრიქონის დარჩენილი ნაწილი მოცემული მომენტისათვის, ე. ი. მისი სუფიქსი, ხოლო y არის მოცემული მომენტისათვის გამოტანილი სტრიქონი ანუ თავდაპირველი სტრიქონის პრეფიქსი.

სასრული გარდამქმნელი ყოველი ტაქტის შესრულებისას გადადის ერთი კონფიგურაციიდან მეორეში. დამოკიდებულება, რომელიც განსაზღვრავს სასრული გარდამქმნელის გადასვლას ერთი კონფიგურაციიდან მეორეში აღვნიშნოთ ⊢-ით. ანალოგიურად, სასრული ავტომატისა, აქ ჩვენ შემოვიტანოთ ამ დამოკიდებულებისათვის ^{i}, $^{+}$ და * ოპერატორები. დამოკიდებულება განისაზღვრება ასე: $(q,ax,y) \vdash (r,x,yz)$ სადაც

$q \in Q$ და $r \in Q$, $a \in (\Sigma \cup \{e\})$, $x \in \Sigma^*$ და $y \in \nabla^*$,

ხოლო (q,a) სიმრავლე უნდა შეიცავდეს (r,z) ელემენტს. ამის შემდეგ ჩვენ შეგვიძლია განვსაზღვროთ სასრული გარდამქმნელით მოცემული (M) თარგმანი:

$$(M) = \{(x,y) \mid (q,x,e) \vdash^{*} (q,e,y) \text{ და } q \in F\}$$

(x,y) წყვილში იგულისხმება, რომ y არის x-ის თარგმანი. მიაქციეთ ყურადღება, რომ სასრული გარდამქმნელი ამთავრებს წარმატებით მუშაობას, როცა შესასვლელი სტრიქონი დაცარიელდება. მაგალითი.

$S \rightarrow a+S \mid a-S \mid +S \mid -\mid a$

$Q=\{q_0,q_1,q_2,q_3,q_4\}$

$\Sigma=\{a,+,-\}$

2.2 გამოყვანის ხეები. გრამატიკაში შეიძლება იყოს რამოდენიმე გამოყვანა ექვივალენტური იმ გაგებით რომ ყოველ მათგანში ერთი და იგივე გამოყვანის წესები გამოიყენება ერთი და იგივე ადგილებში,მაგრამ განსხვავებული რიგით. ექვივალენტობის ცნების განმარტება ორი გამოყვანისათვის ნებისმიერი სახის გრამატიკისათვის რთულია, მაგრამ CF გრამატიკისათვის შეიძლება შემოვიტანოთ ექვივალენტურ გამოყვანათა კლასის მოხერხებული გრაფიკული წარმოდგენა, რომელსაც ეწოდება გამოყვანის ხე. გამოყვანის ხე CF გრამატიკაში G(N,Σ,P,S) ეს არის დალაგებული ხე, რომლის ყოველი წვერო მონიშნულია რაიმე სიმბოლოთი NUSU{e} სიმრავლიდან. თუ შიგა წვერო მონიშნულია A სიმბოლოთი, ხოლო მისი პირდაპირი შთამომავლები - X_1 X_2...X_n სიმბოლოებით, მაშინ $A \rightarrow X_1X_2...X_n$ არის ამ გრამატიკის წესი.

განმარტება. მონიშნულ დალაგებულ D ხეს ეწოდება გამოყვანის ხე(ან გარჩევის ხე) CF გრამატიკაში G(A)=(N,Σ,P,A) თუ შესრულებულია შემდეგი პირობები:

1. D ხის ძირი მონიშნულია A სიმბოლოთი.

2. თუ $D_1,D_2,...,D_k$ ქვეხეებია, რომლებზედაც დომინირებენ ხის ძირის პირდაპირი შთამომავლები და D_i ხის ძირი მონიშნულია X_i-თი, მაშინ წესი $A \rightarrow X_1...X_k$ ეკუთვნის P-ს. D_i უნდა იყოს გამოყვანის ხე $G(X_i) =(N, \Sigma,P,X_i)$ გრამატიკაში თუ X_i არატერმინალია, წინააღმდეგ შემთხვევაში D_i შედგება ერთადერთი წვეროსაგან X_i.

3. თუ ხის ძირს გააჩნია ერთადერთი შთამომავალი მონიშნული e-თი, მაშინ ეს შთამომავალი ქმნის ხეს.

შევნიშნოთ,რომ არსებობს დალაგებული ხის წვეროების ერდადერთი დალაგება,რომელშიც წვეროების პირდაპირი შთამომავლები ლაგდებიან მარცხნიდან მარჯვნივ, განვმარტოთ ეს დალაგება შემდეგნაირად. დავუშვათ, რომ

n არის წვერო და $n_1,n_2,...,n_k$ მისი პირდაპირი შთამომავლებია, მაშინ

განმარტება: გამოყვანის ხის კრონი ვუწოდოთ ჯაჭვს, რომელიც მი იღება, თუ ამოვწერთ მარცხნიდან მარჯვნივ ფოთლების მონიშვნებს.

ახლა ვაჩვენოთ, რომ გამოყვანის ხეები ადექვატურად წარმოადგენენ გამოყვანებს იმ აზრით, რომ CF გრამატიკაში $G\alpha$ გამოყვანილი ჯაჭვის ყოველი გამოყვანისათვის შეიძლება აიგოს გამოყვანის ხე G-ში α კრონით და პირიქით. ამისათვის, შემოვიტანოთ რამოდენიმე ახალი ცნება. ვთქვათ, D არის გამოყვანის ხე CF გრამატიკაში $G=(N,\Sigma,P,S)$.

განმარტება: D ხის კვეთა ვუწოდოთ D ხის წვეროთა ისეთ C სიმრავლეს, რომ

1. არცერთი ორი წვერო C-დან არ ძევს ერთსა და იმავე გზაზე D-ში.

2. D ხის არცერთი წვეროს დამატება არ შეიძლება C სიმრავლისათვის, რომ არ დაირღვეს (1) თვისება.

მაგალითი. ხის წვეროების სიმრავლე, რომლებიც შეიცავენ მხოლოდ ძირს არის კვეთა. ფოთლები აგრეთვე ქმნიან კვეთას.

განმარტება. განვმარტოთ D ხის კვეთის კრონი როგორც ჯაჭვი, რომელიც მიიღება მარცხნიდან მარჯვნივ იმ წვეროების მონიშვნებით კონკატენაციით, რომლებიც ქმნიან რომელიმე კვეთას.

ლემა 1. $S=\alpha_0,\alpha_1,...,\alpha_n$ არის α_n ჯაჭვის გამოყვანა CF გრამატიკაში $G=(N,\Sigma,P,S)$. მაშინ, G-ში შეიძლება აიგოს გამოყვანის ხე D, რომლისთვისაც α_n - კრონია, ხოლო $\alpha_0,\alpha_1,...,\alpha_{n-1}$ რომელიმე კვეთების კრონებია. დამტკიცება. ავაგოთ $D_i(0\le i\le n)$ გამოყვანის ხეების ისეთი მიმდევრობა, რომ α_i იყოს კრონი D_i ხისა.ვთქვათ D_0 არის ხე, რომელიც შედგება

ერთადერთი წვეროსაგან, რომელიც მონიშნულია S-ით. დავუშვათ რომ $\alpha_i = \beta_i A \gamma_i$ და $A \rightarrow X_1 X_2 \ldots X_k$ წესის გამოყენების შემდეგ გამოყოფილ A ნაწილზე მიიღება $\alpha_{i+1} = \beta_i X_1 X_2 \ldots X_k \gamma_i$, მაშინ D_{i+1} ხე მიიღება D_i-საგან თუ A-თი მონიშნულ ფოთოლზე (იგი არის $(|\beta_i|+1)$ სიმბოლო D_i ხის კრონისა) დავუმატებთ K პირდაპირ შთამომავლებს და მათ მოვნიშნავთ $X_1, X_2, \ldots, X_k$-თი შესაბამისად. ცხადია რომ D_{i+1} ხის კრონი იქნება α_{i+1}. D_n ხე იქნება საძიებელი გამოყვანის ხე. დავამტკიცოთ შებრუნებული ლემა, ე.ი. ვაჩვენოთ, რომ ყოველი გამოყვანის ხისათვის G-ში არსებობს ერთი მაინც შესაბამისი გამოყვანა.

ლემა 2. ვთქვათ D - გამოყვანის ხეა CF გრამატიკაში $G=(N,\Sigma,P,S)$ კრონით α .მაშინ, $S \Rightarrow^* \alpha$.

დამტკიცება.ვთქვათ $C_0, C_1, \ldots C_n$ D ხის ისეთი კვეთებია, რომ

1. C_0 შეიცავს მხოლოდ D ხის ძირს.

2. $C_{i+1} (0 \le i < n)$ მიიღება C_i-დან მასში ერთერთი არატერმინალური წვეროს შეცვლით მისი პირდაპირი შთამომავლებით.

3. C_n - არის D ხის კრონი.

ცხადია, რომ ერთი მაინც ასეთი მიმდევრობა არსებობს. თუ α_i არის C_i-სი, მაშინ $\alpha_0, \alpha_1, \ldots, \alpha_n$ არის α_n ჯაჭვის გამოყვანა α_0-დან G-ში.

გამოყვანებს შორის, რომლებიც შეიძლება მოცემული ხისაგან აიგოს, ორი მათგანი ჩვენ განსაკუთრებით გვაინტერესებს.

განმარტება. თუ ლემის დამტკიცებაში C_{i+1} მიიღება C_i-საგან მისი ყველაზე მარცხენა არატერმინალური წვეროს შეცვლით მისი პირდაპირი შთამომავლებით, მაშინ სათანადო გამოყვანას $\alpha_0, \alpha_1, \ldots, \alpha_n$ ეწოდება G გრამატიკაში α_0-საგან α_n ჯაჭვის მარცხენა გამოყვანა. მარჯვენა გამოყვანა

განიმარტება ანალოგიურად, საჭიროა მხოლოდ წინა წინადადებაში ყველაზე მარცხენა წვეროს ნაცვლად განვიხილოთ ყველაზე მარჯვენა წვერო. შევნიშნოთ რომ მარცხენა(ან მარჯვენა) გამოყვანა განისაზღვრება ცალსახად გამოყვანის ხით.

თუ $S=\alpha_0, \alpha_1,\ldots,\alpha_n=W$ არის W ჯაჭვის მარცხენა გამოყვანა, მაშინ α_i-ს($0\le i< n$). აქვს სახე $X_iA_iB_i$ სადაც $X_i\in\Sigma$

ყოველი შემდგომი მარცხენა გამოყვანის α_{i+1} ჯაჭვი მიიღება α_i-საგან მისი მარცხენა არატერმინალური A_i სიმბოლოს შეცვლით რომელიმე წესის მარჯვენა მხარით. მარჯვენა გამოყვანაში იცვლება მარჯვენა არატერმინალი.

მაგალითი. განვიხილოთ CF G_0 გრამატიკა წესებით

$E\to E+T|T$

$T\to T*F|F$

$F\to(E)|a$

მისი მარცხენა გამოყვანაა

$E\to E+T\to T+T\to F+T\to a+T\to a+F\to a+a$

ხოლო მარჯვენა გამოყვანაა

$E\to E+T\to E+F\to E+a\to T+a\to F+a\to a+a.$

განმარტება. α ჯაჭვს ვუწოდებთ მარცხნივ გამოყვანადს (G გრამატიკაში), თუ არსებობს $S=\alpha_0,\alpha_1,\ldots,\alpha_n=\alpha$ მარცხენა გამოყვანა და დავწეროთ $S\Rightarrow_{GL}{}^*\alpha$ (ან $S\Rightarrow_L{}^*\alpha$, როცა ცხადია თუ რომელი G გრამატიკა იგულისხმება). ანალოგიურად α-ს ვუწოდებთ მარჯვნივ გამოყვანადს, თუ არსებობს მარჯვენა გამოყვანა $S=\alpha_0,\alpha_1,\ldots,\alpha_n=\alpha$ და ვწერთ $S\Rightarrow S\Rightarrow_{GR}{}^*\alpha$ (ან $S\Rightarrow_R{}^*\alpha$). მარცხენა გამოყვანის ერთ ნაბიჯს აღვნიშნავთ $\Rightarrow_1$ ხოლო მარჯვენა გამოყვანისას $\Rightarrow_r$-ით.1 და 2 ლემები შეიძლება გავაერთიანოთ ერთ თეორემად:

თეორემა. ვთქვათ $G=(N,\Sigma,P,S)$ CF გრამატიკაა. $S\Rightarrow^*\alpha$ მაშინ და მხოლოდ მაშინ, როცა G-ში არსებობს გამოყვანის ხე კრონით α .

დამტკიცება. ეს უშუალოდ გამომდინარეობს 1 და 2 ლემებიდან.

შევნიშნოთ, რომ ჩვენ თავი ავარიდეთ იმის თქმას,რომ თუ მოცემულია გამოყვანა $S\Rightarrow^{*}\alpha$ CF გრამატიკაში, მაშინ შეიძლება ვიპოვოთ G-ში ერთადერთი გამოყვანის ხე კრონით α. ამის მიზეზი მდგომარეობს იმაში, რომ არსებობენ CF გრამატიკები, რომელთაც შეიძლება ჰქონდეთ რამდენიმე განსხვავებული გამოყვანის ხეები ერთი და იმავე კრონით.

განმარტება. CF გრამატიკას G ეწოდება არაცალსახა თუ არსებობს ერთი მაინც $W\in L(G)$ ჯაჭვი, რომელიც არის ორი ან მეტი განსხვავებული გამოყვანის ხის კრონი G-ში. ეს იგივეა,რომ რომელიმე $W\in L(G)$ ჯაჭვს აქვს ორი ან მეტი მარცხენა (მარჯვენა) გამოყვანა, წინააღმდეგ შემთხვევაში CF გრამატიკას G-ს ცალსახა გრამატიკა ეწოდება.

2.3 CF გრამატიკათა გარდაქმნები. ხშირად საჭიროა მოცემული გრამატიკის ისეთი მოდიფიცირება,რომ მის მიერ წარმოქმნილმა ენამ მიიღოს საჭირო სტრუქტურა. განვიხილოთ მაგალითად, $L(G_0)$ ენა. იგი შეიძლება მივიღოთ G გრამატიკის წესებით

$E\rightarrow E+T|\ E*T|T$

$T\rightarrow (E)|a$

G გრამატიკის მეორე ნაკლი იმაში მდგომარეობს, რომ + და * ოპერაციებს აქვთ ერთი და იგივე პრიორიტეტი. სხვანაირად რომ ვთქვათ, a+a*a და a*a+a გამოსახულებების სტრუქტურა,რომელსაც ანიჭებს მათ G გრამატიკა, გულისხმობს ოპერაციების შესრულების იგივე რიგს, რასაც გამოსახულებებში (a+a)*a და (a*a)+a შესაბამისად.

რომ მივიღოთ + და * ოპერაციების ჩვეულებრივი პრიორიტეტი, რომლის დროსაც * წინ უსწრებს + ოპერაციას და

გამოსახულება a+a*a გაიგება როგორც a+(a*a)-ზე საჭიროა გადავიდეთ G_0 გრამატიკაზე.

ზოგადი ალგორითმული მეთოდი, რომელიც მისცემდა მოცემულ ენას ნებისმიერ სტრუქტურას, არ არსებობს. მაგრამ, მთელი რიგი გარდაქმნების გამოყენებით შეიძლება შევუცვალოთ სახე გრამატიკას ისე, რომ არ შევცვალოთ მის მიერ წარმოქმნილი ენა. დავიწყოთ ყველაზე ცხადით, მაგრამ მნიშვნელოვანი გარდაქმნებით. ზოგიერთ შემთხვევებში CF გრამატიკა შეიძლება შეიცავდეს უსარგებლო სიმბოლოებს წესებში. მაგალითად, $G=(\{S,A\},\{a,b\},P,S)$ გრამატიკაში,სადაც $P=\{S\to a, A\to b\}$, არატერმინალი A და b არ შეიძლება შეგვხვდეს არცერთ გამოყვანად ჯაჭვში. ამგვარად, ამ სიმბოლოებს არა აქვთ არავითარი დამოკიდებულება L(G) ენასთან და ისინი შეიძლება ამოვაგდოთ G გრამატიკის განსაზღვრიდან ისე, რომ არ შევეხოთ L(G) ენას.

განმარტება. $X\in N\cup\Sigma$ სიმბოლოს ვუწოდოთ უსარგებლო CF გრამატიკაში $G=(N,\Sigma,P,S)$ თუ მასში არ არსებობს $S\Rightarrow^{*}WX\gamma$ სახის გამოყვანა,სადაც W,X,γ ეკუთვნიან Σ^{*}-ს.

იმისათვის,რომ დავადგინოთ A არატერმინალი უსარგებლოა თუ არა თავდაპირველად ავაგოთ ალგორითმი, რომელიც გაარკვევს შეიძლება თუ არა ტერმინალი ჰქონდეს რომელიმე ტერმინალურ ჯაჭვს, ე.ი. რომელიც ხსნის $\{W|\ A\to^{*}W, W\in\Sigma^{*}\}$ სიმრავლის სიცარიელის პრობლემას. ასეთი ალგორითმის არსებობიდან გამომდინარეობს CF გრამატიკისათვის სიცარიელის პრობლემის გადაწყვეტა.

ალგორითმი2.3.1.ცარიელია თუ არა L(G) ენა?

შესასვლელი. CF გრამატიკა $G=(N,\Sigma,P,S)$.

გამოსასვლელი. "დიახ", თუ $L(G)\neq\emptyset$. " არა" წინააღმდეგ შემთხვევაში.

მეთოდი. ვაგებთ $N_0,N_1,...$ სიმრავლეებს რეკურსიულად:

1. დავუშვათ $N_0=\emptyset$ და $i=1$.

2. ავაგოთ $N_i=\{A|A\to\alpha\in P$ და $\alpha\in(N_{i-1}\cup\Sigma)^*\}\cup N_{i-1}$

3. თუ Ni არ უდრის Ni_{-1}, მაშინ გავზარდოთ i ერთით და გადავიდეთ (2) ნაბიჯზე. წინააღმდეგ შემთხვევაში მივიღოთ $N_e=N_i$.

4. თუ $S\in N_e$, მაშინ გამოსასვლელზე გამოვიტანოთ დიახ " , წინააღმდეგ შემთხვევაში " არა "

რადგან $N_e\subseteq N$, ამიტომ 2.3.1 ალგორითმი უნდა გაჩერდეს ყველაზე მეტი n+1-ჯერ (2) ნაბიჯის განმეორების შემდეგ, თუ N შეიცავს n რაოდენობა არატერმინალს. დავამტკიცოთ ალგორითმის კორექტულობა. დამტკიცება მარტივია და შემდგომში გამოდგება როგორც მოდელი რამოდენიმე ანალოგიური დამტკიცებისათვის.

თეორემა. ალგორითმი ამბობს "დიახ" მხოლოდ და მხოლოდ მაშინ, როცა $S\Rightarrow^*W$ რომელიმე $W\in\Sigma^*$ ჯაჭვისათვის.

დამტკიცება. თავდაპირველად დავამტკიცოთ i-ს მიხედვით ინდუქციით, რომ

თუ $A\in N_i$, მაშინ $A\Rightarrow^*W$ რომელიმე $W\in\Sigma^*$ ჯაჭვისათვის.(2.3.1)

ბაზისი: i=0, არ საჭიროებს დამტკიცებას, რადგან $N_0\neq\emptyset$.

ვიგულისხმოთ, რომ (2.3.1) ჭეშმარიტია i-სთვის, და ავიღოთ $A\in N_{i+1}$. თუ A ეკუთვნის აგრეთვე N_i-ს, მაშინ ინდუქციის ნაბიჯი ტრივიალურია. თუ $A\in N_{i+1}-N_i$, მაშინ არსებობს წესი $A\to X_1X_2...X_k$, სადაც ყოველი X_i სიმბოლო ეკუთვნის Σ ან N_i-ს. ამგვარად, ყოველი X_j-სთვის შეიძლება ვიპოვოთ ისეთი W_j ჯაჭვი, რომ $X_j\Rightarrow^*W$ თუ $X_j\in\Sigma$, მაშინ $W=X_j$, წინააღმდეგ შემთხვევაში W_j-ს არსებობა

გამომდინარეობს (2.3.1)-დან. ადვილად ჩანს, რომ $A\Rightarrow X_1...X_k\Rightarrow^* W_1X_2...X_k\Rightarrow^*...\Rightarrow^* W_1W_2...W_k$

შემთხვევა K=0 (ე.ი. A→e წესი) არაა გამორიცხული. ინდუქციის ნაბიჯი დამთავრებულია.

N_i სიმრავლეების განმარტება უზრუნველყოფს, რომ თუ $N_i=N_{i+1}$, მაშინ $N_i=N_{i+1}=...$ ჩვენ უნდა ვაჩვენოთ, რომ თუ $A\Rightarrow^* W$ რომელიმე $W\in\Sigma^*$ ჯაჭვისათვის, მაშინ $A\in N_e$. ზემოთ გაკეთებული შენიშვნის ძალით ყველაფერი, რის ჩვენებაც საჭიროა ესაა ის, რომ $A\in N_i$ რომელიმე i-სთვის. n-ის მიხედვით ინდუქციით დავამტკიცოთ, რომ

თუ, $A\Rightarrow^n W$, მაშინ $A\in N_i$ რომელიმე i-სთვის (2.3.2) ბაზისი:n=1, ტრივიალურია, რადგან ამ შემთხვევაში i=1. დავუშვათ, რომ (2.3.2) ჭეშმარიტია n-სთვის, და ვთქვათ $A\Rightarrow^{n+1} W$. მაშინ შეიძლება დაიწეროს $A\Rightarrow X_1...X\Rightarrow^n W$, სადაც $W=W_1...W_k$ ჯაჭვი ისეთია, რომ $X_j\Rightarrow^{n_j} W_j$ ყოველი j-სა და $n_j\leq$ n-სთვის.

(2.3.2)-ის თანახმად, თუ $X_j\in N$, მაშინ $X_j\in N_{ij}$ რომელიმე i_j-სთვის. თუ $X_j\in\Sigma$, მაშინ $i_j=0$. დავუშვათ $i=1+\max(i_1,...,i_k)$, მაშინ განმარტების თანახმად $A\in N_i$ და ინდუქცია დამთავრებულია.

ჩავსვათ A=S 2.3.1-სა და 2.3.2-ში და მივიღებთ თეორემის დამტკიცებას.

შედეგი. G კონტექსტისაგან თავისუფალი გრამატიკისათვის L(G) ენის სიცარიელის პრობლემა გადაწყვეტადია.

განმარტება. $X\in(N\cup\Sigma)$ სიმბოლოს ვუწოდოთ მიუღწევადი CF გრამატიკაში G=(N,Σ,P,S), თუ X არ გვხვდება არცერთ გამოყვანად ჯაჭვში. მიუღწევადი სიმბოლოები შეიძლება მოვიცილოთ CF გრამატიკიდან შემდეგი ალგორითმის საშუალებით:

ალგორითმი 2.3.2. მიუღწევადი სიმბოლოების მოცილება.

შესასვლელი. CF გრამატიკა G=(N,Σ,P,S).

გამოსასვლელი. CF გრამატიკა G′ =(N′,Σ′,P′,S) რომლისთვისაც

1.L(G′)=L(G),

2. ყოველი X∈N′∪Σ′ - არსებობს ისეთი α და β ჯაჭვები (N′ ∪ Σ′)-დან, რომ $S\Rightarrow_{GL}^{*}\alpha X\beta$.

მეთოდი.

1. დავუშვათ $V_0=\{S\}$ და i=1.

2. დავუშვათ $V_i=\{X|$P-ში არის $A\rightarrow\alpha X\beta$ და $A\in V_{i-1}\}\cup V_{i-1}$.

3. თუ $V_i\neq V_{i-1}$, მაშინ დავუშვათ i=i+1 და გადავიდეთ (2) ნაბიჯზე. წინააღმდეგ შემთხვევაში, ვთქვათ

$N'=V_i\cap N$,

$\Sigma'=V_i\cap\Sigma$

P′ შედგება P სიმრავლის ისეთი წესებისაგან, რომლებიც შეიცავენ მხოლოდ სიმბოლოებს V_i-საგან, G′ =(N′,Σ′,P′,S).

და ალგორითმები ძალიან გვანან ერთიმეორეს. შევნიშნოთ რომ ალგორითმის მე-2 ნაბიჯი შეიძლება გავიმეოროთ მხოლოდ სასრულო რიცხვჯერ, რადგანაც $V_i\subseteq N\cup\Sigma$ გარდა ამისა, i-ს მიხედვით ინდუქციით პირდა-პირი დამტკიცება გვიჩვენებს, რომ $S\Rightarrow G'\alpha X\beta$, მაშინ და მხოლოდ მაშინ, როცა $X\in V_i$ რომელიმე i-სთვის. ახლა ჩვენ შეგვიძლია მოვიცილოთ CF გრამატიკიდან ყველა უსარ-გებლო სიმბოლო.

ალგორითმი 2.3.3. უსარგებლო სიმბოლოების მოცილება.

შესასვლელი. CF გრამატიკა G=(N,Σ,P,S), რომელიც $L(G)\neq\emptyset$.

გამოსასვლელი. CF გრამატიკა $G'=(N',\Sigma',P',S)$, რომლისთვისაც $L(G')=L(G)$ და $(N'\cup\Sigma')$-ში არაა უსარგებლო სიმბოლოები.

მეთოდი.

1. G-ზე ალგორითმის გამოყენებით მივიღოთ N_e. დავუშვათ $G_1=(N\cap N_e,\Sigma,P_1,S)$, სადაც P_1 შედგება P სიმრავლის ისეთი წესებისაგან,რომლებიც შეიცავენ მხოლოდ $N_e\cup\Sigma$-ს სიმბოლოებს.

2. G-ზე ალგორითმის გამოყენებით მივიღოთ $G'=(N',\Sigma',P',S)$.

ალგორითმის 1 ნაბიჯზე G-ს ჩამოსცილდება ყველა ისეთი არატერმინალები,რომლებიც არ წარმოშობენ ტერმინალურ ჯაჭვებს. შემდეგ 2 ნაბიჯზე ჩამოსცილდებიან ყველა მიუღწევადი სიმბოლოები. ყოველი X სიმბოლო მიღებული გრამატიკიდან უნდა მონაწილეობდეს ერთ მაინც $S\Rightarrow^*WXY\Rightarrow^*WXY$ სახის გამოყვანაში. შევნიშნოთ, რომ თუ თავდაპირველად გამოვიყენებთ ალგორითმს და შემდეგ - ალგორითმს, მაშინ ყოველთვის ვერ მივიღებთ გრამატიკას, რომელიც არ შეიცავს უსარგებლო სიმბოლოებს.

თეორემა. G' გრამატიკა, რომელსაც აგებს ალგორითმი არ შეიცავს უსარგებლო სიმბოლოებს და $L(G)=L(G')$.

დამტკიცება. იმის დამტკიცებას, რომ $L(G)=L(G')$ მარტივია. ვიგულისხმოთ, რომ $A\in N'$ უსარგებლო სიმბოლოა. მაშინ უსარგებლო სიმბოლოს განმარტების თანახმად შეიძლება წარმოგვიდგეს ორი შემთხვევა:

1 შემთხვევა: $S\Rightarrow_{G1}^{\bullet}\alpha A\beta$

გამოყვანა შეუძლებელია როგორიც არ უნდა იყოს α და β. ასეთ შემთხვევაში A სიმბოლოს ჩამოცილება მოხდება ალგორითმის მე-2 ნაბიჯის დროს.

2 შემთხვევა: $S\Rightarrow_{G1}^{\bullet}\alpha A\beta$ რომელიმე α და β-სთვის, მაგრამ

$A \Rightarrow_{G1}^{\cdot} W$ გამოყვანა $W \in \Sigma'^{\cdot}$ არ არსებობს. მაშინ A არ ჩამოსცილდება მე-(2) ნაბიჯზე და, გარდა ამისა, თუ $A \Rightarrow_{G}^{\cdot} \gamma B\delta$, მაშინ B-ც არ ჩამოსცილდება მე-(2) ნაბიჯზე. ამგვარად, თუ $A \Rightarrow_{G}^{\cdot} W$, მაშინ $A \Rightarrow_{G1}^{\cdot} W$.

აქედან შეიძლება დავასკვნათ, რომ $A \Rightarrow_{G}^{\cdot} W$ $W \in \Sigma^{*}$-სთვის არ არსებობს და A ჩამოსცილდება (1) ნაბიჯზე.

იმის დამტკიცება, რომ G′ -ის არცერთი ტერმინალი არ შეიძლება იყოს უსარგებლო ჩატარდება ანალოგიურად.

მაგალითი 2.3.1. განვიხილოთ G=({S,A,B},{a,b},P,S) გრამატიკა, სადაც P შედგება წესებისაგან:

S→a|A

A→AB

B→b

გამოვიყენოთ G-ს მიმართ ალგორითმი. (1) ნაბიჯზე მივიღებთ N_e={S,B} და G_1=({S,B},{a,b},{S→a,B→b},S).

ალგორითმის გამოყენებით, მივიღებთ $V_2=V_1$={S,a} ამგვარად. G′=({S},{a},{S→a},S). თუ G-ს მიმართ გამოვიყენებთ ჯერ ალგორითმს, მაშინ აღმოჩნდება, რომ ყველა სიმბოლო მიუღწევადია, ასე რომ გრამატიკა არ შეიცვლება. შემდეგ ალგორითმის გამოყენება მოგვცემს N_e={S,B} და შედეგად გვექნება გრამატიკა G_1, რომელიც განსხვავდება G′-საგან.

ხშირად ხელსაყრელია CF გრამატიკიდან G მოვიცილოთ e წესები, ე.ი. A→e სახის წესები.მაგრამ თუ e∈L(G), მაშინ ცხადია, რომ A→e სახის წესებს ვერ მოვიცილებთ.

განმარტება. ვუწოდოთ CF გრამატიკას G=(N,Σ,P,S) გრამატიკა e წესების გარეშე (ან არადაპატარავებადი), თუ

1.P არ შეიცავს e წესებს, ან

2. არსებობს ზუსტად ერთი e წესი S→e და S არ

გვხვდება დანარჩენი წესების მარჯვენა მხარეში.

ალგორითმი 2.3.4. e წესების გარეშე გრამატიკად გარდაქმნა.

შესასვლელი. CF გრამატიკა G=(N,Σ,P,S).

გამოსასვლელი. ექვივალენტური CF გრამატიკა G′=(N′,Σ,P′,S′) e წესების გარეშე.

მეთოდი.

(1) ავაგოთ $N_e=\{A|A\in N$ და $A\Rightarrow^+_G e\}$ ეს ანალოგიურია იმისა, რაც იყო 1 და 2 ალგორითმებში და ვტოვებთ დამტკიცების გარეშე.

(2) ავაგოთ P′ ისე, რომ

a. თუ $A\rightarrow\alpha_0B_1\alpha_1B_2\alpha_2\ldots B_k\alpha_k$ ეკუთვნის P,K≥ 0 და $B_i\in N_e$ $1\leq i\leq k$, ხოლო არცერთი α_j ჯაჭვის სიმბოლო $(0\leq J\leq K)$ არ ეკუთვნის N_e-ს, მაშინ P′-ში ჩავრთოთ ყველა წესები $A\rightarrow\alpha_0X_1\alpha_1X_2\ldots\alpha_{k-1}X_k\alpha_k$, სადაც X_i არის ან B_i ან e, მაგრამ არ ჩავრთოთ $A\rightarrow e$ წესი (ეს შეიძლება იმ შემთხვევაში, თუ ყველა α_i ტოლია e-სი). b. თუ S∈ Ne, მაშინ P′-ში ჩავრთოთ წესი

$S'\rightarrow e|S$,

სადაც S′ ახალი სიმბოლოა დავუშვათ N′ =N∪{S′} .

წინააღმდეგ შემთხვევაში დავუშვათ რომ N′=N და S′=S.

(3) მივიღოთ , რომ G′=(N′,Σ,P′,S′).

თეორემა. ალგორითმი გვაძლევს გრამატიკას e წესების გარეშე, რომელიც შესავალი გრამატიკის ექვივალენტურია.

დამტკიცება. უშუალოდ ჩანს,რომ ალგორითმი გვაძლევს G′ გრამატიკას e წესების გარეშე. იმის საჩვენებლად, რომ L(G)=L(G′), საკმარისია დავამტკიცოთ ინდუქციით W ჯაჭვის სიგრძის მიხედვით, რომ

$A\Rightarrow^*_G W$ მაშინ და მხოლოდ მაშინ, როცა W≠ e და $A\Rightarrow^*_G W$ (2.3.3) რის დამტკიცებაც მარტივია. ჩავსვათ S A-ს

ნაცვლად (2.3.3)-ში. ვხედავთ, რომ $W \in L(G)$ $W \neq e$-სთვის მხოლოდ და მხოლოდ მაშინ,როცა $W \in L(G')$. ცხადია, რომ $e \in L(G)$ მხოლოდ და მხოლოდ მაშინ, როცა $e \in L(G')$. ამგვარად $L(G)=L(G')$. გრამატიკის სხვა სასარგებლო გარდაქმნაა $A \to B$ სახის წესების მოცილება, რომლებსაც ჩვენ ვუწოდებთ ჯაჭვურ წესებს.

ალგორითმი 2.3.5. ჯაჭვური წესების მოცილება.

შესასვლელი. CF გრამატიკა G e წესების გარეშე.

გამოსასვლელი. ექვივალენტური CF გრამატიკა G′ e წესებისა და ჯაჭვური წესების გარეშე.

მეთოდი.

1. ყოველი $A \in N$-სთვის ავაგოთ $N_A=\{B|A \Rightarrow^{*} B\}$

შემდეგნაირად:

a. მივიღოთ $N_0=\{A\}$ და $i=1$.

b. მივიღოთ $N_i=\{C|\ B \to C$ ეკუთვნის P და $B \in N_{i-1}\} \cup N_{i-1}$.

c. თუ $N_i \neq N_{i-1}$ დავუშვათ $i=i+1$ და (b) ნაბიჯი გავიმეოროთ. წინააღმდეგ შემთხვევაში მივიღოთ $N_A=N_i$.

2. ავაგოთ P′ ასე: თუ $B \to \alpha$ ეკუთვნის P-ს და იგი ჯაჭვური წესი არაა, ჩავრთოთ P′ -ში $A \to \alpha$ წესი ყველა ისეთი A-სთვის, რომ $B \in N_A$.

3. მივიღოთ $G' =(N,\Sigma,P',S)$.

მაგალითი. გამოვიყენოთ 2.3.5 ალგორითმი G_0 გრამატიკისათვის, რომლის წესებია

$E \to E+T|T$

$T \to T*F|F$

$F \to (E)|a$

(1) ნაბიჯზე $N_E=\{E,T,F\}$, $N_T=\{T,F\}$, $N_F=\{F\}$. (2) ნაბიჯის შემდეგ P′ სიმრავლე იქნება ასეთი:

$E \to E+T|T*F|(E)|a$

$T \to T*F|(E)|a$

F→ (E)|a.

თეორემა. G′ გრამატიკა, რომელსაც აგებს ალგორითმი ჯაჭვური წესები არ გააჩნია.

თავდაპირველად ვაჩვენოთ, რომ L(G′)⊆L(G). ვთქვათ, W∈ L(G′). მაშინ G′ გრამატიკაში არსებობს გამოყვანა $S\Rightarrow\alpha_0 \Rightarrow\alpha_1\Rightarrow\alpha_n=W$ თუ α_i-დან α_{i+1}-ზე გადასვლისას A→β წესი გამოიყენება, მაშინ არსებობს ისეთი სიმბოლო B∈N შესაძლებელია B=A, რომ $A\Rightarrow^*_G B$

და $B\Rightarrow^*_G\beta$ ამგვარად, $A\Rightarrow^*_G\beta$ და $\alpha_i\Rightarrow^*_G\alpha_{i+1}$ აქედან გამომდინარეობს, რომ $S\Rightarrow^*_G W$ და W∈L(G), ისე რომ L(G′)⊆L(G).

ახლა ვაჩვენოთ, რომ L(G)⊆L(G′). ვთქვათ W∈L(G) და $S\Rightarrow\alpha_0\Rightarrow\ldots\Rightarrow\alpha_n=W$ W ჯაჭვის მარცხენა გამოყვანაა G გრამატიკაში. შეიძლება ვიპოვოთ $i_1,i_2,\ldots i_k$ ინდექსების მიმდევრობა, რომელიც შედგება ზუსტად იმ J-სგან, რომელთათვისაც ნაბიჯზე $\alpha_{j-1}\Rightarrow\alpha_j$ გამოიყვანება არა ჯაჭვური წესით. რადგანაც, ჩვენ ვიხილავთ მარცხენა გამოყვანას, ამიტომ ჯაჭვური წესების მიმდევრობით გამოყენება ცვლიან სიმბოლოს, რომელსაც უჭირავს ერთი და იგივე პოზიცია მარცხენა გამოყვანად ჯაჭვებში. აქედან სჩანს, რომ $S\Rightarrow_{G'}\alpha_{i1}\Rightarrow_{G'}\ldots\Rightarrow_{G'}\alpha_{ik}=W$.

ამგვარად, W∈L(G′), რაც იმას ნიშნავს, რომ L(G′)=L(G).

განმარტება. CF გრამატიკას G=(N,∑,P,S) ეწოდება გრამატიკა ციკლების გარეშე,თუ მასში არაა გამოყვანა $A\Rightarrow^+A$ A∈N. G გრამატიკას ეწოდება დაყვანილი, თუ იგი არ შეიცავს ციკლებს, e-წესებს და უსარგებლო სიმბოლოებს.

e-წესების ან ციკლების მქონე გრამატიკების ანალიზები ზოგჯერ უფრო ძნელია, ვიდრე e-წესების არ მქონე და უციკლო გრამატიკების ანალიზი. გარდა ამისა, ნებისმიერ პრაქტიკულ სიტუაციაში უსარგებლო სიმბოლოები

აუცილებლობის გარეშე ზრდიან ანალიზატორის მოცულობას. ამიტომ სინტაქსური ანალიზის ზოგიერთი ალგორითმებისათვის, ჩვენ მოვითხოვთ, რომ მათში ფიგურირებული გრამატიკები იყვნენ დაყვანილნი. დავამტკიცოთ, რომ ამ მოთხოვნის მიუხედავად მაინც განიხილება ნებისმიერი CF ენები.

თეორემა. თუ L - CF ენაა, მაშინ L=L(G) რომელიმე დაყვანილი CF გრამატიკისათვის G.

დამტკიცება. გამოვიყენოთ ალგორითმები CF გრამატიკისათვის, რომელიც განსაზღვრავს L ენას.

განმარტება. CF გრამატიკის A წესი ეწოდება $A\to\alpha$ სახის წესს (არ აგერიოთ A წესი e-წესში, რომელსაც აქვს სახე $B\to e$.

შემოვიტანოთ კიდევ გარდაქმნა, რომლის საშუალებითაც შეიძლება ამოვაგდოთ გრამატიკიდან ერთი $A\to\alpha B\beta$ სახის წესი. იმისათვის, რომ ამოვაგდოთ ეს წესი, საჭიროა გრამატიკას დავუმატოთ ახალი წესები, რომლებიც მიიღება მასში B არატერმინალის შეცვლით მარჯვენა მხარით ყველა B წესიდან. ლემა. ვთქვათ $G=(N,\sum,P,S)$ - CF გრამატიკაა და P შეიცავს წესს $A\to\alpha B\beta$, სადაც $B\in N$, ხოლო α და β ეკუთვნიან $(N\cup\sum)^*$ ვთქვათ $B\to\gamma_1|\gamma_2|\ldots|\gamma_k$ ამ გრამატიკის ყველა B წესია. ვთქვათ

$G'=(N,\sum,P',S)$, სადაც

$P'=(P-\{A\to\alpha B\beta\})\cup\{A\to\alpha\gamma_1\beta|\alpha\gamma_2\beta|\ldots|\alpha\gamma_k\beta$, მაშინ $L(G)=L(G')$. დამტკიცება მარტივია.

2.4 ხომსკის ნორმალური ფორმა.

განმარტება 2.4.1 CF გრამატიკას $G=(N,\sum,P,S)$ ეწოდება გრამატიკა ხომსკის ნორმალურ ფორმაში (ანდა ბინარულ ნორმალურ ფორმაში), თუ ყოველ წესს P-დან აქვს ერთერთი შემდეგი სახეებიდან:

1. $A \to BC$, სადაც A,B და C ეკუთვნიან N-ს,

2. $A \to a$, სადაც $a \in \Sigma$,

3. $S \to e$, თუ $e \in L(G)$ და S არ გვხვდება სხვა წესების მარჯვენა მხარეში. ვაჩვენოთ, რომ ყოველი CF ენა მიიღება ხომსკის ნორმალურფორმიანი გრამატიკიდან. ეს შედეგი სასარგებლოა იმ შემთხვევებში, როცა გვჭირდება CF ენის წარმოდგენის მარტივი ფორმა.

ალგორითმი 2.4.1 ხომსკის ნორმალურ ფორმად გარდაქმნა.

შესასვლელი. დაყვანილი CF გრამატიკა $G=(N,\Sigma,P,S)$.

გამოსასვლელი. CF გრამატიკა G' ხომსკის ნორმალურ ფორმაში, რომელიც G-ს ექვივალენტურია, ე.ი. $L(G')=L(G)$.

მეთოდი. გრამატიკა G' იგება G-სგან შემდეგნაირად:

1. ჩავრთოთ P'-ში $A \to a$ სახის ყოველი წესი P-დან.

2. ჩავრთოთ P'-ში $A \to BC$ სახის ყოველი წესი P-დან.

3. ჩავრთოთ P'-ში $A \to e$ წესი, თუ იგი იყო P-ში.

4. $A \to X_1 \ldots X_k$ სახის ყოველი წესისათვის P-დან, სადაც $K > 2$ ჩავრთოთ P'-ში წესები

$A \to X'_1 < X_2 \ldots X_k >$

$< X_2 \ldots X_k > \to X'_2 < X_3 \ldots X_k >$

&

&

$< X_{k-2}\ X_{k-1}\ X_k > \to X'_{k-2} < X_{k-1}\ X_k >$

$< X_{k-1}\ X_k > \Rightarrow X'_{k-1}\ X'$,

სადაც $X'_i = X_i$, თუ $X_i \in N$; X'_i ახალი არატერმინალია, თუ $X_i \in \Sigma$;

$< X_i \ldots X_k >$ - ახალი არატერმინალია.

5. $A \to X_1X_2$ სახის ყოველი წესისათვის P-დან, სადაც ერთი მაინც X_1X_2 სიმბოლოებიდან ეკუთვნის Σ, ჩავრთოთ P′ -ში $A \to X'_1X'_2$ წესი.

6. a′ სახის ყოველი არატერმინალისთვის, რომლებიც შემოტანილია (4) და (5) ნაბიჯებზე, ჩავრთოთ P′-ში $a' \to a$ წესი. დაბოლოს, N′ შეიცავს N-ს და ყველა ახალ არატერმინალებს, რომლებიც შემოტანილია P′ აგების დროს. მაშინ, საძიებელი გრამატიკა იქნება $G' = (N', \Sigma, P', S)$.

თეორემა. ვთქვათ L არის CF ენა. მაშინ $L=L(G')$ რომელიმე CF გრამატიკისათვის G′ ხომსკის ნორმალურ ფორმაში. დამტკიცება. თეორემის თანახმად L განისაზღვრება დაყვანილი G გრამატიკით. ალგორითმი G-დან აგებს G′ გრამატიკას, რომელიც, ცხადია, წარმოდგენილია ხომსკის ნორმალური ფორმით. დაგვრჩენია ვაჩვენოთ, რომ $L(G)=L(G')$. ეს მტკიცდება ლემის გამოყენებით G′ გრამატიკის ყოველი წესისათვის, რომელთა მარჯვენა მხარეში შედის a′, და შემდეგ წესებისათვის $< X_i...X_j >$ სახის არატერმინალებით.

მაგალითი 2.4.1. ვთქვათ G დაყვანილი გრამატიკაა, განსაზღვრული წესებით $S \to aAB \mid BA$

$A \to BBB \mid a$

$B \to AS \mid b$

ვაგებთ P′ 2.4.1 ალგორითმით, ვტოვებთ რა წესებს $S \to BA$, $A \to a$, $B \to AS$ და $B \to b$. ვცვლით $S \to aAB$ წესებით

$S \to a'\langle AB\rangle$ და $\langle AB\rangle \to AB$, ხოლო $A \to BBB$-ს წესებით – $A \to B\langle BB\rangle$ და $\langle BB\rangle \to BB$. და ბოლოს, ვუმატებთ $a' \to a$, შედეგად ვიღებთ გრამატიკას $G' = (N', \{a,b\}, P', S)$, სადაც

$N' = \{S, A, B, \langle AB\rangle, \langle BB\rangle, a'\}$, ხოლო P′ შედგება წესებისაგან

$S \to a'\langle AB\rangle \mid BA$

$A \to B\langle BB\rangle \mid a$

B→AS|b

<AB>→AB

<BB>→BB

a′→a

2.5 გრეიბახის ნორმალური ფორმა. ახლა ვაჩვენოთ, რომ ყოველი CF ენისათვის შეიძლება ვიპოვოთ გრამატიკა, რომელშიც წესების მარჯვენა მხარეები იწყებიან ტერმინალებით. ასეთი გრამატიკის აგება დაფუძნებულია ეგრეთ წოდებული მარცხენა რეკურსიის აცილებაზე.

განმარტება 2.5.1. G=(N,Σ,P,S) CF გრამატიკის A არატერმინალს ეწოდება რეკურსიული, თუ $A \Rightarrow^{+\alpha} A\beta$ რომელიმე α და β-სთვის. თუ α=e, მაშინ A-ს ეწოდება მარცხნივ რეკურსიული. ანალოგიურად,თუ β=e, მაშინ A-ს ეწოდება მარჯვნივ რეკურსიული. გრამატიკას, რომელსაც გააჩნია ერთი მაინც მარცხნივ რეკურსიული არატერმინალი, ეწოდება მარცხნივ რეკურსიული.

ანალოგიურად განიმარტება მარჯვნივრეკურსიული გრამატიკა. გრამატიკას, რომელშიც ყველა არატერმინალები, გარდა შეიძლება საწყისი სიმბოლოსა, რეკურსიულებია ეწოდება რეკურსიული.

ზოგიერთ გარჩევის ალგორითმებს, რომლებზეც ჩვენ შემდგომში განვიხილავთ არ შეუძლიათ იმუშაონ მარცხნივ რეკურსიულ გრამატიკებზე. ვაჩვენოთ, რომ ნებისმიერი CF ენა განისაზღვრება ერთი მაინც არა მარცხნივ რეკურსიული გრამატიკით. დავიწყოთ CF გრამატიკიდან უშუალოდ მარცხნივ რეკურსიულობის ჩამოშორებით.

ლემა 2.5.1. ვთქვათ G=(N,Σ,P,S) CF გრამატიკაა, რომელშიც

$A \to A\alpha_1|A\alpha_2|...|A\alpha_m|\beta_1|\beta_2|...|\beta_n$

ყველა A წესებია P-დან და არცერთი $\beta_i (i=1,2,...n)$ ჯაჭვებიდან არ იწყება A-თი. ვთქვათ

G′ =(N∪{A′},∑,P′,S), სადაც A′-ახალი არატერმინალია, ხოლო P′ მიიღება P-საგან A წესების შეცვლით

$A \to \beta_1|\beta_2|...|\beta_n|\beta_1A'|\beta_2A'|...|\beta_nA'$

$A' \to \alpha_1|\alpha_2|...|\alpha_m|\alpha_1A'|\alpha_2A'|...|\alpha_mA'$

წესებით: მაშინ L(G′)=L(G).

დამტკიცება.ჯაჭვები,რომლებიც შეიძლება მივიღოთ G გრამატიკაში A არატერმინალისაგან A წესების გამოყენებით ყველაზე მარცხენა არატერმინალზე ჰქმნიან $(\beta_1+\beta_2+...+\beta_n)\ (\alpha_1+\alpha_2+...+\alpha_m)^*$ რეგულარულ სიმრავლეს. ეს ზუსტად ის ჯაჭვებია, რომლებიც შეიძლება მივიღოთ G′-ში A-სგან მარჯვენა გამოყვანებით, გამოვიყენებთ რა ერთხელ A წესს და რამოდენიმეჯერ A′ წესებს. (შედეგად მთელი გამოყვანა არ იქნება მარცხენა). გამოყვანის ყველა ნაბიჯები G-ში, რომლებშიც არ გამოიყენება A წესები შეიძლება უშუალოდ ჩავატაროთ G′-ში, რადგანაც A წესების გარდა G და G′ ერთნაირია. აქედან შეიძლება დავასკვნათ, რომ $L(G') \subseteq L(G)$.

შებრუნებული ჩართვა მტკიცდება არსებითად ისევე. G′ -ში იღება მარჯვენა გამოყვანა და განიხილებიან ნაბიჯების მიმდევრობები, რომლებიც შედგებიან მხოლოდ ერთხელ A წესების გამოყენებისაგან და რამოდენიმეჯერ A′ წესების გამოყენებებისაგან. ამგვარად, L(G)=L(G′).

მაგალითი 2.5.2. ვთქვათ G_0 ჩვენი ჩვეულებრივი გრამატიკაა წესებით.

$E \to E+T|T$

$T \to T*F|F$

$F \to (E)|a$

თუ ამ გრამატიკის მიმართ გამოვიყენებთ ლემის კონსტრუქციას, მაშინ მიიღება ექვივალენტური G′ გრამატიკა წესებით

E→T|TE′

E′→+T|+TE′

T→F|FT′

T′→ F|FT′

F→(E)|a.

ახლა ჩვენ მზად ვართ აღვწეროთ ალგორითმი, რომელიც დაყვანილი CF გრამატიკას მოაშორებს მარცხენა რეკურსიას. თავისი იდეით ეს ალგორითმი ჰგავს რეგულარულ კოეფიციენტებიანი განტოლების ამოხსნის ალგორითმს.

ალგორითმი 2.5.1. მარცხენა რეკურსიის მოცილება.

შესასვლელი. დაყვანილი CL გრამატიკა G=(N,∑,P,S). გამოსასვლელი. ექვივალენტური CL გრამატიკა G′ მარცხენა რეკურსიის გარეშე.

მეთოდი.

1. ვთქვათ $N=\{A_1,...,A_n\}$. გარდავქმნათ G ისე, რომ $A_i\rightarrow\alpha$ წესში α ჯაჭვი იწყებოდეს ტერმინალით ან ისეთი A_j -თი, რომ j> i. ამ მიზნით ვთქვათ i=1.

2. ვთქვათ A_i წესების სიმრავლეა $A_i\rightarrow A_i\alpha_1|...|A_i\alpha_m|\beta_1|...\beta_p$, სადაც არცერთი β_j ჯაჭვებიდან არ იწყება A_k-თი, თუ K≤ i. (ეს ყოველთვის შესაძლებელია). შევცვალოთ A_i წესები

$A_i\rightarrow\beta_i|...|\beta_p|\beta_1A'_i|...|\beta_pA'_i$

$A'_i\rightarrow\alpha_1|...|\alpha_m|\alpha A'_i|...|\alpha_mA'_i$

წესებით სადაც A'_i ახალი არატერმინალია. ყველა A_i წესების მარჯვენა მხარეები ახლა იწყებიან ტერმინალით ანდა A_k-თი, სადაც K> i.

3. თუ i=n, მაშინ მიღებული გრამატიკა ჩავთვალოთ საბოლოო შედეგად და გავჩერდეთ. წინააღმდეგ შემთხვევაში მივიღოთ i+1 და j=1.

4. შევცვალოთ ყოველი A_i $Aj\alpha$ სახის წესი წესებით $A_i\rightarrow\beta_1\alpha|...|\beta_m\alpha$, სადაც $A_j\rightarrow\beta_1|\beta_2|...|\beta_m$ ყველა A_j წესებია.

რადგანაც ყოველი A_j წესის მარჯვენა მხარე იწყება უკვე ტერმინალით ან A_k-თი $K>j$-სთვის. ამიტომ ყოველი A_i წესის მარჯვენა მხარესაც ახლა ექნება ეს თვისება.

5. თუ $j=i-1$ გადავიდეთ მე-(2) ნაბიჯზე. წინააღმდეგ შემთხვევაში მივიღოთ $j=j+1$ და გადავიდეთ მე-(4) ნაბიჯზე.

თეორემა 2.5.1. ყოველი CF ენა განისაზღვრება არამარცხნივრეკურსიული გრამატიკით.

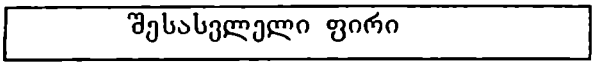
დამტკიცება. ვთქვათ G დაყვანილი გრამატიკაა, რომელიც წარმოშობს CL L ენას. მასზე ალგორითმის გამოყენებისას, გამოიყენება მხოლოდ ის გარდაქმნები, რომლებიც მოხსენებულია ლემებში. ამიტომ საბოლოო G′ გრამატიკა წარმოქმნის L ენას.

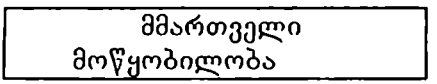
2.6 ავტომატები მაღაზიური მეხსიერებით.

დაპროგრამების ცნობილი ენები განისაზღვრებიან კონტექსტისაგან თავისუფალი გრამატიკებით. ეს იმას ნიშნავს, რომ თუ ჩვენ ავიღებთ რაიმე დაპროგრამების ენას მაგალითად, FORTRAN-IV-ს. ჩვენ შეგვიძლია შევადგინოთ კონტექსტისაგან თავისუფალი გრამატიკა GF, რომელიც უშვებს მხოლოდ FORTRAN-IV-ზე სინტაქსურად სწორად ჩაწერილ პროგრამებს. ე. ი. ნებისმიერი FORTRAN-IV-ზე სწორად ჩაწერილი პროგრამისათვის, რომელიც არის რაიმე P ტექსტი, მოიძებნება GF გრამატიკაში გამოყვანა $S\Rightarrow^*P$ და რაიმე P_1 არასწორად ჩაწერილი პროგრამისათვის არ არსებობს $S\Rightarrow^*P_1$ GF გრამატიკაში. აქედან გამომდინარეობს, რომ თუ ჩვენ გვექნება ალგორითმი $S\Rightarrow^*P$ ნებისმიერი P-ს საპოვნელად, მაშინ ჩვენ შეგვიძლია ამ ალგორითმის რეალიზაცია კომპიუტერზე SA პროგრამის სახით და მისი საშუალებით შევამოწმოთ P პროგრამა არის თუ არა FORTRAN-IV-ზე სწორად ჩაწერილი პროგრამა. ასეთ SA

პროგრამას ეწოდება სინტაქსური ანალიზატორი. იგივე ამოცანა შეიძლება გადაწყვეტილ იქნეს ნებისმიერი კონტექსტისაგან თავისუფალი გრამატიკისათვის სპეციალური მოწყობილობით, რომელსაც ეწოდება ავტომატი მაღაზიური მეხსიერებით. ასეთი ავტომატები სასრული ავტომატებისაგან ძირითადად განსხვავდებიან იმით, რომ მათ შეუძლიათ სპეციალურ მეხსიერებაში დაიმახსოვრონ თუ რა მდგომარეობები გაიარეს მოცემული მომენტისათვის და ამით აღადგინოთ წინაისტორია, რომელიც გაიარა ავტომატმა საწყისი მდგომარეობიდან მოცემულ მდგომარეობამდე. ასეთ სპეციალურ მეხსიერებას ეწოდება მაღაზიური მეხსიერება ან რასაც პროგრამისტები მეორენაირად უწოდებენ სტეკს. სტეკში სიმბოლოს ჩაწერა ხდება სტეკის თავში და სტეკიდან სიმბოლოს აღება ხდება იგივე სტეკის თავიდან. სტეკის თავი ყოველთვის უთითებს ბოლოს ჩაწერილ სიმბოლოს. როცა ვწერთ სიმბოლოს სტეკში, სტეკის თავი ერთი ადგილით გადაიწევს წინ და იქ ჩაიწერება მოცემული სიმბოლო ე. ი. წინათ ჩაწერილი სიმბოლოს წინ, ხოლო როცა ვიღებთ სიმბოლოს სტეკიდან მაშინ ხდება პირიქით. სტეკის თავი დაიწევს ერთი სიმბოლოთი უკან და სტეკის თავში მოხვდება შემდეგი სიმბოლო. მგვარად სტეკის თავი მოძრაობს წინ და უკან მასში სიმბოლოს ჩაგდება ამოგდების მიხედვით. ეს კი უზრუნველყოფს სტეკში ბოლოს ჩაგდებული სიმბოლოს პირველად ამოღებას. ასეთ სტეკებს უწოდებენ LIFO(Last Input First Output) სტეკებს. იმისათვის, რომ ისინი განვასხვაოთ FIFO(First Input First Output) სტეკებისაგან, რომელთა საშუალებით ხდება რიგების მოდელირება კომპიუტერში. ზოგადად, ავტომატი მაღაზიური მეხსიერებით შედგება:

1. შესასვლელი ფირისაგან, საიდანაც ავტომატი





დიაგრამა

კითხულობს განსახილველი სიტყვის სიმბოლოებს, ისევე როგორც სასრული ავტომატის შემთხვევაში;

2. მართველი მოწყობილობისაგან და

3. სასრული მაღაზიური მეხსიერებისაგან ანუ LIFO სტეკისაგან (იხ. დიაგრამა).

ფორმალურად ავტომატი მაღაზიური მეხსიერებით განიმარტება როგორც შვიდეული:

$AM=(Q,\Sigma,\delta,\Gamma,q_0,z_0,F)$

სადაც, Q მდგომარეობათასიმრავლეა,

Σ შესასვლელი სიმბოლოების სიმრავლეა,

Γ მაღაზიურ სიმბოლოთა სიმრავლეა,

$q_0 \in Q$ საწყისი მდგომარეობაა,

z_0 მაღაზიის ძირში მოთავსებული სიმბოლოა,

რომელიც მიუთითებს მაღაზიის დაცარიელებას. $F \subseteq Q$ საბოლოო მდგომარეობათა სიმრავლეა და არის $QX(\{\sum \cup e\})X\Gamma$ სიმრავლის გადასახვა ისეთ სიმრავლეში, რომლის ელემენტებია $QX\Gamma^*$ სიმრავლის ქვესიმრავლეები.

სასრული ავტომატების ანალოგიურად განისაზღვრება AM ავტომატის კონფიგურაცია და ტაქტი. (q,W,d) არის AM ავტომატის მიმდინარე კონფიგურაცია თუ ავტომატი იმყოფება q მდგომარეობაში, შესავალი სტრიქონია ამ მომენტში W და არის მაღაზიის შიგთავსი. ქედან გამომდინარე (q,w,α) სამეული არის $QX\Sigma^*X\Gamma^*$ ის ელემენტი. როცა $\alpha=e$ მაღაზია არის ცარიელი. დამოკიდებულებას $(q,aw,y\alpha) \vdash (q_1,W,\gamma\alpha)$ ეწოდება AM ავტომატის ტაქტი, როცა AM ავტომატი $(q,aw,y\alpha)$ კონფიგურაციიდან გადადის $(q,w,\gamma\alpha)$ ტაქტში და (q,a,y) სიმრავლე შეიცავს (q_1,γ) ელემენტს. ეს ნიშნავს რომ AM ავტომატის მიმდინარე მდგომარეობაა q და შესასვლელის მიმდინარე სიმბოლოა a, მაღაზიის თავში გვაქვს y სიმბოლო და თუ (q,a,y) სიმრავლე შეიცავს (q,γ)-ს, მაშინ AM ავტომატის მიმდინარე მდგომარეობა გახდება q_1, მიმდინარე შესასვლელი სიმბოლო იქნება a-ს მომდევნო სიმბოლო მარცხნიდან მარჯვნივ მიმართულებით და მაღაზიაში y სტრიქონი შეიცვლება γ-თი. თუ $a=e$ ასეთ შემთხვევაში ტაქტს ეწოდება ცარიელი ტაქტი. ცარიელი ტაქტი შესაძლებელია მაშინაც, როცა შესასვლელი სტრიქონი ცარიელია, ხოლო თუ მაღაზია ცარიელია ე. ი. მაღაზიაში გვაქვს მხოლოდ z_0 სიმბოლო, შემდგომი ტაქტი შეუძლებელია. $\vdash^i$ აღინიშნება $\vdash$ დამოკიდებულების i ხარისხი ე.ი. მიმდევრობით შესრულებული ტექსტების i რაოდენობა, ხოლო $\vdash^*$ და $\vdash^+$ აღინიშნებიან შესაბამისად

⊢ დამოკიდებულების ტრანზიტულ-რეფლექსური და ტრანზიტული ჩაკეტვები. (q_0,w,z_0) არის საწყისი კონფიგურაცია, სადაც $W\in\Sigma^*$, ხოლო საბოლოო კონფიგურაციაა (q,e,α) თუ $q\in F$ და $\alpha\in\Gamma^*$. თუ $(q_0,W,z_0)\vdash(q,e,\alpha)$ რაიმე $q\in F$ და $\alpha\in\Gamma^*$ მაშინ ვიტყვით, რომ W სტრიქონს უშვებს AM ავტომატი ან რაც იგივეა $W\in L(AM)$ ე. ი. W ეკუთვნის AM ავტომატით განსაზღვრულ ენას და L(AM) არის AM ავტომატით დაშვებული ყველა სტრიქონის სიმრავლე.

3 თავი
ბუნებრივი ენების კომპიუტერული დამუშავება

3.1 PTQ ფრაგმენტი. ამ თავში წარმოდგენილია ინგლისური ენის ფრაგმენტი, რომელიც აღწერილია მონტეგიუს შრომაში[2]. ვიწყებთ პატარა ფრაგმენტით და მას ვაფართოვებთ თანდათანობით. ყოველ ნაბიჯზე სინტაქსი და სემანტიკა გაანალიზებულია ფართოდ. სპეციალური ყურადღება ექცევა მოტივაციას და ანალიზის დაზუსტებას. სპეციალური ყურადღება მიექცევა აგრეთვე დამუშავების ალგებრულ და ალგორითმულ ასპექტებს. ალგებრული განხილვა ითვალისწინებს იმის ახსნას, თუ რატომ ზოგიერთი დეტალები არიან ასე და არა სხვანაირად. ალგორითმული ასპექტები ეხება იმ მეთოდს, რომლის საშუალებითაც მარტივად წარმოიდგინება მნიშვნელობები. იმსათვის რომ მივაღწიოთ ამას ზოგიერთი წესები მოცემული იქნება ფორმულების გასამარტივებლად. ფრაგმენტი მდიდარია სემანტიკურად საინტერესო მოვლენებით. ქვემოთ მე მივუთითებ ზოგიერთ მათგანს კომენტარებით.

მოვლენის პირველი სახეობა, რომელიც ეხება ისეთ წინადადებებს რომელთა აზრი ნათელია და განიხილება თუ როგორ უნდა წარმოვადგინოთ ისინი

სტანდარტული პრედიკატული ლოგიკის გამოყენებით.

განვიხილოთ (1) და (2)

(1) John runs.

(2) Every man runs.

ეს წინადადებები გარეგნული ფორმით ჰგვანან ერთი მეორეს. მხოლოდ ქვემდებარით განსხვავდებიან. მაგრამ მათი აზრების წარმოდგენები არიან განსხვავებული. სტანდარტულ ლოგიკაში მათი აზრების წარმოდგენებია:

(3) run (John)

(4) ∀ X [man (x)→ run (x)] .

ეს გვაძლევს გამოსავალს იმისა, თუ როგორ მივიღოთ განსხვავებული ფორმულები მსგავსი წინადადებებისათვის. ანალოგიური საკითხი წარმოიშვება (5)-ის ომონიმიურობისას.

(5) Every man loves a woman.

ეს წინადადება შეიძლება იყოს გაგებული როგორც კონკრეტული რომელიმე ქალი შეყვარებულია ყოველი მამაკაცის მიერ (მაგალითად Brigitte Bardot), ან როცა ყოველი მამაკაცისათვის შეიძლება იყოს სხვადასხვა ქალი (ვთქვათ თავისი საკუთარი დედა). ამგვარად (5) არის ომონიმიური. ამ ორი შესაძლებლობის გამო. ასეთი სახის ომონიმიას ეწოდება "საზღვრების ომონიმია" ქვანტიფიკატორებისათვის. ორი სხვადასხვანაირი წაკითხვა ამ წინადადებისა გრამატიკულად მოცემულია (6)-სა და (7)-ში.

(6) ∀X[man(x)→∃y[woman(y)∧love(x,y)]]

(7) ∃y[woman(y)∧∀x[man(x)→love(x,y)]] .

მოვლენის მეორე სახეობა ეხება წინადადებებს, რომელთათვისაც ძნელია იმის თქმა, თუ როგორ უნდა იყოს წარმოდგენილი მათი მნიშვნელობები. განვიხილოთ (8) და (9)

(8) John seeks a unicorn.

(9) John finds a unicorn.

ამ ორ წინადადებას აქვს ერთნაირი ფორმა და მხოლოდ მათი ზმნები განსხვავდებიან ერთიმეორისაგან. ვინმემ შეიძლება იფიქროს რომ მათ უნდა ჰქონდეთ ერთნაირი აზრები აგრეთვე. ერთადერთი განსხვავება არის ის, რომ სხვადასხვა დამოკიდებულებებს გამოხატავენ ჯონსა და მარტორქას შორის. მაგრამ ეს ასე არაა. მიდგომა, რომელიც ამბობს, რომ seek-დამოკიდებულება არის ყოველთვის რაიმე დამოკიდებულება ორ ინდივიდს შორის არ უნდა იყოს მისაღები. ჩვენ უნდა გავითვალისწინოთ (8)-სთვის რაიმე

აზრი, რომლიდანაც არ გამომდინარეობს რომ მარტორქები არსებობენ. მაგრამ (8) შეიძლება იყოს გამოყენებული იმ სიტუაციაშიც როცა მარტორქები არსებობენ. ამგვარად გვაქვს ომონიმია ამ ორ შესაძლებლობას შორის. მას აქვს წაკითხვა, რომლიდანაც გამომდინარეობს, რომ ყველაზე მცირე ერთი მარტორქა მაინც არსებობს (რეფერენციალური წაკითხვა) და წაკითხვა, რომლიდანაც ეს არ გამომდინარეობს(არარეფერენციალური წაკითხვა). რეფერენციალური და არარეფერენციალური წაკითხვის სხვა მაგალითებია (10), (11) და (12).

(10) John talks about a unicorn. (ჯონი საუბრობს მარტორქის შესახებ)

(11) John wishes to find a unicorn and eat it.(ჯონს სურს იპოვოს მარტორქა და შეჭამოს იგი)

(12) Mary believes that John finds a unicorn and that he eats it.(მერის სჯერა, რომ ჯონი იპოვის მარტორქას და შეჭამს მას)

მე-(9) წინადადება უზრუნველყოფს მხოლოდ რეფერენციალურ წაკითხვას. იგივეს ადგილი აქვს (13)-ისთვის.

(13) John tries to find a unicorn and wishes to eat it.(ჯონი ცდილობს იპოვოს მარტორქა და შეჭამოს იგი)

ომონიმია, რომელსაც ჩვენ ვასხვავებთ (8), (9), (11) და (12) წინადადებებში ლიტერატურაში ცნობილია როგორც "de-dicto/de-re" ომონიმია ან "სპეციფიკური/არასპეციფიკური" ომონიმია.

2. John runs (ჯონი მირბის)

ეს ფრაგმენტი შედგება ძალიან მარტივი წინადადებებისაგან. როგორიცაა John runs. გვაქვს სამი კატეგორია (სორტი): თერმების კატეგორია T, გარდაუვალი ზმნური ფრაზების კატეგორია IV და წინადადებების კატეგორია S. არსებობენ ძირითადი ბაზისური გამოსახულებები (გენერატორები) T და IV კატეგორიებისათვის. T კატე-

გორიის BT გენერატორების სიმრავლე შეიცავს PTQ ფრაგმენტის საკუთარ სახელებს. შემდგომში მათ დაემატება სპეციალური სახელი საილუსტრაციოდ Bigboss. სიმრავლეები B_T და B_{IV} განისაზღვრებიან ასე:

2.1. B_T={John,Bill,Mary,Bigboss}

2.2. B_{IV} ={run,walk,talk,rise,change}

2.3. End

ლოგიკაში B_T-ს ელემენტებს შეესაბამება e ტიპის კონსტანტები, Bigboss-ის გარდა. ინტენსიონალურ ლოგიკაში e ტიპის კონსტანტებს შორის ჩვენ განვასხვავებთ სამ სპეციალურ მათგანს:

2.3. {John,bill,mary}$\subset CON_e$

2.3. end

$C^{A,i,g}$=F(C)(i)(C$\in CON_e$)

2.4. მნიშვნელობის 1 პოსტულატი: ∃ u [u = α] , სადაც α ∈ { John, bill, mary} 2.4. end

2.5. განსაზღვრება. [α/Z]Φ აღნიშნავს ყველა თავისუფალი Z-ის ჩანაცვლებას α-თი Φ-ში.

2.6. თეორემა λU[Φ]α = [α/]Φ სადაც α ∈{John,bill,mary}

2.7. IL ფორმულას ეწოდება მოდალურად ჩაკეტილი თუ იგი არის შემდეგი IL ქვეალგებრის ელემენტი: <[{John,mary,bill}] , (VARτ)τ∈TY,RU{R∧,R}>

2.8. გამარტივების წესი

1. ვთქვათ Z∈VAR_{τ_1}, α∈ME_{τ_2} და β∈MEτ2

შემდეგ შევცვალოთ λZ[β](α)[α/Z]β თუ

1) β-ს არცერთი ცვლადი არ ხდება დაბმული ამ ჩასმის შედეგად და ან

2) Z არაა დაბმული β-ში ^ , H , W ან □ საზღვრებში ან

3) β არის მოდალურად ჩაკეტილი

Bigboss არ შეიძლება ითარგმნოს როგორც e ტიპის ხისტი კონსტანტა. ჩვენ შეგვიძლია იგი ვთარგმნოთ <s,e> ტიპის კონსტანტაში ანდა e ტიპის რაიმე კონსტანტაში ვთარგმნოთ და შემდეგ მას გავუკეთოთ ინტერპრეტაცია არახისტად. ჩვენ ავირჩევთ პირველ გზას.

2.9. bigboss ∈CON<s,e>

2.9. end.

bigboss კონსტანტის ინტერპრეტაცია არის ფუნქცია ინდექსიდან ინდივიდუუმში. ასეთ ფუნქციას ეწოდება ინდივიდუალური ცნება (კონცეპტი). ამგვარად ∧ John აღნიშნავს ინდივიდუალურ ცნებას. ∧ John-ით აღნიშნული ინდივიდუალურ ი ცნება არის კონსტანტა ფუნქცია, მაშინ როცა bigboss არაა კონსტანტა ფუნქცია. ვთქვათ ძალთა ბალანსი იცვლება და ბრეჟნევი ხდება bigboss რეიგანის ნაცვლად. ეს შეიძლება გამოისახოს წინადადებით (14).

(14) Bigboss changes.

(14) აზრი არ იქნება სწორად წარმოდგენილი ფორმულით, რომელიც ამბობს, რომ change პრედიკატი გამოიყენება რაღაცა ინდივიდუმზე. შეიძლება შეიცვალა აბსოლუტური ძალა რეიგანისა (შემცირდა) ან შეიცვალა აბსოლუტური ძალა ბრეჟნევისა (გაიზარდა). ანდა შეიძლება ორივეს ძალა შეიცვალა. წინადადება (14) ამბობს რომ ცნება ' Bigboss' შეიცვალა იმ აზრით რომ იგი ეხება სხვა პიროვნებას. ამგვარად (14)-ის აზრი შეიძლება იყოს წარმოდგენილი - ფორმულით, რომელიც ამბობს, რომ პრედიკატ changes-ს ადგილი აქვს ინდივიდუალური ცნებისათვის, რომელიც დაკავშირებულია Bigboss-თან. ასეთი ანალიზისათვის change უნდა იყოს < < s,e> , t > ტიპის. სინტაქსსა და სემანტიკას შორის ჰომომორფული დამოკიდებულების გამო ეს ნიშნავს, რომ ყველა გარდაუვალი ზმნები < < s,e> , t> ტიპის უნდა იყოს. ამგვარად,

2.10. { run,walk,rise,change}⊂CON<<s,e>,t>

2.11. run=run, walk′=walk, talk′=talk, rise=rise, change′=change

2.11. end

3.2 განსაზღვრულ ფრაზიანი გრამატიკები[3](DCG). როგორც ცნობილია ბუნებრივი ენის ძირითადი სინტაქსური ანალიზი შეიძლება აღიწეროს კონტექსტისაგან თავისუფალი გრამატიკით (CFG). კონტექსტისაგან თავისუფალი გრამატიკათა შემდგომ განზოგადოებას წარმოადგენს განსაზღვრულ ფრაზიანი გრამატიკები. ასეთ გრამატიკაში არატერმინალურ სიმბო-ლოდ შეიძლება ავიღოთ ისეთი გრამატიკული კატეგორიები, როგორიცაა წინადადება, ქვემდებარის ჯგუფი, ზმნის ჯგუფი, დამატების ჯგუფი და სხვა. შეგვიძლია განვმარტოთ, რომ წინადადება შედგება ქვემდებარის ჯგუფისაგან, ზმნის ჯგუფისაგან და ერთი ან ორი დამატების ჯგუფისაგან. თუ ჩვენ გავაგრძელებთ თვითეული ჯგუფის განმარტებას დავინახავთ, რომ ქვემდებარის ჯგუფი და დამატების ჯგუფები თავიანთი ზოგადი აღნაგობით არ განსხვავდებიან ერთიმეორესაგან. ამიტომ მათთვის შეიძლება შემოვიღოთ ერთი საერთო სახელი სახელის ჯგუფი და იგი განვმარტოთ. მეორეს მხრივ წინადადებაში რამდენი სახელის ჯგუფია დამოკიდებულია ზმნის ჯგუფზე. ამიტომ მიზანშეწონილია წინადადების განმარტება დავუკავშიროთ ზმნის ჯგუფის განმარტებას და ზმნის ჯგუფის მიხედვით შემოვიტანოთ სახელთა ჯგუფები წინადადებაში და ეს პროცესი გავაგრძელოთ მანამ სანამ არ დავალთ ისეთ კატეგორიებამდე როგორიცაა მეტყველების ნაწილები. მაგალითად არსებითი სახელები, ზედსართავები, კავშირები, ზმნის-ზედები და სხვა. მათი განმარტება შეიძლება მათში შემავალი ელემენტების (სიტყვების) ჩამოთვლით, რომლებიც შემდგომ განმარტებას არ ექვემდებარებიან.

უნდა იყოს მიცემით ბრუნვაში (შეესაბამება პირდაპირი ობიექტის ჯგუფს სიღრმისეულ დონეზე). ანალოგიურად ჩამოყალიბდება წესები ზმნის ფორმებისათვის მეორე და მესამე სერიებიდან. ყველა შემთხვევაში ზმნის ფორმა უნდა იყოს შეთანხმებული რიცხვში სუბიექტის ჯგუფთან და პირში სუბიექტისა და ობიექტის ჯგუფებთანაც. ამავე დროს უნდა იყოს გათვალისწინებული, რომ ხსენებული ჯგუფები შეიძლება იყვნენ დალაგებულნი წინადადებაში ნებისმიერად. ახლა ვნახოთ თუ როგორ შეიძლება სინტაქსური წესები გამოვსახოთ DCG-ში და შემდეგ როგორ გადაიწერებიან ისინი პროლოგზე. S(X) პრედიკატით აღვნიშნოთ ის ფაქტი, რომ X არის სია, რომლის ელემენტებია რაიმე წინადადების სიტყვები, იგივე მიმდევრობით, რომლითაც ისინი გვხვდებიან წინადადებაში, ე. ი. P(X) არის ერთარგუმენტიანი პრედიკატი, სადაც X არის სია და სიის ელემენტები არიან სტრიქონული ტიპის კონსტანტები ან ცვლადები). ანალოგიურად NP(X) გამოსახავს სახელის ჯგუფს და VP(X) ზმნის ჯგუფს. ამგვარად, ზემოთ მოცემული წესები შეიძლება გამოვსახოთ ჰორნის წინადადებებით შემდეგნაირად:

S(X)←VP(X).

VP(X)← append(X1,X4,X)& append(X2,X3,X4,)& (2)

np| (X1)&VP1(X2)& np2(X3)&

აქ append(X1,X2,X3) აღნიშნავს X3 სიის გახლეჩას X1 და X2 სიებად, სადაც X1 და X2 სიების ელემენტთა კონკატენაცია იგივეა რაც X3-ის ელემენტთა კონკატენაცია. ზემოთ მოცემული (2) წინადადებები ჩვენ შეგვიძლია გადავწეროთ უფრო მოხერხებული სახით. ამისათვის, რაიმე X სია ჩვენ შეგვიძლია წარმოვიდგინოთ ორ სიად Y და Z, სადაც X არის Y და Z-ის კონკატენაცია. კერძოდ, Z შეიძლება იყოს ცარიელი სია, რომელიც აღვნიშნოთ nil-ით. ამგვარად, S(X) პრედიკატი შეიძლება შევცვალოთ ახალი

პრედიკატით S(Y,Z), ზემოთ მიღებული შეთანხმების გათვალისწინებით. ამის შემდეგ, (2) წინადადებები გადაიწერებიან ასე:

S(X,Z)←VP(X,Z).

VP(X,V)←nP1(X,Y)&VP1(Y,Z)& nP2(Z,V). (3)

განსახილავი წინადადება ჩვენ შეგვიძლია მივაწოდოთ პროლოგ პროგრამას მიზნის საშუალებით. ასე მაგალითად,

← S(პეტრე, აშენებს, ახალ, სახლს, nil, nil)

თუ ჩვენ გამოვიყენებთ მაკკორდის მიერ შემოტანილ აღნიშვნებს – – > და % , სადაც ისარი არის ოპერატორი, რომელიც აცალკევებს წესის დასათაურებას წესის ტანისაგან, ხოლო % ოპერატორი გამოჰყოფს ტანის ელემენტებს ერთიმეორესაგან, მაშინ (3) წინადადებები გადაიწერება ასე:

S– – > VP.

VP– – > nP1 : VP1:nP2. (4)

(4)-ში S, np, np1 და np2 არიან არატერმინალური სიმბოლოები და წარმოადგენენ (3)-ში მოცემული პრედიკატების შემოკლებულ ჩაწერას. ახლად შემოტანილი ოპერატორებისათვის უნდა დადგინდეს მათი პრიორიტეტები. ცხადია %-ს უნდა ჰქონდეს უფრო მაღალი პრიორიტეტი ვიდრე – – > -ს. IBM პროლოგში ოპერატორის პრიორიტეტი მოიცემა ჩაშენებული პრედიკატით OP. მაგ. OP("– – >",rl,20), სადაც rl აღნიშნავს ორ ოპერანდიან მარჯვნივ ასოციატიურ ოპერატორს. არატერმინალები განსაზღვრავენ სიტყვების სიათა კლასებს, ხოლო ტერმინალები განსაზღვრავენ ცალკეულ სიტყვებს. ტერმინალები გამოისახებიან +T ფორმით, სადაც + არის პრეფიქსული ოპერატორი და უნდა ჰქონდეს სხვა ოპერატორებზე უფრო მაღალი პრიორიტეტი. ზემოთ მიღებული შეთანხმებებით, ახლა შევადგინოთ მარტივი გრამატიკა, რომელიც გამოდგება "პეტრე აშენებს ახალ

სახლს"-ის მსგავსი წინადადებების გამოცნობისათვის:

s – – > p

VP – – > np1:vp1:np2.

vp1– – > v

np1– – > პეტრე

np2 – – > ad : N; N.

v – – > + აშენებს. (5)

N – – > +სახლს.

ad – – > + ახალ.

ეს გრამატიკა ჩვენ შეგვიძლია იოლად გადავიყვანოთ ჰორნის წინადადებებში თუ გავიხსენებთ, რომ ყოველი არატერმინალი nt გადაიყვანება ორადგილიან პრედიკატში nt(x,y) და : ოპერატორს შევცვლით -ით. + t, სადაც t არის ტერმინალური სიმბოლო, უნდა გადავიყვანოთ პირობაში X=t, y წესში მისი პოზიციის გათვალისწინებით. მაგალითად. წესი

vp– – > np : vp1 : + ახალ : n, გადაიყვანება წინადადებაში

vp(x,v)← np(x,y)& vp1(y,z)& z= ახალ.u& n(u,v)

შესაძლებელია ამ წესის შემდგომი გამარტივება თუ z-ს შევცვლით მისი მნიშვნელობით:

vp(x,v)← np(x,y)& vp1(y,ახალ.u)& n (u,v).

ასევე წესი n– – > + სახლს გადაიყვანება წინადადებაში

n(x,y)← x= სახლს. y.

ხოლო X-ის გამორიცხვით მიიღება

n(სახლს.y,y).

ამგვარად, (5) გრამატიკა გადაიყვანება შემდეგ პროლოგ პროგრამაში:

s(x,y)← vp(x,y).

vp(x,u)← np1(x,y)& vp1(y,z)& np2(z,u).

vp1(x,y)← v(x,y).

np2(x,z)← ad(x,y)& n(y,z).

np2(x,y)← n(x,y).

np1(პეტრე.x,x).

v(აშენებს.x,x).

n(სახლს.x,x).

ad(ახალ.x,x).

მიღებული პროლოგ პროგრამა შეიძლება გამოყენებულ იქნეს როგორც (5) გრამატიკით განსაზღვრული წინადადებების გამოსაცნობად, ასევე (5)-ით განსახღვრული წინადადებების გენერირებისათვის. თუ (6) პროგრამას მივცემთ მიზანს ←s(x, nil)& write (x) & fail. პასუხად მივიღებთ ყველა წინადადებას განსაზღვრულს (5) გრამატიკით, ხოლო თუ მივცემთ მიზანს

←S(პეტრე.აშენებს.ახალ.სახლს,nil).

მოგვცემს დადებით პასუხს, რაც ნიშნავს რომ " პეტრე აშენებს ახალ სახლს" წინადადება ეკუთვნის (5) გრამატიკით განსაზღვრულ ენას. როგორც ადვილი შესამჩნევია ასეთი გრამატიკა პრაქტიკულად უვარგისია მისი შემდგომი გაფართოების გარეშე. პირველ რიგში ძალზე მოუხერხებელია პრაქტიკული რეალიზაციის თვალსაზრისით ქართული ენის სიტყვის ფორმების ტერმინალურ სიმბოლოებად გამოცხადება, რაც იმას ნიშნავს რომ სიტყვათა ყველა ფორმები უნდა შევიტანოთ კომპიუტერულ ლექსიკონში. ამიტომ, ამ ეტაპზე შეიძლება ვიგულისხმოთ, რომ ყოველი სიტყვა შეიძლება დახასიათებულ იქნეს მისი უცვლელი ნაწილით (ლექსიკური მნიშვნელობით) და იმ მორფოლოგიური კატეგორიებით, რომლებიც აღწერენ სრულად ამ სიტყვას. ჩვენ ვიგულისხმებთ რომ სიტყვის ფორმა შეცვლილია მისი უცვლელი ნაწილით და მორფოლოგიური კატეგორიებით, რომლებიც ამ სიტყვას აღწერენ. ამისათვის შემოვიტანთ აქ ჩვენ სათანადო პრედიკატებს, რომლებსაც არგუმენტებად ექნებათ საჭირო მორფოლოგიური კატეგორიები და ასევე სხვა ინფორმაციაც. მაგალითად, არსებითი სახელებისათვის შეიძლება

შემოვიტანოთ პრედიკატი არს-სახ, მაშინ იმისათვის, რომ მივუთითოთ სიტყვის ფორმა სახლს საჭიროა არს-სახ პრედიკატს ჰქონდეს არგუმენტები, რომლებიც გვიჩვენებენ ბრუნვას, რიცხვს და სიტყვის უცვლელ ნაწილს. კერძოდ თუ დავწერთ არს-სახ (სახლ,მიც,მხ), სადაც პირველი არგუმენტი მიუთითებს სიტყვის უცვლელ ნაწილს, მეორე არგუმენტი ბრუნვას და მესამე არგუმენტი რიცხვს და ჩანაწერები: სახლ, მიც და მხ არიან არგუმენტების მნიშვნელობები. სახლ წარმოადგენს სიტყვის " სახლს" უცვლელ ნაწილს, ხოლო მიც და მხ წარმოაგენენ მიცემითი ბრუნვისა და მხოლობითი რიცხვს აღნიშვნებს შესაბამისად. როგორც მაგალითიდან ჩანს, პრედიკატის დასახასიათებლად არაა საკმარისი მისი სახელისა და არგუმენტების რაოდენობისა და რიგის ცოდნა, ასევე საჭიროა თვითეული არგუმენტის ტიპის ცოდნა ანუ დავახასიათოთ თვითეული არგუმენტისათვის მისი მნიშვნელობათა სიმრავლე. მოყვანილ მაგალითში პრედიკატს არს-სახ აქვს სამი არგუმენტი, რომელიც საკმარისი იყო მოცემული თერმის "სახლს" დასახასიათებლად, მაგრამ ეს იმას არ ნიშნავს, რომ საზოგადოდ არსებით სახელთა დასახასიათებლად საკმარისი იქნება სამი არგუმენტი. არგუმენტთა რაოდენობა დამოკიდებულია იმაზე, თუ რას ვისახავთ მიზნად ამ პრედიკატის შემოტანით. თუ ჩვენ ვისახავთ მიზნად დავახასიათოთ ყველა არსებითი სახელები მორფოლოგიურ დონეზე, მაშინ არგუმენტების რაოდენობა იქნება ვთქვათ m., ხოლო თუ ასევე ვიგულისხმებთ, რომ ეს პრედიკატი შეიცავდეს ინფორმაციას არსებით სახელთა დასახასიათებლად სემანტიკურ დონეზეც, მაშინ არგუმენტების რაოდენობა იქნება უფრო მეტი. თუ ეს პრედიკატი გვჭირდება როგორც საწყისი ინფორმაცია არსებითი სახელის მორფოლოგიური კატეგორიების დასადგენად, მაშინ მას უნდა ჰქონდეს არგუმენტები, სიტყვის ფუძისათვის, ბრუნების ტიპისათვის და სხვა ისეთ-ინფორმაციისათვის, რომელიც ჩვეულებრივ მოიცემა

მანქანური თარგმნის კომპიუტერულ ლექსიკონებში არსებითი სახელის ფუძეებისათვის. ამგვარად, ტერმინალური სიმბოლოები შეიცვლება პრედიკატებით, რომლებიც იღებენ მნიშვნელობებს თავიანთი არგუმენტებისათვის ცოდნის ბაზიდან. ე.ი. ასეთი პრედიკატები უნდა იყოს განსაზღვრული ცოდნის ბაზაში. ასეთი პრედიკატების ცოდნის ბაზაში შეტანა ხდება უფრო დაბალი დონის ანალიზის დროს. მაგალითად, სინტაქსური ანალიზისათვის უფრო დაბალი დონის ანალიზი იქნება მორფოლოგიური ანალიზი. გარდა ამისა, იმისათვის, რომ წესებში მივუთითოთ ჯგუფებს შორის არსებული მოთხოვნების დაცვა, არატერმინალური სიმბოლოების შესაბამისი პრედიკატებისათვის უნდა შემოვიტანოთ კიდევ დამატებითი არგუმენტები. მაგალითად, არსებითი სახელის ჯგუფსა და ზმნის ჯგუფს შორის რიცხვში შეთანხმებისათვის და სხვა. ასეთნაირად გაფართოებული გრამატიკა უკვე გამოდგება იმისათვის, რომ დავადგინოთ რაიმე წინადადება არის თუ არა სინტაქსურად სწორად შედგენილი. იმისათვის, რომ ასეთმა გრამატიკამ მოგვცეს აგრეთვე მოცემული წინადადების სინტაქსური სტრუქტურა, საჭიროა ყოველ არატერმინალურ სიმბოლოს შესაბამის პრედიკატს დავუმატოთ ახალი არგუმენტი, რომელიც იქნება სია და დასაშვებია, რომ ამ სიის ელემენტები იყოს კვლავ სია და ა. შ.

3.3 გ. ერბახი ანალიზისა და გენერაციის ქვემოდან ზემოთ მიმართული ალგორითმი[4]. ეს ალგორითმი არის ქვემოდან ზემოთ მიმართული ცხრილური ანალიზის ალგორითმი, რომლის ლექსიკური ერთეულების მოძებნის ფაზა დაზუსტებულია ისე, რომ იგი გამოდგეს გენერაციისათვისაც.

ძირითადი განსხვავება ანალიზსა და გენერაციას შორის არის შემდეგი:

1. ანალიზის დროს გამოიყენება ის სიტყვები, რომლებიც გვხვდება შესავალ სტრიქონში, მაშინ როცა

გენერაციის დროს გამოიყენება ლექსიკონის ყველა ერთეულები.

2. ერთეულები, რომლებიც გამოიყენებიან ანალიზისას არიან დალაგებულნი შესასვლელ სტრიქონში მათი შეხვედრის რიგის მიხედვით, მაშინ როცა გენერაციის დროს ერთეულები შეიცავენ ინფორმაციას იმის შესახებ, თუ რომელ სტრიქონებში შეიძლება შეგვხვდნენ ისინი. აქედან გამომდინარე შეიძლება დავასკვნათ, რომ ანალიზი არის გენერაციის უფრო შეზღუდული სახეობა, რადგან ანალიზის დროს ცნობილია რომელია ასაგები სტრიქონი[5]. ამიტომ, აგებულია ერთიანი ალგორითმი, როგორც გენერაციისათვის ასევე ანალიზისათვის, სადაც განსხვავება თავს იჩენს მხოლოდ ალგორითმის ინიციალიზაციის ფაზაში. ალგორითმი განასხვავებს აქტიურ და პასიურ ერთეულებს. რაიმე წესი არის გამოყენებული რაიმე ერთეულზე, თუ ერთეულის კატეგორია უკავშირდება წესის მარჯვენა მხარის პირველ კატეგორიას მაშინ ახალი ერთეულის დაბოლოება არის წესის მარჯვენა მხარის დარჩენილი ნაწილი. თუ დარჩენილი ნაწილი არაა ცარიელი, მაშინ ერთეულს ეწოდება აქტიური ერთეული და მას შეუძლია მიუერთდეს რაიმე პასიურ ერთეულს, რომლის კატეგორია უკავშირდება დარჩენილი ნაწილის ყველაზე მარცხენა ელემენტის კატეგორიას. რაიმე ერთეული არის <Span,String,Category, Remainder>. Span არის წყვილი <დასაწყისი ძირი, ბოლო ძირი> .

String არის სიტყვების სია.

Category არის გრამატიკის რაიმე არატერმინალური სიმბოლო.

Remander არის კატეგორიების სია. თუ სია ცარიელია, მაშინ ერთეულს ეწოდება პასიური(დასრულებული), წინააღმდეგ შემთხვევაში იგი არის აქტიური დაუსრულებელი). ერთეულები არიან აქტიურები ან დაკიდებულები. აქტიურ ერთეულთა სიმრავლეს ეწოდება ცხრილი, ხოლო დაკიდებულ ერთეულთა სიმრავლეს-ნიმუში. დაკიდებულ

ერთეულებს აქვთ პრიორიტეტები და ისინი ემატებიან ცხრილს მათი პრიორიტეტების კლების მიხედვით. როცა დაკიდებული ერთეული ემატება ცხრილს, იგი ხდება აქტიური და ახალი დაკიდებული ერთეულები იქმნებიან რაიმე წესის გამოყენებით ერთეულზე ან ერთეულთა კომბინირებით, როცა ერთერთი აქტიურია და მეორე პასიური. ასეთი ალგორითმი საშუალებას იძლევა გამოვიყენოთ სხვადასხვა გარჩევის სტრატეგიები.

3.4 ხეთა შეერთების გრამატიკები (TAG). ხეთა შეერთების გრამატიკა (Tree-Adjoining Grammar) გამიზნულია სტრიქონთა გენერაციის სისტემისათვის, უფრო ზუსტად, იმ სტრუქტურების გენერაციისათვის, რომლებიც წარმოდგენილი არიან ხეების საშუალებით. იმისათვის, რომ ავღწეროთ ხეების გამოყვანა ობიექტურ ენაზე, საჭიროა ეს ხეები გამოვიყვანოთ ობიექტური ენის ხეებისაგან. ასეთი გამოყვანები საინტერესოა, როგორც სინტაქსური - ასევე სემანტიკური თვალსაზრისით. TAG შემოტანილია E. Joshi-ის მიერ[6]. ხეთა შეერთების ენა (TAG) ეკუთვნის კონტექსტზე დამოკიდებულ ენებს, უფრო ზუსტად, შუალედურ კონტექსტის მგრძნობიარე ენათა კლასს. ხეთა შეერთების გრამატიკა ეწოდება ხუთეულს (Σ, N , I , A , S), სადაც:

(1) Σ არის ტერმინალურ + სიმბოლოების სასრული სიმრავლე;

(2) N არის არატერმინალური სიმბოლოების სასრული სიმრავლე;

(3) S არის გამოყოფილი არატერმინალური სიმბოლო, S∈ N;

(4) I არის სასრული ხეების სიმრავლე, რომელსაც ეწოდება საწყისი ხეების სიმრავლე.

(5) A არის სასრული ხეების სასრული სიმრავლე რომელსაც ეწოდება დამხმარე ხეთა სიმრავლე.

ხის შიგა კვანძები მონიშნულია არატერმინალური სიმბოლოებით, რომელთაც აღვნიშნავთ დიდი ლათინური ასოებით. ხოლო ხის ფოთლები მონიშნულია ტერმინალური სიმბოლოებით ან არატერმინალური სიმბოლოებით. ტერმინალურ სიმბოლოებს აღვნიშნავთ პატარა ლათინური ასოებით. ხის ფოთოლი, რომელიც მონიშნულია არატერმინალური სიმბოლოთი მიწერილი აქვს სიმბოლო ↓, რომელიც აღნიშნავს, რომ ეს კვანძი განკუთვნილია ჩასმისათვის. არსებობს მხოლოდ ერთი კვანძი, რომელიც მონიშნულია იგივე სიმბოლოთი, რაც ხის ძირი და ამ კვანძს ეწოდება კვალი და იგი დამატებით მონიშნულია სიმბოლოთი * ლექსიკალიზებულ TAG-ში ყველაზე მცირე ერთი მაინც ტერმინალური სიმბოლო (კვალი) უნდა იყოს საწყის ან დამხმარე ხეში. ხეს აღებულს {IUA}-დან ეწოდება ელემენტარული ხე. ასეთ ხეს ვუწოდებთ X ტიპის ელემენტალურ ხეს თუ მისი ძირი მონიშნულია X-ით. ხე, რომელიც შედგენილია ორი ხის კომპოზიციით ეწოდება გამოყვანილი ხე. TAG იყენებს შეერთებისა და ჩასმის კომპოზიციურ ოპერაციებს. შეერთების ოპერაცია

β დამხმარე ხისა და ნებისმიერი α ხისაგან აგებს ახალ γ ხეს. ვთქვათ n არის α-ს არა ჩასმითი კვანძი მონიშნული X-ითა და β-ს ძირი მონიშნულია X-ით. γ, რომელიც მიიღება β-ს α-ს n-ურ კვანძთან შეერთებით და განისაზღვრება შემდეგნაირად:

1. n-ით განსაზღვრული ქვეხე, ვუწოდოთ მას δ, ჩამოეჭრება α-ს.

2. β-დამხმარე ხე მიუერთდება n კვანძში α-ს და β-ს ძირი გაიგივდება n კვანძთან.

3. β-ს კვალს მიუერთდება δ ქვეხე და δ ძირი გაიგივდება β -ს კვალთან.

ჩასმის ოპერაცია მოქმედებს მხოლოდ ხის ფოთლებზე, რომლებიც მონიშნული არიან არატერმინალური

სიმბოლოებით და ↓-ით. კვანძი, სადაც ხდება ჩასმა შეიცვლება ჩასასმელი ხით. შესაძლებელია შემოვიტანოთ შეზღუდვები შეერთების ოპერაციაზე. მაგალითად, შესაძლებელია მოცემულ კვანძში განვსაზღვროთ T დამხმარე ხეების სიმრავლე T⊆A, რომლის ელემეტი შეიძლება შევაერთოთ მოცემულ კვანძში. ასეთი შეზღუდვა აღვნიშნოთ SA(T)-თი. ასეთ შემთხვევაში შეერთება არაა სავალდებულო. NA-თი აღვნიშნოთ SA(T), სადაც T ცარიელი სიმრავლეა. ე. ი. მოცემულ კვანძში შეერთება აკრძალულია. OA(T)-თი აღვნიშნოთ აუცილებელი შეერთება, როცა მოცემულ კვანძში აუცილებლად უნდა მოხდეს შეერთება რომელიმე t∈T-თი. თუ არავითარი შეზღუდვები არაა შეერთებაზე და არა გვაქვს ჩასმის კვანძები ელემენტარულ ხეებში ასეთ TAG-ს ეწოდება ხეთა შეერთების მკაცრი გრამატიკა. გამოყვანა გრამატიკაში განსაზღვრულია ისეთი ხის საშუალებით, რომელიც გვიჩვენებს თუ როგორ მიიღება საბოლოო ხე საწყისი ხისაგან შეერთებისა და ჩასმის ოპერაციების გამოყენებით; გრამატიკას ეწოდება ლექსიკალიზებული თუ იგი შედგება:

1. სტრუქტურათა სასრული სიმრავლისაგან, სადაც ყოველი სტრუქტურა დაკავშირებულია რაიმე ლექსიკურ ერთეულთან და ამ ლექსიკურ ერთეულს ეწოდება ამ სტრუქტურის კვალი.

2. რაიმე ოპერაციისაგან ან ოპერაციებისაგან სტრუქტურათა კომპოზიციისათვის მოითხოვება, რომ კვალი არ უნდა იყოს ცარიელი ლექსიკური ერთეული. ლექსიკონი შედგება სტრუქტურათა სასრული სიმრავლისაგან, სადაც ყოველი სტრუქტურა დაკავშირებულია რაიმე კვალთან. ლექსიკონით განსაზღვრულ სტრუქტურებს ეწოდებათ ელემენტარული სტრუქტურები, ხოლო სტრუქტურები რომლებიც მიიღებიან სტრუქტურათა კომბინირებით ეწოდებათ გამოყვანილი სტრუქტურები. მოითხოვება, რომ სტრუქტურა იყოს სასრული ზომის და

ოპერაციას გადაჰყავდეს სტრუქტურათა სასრული სიმრავლე სასრული რაოდენობის მქონე სტრუქტურებში. სტრუქტურა არის ლექსიკალიზებული თუ არსებობს ერთი მაინც ლექსიკური ერთეული (არაცარიელი), რომელიც გვხვდება მასში. გრამატიკა, რომელიც შედგება ლექსიკალიზებული სტრუქტურებისაგან არის ლექსიკალიზებული. შესაძლებელია კონტექსტისაგან თავისუფალი გრამატიკების ლექსიკალიზაცია TAG-ებით. გრამატიკული ფორმალიზმების ლექსიკალიზაცია საინტერესოა როგორც ლინგვისტური, ასევე ფორმალური თვალსაზრისით. თუ მივიღებთ თვალსაზრისს, რომ არ იყოს სრულად განცალკევებული თავიანთი ლექსიკური რეალიზაციისაგან, მაშინ ყოველი ელემენტარული სტრუქტურა სისტემურად დაკავშირებულია თავის ლექსიკურ კვალთან ლექსიკალიზებული მიდგომის დროს. ეს სტრუქტურები აზუსტებენ გაფართოებულ არეებს, რომლებზედაც შეიძლება დავადოთ შეზღუდვები. კონტექსტისაგან თავისუფალი წესების ლექსიკალიზაციის პროცესი გვაიძულებს ჩვენ გამოვიყენოთ ჩასმისა და შეერთების ოპერაციები სტრუქტურათა კომბინირებისათვის, ეს კი ხდის ფორმალიზმს ისეთს, რომელიც ხვდება კონტექსტის მგრძნობიარე ენათა კლასში. ჩასმისა და შეერთების ოპერაციები აიოლებენ კონტექსტისაგან თავისუფალი ენების ლექსიკალიზაციას. ასეთი ფორმალიზმი კი გვაძლევს ლექსიკალიზებულ ხეთა შეერთების ენას. TAG, თავის მხრივ, ჩაკეტილია ლექსიკალიზაციის მიმართ.

3.5 სტეკიანი ავტომატები[7]. ახლა განვიხილოთ უფრო სრულყოფილი ავტომატების მოდელი ვიდრე განხილული მაღაზიური ავტომატები იყო. ამ ავტომატების გამოთვლითი სიძლიერე აღწევს ტიურინგის მანქანის შესაძლებლობებს. ასეთი ავტომატი შეიძლება ეფექტურად გამოყენებულ იქნეს როგორც პროგრამირების რაიმე ენისათვის კომპილიატორის შესაქმნელად, ასევე ბუნებრივი ენების

სინტაქსური ანალიზისათვის. სტეკიანი ავტომატი შეიცავს შესასვლელ ჯაჭვს, მდგომარეობებსა და სტეკს. მას შეუძლია წაიკითხოს სიმბოლო შესასვლელი ჯაჭვიდან ე. ი. მისმა მიმთითებელმა იმოძრაოს ჯაჭვის თავიდან მისი ბოლოს მიმართულებით ან პირიქით, შეუძლია სტეკის თავში ჩაწეროს რაიმე სიმბოლო ან წაშალოს, ასევე შესაძლებელია სტეკის მიმთითებლის მოძრაობა სტეკის შიგნით. თუ ასეთ ავტომატს მივცემთ შესაძლებლობას მოახდინოს ჩაწერა სტეკის შიგნითაც, მაშინ მისი შესაძლებლობები კიდევ უფრო გაიზრდება. სტეკური ავტომატი მუშაობს შემდეგნაირად:

1. იგი გადადის ერთი მდგომარეობიდან მეორეში.

2. მას შეუძლია გადაადგილდეს ერთი სიმბოლოთი შესასვლელი ჯაჭვიდან წაკითხული სიმბოლოს მარჯვნივ ან მარცხნივ.

3. მას შეუძლია შეასრულოს შემდეგი მოქმედებებიდან ერთ-ერთი.

a. გადაადგილოს სტეკის მიმთითებელი ერთი სიმბოლოთი სტეკიდან წაკითხული სიმბოლოს მარჯვნივ ან მარცხნივ.

b. თუ იგი კითხულობს სტეკის თავში მოთავსებულ სიმბოლოს, მაშინ მას შეუძლია ჩაწეროს სტეკის თავში სიმბოლოები და წაშალოს სტეკის თავიდან წაკითხული სიმბოლო. ფორმალურად $A = (K,\Sigma,\notin\hat{}\$,\Gamma,\delta,q_0,Z_0,F)$

სტეკიანი ავტომატი შედგება შემდეგი ელემენტებისაგან:

1. K არის მდგომარეობის სასრული არაცარიელი სიმრავლე;

2. Σ არის შესასვლელი ჯაჭვის სიმბოლოების სასრული არაცარიელი სიმრავლე;

3. $\notin$ და $ არიან სიმბოლოები, რომლებიც არ ეკუთვნიან Σ -ს და გვხვდებიან შესასვლელი ჯაჭვის თავსა და ბოლოში

შესაბამისად, რომლებიც გამოიყენებიან შესასვლელი ჯაჭვის თავსა და ბოლოს დასაფიქსირებლად;

4. Γ არის სტეკის სიმბოლოების სასრული არაცარიელი სიმრავლე;

5. $Z_0 \in \Gamma$ და გვხვდება სტეკის ბოლოში მხოლოდ. იგი გამოიყენება იმის აღსანიშნავად, რომ მივაღწიეთ სტეკის ძირს;

6. δ არის ფუნქცია, რომლის განსაზღვრის

არეა KX(∑∪{¢^$}`XΓ და მნიშვნელობათა არეა

$\{-1,0,1\}XKX\{-1,0,1\}X\Gamma^*$ ისეთი, რომ ყოველი q, a, და Z-სათვის:

a. თუ δ(q,a,z)=(d,q′,e,w) და w≠z, მაშინ e = 0;

b. თუ δ(q,a,z_0)=(d,q′,e,y), მაშინ y=z_0w,

$w \in \Gamma^*$;

7. $q_0 \in K$ და ეწოდება საწყისი მდგომარეობა;

8. F∈K და ეწოდება საბოლოო მდგომარეობათა სიმრავლე. სტეკიანი ავტომატის შესასვლელ ჯაჭვი არის Σ^* $-ის ელემენტი. საბოლოო მდგომარეობები გამოიყენება იმისათვის, რომ გამოვიტანოთ ის ჯაჭვები, რომლებიც ამ გრამატიკით განსაზღვრულ ენას ეკუთვნიან. ჩანაწერი δ(a, q,z)=(d,q′,e,z) ნიშნავს. რომ თუ სტეკიანი ავტომატი (SA) იმყოფება q მდგომარეობაში, სტეკის თავში არის Z და შესასვლელ სტრიქონიდან კითხულობს a სიმბოლოს მაშინ:

1. იგი გადადის q′ მდგომარეობაში;

2. იგი მოძრაობს მარცხნივ (შესასვლელი სტრიქონის მიმართ), თუ d=-1 ან მარჯვნივ თუ d=1 ან არ მოძრაობს თუ d=0;

3. იგი მოძრაობს (სტეკის მიმართ) მარცხნივ თუ e=-1 ან მარჯვნივ თუ e=1 ან არ მოძრაობს თუ e=0.

ჩანაწერი δ(q,a,z)=(d,q′,0,w) და Z≠W ნიშნავ რომ თუ SA არის (q,a,z) კონფიგურაციაში, მაშინ

1. იგი მოძრაობს შესასვლელ სტრიქონზე d-ს მიხედვით;

2. გადადის q′ მდგომარეობაში;

3. სტეკში წერს W-ს Z-ის ნაცვლად.

6a პირობა კრძალავს სტეკზე მოქმედებებს, როცა e=0 და 6b პირობა კრძალავს მაღაზიის დაცარიელებას, ე.ი. Z_0 - ის წაშლას. სტეკიანი ავტომატის მიმდინარე მდგომარეობა განისაზღვრება წასაკითხავი სტრიქონით, მიმდინარე q მდგომარეობითა და სტეკის შიგთავსით, ე.ი. SA ავტომატის მდგომარეობა (M) არის $KXQ(\Sigma\cup\{e,\$,a\})^{*}X(\Gamma\cup\{b\})^{*}$ სიმრავლის ელემენტი, სადაც a არის შესავალი სტრიქონის მიმდინარე სიმბოლოს მიმთითებელი, ხოლო b არის სტეკის თავის მიმთითებელი. ახლა განვსაზღვროთ V – დამოკიდებულება SA ავტომატისათვის. ვთქვათ $SA=(K,\Sigma,\notin\$,\Gamma,\delta,q_0,Z_0,F)$, |– დამოკიდებულება SA^{+}c მდგომარეობებს შორის განისაზღვრება შემდეგნაირად: თუ $k,l\geq 1$, $a_1,a_2, \ldots, a_k$ ეკუთვნის $\Sigma\cup\{\notin\$\}$; a_{k+1} |-, $Z_1,\ldots,Z_e$ ეკუთვნის Γ, Y ეკუთვნის Γ^{*} ეკუთვნის Γ-ს, მაშინ

1. თუ $\delta(q,a_i,Z_j)=(d,q',e,z_j)$, სადაც $1\leq i\leq k$ და $1\leq j\leq l$ აკმაყოფილებს პირობებს: i. $d\geq 0$ თუ i=1;

ii. $e\geq 0$ თუ j=1 და

iii. $e\leq 0$ თუ j=l, მაშინ $(q,a_1 \ldots \bar{a}a_i\ldots a_k, z_1\ldots Z_j b\ldots Z_l)\vdash(q',a_1\ldots \bar{a}a_{i+d}\ a_{k+1}, z_1 \ldots z_{j+e}b\ldots z_l)$;

2. თუ $\delta(q,a_i,Z)=(d,q',0,w)$ და $1\leq i\leq k$ აკმაყოფილებს i პირობას, მაშინ $(q,a_1\ldots \bar{a}a_i\ldots a_k, yzb)\vdash (q ,a_1\ldots ba_{i+d}\ldots a_k, ywb)$.

ამ დამოკიდებულებით ჩვენ შეგვიძლია აღვწეროთ SA ავტომატის ერთი ნაბიჯი. ახლა განვიხილოთ ამ დამოკიდებულების ტრანზიტული და რეფლექსური ჩაკეტვა * :

$(q,x\bar{a}y,w_1bw_2)\vdash *(q',x'\bar{a}y',w',bw_2)'$, სადაც X,Y,X′

და y′ ჯაჭვებია $(\Sigma\cup\{\notin\$\})^*$-დან და w_1,w_2,w_1,w_2', ჯაჭვებია Γ^*-დან. თუ $k \geq 0$, f_i, g_i და h_i, სადაც $0 \leq i \leq k$ ისეთი, რომ $f_0=q$, $f_k=q'$, $g_0=x\bar{a}y$; $g_k=x'$ by′ , $h_0=w_1b\ w_2$, $h_k=w'_1bw'_2$, მაშინ $(f_i,g_i,h_i) \vdash (f_{i+1},g_{i+1},h_{i+1})$, სადაც $0\leq i\leq k$.

ვიტყვით, რომ SA ავტომატი უშვებს $X\in\Sigma^*$ ჯაჭვს თუ $(q_0,\bar{a}\in x\$,z_0b) \vdash {}^*(q,\notin x\$\bar{a},w,bw_2)$, სადაც $q\in F$ და $W_1,W_2\in\Gamma^*$.

SA ავტომატის მიერ დაშვებული ყველა ჯაჭვთა სიმრავლე აღვნიშნოთ L(SA)-თი და იგი არის ენა განსაზღვრული SA ავტომატით. მაღაზიური ავტომატის ანალოგიურად, აქაც ჩვენ შეგვიძლია შემოვიტანოთ არადეტერმინისტული SA ავტომატის განმარტება (იხ. Ginsburg...). SA ავტომატზე გარკვეული შეზღუდვების დადებით მიიღება ყველა აქამდე განხილული ავტომატები.

3.6 CAT2 ენის აღწერა[8]. CAT2 არის გამიზნული ბუნებრივი ენის კომპიუტერული დამუშავებისათვის. იგი შეიცავს ისეთ საშუალებებს რომელთა დახმარებით გაადვილებულია ბუნებრივი ენის სტრუქტურების კომპიუტერზე წარმოდგენა. პირველ რიგში იგი გამიზნულია მანქანური თარგმანისათვის, თუმცა მისი საშუალებით შესაძლებელია ბუნებრივი-ენის ანალიზი, ცოდნის დაგროვება და მისი გამოყენება სხვადასხვა მიზნისათვის. CAT2-ს საფუძვლად აქვს მიდგომა, რომელიც შემუშავებული იყო EUROTRA პროექტში კერძოდ <C,A>,T-ს სახელით ცნობილი მიდგომა მანქანური თარგმანისადმი[9]. იგი დაფუძნებულია კონსტრუქტორებისა, ატომებისა და ტრანსლიატორების ცნებებზე. კონსტრუქტორები და ატომები ქმნიან წარმოდგენის რაიმე დონეს, ხოლო ტრანსლატორები აკავშირებენ ერთიმეორესთან რაიმე ორ დონეს. ენობრივი სტრუქტურები წარმოიდგინებიან ხეების საშუალებით. მას აქვს გამარტივებული სინტაქსი და სემანტიკა კარგად განსაზღვრულია, რომელიც ემყარება

თვისებათა ლოგიკასა [10] და თვისებათა უნიფიკაციის ფორმალიზმს [11]. CAT2-ძირითადი განსხვავება სხვა მსგავს ფორმალიზმებისაგან არის ის, რომ იგი იყენებს ლინგვისტურ-სტრუქტურების წარმოსადგენადაც ხეებს, მაშინ როდესაც თვისებათა სტრუქტურები გამოიყენება მაგალითად: LFG-ში [12], FUG-ში [13] და HPSG-ში [14]. CAT2-ის სხვა დამახასიათებელი ნიშანია დონეთა როლები, რომლებიც ახასიათებენ წარმოდგენის დონეებს. ამგვარად, KAT2-ში ჩვენ გვაქვს სხვადასხვა წარმოდგენის დონეები, რომლებიც დაკავშირებული არიან ერთიმეორესთან ტრანსლიატორებით. წარმოდგენის დონე განსაზღვრულია გენერატორით.

3.6.1 ძირითადი ცნებები. რაიმე ბუნებრივი ენის გრამატიკა CAT2-ში წარმოიდგინება როგორც წესების სიმრავლე, რომელიც დაჯგუფებულია გენერატორებად და ტრანსლიატორად. რაიმე გენერატორი არის წესების სიმრავლე, რომელიც განსაზღვრავს წარმოდგენის დონეს და რაიმე ტრანსლიატორი-არის წესების სიმრავლე, რომელიც აკავშირებს ერთიმეორესთან წარმოდგენის ორ დონეს. ერთი გენერატორი განსაზღვრავს სინტაქსურ სტრუქტურებს, რომელსაც ეწოდება სინტაქსური გენერატორი და მეორე გენერატორი განსაზღვრავს რეალურ სტრუქტურებს, რომელსაც ეწოდება რეალური გენერატორი. ასევე არსებობენ მორფოლოგიური გენერატორი და ტექსტის გენერატორი. მორფოლოგიური გენერატორი განსაზღვრავს მორფოლოგიურ სტრუქტურებს და ტექსტის გენერატორი გათვალისწინებულია მრავალ-წინადადებიანი სტრუქტურებისათვის. ამგვარად ავაგებთ რა სინტაქსურ გენერატორს და ერთს ან მეტ რეალურ გენერატორებს, რომელთაც დააკავშირებენ ტრანსლიატორები, შესაძლებელია მოცემული ენისათვის შეიქმნას ლინგვისტური სტრუქტურები კომპიუტერის საშუალებით. ასევე მეორე ენისათვის თუ გავაკეთებთ იგივეს და მათ გენერატორებს დავაკავშირებთ ტრანსლიატორებით ამით შესრულდება

ერთი ენის ლინგვისტური სტრუქტურების გადაყვანა მეორე ენის შესაბამის ლინგვისტურ სტრუქტურებში. მაგალითად, განვსაზღვროთ ინგლისური ენის გრამატიკა, რომელიც შედგება CSEN სინტაქსური გენერატორისაგან და ISEN ინტერფეისული გენერატორისაგან და ისინი დავაკავშიროთ ტრანსლიატორებით: CSEN→ISEN, რომელსაც გადაჰყავს სინტაქსური სტრუქტურები CSEN-იდან რეალურ სტრუქტურებში ISEN-ში და ISEN→CSEN, რომელიც ასრულებს შებრუნებულ მოქმედებას. ასევე განვსაზღვროთ ქართული ენის გრამატიკა CSKR და ISKAR გენერატორებით და ტრანსლიატორებით CSKAR→ISKAR და ISKAR→ CSKAR. დავუმატოთ მათ ტრანსლიატორები: ISEN→ ISKAR და ISKAR→ISEN, მაშინ შეგვიძლია ჩვენ ქართული IS(ინტერფეისული სტრუქტურები) გადავიყვანოთ ინგლისურ IS სტრუქტურებში და პირიქით. ლინგვისტურ სტრუქტურებს, რომელსაც გენერატორი ჰქმნის ეწოდებათ ობიექტები. კონკრეტულ ობიექტს აქვს ხის ფორმა, რომელშიც კვანძები შეიცავენ თვისებების სიმრავლეს. ყოველი თვისება არის წყვილი ატრიბუტი და მნიშვნელობა. ყოველი ატრიბუტი არის უნიკალური ხის კვანძში და ყოველ ატრიბუტს აქვს მნიშვნელობა – ატომური კონსტანტა ან ცვლადი ან თვისებათა სიმრავლე. ობიექტების გენერირებისათვის CAT2 იყენებს უნიფიკაციის ოპერაციას, როგორც ხეების ასაგებად, ასევე თვისებათა შესადგენად. უნიფიკაცია რაიმე ობიექტისა ხდება მარცხნიდან მარჯვნივ ქვეხეების გათვალისწინებით. უნიფიკაციასთან დაკავშირებული მნიშვნელოვანი ცნებაა შეზღუდვის დაკმაყოფილება. შეზღუდვები, რომლებიც არ კმაყოფილდება მოცემულ მომენტში, ხდება მისი გადადება მანამდე, სანამ ახალი ინფორმაციის მიღება არის შესაძლებელი. შეზღუდვათა ტიპებია: დადებითი, უარყოფითი, დიზიუნქციური, არსებობის და პირობითი შეზღუდვები.

3.6.2 თვისებები და შეზღუდვები. თვისებები გამოიყენება

იმისათვის, რომ მივუთითოთ ლინგვისტური სტრუქტურის კონკრეტული მახასიათებლები. თვისება შედგება წყვილისაგან – ატრიბუტი და მნიშვნელობა, რომლებიც ერთიმეორისაგან გამოიყოფა '=' ნიშნით. თვისებათა სიმრავლეს ათავსებენ ფიგურულ ფრჩხილებში და ერთიმეორისაგან გამოიყოფიან მძიმეებით. მაგალითად:

{cat=არს, lex=სახლები, lu=სახლი, arg={ per=3, num=vh} , wh=' – ' }

თვისების მნიშვნელობა შეიძლება იყოს კონსტანტა, ცვლადი ან თვისებათა სიმრავლე. თვისება შეიძლება იყოს რთული (მაგ. arg) ან ატომური (მაგ.cat, lu და სხვა). CAT2-ის კონსტანტებია PROLOG-ის კონსტანტები, მხოლოდ არ გამოიყენება ნამდვილი რიცხვები. თვისების მნიშვნელობა შეიძლება იყოს ცვლადი. ცვლადი შეიძლება დაუკავშირდეს კონსტანტას ან თვისებათა სიმრავლეს უნიფიკაციით და შემდეგ ხდება ინიციალიზებული ცვლადი. ასევე ცვლადი შეიძლება დაუკავშირდეს სხვა ცვლადს ასეთ შემთხვევაში გვაქვს დაკავშირებული ცვლადი. თვისებათა სიმრავლის აღწერა გამოსახავს შეზღუდვებს ობიექტის თვისებებზე. დადებითი შეზღუდვები ისეთი შეზღუდვებია, რომლებიც მოიცემა კონსტანტებით ან ცვლადებით ან შედგენილი თვისებით, რომელიც კვლავ შედგება დადებითი შეზღუდვებით. უარყოფითი შეზღუდვები შეიცავენ უარყოფებს. მაგ.{per=\=3}. დიზიუნქტური შეზღუდვები აცხადებენ, რომ ობიექტის თვისებების მნიშვნელობა შეიძლება იყოს ერთ-ერთი რამოდენიმე ალტერნატივებიდან. არსებობის შეზღუდვები მოითხოვენ, რომ მოცემული შეზღუდვა უნდა არსებობდეს ობიექტში წინააღმდეგ შემთხვევაში იგი განიცდის მარცხს. არსებობის შეზღუდვა აღინიშნება = = ნიშნით. პირობით შეზღუდვები არიან შეზღუდვები რაიმე თვისებათა სიმრავლეზე და არა რაიმე თვისების მნიშვნელობაზე. პირობით შეზღუდვას აქვს სახე

პირობა > > მიმდევრობა

სადაც როგორც პირობა ასევე მიმდევრობა არიან თვისებათა სიმრავლის აღწერები. თუ ობიექტის თვისებათა სიმრავლე აკმაყოფილებს შეზღუდვებს მოცემულს პირობაში, მაშინ იგი უნდა აკმაყოფილებდეს შეზღუდვებს მოცემულს მიმდევრობაში.

არსებობს ორი თვისების აღწერის მნიშვნელობები CAT2-ში, რომელთაც აქვთ სპეციალური დანიშნულება. ერთი არის INDEX უნიკალური მთელი რიცხვების გენერირებისათვის და მეორე არის NUM-ისათვის, რომ მნიშვნელობა იყოს რიცხვითი. ისინი შეიძლება შეგვხვდეს როგორც მნიშვნელობები ნებისმიერი თვისების აღწერაში. ორივენი წარმოადგენენ შეზღუდვებს მნიშვნელობებზე. $INDEX მნიშვნელობა გამოიყენება ანაფორულ მითითებაში ინდექსების გენერირებისათვის.

3.6.3 გენერატორები. რაიმე გენერატორი განსაზღვრავს ობიექტების სიმრავლეს წარმოდგენის რაიმე დონეზე. გენერატორები შეიცავენ ორი ტიპის წესებს: b-წესებსა და f-წესებს. b-წესები (აგების წესები) არიან კონსტრუქტორთა სიმრავლე, რომელიც განსაზღვრავს სტრუქტურულ ხეებს და f-წესები (თვისებათა წესები) მოქმედებენ თვისებებზე ხის შიგნით, ან ამატებენ ახალ თვისებებს ან ამოწმებენ უკვე არსებულ თვისებებს. როცა ჩვენ ვქმნით რაიმე ფაილს, რომელიც შეიცავს რაიმე გენერატორის განსაზღვრას, ჩვენ ვიწყებთ განსაზღვრებას კომპილატორის დირექტივით @ level:

@ level (NAME /TYPE/ATTRIBUTE).

ეს დირექტივა აძლევს გენერატორს სახელს, (NAME) განსაზღვრავს მის ტიპს (TYPE) და პრივილიგირებულ ატრიბუტებს მისთვის. (ATTRIBUTE) სახელი შეიძლება იყოს, ატომური კონსტანტა. ტიპი არის სინტაქსური

(Syntactic) ან რელაციური (relational) და ატრიბუტი არის რაიმე ატომური კონსტანტა. მაგალითად

@ level (CSKAR/syntactic/cat).

მომდევნო ინსტრუქციები ახალ @level ინსტრუქციამდე ეკუთვნიან ამ გენერატორს.

B-წესები განსაზღვრავენ კონსტრუქტორებსა და ატომებს, რომლებიც აღწერენ ცალკეულ ხეებს. ყოველ b-წესს აქვს ფორმა:

NAME=ROOT.BODY.

სადაც სახელი(NAME) არის ატომური კონსტანტა, რომელიც აღნიშნავს წესს, ძირი(ROOT) არის რაიმე თვისებათა სიმრავლის აღწერა, რომელიც აღწერს ხის ძირს და ტანი (BODY) არის უშუალო შთამომავლების თვისებათა სიმრავლეების აღწერები. ატომურიკონსტრუქტორის შემთხვევაში ტანი არის ცარიელი სია. b-წესები იწყება დირექტივით

@ rule(b), ამ დირექტივის მომდევნო ინსტრუქციები შემდეგ @ rule ან @ level დირექტივებამდე განიხილებიან როგორც b-წესები.

3.7 საუბრის წარმოდგენის თეორია[15]. საუბრის წარმოდგენის თეორია (DRT) დაკავშირებულია ბუნებრივი ენის ტექსტის სემანტიკური სტრუქტურის წარმოდგენასთან. უფრო ზუსტად, იგი ცდილობს ტექსტის შემადგენელი მიმდევრობითი წინადადებებისათვის წარმოგვიდგინოს მათი სინტაქსური, სემანტიკური და ლოგიკური სტრუქტურა და ის კავშირები, რომლებითაც ისინი დაკავშირებული არიან ერთიმეორესთან. ამასთანავე ეს წარმოდგენა უნდა იყოს განზოგადებული ე. ი. ისეთი, რომელიც გამოდგებოდა არა მარტო მოცემული ტექსტის წინადადებებისათვის არამედ აგრეთვე ბევრი სხვა გარკვეული თვალსაზრისით მსგავსი წინადადებებისათვის. DRT არის ბუნებრივი ენის სემანტიკის მეთოდი და

წარმოიშვა თავსატეხი წინადადებების (მათი სემანტიკის თვალსაზრისით) სემანტიკის დასადგენად და აგრეთვე დროის და ასპექტის გარჩევისათვის რომანულ ენებში. ამ მიდგომას საფუძვლად უდევს მოდალური თეორიული სემანტიკა, რომელიც შემოტანილია მონტეგიუს [2] მიერ და წარმატებით იყო გამოყენებული მის მიერ ინგლისური ენის ფრაგმენტისათვის. დღეისათვის იგი ცნობილია მონტეგიუს გრამატიკის სახელწოდებით. მონტეგიუს გრამატიკამ განვითარება ჰპოვა პარტიეს (Partee [16]), თომასონის (Thomasson [17]) , ფოდორის (Fodor [18]), კეენანის (Keenan [19]) და იანსენის (Iansen [20]) შრომებში. DRT არ არის წმინდა მოდელურ თეორიული სემანტიკის მეთოდი. DRT ცდილობს გაასწოროს მოდელურ თეორიული პარადიგმების ცალმხრივობა რეფერენციალური პერსპექტივის ინტერპრეტაციაზე ორიენტირებულ თვალსაზრისთან კომბინირებით. იგი არის ლინგვისტური მნიშვნელობის უფრო რთული მეთოდი, რადგანაც იგი ერთის მხრივ საგანზე ორიენტირებულია და მეორეს მხრივ იგი დაკავშირებულია ენის ინტერპრეტაციასთან. ეს თეორია განვითარებულია კამფის (Kamp [21]) და ჰეიმის (Heim [22]) მიერ. ჩვენ შევეცდებით აღვწეროთ ეს თეორია და სათანადო საუბრის წარმოდგენის სტრუქტურები (DRS) და მათი აგების წესები ქართული ენის ფრაგმენტის მაგალითზე. სანამ ამ თეორიის აღწერას შევუდგებოდეთ გავეცნოთ ზოგიერთ ძირითად ცნებებს, რომლებიც ჩვენ დაგვჭირდება DRT-ს აღწერისას. როდესაც, ჩვენ ვკითხულობთ ტექსტს და ვიგებთ მის შინაარსს ეს პროცესი შეიძლება გაგებულ იქნეს როგორც ლინგვისტური ფორმიდან შინაარსის აღების პროცესი და ამ შინაარსის ჩადება ახალ აზროვნებით ფორმაში. პირიქით, როცა ჩვენ გადმოვცემთ შინაარსს ტექსტის გენერირებისას, მაშინ ხდება შინაარსის აზროვნებითი ფორმიდან გადატანა (ჩადება) ლინგვისტურ ფორმაში.

ამგვარად, ეს პროცესები ჩვენ შეგვიძლია განვიხილოთ როგორც თარგმანი ერთი ენიდან მეორეზე (ბუნებრივი ენიდან აზროვნების ენაზე თარგმანი ან პირიქით). მეორეს მხრივ, ისმის საკითხი შინაარსის ჭეშმარიტების შესახებ ე.ი. ბუნებრივი ენის გამოსახულების (ან შესატყვისი აზროვნების ენის გამოსახულების), რომელსაც შინაარსი გააჩნია, ჭეშმარიტების შესახებ. სანამ განვმარტავდეთ თუ რას ნიშნავს ეს მოვიყვანოთ ფოდორის (Fodor[18]) არგუმენტები.

3.7.1 სინტაქსი. ცალსახა ენის მაგალითია წინადადებათა აღრიცხვის ენა U^0, სადაც ინდივიდუალური კონსტანტებია `a`, `b` და `c`. ერთადგილიანი პრედიკატებია(ანუ ერთადგილიანი ზმნები ანუ გარდაუვალი ზმნები) `p` და `q` ერთადგილიანი დამაკავშირებელია `¬` და ორადგილიანი დამაკავშირებლები `∧` და `∨`. ფორმულების სიმრავლე განსაზღვრულია რეკურსიულად, როგორც სემანტიკური სიმრავლე K ისეთი, რომ

(0) თუ არის ერთადგილიანი ზმნა და α არის რაიმე ინდივიდუალური კონსტანტა, მაშინ $\ulcorner(\alpha)\urcorner \in K$

(1) თუ ζ ერთადგილიანი დამაკავშირებელია $\phi \in K$, მაშინ $\ulcorner\zeta\phi\urcorner \in K$ და

(2) თუ ς ორადგილიანი დამაკავშირებელია და ϕ , $\Psi \in K$, მაშინ $\ulcorner\phi\ \zeta\ \Psi\urcorner \in K$. `p(a)`, `¬ p(a)` , `p(a) ∨ Q(a)` და `¬¬ p(a) ∧ ¬ Q(b)` ფორმულებია. ამ ფორმულირებაში მონაწილეობს ხუთი სინტაქსური კატეგორია: ინდივიდუალური კოტსტანტები, ერთადგილიანი ზმნები, ერთადგილიანი დამაკავშირებლები, ორადგილიანი დამაკავშირებლები და ფორმულები. გარდა უკანასკნელისა, ყველა კატეგორია განსაზღვრულია მათი ელემენტების ჩამოთვლით. ფრჩხილების სამი წყვილი არცერთ კატეგორიას არ ეკუთვნიან. ისინი შემოტანილი არიან სინკატეგორიმატიკულად

ფორმულათა სიმრავლის რეკურსიულად განსაზღვრისას. რაც იმას ნიშნავს, რომ ამ ენის სტანდარტული სემანტიკური ინტერპრეტაციის დროს მათ არავითარი დამოუკიდებელი მნიშვნელობა არა აქვთ. ფრჩხილების სემანტიკური გამოყენებაა არა მნიშვნელობათა შექმნა, არამედ ფორმულათა სინტაქსური ცალსახობის უზრუნველყოფა, ურომლისოდაც ენა იქნებოდა ომონიმური. ამის საილუსტრაციოდ განვიხილოთ L ენა, სადაც (2) შეცვლილია შემდეგი წესით: (2,) თუ ζ არის ორადგილიანი დამაკავშირებელი და ϕ,Ψ∈K მაშინ ϕζΨ∈K. L ენა არის ომონიმური, რადგან ფორმულას

`p(a)∨Q(a)∧Q(b)`აქვს ორი გარჩევა და შესაბამისად ორი განსხვავებული სინტაქსური სტრუქტურა:

(S1)

(S2)

სადაც S-ით აღნიშნულია განსახილველი ფორმულა. თუ a არის პეტრე, b არის თამარი, P არის დადის და Q არის ზის, მაშინ (S1) ნიშნავს ,რომ (პეტრე დადის) ან (პეტრე ზის და თამარი ზის), ხოლო (S2) ნიშნავს, რომ (პეტრე დადის ან პეტრე ზის) და (თამარი ზის). მრგვალი ფრჩხილები აღნიშნავენ `ან`-ისა და `და`-ს საზღვრებს. (S1) და (S2) არიან ანალიზის ხეები, რომლებიც გვიჩვენებენ თუ როგორ წარმოიქმნება წესების გამოყენებით S ფორმულა. ანალიზის ხეები გვიჩვენებენ ფორმულათა სინტაქსურ სტრუქტურებს, რომლებიც თავის მხრივ მიიღებიან რეკურსიული განსაზღვრებებით. სინტაქსური კატეგორიები, რომლებიც ფიგურირებენ ამ რეკურსიებში ერთ-ერთ მნიშვნელოვან

თვისებებზე მიუთითებენ. თუ რაიმე ფორმულები ეკუთვნიან ერთდაიმავე სინტაქსურ კატეგორიას, მაშინ ისინი ერთნაირად შეიძლება დაუკავშირდნენ სხვა გამოსახულებებს ერთიდაიგივე სორტის დაკავშირებით. ეს არის გამოყენებული ენის ფორმულების რეკურსიული განსაზღვრისას. დაკავშირების სორტები არიან ფუნქციები, რომლებიც იღებენ გამოსახულებათა მიმდევრობას და გვაძლევენ გამოსახულებებს. მაგალითის სახით განვსაზღვროთ ასეთი ფუნქციები U^0-ისთვის. ვთქვათ F^0_0 იყოს სორტი, რომელიც აკავშირებს ერთადგილიან ზმნას ინდივიდუალურ კონსტანტასთან და ღებულობს ფორმულას, F^0_1 იყოს სორტი, რომელიც იღებს ერთადგილიან დამაკავშირებელს და აკავშირებს მას რაიმე ფორმულასთან, რომ მიიღოს ახალი ფორმულა. F^0_2 იყოს სორტი, რომელიც აკავშირებს ორადგილიან დამაკავშირებელს ორადგილიან ფორმულათა მიმდევრობასთან და ღებულობს ახალ ფორმულას. ამგვარად, გვაქვს განტოლებები, რომლებიც განსაზღვრავენ ამ ფუნქციებს:

$$F^0_0(\delta,\alpha) = \ulcorner \delta(\alpha) \urcorner$$

$$F^0_1(\zeta,\varphi) = \ulcorner \zeta(\phi) \urcorner$$

ეს ფუნქციები ფაქტიურად არიან ოპერაციები, რომლებიც განსაზღვრულია ენის გამოსახულებებზე. F^0_1 ($\neg(QV`,`)b`) = \ulcorner \neg(QV(\ )b\ ) \urcorner$ ამ ფუნქციებს ეწოდებათ U^0-ის სტრუქტურული ოპერაციები და გამოიყენებიან სინტაქსური წესების გამოსაცხადებლად, მაგრამ ისინი თვით არ არიან სინტაქსური წესები. ასე მაგალითად, ასებობს ისეთი წესი, რომელიც გვეუბნება, რომ F^0_0 არის სინტაქსური ოპერაცია და გამოყენებულია ერთადგილიანი ზმნისა და ინდივიდუალური კონსტანტისაგან ფორმულის მისაღებად. იმისათვის, რომ ასეთი წესი ჩამოვაყალიბოთ, ჩვენ პირველ რიგში უნდა ჩამოვთვალოთ U^0-ის სინტაქსური კატეგორიები: ICST,IV,1C,2C და FOR. ვთქვათ ∇ იყოს სიმრავლე

{ICST,IV,1C,2C,FOR} მონტეგიუს მიხედვით, ეს სიმრავლე არის ცალსახა ენის განსაზღვრების ერთ-ერთი კომპონენტთაგანი. სინტაქსური წესი უნდა შეიცავდეს სამი სახის ინფორმაციას: სტრუქტურულ F ოპერაციას, გამოსახულებათა სინტაქსურ კატეგორიებს, რომლებიც არიან F-ის არგუმენტები და F-ის მნიშვნელობის სინტაქსურ კატეგორიას. მაგალითად U^0-ის პირველი წესი ამბობს, რომ თუ ε არის IV და α არის ICST მაშინ ε(α) არის FOR. ამგვარად, მონტეგიუს სინტაქსური წესები არიან დალაგებული სამეულები, რომლის პირველი კომპონენტი არის სტრუქტურული ოპერაცია, მეორე კომპონენტია კატეგორიული ინდექსების მიმდევრობა და მესამე ელემენტია კატეგორიული ინდექსი. ამგვარად S^0 სიმრავლის შემდეგი სამი სამეული შეადგენს U^0-ის სინტაქსური წესების სიმრავლეს:

$< F^0_0$ IV,ICST> , FOR>

$< F^0_1$,< 1C, FOR> , FOR>

$< F^0_2$,< 2C, FOR, FOR > ,FOR>

სემანტიკური მოსაზრებით მიზანშეწონილია დავაფიქსიროთ ის კატეგორია, რომელიც შეადგენს ენის წინადადებებს, რადგან წინადადებებს უნდა ჰქონდეს ჭეშმარიტული მნიშვნელობები. U^0-ის შემთხვევაში ეს არის FOR. სიმრავლეები Δ^0 და S^0 ითვალისწინებენ U^0-ის სტრუქტურულ დახასიათებას. და მთელ რიგ კატეგორიების გამოსახულებებს აქვთ თავისი სტრუქტურა, გარდა იმ გამოსახულებებისა, რომლებიც შეადგენენ რეკურსიის ბაზისს. ასეთი გამოსახულებები არიან δ კატეგორიის გამოსახულებები და ისინი ეკუთვნიან ლექსიკონს, სადაც $\delta \in \Delta$. ლექსიკონი შეიცავს სიმრავლეს X δ , სადაც $\delta \in \Delta$. U^0-სთვის ასეთი სიმრავლეებია:

X^0_{ICST}={`a`,`b`,`c`}

$X_{IV}=\{`p`,`a`\}$

$X^0_{IC}=\{` \neg `\}$

$X^0_{2C}=\{`V`,`\wedge `\}$

$X^0_{FOR}=\wedge$

$X^0_{FOR}=\wedge$ გვიჩვენებს, რომ ფორმულები არიან სინტაქსურად რთული. ვთქვათ, $C^0\delta$ იყოს U^0-ის δ კატეგორიის გამოსახულებათა სიმრავლე

$C^0 <= C^0$

C^0 შეიძლება განისაზღვროს S^0-ით, სინტაქსური წესებით და ოპერაციებით F^0_i და $X\delta^0$-ით $i\in\{0,1,2\}$ და $\delta\in\Delta^0$. ამგვარად,

C^0-ის ინდექსირებული ოჯახი არის ისეთი უმცირესი ოჯახი $C\ \Delta^0$-ით ინდექსირებული სიმრავლეებისა, რომ

(1) $X^{0\delta} \subseteq C\delta_{\in\Delta^0}$;

(2.0) $\varsigma\in C_{IV}$ და $\alpha\in C_{IC}$, $F^0_0(\varsigma,\alpha)\in C_{FOR}$;

(2.1) $\varsigma\in C_{1C}$ და $\phi\in C_{FOR}$, $F^0_1(\varsigma,\phi)\in C_{FOR}$;

(2.2) $\varsigma\in C_{2C}$ და $\phi_{1\Psi}\in C_{FOR}$, $F^{0(\varsigma,\phi,\Psi)\in}_2 C_{FOR}$

განვიხილოთ U^0-ის სწორი გამოსახულების A_0 სიმრავლე იმ გამოსახულებებთან ერთად, რომლებიც მიიღებიან U^0-ის სტრუქტურული ოპერაციების გამოყენებით. იგი შეიცავს მთელ რიგ გამოსახულებებს, რომლებიც ნაკლებად საინტერესოა სინტაქსური თვალსაზრისით, მაგრამ საინტერესოა U^0-ის ზოგადი თვისებების შესწავლის თვალსაზრისით. ეს სიმრავლე შეგვიძლია განვიხილოთ როგორც ალგებრა U^0-ის სტრუქტურული ოპერაციებით. ამ ალგებრის საშუალებით შეგვიძლია შემოვიტანოთ ისეთი მათემატიკური ცნებები როგორიცაა ჰომომორფიზმი და დავამტკიცოთ მეტათეორემები. ჩვენ შეგვიძლია განვსაზღვროთ ენა, რომელიც იგივურია U^0-თან, როგორც

ინდექსირებული ოჯახი ბ A^{0}, F^{0}_{i}, $X\delta^{0}$,S^{0}, FOR> $i \in \{0,1,2\}$, $\delta \in \Delta^{0}$. იმისათვის რომ ეს ენა იყოს ცალსახა იგი უნდა აკმაყოფილებდეს ორ პირობას: სტრუქტურული ოპერაციებით მიღებული არცერთი გამოსახულება არ უნდა იყოს U^{0}-ის ძირითადი გამოსახულება. (ე. ი. $X\delta^{0}$-ის ელემენტი) და ყოველი სწორი რთული გამოსახულება უნდა მიიღებოდეს მხოლოდ ერთი სტრუქტურული ოპერაციით და არგუმენტის მხოლოდ ერთი მნიშვნელობებით. მაგალითად. `(p(a))Λ(Q(a)) შეიძლება ჰქონდეს მხოლოდ ფორმა $F_{2}^{0}$(`Λ , `P( a)`, `Q(a)`) U^{0}-ში და არ უნდა არსებობდეს სხვა ოპერაცია და არგუმენტები, რომლებიც მოგვცემს ამ ფორმულას. ეს იძლევა იმის გარანტიას, რომ ყოველ ფრაზას ექნება მხოლოდ ერთი ანალიზის ხე.

3.7.2 პრაგმატიკა (მონტეგიუ). ენის შესწავლა (სემიოტიკა) მორისის [23] მიერ დაყოფილია სამ ნაწილად: სინტაქსი, სემანტიკა და პრაგმატიკა. სინტაქსი ეხება ლინგვისტურ გამოსახულებებს შორის დამოკიდებულებებს; სემანტიკა კი ეხება დამოკიდებულებებს გამოსახულებებსა და იმ ობიექტებს შორის, რომლებსაც ეს დამოკიდებულებები მიუთითებენ. პრაგმატიკა ეხება დამოკიდებულებებს გამოსახულებებსა, იმ ობიექტებსა, რომლებსაც ისინი მიუთითებენ, და გამომყენებლებს ან გამოსახულებათა გამოყენების კონტექსტებს შორის.

3.7.3 ენები და ინტერპრეტაციები. რაიმე პრაგმატული ენა განისაზღვრება როგორც ენა, რომელსაც აქვს შემდეგი კატეგორიის სიმბოლოები (ან ატომური გამოსახულებები):

(1) ლოგიკური კონსტანტები ¬,Λ,V,→,↔,Λ,V,=(შესამისად: `არაა შემთხვევა რომ`, `და`, `ან`, `თუ...მაშინ`,

`მაშინ და მხოლოდ მაშინ`, ყველასათვის`, `ზოგიერთისთვის`, ` იგივურია `);

(2) მრგვალი ფრჩხილები;

(3) ინდივიდუალური ცვლადები: $V_0, V_1, ..., V_m, ...$;

(4) n ადგილიანი პრედიკატები ყოველი ნატურალური რიცხვისათვის n , ერთადგილიანი E (არსებობს) პრედიკატის ჩათვლით;

(5) (n) ადგილიანი ოპერაციული სიმბოლოები ყველა ნატურალური რიცხვისათვის n;

(6) (n) ადგილიანი ოპერატორები ყოველი დადებითი მთელი n ისათვის. ყოველი პრაგმატული ენისათვის მოითხოვება, რომ იგი შეიცავდეს (1), (2) და (3) კატეგორიის სიმბოლოებს. ამგვარად, პრაგმატული ენა არის საზოგადოდ პრედიკატების, ოპერაციების და ოპერატორების სიმრავლე.

რაიმე n ადგილიანი ოპერატორი არის რაიმე სიმბოლო, რომლის შემდეგ თუ ჩვენ მოვათავსებთ n წინადადებისაგან შედგენილ სტრიქონს, მოგვცემს ახალ წინადადებას. თერმები (აღმნიშვნელი გამოსახულებები) და ფორმულები განისაზღვრებიან ჩვეულებრივად რეკურსიული წესების გამოყენებით. L-ის თერმების სიმრავლე არის უმცირესი ისეთი Γ სიმრავლე რომ

(1) ყველა ცვლადები არიან Γ-ში და

(2) Γ შეიცავს $A\zeta_1...\zeta_n$-ს სადაც A - არის n ადგილიანი ოპერაციის სიმბოლო L-ში და $\zeta_1...\zeta_n$ ეკუთვნის Γ-ს. L-ის ფორმულების სიმრავლე არის უმცირესი ისეთი Δ სიმრავლე, რომ

(1) Δ შეიცავს $P\zeta_1...\zeta_n$-ს, სადაც P არის L-ის n ადგილიანი პრედიკატი და $\zeta_1...\zeta_n$ არიან L-ის თერმები და

(2) Δ შეიცავს $\neg, (\phi \wedge \Psi), (\phi \vee \Psi), (\phi \rightarrow \Psi)$ და $(\phi \equiv \Psi$

) როცა ϕ და Ψ ეკუთვნის Δ-ს.

(3) Δ შეიცავს $\Lambda_u\phi$ და $V_u\phi$, სადაც u არის ცვლადი და ϕ არის Δ-დან და

(4) Δ შეიცავს $N\phi_1...\phi_n$-ს, როცა N არის L-ის n ადგილიანი ოპერატორი და $\phi_1,\ldots,\phi_n$ ეკუთვნის Δ -ს.

L-ის ინტერპრეტაციისას, ჩვენ უნდა გავითვალისწინოთ გამოყენების შესაძლო კონტექსტები. ამ კონტექსტებიდან უნდა ავარჩიოთ რელევანტური ასპექტების კომპლექსები და მათ ჩვენ ვუწოდებთ ინდექსებს ან კიდევ მითითების წერტილებს. მაგალითად, თუ L-ის ინდექსური თვისებებია მხოლოდ დროის ოპერატორების შეხვედრა, მაშინ მითითების წერტილები იქნებიან მომენტები, რომლებიც შესაძლებელია შეგვხვდეს გამოთქმებში როგორც დროის განსხვავებული მომენტები. თუ L შეიცავს აგრეთვე პირთა ნაცვალსახელებს, მაშინ მითითების წერტილები იქნებიან დალაგებული წყვილები, რომელთა ერთი კომპონენტი იქნება დროის მომენტი და მეორე კომპონენტი - პირი. სხვა სახის ინფორმაცია, რომელიც დაგვჭირდება L-ის ინტერპრეტაციისათვის არის L -ის თვითეული P პრედიკატის ინტენსიონალი(მნიშვნელობა). ამისათვის, უნდა განისაზღვროს მითითების ყოველ წერტილში i P-ს ექსტენსიონალი (ანუ დენოტაცია). მაგალითად, თუ მითითების წერტილები არიან დროის მომენტები და P არის ერთადგილიანი პრედიკატი `არის თეთრი`, მაშინ დროის ყოველი მომენტისათვის უნდა დავადგინოთ სიმრავლე იმ ობიექტებისა, რომლებიც შეიძლება იყვნენ თეთრი.

მესამე სახის ინფორმაცია არის L-ის ყოველი ოპერაციის ინტენსიონალი, რომელიც ისევე შეიძლება განისაზღვროს, როგორც პრედიკატის შემთხვევაში. მაგალითად, განვიხილოთ ერთადგილიანი ოპერაცია ``ვისიმე ქმარი ``. დროის ყოველი მომენტისათვის არის ფუნქცია, რომელიც განსაზღვრავს ყოველი შეუღლებული

ქალისათვის - მის ქმარს. მეოთხე სახის ინფორმაციაა L-ის ოპერატორების ინტერპრეტაცია. ამისათვის, L-ის ყოველ n ადგილიან ოპერატორთან მითითების i წერტილში ვაკავშირებთ ამ ოპერატორის ექსტენციონალს i-ს გათვალისწინებით, როგორც ამ ოპერატორის რაიმე n ადგილიან დამოკიდებულებას მითითების წერტილების სიმრავლეებს შორის. L-ის ინტერპრეტაციისათვის ჩვენ უნდა დავაზუსტოთ განსახილავი შესაძლო ობიექტების სიმრავლე. ამისათვის შემოვიღოთ განმარტება.

განმარტება. I შესაძლო ინტერპრეტაცია L პრაგმატული ენისათვის არის დალაგებული სამეული <I,U F> ისეთი , რომ

(1) I და U არიან სიმრავლეები;

(2) F არის L-ზე განსაზღვრული ფუნქცია;

(3) ყოველი A სიმბოლოსათვის L-დან F_A არის I-ზე განსაზღვრული ფუნქცია;

(4) ყოველთვის, როცა P არის L-ის n ადგილიანი პრედიკატი და i∈I, EP(i) არის n ადგილიანი დამოკიდებულება U-ზე;

(5) ყოველთვის, როცა A არის L-ის n ადგილიანი ოპერაცია და i∈I, $F_A(i)$ არის n+1 ადგილიანი დამოკიდებულება U-ზე ისეთი, რომ როცა $X_1 \ldots X_n$ ეკუთვნიან U-ს , არსებობს ზუსტად ერთი ობიექტი Y∈ U ისეთი, რომ $< X_1, \ldots, X_n, Y \in FA(i)$, და

(6) ყოველთვის, როცა N არის L-ის n-ადგილიანი ოპერატორი და i∈I, $F_n(i)$ არის n ადგილიანი დამოკიდებულება I-ის ყველა ქვესიმრავლეების სიმრავლეზე.

3.7.4 მნიშვნელობა და მითითება. იმისათვის, რომ დავახასიათოთ ჭეშმარიტება, ლოგიკური სისწორე (ვარგისიანობა, დასაბუთება) და ლოგიკური შედეგი შემოვიტანოთ

ინტენსიონალისა და ექსტენსიონალის ცნებები. თერმის ექსტენსიონალი მოცემულ მითითების წერტილში არის ისეთი H ფუნქცია, რომელიც ნიშნავს მისი ცვლადების მნიშვნელობათა ყოველ შესაძლო სისტემისათვის ამ თერმის მნიშვნელობას მითითების მოცემულ წერტილში. თერმები შეიძლება შეიცავდნენ ცვლადების სხვადასხვა რაოდენობას, რომელიც არაა ზემოდან შემოსაზღვრული, ამიტომ მოსახერხებელია მივიღოთ, რომ H გამოიყენება შესაძლებელ ობიექტთა უსასრულო მიმდევრობებზე, ე. ი. H- აქვს არგუმენტების უსასრულო რაოდენობა. მიმდევრობის წევრის ნომერი ამყარებს თანადგომას მიმდევრობის წევრსა და თერმის შესატყვის ცვლადს შორის. ე.ი. მიმდევრობის k-ური წევრი არის თერმის k-ური ცვლადის მნიშვნელობა. განსაზღვრება II. დავუშვათ C არის L პრაგმატული ენის რაიმე დასაშვები ინტერპრეტაცია

C=<I ,U,F> და i∈I . მაშინ $Ext_{i, C}(\zeta)$ ან ζ-ს ექსტენსიონალი i ში (C-თვის), სადაც ∑ არის L-ის რაიმე თერმები, განისაზღვრება შემდეგი რეკურსიული განმარტებით:

(1) $Ext_{i,C}(V_n)$ არის ისეთი ფუნქცია H განსაზღვრული C-ს შესაძლო ობიექტების ყველა უსასრულო მიმდევრობათა სიმრავლეზე, რომ თუ X არის რაიმე ისეთი მიმდევრობა, მაშინ $H(X)=X_n$.

(2) თუ A არის L-ის n-ადგილიანი ოპერაცია და $\zeta_1,\ldots,\zeta_n$ არიან L-ის თერმები, მაშინ $Ext_{i,c}(A\zeta_1,\ldots,\zeta_n)$ არის ისეთი H ფუნქცია, რომლის განსაზღვრის არეა C-ს შესაძლო ობიექტების ყველა უსასრულო მიმდევრობათა სიმრავლე და X არის რაიმე ასეთი მიმდევრობა, მაშინ H(X) არის უნიკალური ობიექტი y, რომლისთვისაც

$<Ext_{i,c}(\zeta_1)X,\ldots,Ext_{i,c}(\zeta_n)\ X,Y>$ არის $F_A(i)$-ის წევრი.

განსაზღვრება III. დავუშვათ C არის L პრაგმატული

ენის რაიმე დასაშვები ინტერპრეტაცია (I,U,F) და E არის L-ის თერმი. მაშინ $Int_C(\zeta)$ არის H ფუნქცია განსაზღვრის არით I ისე, რომ ყოველი $i \in I$ -სთვის

$H(i) = Ext_{i,c}(\zeta)$.

რადგანაც ფორმულის ექსტენსიონალი საზოგადოდ დამოკიდებულია ფორმულის ნაწილების ინტენსიონალზე, ამიტომ ფორმულის ექსტენსიონალის განსასაზღვრად პირველად უნდა განისაზღვროს ფორმულის ინტენსიონალი და შემდეგ მარტივი რეკურსიით შეიძლება ფორმულის ექსტენსიონალის განსაზღვრა.

განსაზღვრება IV. დავუშვათ C არის L პრაგმატული ენის დასაშვები ინტერპრეტაცია <I,U,F> , მაშინ Int_C, სადაც ф არის L-ის რაიმე ფორმულა, განისაზღვრება შემდეგნაირად:

(1) თუ ζ და η არიან L-ის თერმები, მაშინ $Int_c(\zeta = \eta)$ არის ისეთი H ფუნქცია განსაზღვრის არით I, რომ ყოველი $i \in I$-სთვის H(i) არის U-ს წევრების X უსასრულო მიმდევრობათა სიმრავლე, რომლისთვისაც $Ext_{i,c}(\zeta)X$ იგივეა რაც $Ext_{i,c}(\eta)(X)$.

(2) თუ P არის L-ის n ადგილიანი პრედიკატი და $\zeta_1,\ldots,\zeta_n$ არიან L-ის თერმები, მაშინ $Int_{i,c}P(\zeta_1 \quad \zeta_n)$ არის ისეთი ფუნქცია H არით I, რომ ყოველი $i \in I$ -სთვის, H(i) არის U-ს წევრების X – უსასრულო მიმდევრობათა სიმრავლე, რომლისთვისაც $< Ext_{i,c}\zeta_1(\zeta_n)X,\ldots,Ext_{i,c}(\zeta_n)X>$ არის $F_p(i)$-ს წევრი.

(3). თუ ф არის L-ის ფორმულა, მაშინ $Int_c(t\phi)$ არის ისეთი ფუნქცია H არით I, რომ ყოველი $i \in$ I-სთვის, H(i) არის U-ს წევრების X უსასრულო მიმდევრობათა სიმრავლე, რომლისთვისაც X არის $Int_c(\phi)(i)$-ში და ანალოგიურად განიმარტება სხვა დამაკავშირებლე-ბისთვისაც.

(4) თუ ф არის L-ის ფორმულა, მაშინ $Int_c(\forall V_n \phi)$ არის

ისეთი ფუნქცია H არით I, რომ ყოველი i∈I -სთვის H(i) არის U-წევრების X უსასრულო მიმდევრობების სიმრავლე, რომლისთვისაც არსებობს U-ში ისეთი Y, რომ უსასრულო მიმდევრობა $<X_1,\ldots,X_n,Y,X_{n+1},\ldots>$ არის $Int_c(\phi)(i)$-ში. ანალოგიურად განიმარტება $\Lambda V_{n\phi}$ -სთვის.

(5) თუ N არის n ადგილიანი ოპერატორი L-ში და $\phi_1, \ldots, \phi_n$ არიან L-ის ფორმულები, მაშინ $Int_c(N\phi_1,\ldots, \phi_n)$ არის ისეთი ფუნქცია H არით I, რომ ყოველი L∈I-თვის, H(i) არის U-ს წევრების X უსასრულო მიმდევრობათა სიმრავლე ისეთი, რომ $< J_1\hat{}\&\&\&\hat{}J_n>$ არის $F_N(i)$-ში , სადაც ყოველი $k<n$-სთვის, J_k - არის J წევრების სიმრავლე, რომლისთვისაც X არის $Int_c(\phi_k)(j)$-ს წევრი. განსაზღვრება V. დავუშვათ, რომ C არის L-ის დასაშვები ინტერპრეტაცია, i არის C-ს მითითების წერტილი და ϕ არის L-ის ფორმულა, მაშინ $Ext_{i,c}(\phi)$ არის $Int_c(\phi)(i)$.

განსაზღვრება VI. თუ C არის L-ის დასაშვები ინტერპრეტაცია, i არის C-ს მითითების წერტილი და ϕ არის L-ის წინადადება, მაშინ ϕ არის ჭეშმარიტი i-ში მხოლოდ და მხოლოდ მაშინ, როცა $Ext_{i,c}(\phi)$ არის C-ს შესაძლო ობიექტების ყველა შესაძლო მიმდევრობათა სიმრავლე.

განსაზღვრება VII. ჩვენ ვიტყვით, რომ ϕ არის რაიმე Γ სიმრავლის ლოგიკური შედეგი მაშინ და მხოლოდ მაშინ როცა არსებობს L პრაგმატული ენა ისეთი რომ ϕ და Γ-ს ყველა წევრები არიან L-ის წინადადებები და L-ის ყოველი დასაშვები ინტერპრეტაციისათვის C და C-ს ყოველი მითითების წერტილისათვის i, თუ Γ-ს ყველა წევრები არიან ჭეშმარიტი i-ში, მაშინ ϕ არის ჭეშმარიტი i-ში; და ϕ არის ლოგიკურად სწორი მაშინ და მხოლოდ მაშინ თუ ϕ არის

ცარიელი სიმრავლის ლოგიკური შედეგი (ე. ი. მხოლოდ და მხოლოდ მაშინ თუ არსებობს L პრაგმატული ენა, რომელშიც ϕ არის წინადადება და ისეთი, რომ ϕ არის ჭეშმარიტი i-ში ყოველთვის როცა C არის L-ის დასაშვები ინტერპრეტაცია და i არის C-ს მითითების წერტილი).

3.7.5 სპეციალიზაციები. განვიხილოთ სპეციალური დისციპლინები, რომლებიც შედიან პრაგმატიკაში, როგორიცაა დროის ლოგიკა, მოდალური ლოგიკა და სხვები. ისინი წარმოადგენენ ინტერპრეტაციათა სპეციალურ კლასებს.

3.8 ჩვეულებრივი დროის ლოგიკა. განვიხილოთ L პრაგმატული ენა, რომელსაც აქვს მხოლოდ ერთადგილიანი ოპერატორები P და F. ვთქვათ $K_I(L)$ იყოს < I ,U,C> შესაძლო ინტერპრეტაციების კლასი L-ისათვის ისეთი , რომ

(1a) I არის ნამდვილი რიცხვების სიმრავლე;

(1b) ყოველი i∈I -სთვის, $G_P(i)$ არის ერთეული წყვილების <J> სიმრავლე ისეთი რომ J⊆I და არსებობს J∈J ისეთი, რომ J< i; და

(1c) ყოველი L∈I-სთვის $G_F(i)$ არის ერთადგილიანი < j > მიმდევრობების სიმრავლე Δ ისეთი, რომ J⊆ I და არსებობს J⊆ J ისეთი რომ i<J. აქ ჩვენ ვუყურებთ ნამდვილ რიცხვებს როგორც დროის მომენტებს. თუ (1a) - (1c) ადგილი აქვს, ϕ არის L-ის წინადადება და i∈I. ადვილია ვაჩვენოთ, რომ Fϕ არის ჭეშმარიტი i-ში < I , U,G>-ს მიხედვით მაშინ და მხოლოდ მაშინ, როცა არსებობს J i-ში ისეთი, რომ J< i და ϕ არის ჭეშმარიტი J-ში < I ,U,G > მიხედვით და Fϕ არის ჭეშმარიტი i-ში <I,U,G> -ს მიხედვით მაშინ და მხოლოდ მაშინ, როცა არსებობს J∈ I ისეთი რომ J > i და ϕ არის ჭეშმარიტი j-ში <I,U,G>-ს მიხედვით. ამგვარად, P და F

ოპერატორები გამოსახავენ შესაბამისად წარსულ და მომავალ დროს და I სიმრავლე არის უწყვეტი სიმრავლე. თუ ჩვენ მოვითხოვთ რომ I სიმრავლე იყოს დალაგებული, მაშინ მივიღებთ განზოგადოებულ დროის ლოგიკას. ბუნებრივ ენებში პირისა და ჩვენებითი ნაცვალსახელების შესასწავლად ჩვენ შეგვიძლია მოვითხოვოთ რომ L არ შეიცავდეს არცერთ ოპერატორს და ჰქონდეს განსაკუთრებული ნულადგილიანი ოპერაციული სიმბოლო c, და შესაძლო ინტერპრეტაციებზე თუ დავადებთ გარკვეულ მოთხოვნებს, ასეთი ლოგიკური ენა გამოიყენება პირისა და ჩვენებითი ნაცვალსახელების შესასწავლად. ასევე, გარკვეული L ენის განხილვით და შესაძლო ინტერპრეტაციებზე სხვა შეზღუდვების დადებით მიიღებიან სტანდარტული და განზოგადოებული მოდალური ლოგიკები, სპეციალური დეონტიური ლოგიკა და სხვები.

3.9 ბუნებრივი ენის კომპიუტერული დამუშავება და პროლოგი. ბუნებრივი ენის კომპიუტერული დამუშავება პროლოგის საშუალებით შედგება შემდეგი ნაწილებისაგან. პირველ ნაწილში ხდება ბუნებრივი ენის გრამატიკების წარმოდგენა პროლოგ-პროგრამის სახით. ეს წარმოდგენა ბუნებრივად ხდება, რადგან გრამატიკის წესები შეიძლება პირდაპირ წარმოდგენილ იქნეს პროლოგის წესებისა და ფაქტების სახით და შემდეგ ანალიზი და სინთეზი შეიძლება ეფექტურად შესრულდეს პროლოგის გამოთვლის მექანიზმით. მეორე ანალიზის შედეგი შეიძლება წარმოდგენილ იქნეს სიური სტრუქტურის საშუალებით. შემდეგი ნაწილი არის ბუნებრივი ენის სემანტიკური წარმოდგენა ლოგიკაში. ეს წარმოდგენაც არის ბუნებრივი, რადგანაც პროლოგს უშუალო კავშირი აქვს ლოგიკასთან, ლოგიკური აქსიომები შეიძლება უშუალოდ წარმოდგენილ იქნეს პროლოგის წესების საშუალებით და პროლოგის გამოყვანის მექანიზმი უშუალოდ აღებულია ლოგიკიდან და ამიტომ კონკრეტული ლოგიკური გამოყვანის მექანიზმი (თუ იგი არ ემთხვევა პროლოგის გამოყვანის მექანიზმს) შეიძლება ტრანსლი-

რებულ იქნეს ადვილად პროლოგის გამოყვანის მექანიზმში ისევ სპეციალური პროლოგ- პროგრამის საშუალებით. ეს იდეა იყო სწორედ ჩადებული პროლოგში მათი შემქმნელების მიერ (კოლმერაუერი [24]). ჩვენი ძირითადი მიზანია განვიხილოთ ბუნებრივი ენის ტექსტის სემანტიკის წარმოდგენის ლოგიკური ფორმის მქონე ენა. განვიხილავთ LFL ენას (ლოგიკურ ფორმიანი ენა[25]). ეს ენა აღწერილი იქნება ქართული ენის წინადადებების ამ ენაზე წარმოდგენის მოცემით. ბუნებრივი ენის კომპიუტერული დამუშავება ამ ენის გამოყენებით ხდება შემდეგნაირად: პირველად იწერება პროლოგ-პროგრამა ბუნებრივი ენის წინადადებებისათვის სინტაქსური წარმოდგენის მისაღებად, ე.ი. ამ პროგრამისათვის შესასვლელია ბუნებრივი ენის წინადადება და გამოსასვლელია ამ წინადადების სინტაქსური წარმოდგენა. შემდეგ, ეს წარმოდგენა გადაიყვანება კვლავ ახალი პროლოგ-პროგრამით წინადადების სემანტიკური წარმოდგენის ენაზე. ე. ი. ახალი პროგრამის შესასვლელია სინტაქსური წარმოდგენა, ხოლო გამოსასვლელია ამ წინადადების სემანტიკური წარმოდგენა გამოსახული ლოგიკური ენის საშუალებით. შემდეგ, ახალი პროლოგ პროგრამით ხდება მიღებული წარმოდგენის გადაყვანა პროლოგ პროგრამაში და მისი შესრულება იძლევა საბოლოო შედეგს, რომელიც გათვალისწინებული იყო თავდაპირველი წინადადების კომპიუტერული დამუშავებით.

3.10 LFL ლოგიკური ენა. LFL-ს ჩვენ გამოვიყენებთ როგორც მნიშვნელობის წარმოდგენის ენას ბუნებრივი ენისათვის. წინადადებათა მნიშვნელობები არიან ლოგიკური ფორმები და ისინი არიან გამოსახულებები LFL-ში. მთავარი პრედიკატი LFL-ში არის სიტყვის აზრები ბუნებრივ ენაში. სიტყვის აზრები შეესატყვისებიან სიტყვათა განსხვავებულ მნიშვნელობებს, რომლებიც მოცემულია სტანდარტულ ლექსიკონში. მაგალითის სახით განვიხილოთ ლოგიკური ფორმა წინადადებისათვის

პეტრეს ჰგონია, რომ ყოველ მამაკაცს უყვარს ხათუნა (1)

ეს წინადადება შეიძლება ჩაიწეროს ასე:

ჰგონია1(პეტრე, ყოველი(მამაკაცი1(X), უყვარს3(X, ხათუნა))) (2)

სადაც ჰგონია1 არის ერთ-ერთი აზრი ზმნისა ` გონება``, მამაკაცი1 არის არსებითი სახელის ``მამაკაცი`` ერთ-ერთი აზრი და ა.შ. გარდა ლექსიკური პრედიკატებისა LFL-ში არის არალექსიკური პრედიკატებიც, როგორიცაა ზმნის დროს მითითება ან კიდევ არსებითი სახელის რიცხვის მითითება და სხვა. მაგალითად,

პეტრემ შეიყვარა ხათუნა (3)

შეიყვარა არის შეყვარება ზმნის წარსული დრო და იგი გამოისახება სპეციალური პრედიკატით.

წარსული- დრო (შეყვარება1(პეტრე, ხათუნა)) (4)

შევნიშნოთ, რომ მე-(2) წარმოდგენაში ყოველი არაა პრედიკატი, იგი არის ქვანტორი ``ყოველი``, რომლის განსაზღვრის არეა ფრჩხილებში მოთავსებული ფორმა და ამ ქვანტორით დაბმული ცვლადია X. განვიხილოთ სხვა მაგალითი:

უყვარს პეტრეს ხათუნა? (5)

იგი წარმოიდგინება ასე:

პასუხი(true, უყვარს(პეტრე, ხათუნა))

ამ პრედიკატის შესრულების შედეგია true თუ ბაზაში არსებობს ფორმა უყვარს (პეტრე, ხათუნა), წინააღმდეგ შემთხვევაში - false, სადაც true და false ლოგიკური კონსტანტებია. LFL-ის ყოველ პრედიკატს აქვს არგუმენტების ფიქსირებული რაოდენობა. არგუმენტებად შეიძლება იყვნენ ცვლადები, კონსტანტები ან რაიმე ლოგიკური ფორმები. აქედან გამომდინარე ლოგიკური თორმა ანუ LFL-ის გამოსახულება შეიძლება განისაზღვროს ასე:

1. თუ P არის LFL-ის რაიმე პრედიკატი (სიტყვის რაიმე აზრი ან რაიმე არალექსიკური პრედიკატი), რომელსაც აქვს

n არგუმენტი X1,...,Xn, სადაც თვითეული მათგანი არის რაიმე ცვლადი ან რაიმე კონსტანტა ან რაიმე ლოგიკური ფორმა (როგორც ეს მოითხოვება p-ს ამ არგუმენტისათვის), მაშინ p(X1,...,Xn) არის ლოგიკური ფორმა.

2. თუ P და Q ლოგიკური ფორმებია, მაშინ P&Q არის ლოგიკური ფორმა.

3. თუ P არის ლოგიკური ფორმა და E არის რაიმე ცვლადი, მაშინ P:E (იკითხება P ინდექსით E) არის ლოგიკური ფორმა.

`` ოპერატორს ეწოდება ინდექსის ოპერატორი. როდესაც კონტექსტი იგნორირებულია ლოგიკური ფორმა P:E შეიძლება წარმოიდგინოს წინადადებით

P P ← P ან რაც იგივეა P. სანამ სიტყვათა კლასების აღწერაზე გადავიდოდეთ განვიხილოთ სინტაქსის ზოგიერთი საკითხი, რადგან ლოგიკური ფორმა მჭიდროდაა დაკავშირებული სინტაქსთან.

3.10.1 ზმნები. წინადადების ძირითადი წევრია ზმნა. რადგან იგულისხმება რომ წინადადების ყოველ შემადგენელ ნაწილს აქვს მთავარი სიტყვა-წინადადებისათვის ასეთ მთავარ სიტყვას წარმოადგენს ზმნა. რაიმე წინადადებაში მთავარი ზმნა გარშემორტყმულია მოდიფიკატორებით როგორიცაა დამატებები და დამხმარე მოდიფიკატორები. მაგალითად, სუბიექტი და ობიექტი არიან დამატებები, ხოლო ზმნისზედა არის დამხმარე მოდიფიკატორი. ლოგიკურ ფორმაში მთავარი ზმნა წარმოადგენს პრედიკატს, რომლის არგუმენტებია მისი დამატებები. ამგვარად სინტაქსური ცნება-დამატება მჭიდროდაა დაკავშირებული სემანტიკურ ცნებასთან-არგუმენტი. მაგალითად:

პეტრე მიდის. მისვლა1(პეტრე).

პეტრე მიაცილებს თინას. მიაცილებს1(პეტრე, თინა).

პეტრე ჩუქნის ყვავილს თინას. ჩუქება1(პეტრე, ყვავილი, თინა).

შევნიშნოთ, რომ სიტყვათა ბრუნვის ნიშნები და ზმნის

ფორმები არაა ცხადად მითითებული ლოგიკურ ფორმაში, რადგან სიტყვის ფუნქციას მკაცრად განსაზღვრავს არგუმენტის ადგილი ფორმაში, ხოლო ზმნის ფორმა ჩადებულია პრედიკატის მნიშვნელობაში. სიტყვის აზრი არის ინტენსიონალური თუ ამ სიტყვის ერთი არგუმენტი მაინც არის ლოგიკური ფორმა. მაგალითად

პეტრეს სჯერა, რომ თინა მოვა.

დაჯერება1(პეტრე, მოსვლა2(თინა)).

ენებითი გვარის კონსტრუქციები შეიძლება იქნეს წარმოდგენილი შემდეგნაირად:

თინა არის მიცილებული პეტრეს მიერ.

პასივი(პეტრე, მიცილება(პეტრე, თინა)).

აქ პასივი(X,P) არის არალექსიკური პრედიკატი, სადაც P აღწერს თუ რა გადახდა პეტრეს. ზოგ შემთხვევაში შესაძლებელია პასივის იგნორირება და მაშინ გვექნება:

პასივი (X,P)← P

ზოგიერთ ზმნებს შეიძლება არ ჰქონდეს არგუმენტი. მაგ. თოვს. ასეთ შემთხვევაში გვაქვს უარგუმენტო პრედიკატი თოვს.

3.10.2 არსებითი სახელები. უმეტესობა არსებითი სახელებისა არიან ერთარგუმენტიანი პრედიკატები. მაგ. კაცი, რომელიც წარმოიდგინება კაცი(X), მაგრამ არიან არსებითი სახელები, რომლებიც აღნიშნავენ დამოკიდებულებას და ასეთ შემთხვევაში მათ შეესატყვისება ორარგუმენტიანი პრედიკატები. მაგ. მამა პრედიკატის წარმოდგენაა მამა(X,Y), რომელიც აღნიშნავს რომ X არის Y-ის მამა. საკუთარი სახელები ხელს უწყობენ ცვლადის დაკავშირებას შესატყვის კონსტანტასთან. სხვა არსებით სახელთა ჯგუფებს აქვთ რაიმე დაკავშირებული ქვანტიფიკატორები, რომლებიც მოდიან წინადადების დარჩენილი ნაწილიდან. არსებითი სახელთა ჯგუფებისათვის ქვანტიფიკატორები ძირითადად მოდის ზედსართავი სახელებისაგან და განმსაზღვრელებისაგან.

3.10.3 განმსაზღვრელი სახელები. არსებითი სახელებისათვის მოდიფიკატორების უმთავრესი ტიპები არიან განმსაზღვრელი სახელები, ზედსართავი სახელები და სხვა არსებითი სახელები. განმსაზღვრელ სახელთა რაოდენობა ფიქსირებულია. სემანტიკურად განმსაზღვრელები არიან ქვანტიფიკატორები. მაგალითებია: ზოგიერთი, ყოველი, მრავალი, ყველა, არცერთი, ცოტა, მისი, მათი, ეს, ის, ესენი, ისინი, რომელი, ვისი, რისი, ერთერთი და სხვა. მათ როგორც პრედიკატებს უმეტესად აქვთ არგუმენტი, რომლებიც არიან ლოგიკური ფორმები. პირველ არგუმენტს ეწოდება განმსაზღვრელის ბაზა, ხოლო მეორეს ფოკუსი. მათი ზოგადი სახეა:

განმსაზღვრელი(ბაზა,-ფოკუსი).

წყვილს (-ბაზა,-ფოკუსი) ეწოდება განმსაზღვრელის საზღვრები. მაგალითები:

ყოველ ადამიანს მოსწონს პეტრე.

ყოველი(ადამიანი (X), მოსწონს(X, პეტრე)).

ყოველ მამაკაცს უყვარს რომელიმე ქალი.

ყოველი(მამაკაცი(X), რომელიმე(ქალი(Y), უყვარს(X,Y))).

ყოველი მამაკაცი, რომელსაც უყვარს თინა, აძლევს მას საჩუქარს.

ყოველი(მამაკაცი(X)&საჩუქარი(Y)&,უყვარს(X,თინა), აძლევს(X,Y,თინა)).

სტანდარტულ უნივერსალურ ქვანტიფიკატორს საზოგადოდ აქვს სახე: (ყველა(X,P) რაც ნიშნავს, რომ ყოველი X-ისათვის ადგილი აქვს P-ს. LFL-ში ქვანტიფიკატორები განსხვავდებიან სტანდარტული განსაზღვრისაგან ორი რამით: (1) ქვანტიფიკატორის საზღვრები გახლეჩილია ორ ნაწილად - ბაზა და ფოკუსი და ჩვენ ვწერთ ყველა (P,Q) ნაცვლად ყველა (X,P→Q). ქვანტიფიკაცია შეიძლება შესრულდეს რამოდენიმე ცვლადზე ერთდროულად და ქვანტიფიკაცია ხდება ბაზის ყველა თავისუფალი ცვლადის მიხედვით. ეს კეთდება იმიტომ, რომ ბუნებრივი ენის

ქვანტიფიკატორების გამოყენება განსხვავდება ლოგიკის სტანდარტული ქვანტიფიკატორისაგან. მაგალითად: ბევრ მამაკაცს მოსწონს მონრო.

ბევრი(მამაკაცი(X), მოსწონს(X, მონრო))

იმისათვის, რომ განვსაზღვროთ ``ბევრი`` უნდა ვიცოდეთ მამაკაცების რაოდენობა და მამაკაცების რაოდენობა, რომელთაც მოსწონს მონრო. ჩვენ უნდა განვსაზღვროთ card(P,N), სადაც N არის P-ების რაოდენობა

card(P,N)← set-of(P,D,S) & length(S,N).

set-of(X,Y,Z) არის ჩაშენებული პრედიკატი, სადაც Z არის იმ X-ების სია, რომლისთვისაც ადგილი აქვს Y-ს. length(S,N)-ში N გვაძლევს S სიაში ელემენტების რაოდენობას. ამის შემდეგ ბევრი(-ბაზა, -ფოკუსი) შეიძლება განისაზღვროს ასე:

ბევრი(-ბაზა, -ფოკუსი)← card(ბაზა, A)& card(-ფოკუსი &-ბაზა, A1) & მეტი(A1,A).

აქ ქვანტორის საზღვრის ბაზად და ფოკუსად დაშლამ საშუალება მოგვცა ეფექტურად გაგვესაზღვრა IBM პროლოგზე ქვანტორი ბევრი. ბევრის ასეთ გაგებას ეწოდება ``ბევრი``-ს გაგება ფარდობითი აზრით, რაც ჩვენი მაგალითის შემთხვევაში ნიშნავს, რომ მონრო უფრო მეტ მამაკაცს მოსწონს, ვიდრე სხვა მამაკაცებს მოსწონს რომელიმე სხვა ქალი.

3.10.4 ნაცვალსახელები. ბევრი ნაცვალსახელი იქცევა ისევე როგორც განმსაზღვრელნი და ზოგიერთი მათგანი გრაფიკულადაც ემთხვევა განმსაზღვრელებს. ფორმალურად იმის დადგენა თუ კონკრეტული ნაცვალსახელი რომელ სახელს მიუთითებს არის ძნელი საკითხი და იგი საჭიროებს სპეციალურ შესწავლას და ამ მიმართულებით მრავალი გამოკვლევები არსებობენ, ხოლო ქართული ენისათვის ასეთი გამოკვლევები არაა ჩატარებული. ნაცვალსახელები ლოგიკურ ფორმაში შეიძლება მიუთითებდნენ ცვლადებს,

კონსტანტებს ან ლოგიკურ ფორმებს. მაგალითად.

პეტრეს ჰყავს ძაღლი. მას უყვარს იგი.

ძაღლი(X) & ჰყავს(პეტრე, X) & უყვარს(პეტრე, X).

ყოველ ადამიანს უყვარს თავისი თავი.

ყოველი(ადამიანი(X), უყვარს(X,X))

3.10.5 ზმნისზედები. ზმნისზედები ჰქმნიან სიტყვათა ღია კლასს, რომლებიც აზუსტებენ ზმნებს, ზედსარვავ სახელებს და სხვა ზმნისზედებს. ისინი არიან ინტენსიონალურები. ზოგიერთებს აქვთ ერთი არგუმენტი და აზუსტებენ არგუმენტის ლოგიკური ფორმის დროს. მაგალითად,

გუშინ პეტრემ იყიდა ვაშლი

გუშინ(ex(ვაშლი(x), ყიდულობს(პეტრე,x)))

ფოკუსის ფუნქცია კარგად ჩანს შემდეგ მაგალითებში:

პეტრე ყოველთვის ყიდულობს წიგნებს პავლესთან.

ყოველთვის(წიგნი(x) & ყიდულობს(პეტრე,x)):E, თან(პავლე,E)).

პეტრე ყოველთვის ყიდულობს წიგნებს პავლესთან.

ყოველთვის(თან(პავლე, ყიდულობს(პეტრე, x)), წიგნი(x)).

ამ მაგალითებიდან ჩანს რომ ერთიდაიგივე სიტყვებისაგან შედგენილი წინადადებები, რომლებიც განსხვავდებიან მხოლოდ ფოკუსით აქვთ განსხვავებული ლოგიკური ფორმები. აღმატებითი ხარისხებიანი ზმნისზედებისათვის ლოგიკური ფორმები მიიღებიან ანალოგიურად იმისა, რაც იყო გაკეთებული ``ბევრი ``-სთვის.

3.10.6 ზედსართავი სახელები. ზედსართავები ჰქმნიან ღია კლასს და მოდიფიკაციას უკეთებენ არსებით სახელებს. შეიძლება გამოვიდნენ დამატების როლში ზოგიერთი ზმნებისათვის როგორიცაა ``ყოფნა`` და ``ჩანს``. ზედსართავიდან მიიღებიან აგრეთვე ზმნისზედები. უმარტივესი ზედსართავები სემანტიკურად არიან ექსტენსიონალურები. მაგალითად,

პეტრე ყიდულობს წითელ ვაშლს.

ex(ვაშლი(x)& წითელი(x), ყიდულობს(პეტრე,x)).

ინტენსიონალურ ზედსართავ სახელებს აქვთ ერთი არგუმენტი, რომელიც მოდის ძირითადი არსებითი სახელიდან ან სხვა მოდიფიკატორიდან. მაგალითად,

პეტრე იცნობს სახელგანთქმულ ფეხბურთელს.

ex(სახელგანთქმული(ფეხბურთელი(x)), იცნობს(პეტრე, x))

პეტრე არის პავლეს ერთადერთი შვილი

ერთადერთი(შვილი(x, პავლე), x=პეტრე)

3.10.7 თანდებულები. თანდებულები ჰქმნიან სიტყვათა ჩაკეტილ კლასს და შემოჰყავთ თანდებულიანი ფრაზები. ესენი არიან არსებით სახელიანი ფრაზები დაბოლოებულნი თანდებულებით. თანდებულთა აზრი არის ორადგილიანი პრედიკატი. პირველი არგუმენტი არის ის არსებით სახელიანი ფრაზა, რომელსაც ახლავს თანდებული და მეორე არგუმენტი – ფრაზა, რომელიც დაზუსტებულია ამ თანდებულიანი ფრაზით. მაგალითები:

პეტრე მიდის სახლში

ში-ადგილი(სახლი(x), მიდის(პეტრე))

პეტრე მიდის თბილისში

ში-ადგილი(პეტრე, მიდის(თბილისი))

3.10.8 კავშირები. კავშირები ჰქმნიან სიტყვათა ჩაკეტილ კლასს. კავშირები არიან ორი სახის: მაკოორდინებელი და დაქვემდებარებული კავშირები. კავშირები არგუმენტებად იღებენ ორ ლოგიკურ ფორმას. მაგალითები:

პეტრე აღებს კარებს და შედის ოთახში.

და(კარები(x) & აღებს(პეტრე,x), ში-ადგილი(ოთახი(Y), შედის(პეტრე))) ამ შემთხვევაში ``და`` არაა იგივე რაც კონიუნქცია, რადგან იგი გამოსახავს აგრეთვე მოქმედებათა თანმიმდევრობას. მაქვემდებარებელი კავშირების მაგალითებია:

როცა ძაღლი ავია იგი ყეფს.

როცა(ძაღლი(X) & ავი(X), ყეფს(X))

თუ A> B მაშინ B< A

თუ-მაშინ(მეტი(A , B) , ნაკლები(B , A))

3.10.9 არალექსიკური პრედიკატები LFL-ში. ზოგიერთი არალექსიკური პრედიკატები დაკავშირებულია სიტყვების ფლექსიურ ცვლილებებთან, როგორიცაა გრამატიკული რიცხვი და დრო. მაგალითად, past(p) ნიშნავს, რომ p ლოგიკური ფორმა განხილულია წარსულ დროში. არალექსიკურ პრედიკატებს მიეკუთვნება აგრეთვე პრედიკატი ``კი-არა`` , რომელიც დასმულ კითხვაზე იძლევა პასუხს კი ან არა. მაგალითად,

ნახა პეტრემ ნანა გუშინ?

კი-არა(გუშინ(ნახულობს(პეტრე, X), X=ნანა)).

მრავალი ლექსიკური პრედიკატი წარმოიშვება ქართული ზმნების მორფოლოგიური კატეგორიების დასადგენად.

3.10.10 მაინდექსირებელი ოპერატორი. ბუნებრივი ენის წინადადებები აღწერენ სიტუაციებს, რომლებიც დაკავშირებულია დროსთან და ადგილთან. ე. ი. მათ მიერ აღწერილი მოქმედებები მიმდინარეობენ გარკვეულ დროში ან ადგილზე ან სხვა კონკრეტულ სიტუაციაში. მაგალითად,

პეტრემ ირბინა უსწრაფესად გუშინ.

უსწრაფესად(დარბის(პეტრე) :E გუშინ(E)).

აქ E აღნიშნავს დროის მომენტს გუშინ და ჩანაწერი ამბობს რომ პეტრემ გაირბინა უსწრაფესად E მომენტში და E-ს მნიშვნელობაა გუშინ.

პეტრემ დაინახა თინა. ეს მოხდა 11 საათამდე.

ხედავს(პეტრე, თინა) :E & მდე(11:00,E).

აქ წარსული დრო არაა განხილული სიმარტივისათვის, როცა ჩვენ გვაქვს ისეთი სამყარო, სადაც კონტექსტი არავითარ როლს არ თამაშობს, მაშინ ჩვენ შეგვიძლია მივიღოთ:

P:P← P.

როცა მხოლოდ დროს აქვს მნიშვნელობა, მაშინ P E შეიძლება ჩავწეროთ პროლოგზე როგორც ადგილი აქვს(P,T), სადაც T არის დროის მომენტი E. ასევე, შეიძლება გამოვიმუშაოთ სპეციალური წესები ინდექსირებული პრედიკატების პროლოგში გადასაყვანად ყოველ განსაკუთრებულ შემთხვევებში

4 თავი

კომპიუტერული მასწავლებელი სისტემები

ეს თავი ეხება ისეთი კომპიუტერული მასწავლებელი სისტემის აგების საკითხებს, როცა შესაძლებელი იქნება სწავლების პროცესში კომპიუტერსა და მოსწავლეს შორის ორმხრივი დიალოგი ბუნებრივ ენაზე (იგულისხმება,რომ დიალოგი არაა შეზღუდული იმით, რომ მარტო კომპიუტერს შეეძლოს შეკითხვის დასმა მოსწავლისათვის არამედ პირიქითაც). კომპიუტერული მასწავლებელი სისტემის აგება ფართოდ დაიწყო 80-იან წლებში და დღესაც გრძელდება ამ მიმართულებით კვლევები. შეიქმნა მთელი რიგი კომპიუტერული სასწავლო კურსები, რომლებიც გამოიყენებიან როგორც სასწავლო დაწესებულებებში, ასევე ინტერნეტის საშუალებით სწავლებისათვის. ასეთი სისტემებიდან ჩვენი შეხედულებით ყველაზე მნიშვნელოვანი სისტემები ფართოდ მიმოხილულია სადისერტაციო ნაშრომის პირველ თავში. უმეტესი კომპიუტერული მასწავლებელი სისტემების ძირითადი ნაკლი მდგომარეობს იმაში, რომ ამ სისტემებში არ არსებობს დიალოგი მოსწავლესა და მასწავლებელ სისტემას შორის, ან თუ არსებობს, ასეთი დიალოგი ძალზე შეზღუდული სახისაა, რადგან სრულყოფილი ფორმალურ წარმოდგენას, რომელიც ფორმით განსხვავებული, მაგრამ ერთიდაიგივე შინაარსის მქონე ორი წინადადებების გამოცნობის საშუალებას მოგვცემდა. ამ მიმართულებით მთელ მსოფლიოში მიმდინარეობს ინტენსიური მუშაობა და იგი წარმოადგენს ერთ-ერთ მნიშვნელოვან პრობლემას. ამ თავში განხილულია არსებული კომპიუტერული მასწავლებელი სისტემები, დადგენილია მათი ნაკლოვანი და დადებითი თვისებები და მათი კრიტიკული ანალიზის

საფუძველზე შედგენილია უფრო სრულყოფილი მასწავლებელი სისტემის აგების სქემა, შემუშავებულია ასევე ქართული ტექსტების მორფოლოგიური და სინტაქსური ანალიზის ალგორითმები და ამით მომზადებულია საფუძველი ქართული ტექსტების სემანტიკური ანალიზისთვის, რომლის გარეშედაც შეუძლებელია სრულყოფილი მასწავლებელი სისტემის შექმნა. პირველ პარაგრაფში მიმოხილულია არსებული მასწავლებელი სისტემები და მოცემულია მათი კრიტიკული შეფასება. მეორე პარაგრაფში აღწერილია ზმნების ორიგინალური კლასიფიკაცია, რომელიც საფუძვლად უდევს როგორც ქართული ზმნების დაშლას მორფემებად და ზმნის ფორმის მორფოლოგიურ კატეგორიათა დადგენას, ასევე შეიძლება გამოყენებულ იქნეს ზმნის მოცემულ მორფოლოგიურ კატეგორიათა საშუალებით შესატყვისი ზმნის ფორმის ასაგებად; აღწერილია ქართულ ზმნათა მორფოლოგიური ანალიზის ალგორითმის გრაფული წარმოდგენა, რომელიც შეიძლება გამოყენებულ იქნეს ქართული ზმნის ფორმათა აღრიცხვისათვის. მოცემულია ზმნის ფორმათა მორფემებად დაშლის პროგრამა ტურბოპროლოგზე და მისი აღწერა. ამავე პარაგრაფში აღწერილია ქართული ფრაზების კომპიუტერული სინტაქსური ანალიზის ორიგინალური წესები და პროგრამა ტურბოპროლოგზე, რომელიც იძლევა ქართული ფრაზების სინტაქსურ გარჩევას. მესამე პარაგრაფში აღწერილია განზოგადოებული ქსელური გრამატიკა (ATN), რომელიც შეიძლება საფუძვლად დაედოს ამავე თავში წარმოდგენილი კომპიუტერული მასწავლებელი სისტემის სქემის რეალიზაციას. აქვე აღწერილია ასევე ცოდნის ბაზის სტრუქტურა მორფოლოგიური და სინტაქსური ცოდნის წარმოსადგენად.

გამოყენებულია ის მეთოდები, რომლებიც ფართოდ გამოიყენება მასწავლებელი სისტემის აგებისათვის. ეს მეთოდები ემყარება ფორმალურ გრამატიკათა თეორიას და გამოიყენება ბუნებრივენოვანი ტექსტის კომპიუტერული

ანალიზისათვის. შემუშავებულია ბუნებრივენოვანი კომპიუტერული სწავლებადი სისტემის აგების სქემა, რომელიც შეიძლება გამოყენებულ იქნეს როგორც მასწავლებელი სისტემის აგებისათვის, ასევე ბუნებრივენოვანი ტექსტებიდან ცოდნის კომპიუტერული დაგროვებისათვის. შემუშავებულია ქართული ენის მორფოლოგიური და სინტაქსური ანალიზისადმი ორიგინალური მიდგომა, რომელიც საფუძვლად დაედება ქართული ტექსტის სემანტიკურ ანალიზს.

4.1 **მასწავლებელი სისტემები.** ავტომატიზირებული მასწავლებელი სისტემა უზრუნველყოფს სწავლების პროცესის ადაპტაციას მოსწავლეთა ინდივიდუალურ ხასიათთან, განტვირთავს მასწავლებლებს მთელი რიგი შრომატევადი და ხშირად განმეორებადი ოპერაციებისაგან სასწავლო ინფორმაციის წარმოდგენისა და ცოდნის შემოწმებისას, ხელს უწყობს ცოდნის შემოწმების ობიექტური მეთოდების შემუშავებას და აადვილებს სასწავლო მეთოდური გამოცდილების დაგროვებას. ცხადია, ავტომატიზირებული მასწავლებელი სისტემებით აღჭურვილ კლასებში მეცადინეობის ჩატარებისას იზრდება მოსწავლეთა აქტიურობა. ისინი დამოუკიდებლად ამუშავებენ დიდი მოცულობის სასწავლო ინფორმაციას, იზრდება ასევე მასწავლებელთა შესაძლებლობები სწავლის პროცესის მართვაში.

ამ პარაგრაფში მასწავლებელი სისტემები დაყოფილია ორ კლასად:

1) არაინტელექტუალური მასწავლებელი სისტემები ან ტრადიციული მასწავლებელი სისტემები და

2) ინტელექტუალური მასწავლებელი სისტემები.

მე-2 პარაგრაფში როგორც ტრადიციული მასწავლებელი სისტემის მაგალითი, განხილულია სისტემა — SAM. ეს სისტემა ახორციელებს დიალოგური სწავლების მეთოდს. სწავლების სცენარს ადგენს სასწავლო მასალის ავტორი, აწარმოებს ინფორმაციული ფრაგმენტების ფორმირებას. ამ

სისტემის ნაკლოვან მხარეს წარმოადგენს ის, რომ ძირითადი სწავლების პროცესი რეალიზებულია კითხვა-პასუხის საშუალებით. კომპიუტერი უსვამს კითხვას მოსწავლეს და მოსწავლემ მხოლოდ უნდა გასცეს პასუხი, თანაც პასუხის ფორმა შეზღუდულ ჩარჩოებშია მოქცეული. ასეთ სისტემებში არ იგულისხმება ტექსტის სემანტიკური ანალიზი და ცხადია, კომპიუტერი ვერ აგროვებს ცოდნას ასათვისებელი საგნის შესახებ. არსებული ნაკლის თავიდან აცილებისათვის აუცილებელი პირობაა კომპიუტერმა შეძლოს ტექსტში არსებული ცოდნის გადატანა ცოდნის ბაზაში ექსპერტის დახმარებით და პირიქით, ცოდნის ბაზაში არსებული ცოდნის საშუალებით ბუნებრივენოვანი ტექსტის გენერირება რადგან ცოდნის ავტომატურად დაგროვების მეთოდები ჯერჯერობით არაა სრულყოფილი. იმის მიხედვით, თუ როგორი წარმატებით იქნება გადაწყვეტილი ეს საკითხი, შეიძლება ვილაპარაკოთ მასწავლებელი სისტემის ვარგისიანობაზე. მე-3 პარაგრაფში განხილულია რამოდენიმე მნიშვნელოვანი არასრული სწავლებადი ბუნებრივენოვანი სისტემა, ნამოწეულია წინა პლანზე ის თვისებები, რომლებიც აუცილებლად უნდა გააჩნდეთ სწავლებად სისტემებს. ასეთ სისტემებს მიეკუთვნებიან Nanoklaus, Unix Consultant, Kalipsos, Lilog და სხვა. განხილული სისტემებიდან ნაწილი დაპროექტებულია ნებისმიერი ცოდნის დიალოგის საშუალებით ასათვისებლად, ნაწილში კი ცოდნის შეტანა ხდება ხელით და არა დიალოგით. ასეთი სისტემის მაგალითად შეიძლება დავასახელოთ სისტემა Cyc. აქ ცოდნა კოდირებულია ორ განსხვავებულ, მაგრამ ერთიმეორესთან დაკავშირებულ ენაზე: ფრეიმების ენა და შეზღუდვათა ენა.

ყველა ზემოთ აღნიშნული სისტემის განხილვა, მათ ნაკლოვანებებსა და შესაძლებლობებზე დაკვირვება იძლევა როგორც მასწავლებელი სისტემის აგების ტექნიკის გაცნობის, ასევე მასწავლებელი სისტემის აგებისადმი საკუთარი მიდგომის გამომუშავების საშუალებას.

***4.2* დიალოგური მასწავლებელი სისტემები.** მასწავლებელი სისტემის აგების უმარტივეს სქემას წარმოადგენს მოსწავლისათვის ასათვისებელი მასალის კომპიუტერის ეკრანზე ტექსტის სახით მიწოდება და ასათვისებელ მასალაზე ამომწურავი კითხვებისა და მათზე სწორი პასუხების მომზადება. თუ მიზნად დავისახავთ ასათვისებელი ცოდნის კომპიუტერში მხოლოდ ტექსტის სახით შენახვას, მაშინ გადასაწყვეტი რჩება მხოლოდ ისეთი კომპიუტერული სისტემის შექმნა, რომელიც მოახდენდა სხვადასხვა საგნის მასწავლებელი პროგრამების შედგენის პროცესის ავტომატიზაციას. მოცემულ პარაგრაფში განხილულია სწორედ ამ მიზნით შექმნილი სისტემა SAM, რომელიც ითვალისწინებს სპეციალიზირებული ენის შექმნას მასწავლებელი პროგრამის შესადგენად. ი.ვეკუას სახელობის გამოყენებითი მათემატიკის ინსტიტუტის სისტემური პროგრამირების განყოფილებაში TURBO PROLOG ენაზე შედგენილ იქნა ინტერპრეტატორი, რომელიც საშუალებას იძლევა ავტომატიზირებული მასწავლებელი სისტემის "ZJR" ენის ბრძანებებით შედგენილი ავტორის კურსი შესრულდეს პერსონალურ კომპიუტერზე. ინტერპრეტატორი, მისი ფუნქციონირებისათვის საჭირო კონკრეტული სასწავლო კურსი და დამხმარე ფაილები ერთად ცნობილია მასწავლებელი სისტემა SAM –ის სახელწოდებით. მასწავლებელი სისტემა SAM [26,27] გამოიყენება სასწავლო დაწესებულებებში დამოუკიდებელი დიალოგური მუშაობის რეჟიმში საგნის გარკვეული კურსის შესასწავლად. იგი განიხილება როგორც სწავლების დამხმარე საშუალება. ავტორის მიერ შედგენილი კურსი იწერება ZJR ენაზე. შემდეგ კი სრულდება SAM სისტემის საშუალებით ინტერპრეტაციის რეჟიმში.

სასწავლო კურსი იქმნება კონკრეტული საგნის სპეციალისტის მიერ. ავტორი განსაზღვრავს კურსის შინაარს, ავტომატიზირებული სწავლების მეთოდიკას, კონკრეტულ

პროცედურებს და სასწავლო მასალის გავლის ეტაპებს. ავტორის ძირითადი ამოცანაა ავტომატიზირებული მასწავლებელი კურსის სცენარის შედგენა. ავტორის მიერ შედგენილი სასწავლო კურსის ალგორითმი იყოფა ნაწილებად. ყოველი ნაწილი ამუშავებს ერთ კონკრეტულ საკითხს. თავდაპირველად ხდება იმ მასალის გადმოცემა, რომლის წარმოდგენას აპირებს ავტორი მოცემულ ნაწილში;შემდეგ ავტორმა უნდა შეადგინოს კონკრეტული ალგორითმი იმის დასადგენად, თუ როგორ აითვისა მოსწავლემ მასალა და ნება დართოს მოსწავლეს გადავიდეს შემდეგ ნაწილზე ან დააბრუნოს შესატყვისი ნაწილის გამეორებაზე. ავტორმა უნდა გაითვალისწინოს პასუხის წარმოდგენის ყველა შესაძლო ხერხი: მაგალითად, თუ პასუხი რიცხვია, რიცხვი შეიძლება წარმოვადგინოთ ციფრებით - 25 და შეიძლება სიტყვითაც - ოცდახუთი. ასე მომზადებული კურსი შემდეგ აღიწერება კურსების აღწერის ენაზე ZJR. SAM-ში ეს ენა გაფართოებულია ისეთნაირად, რომ შესაძლებელია გრაფიკული ობიექტების შემოტანა კურსის პროგრამაში PAINTBRUSH პროგრამის საშუალებით. ენის ოპერატორები დაყოფილია 3 ჯგუფად:

I. პრობლემური — PR, QU, RD.

II პასუხის დამამუშავებელი — CA, CA(L), CA(W).
CB, CB(L), CB(W).
WA, WA(L), WA(W).
WB, WB(L), WB(W).
UN, UN(L), UN(W).

III მომსახურე (დამხმარე) ბრძანებები — EP, LB, TY, LL, DE, PA,

მაგალითად, პრობლემური ოპერატორები გამოიყენება სასწავლო მასალის ეკრანზე გამოტანისა და მოსწავლისთვის კითხვის დასმისათვის.

ზემოთ ჩამოთვლილი ოპერატორების არჩევა ხდება ავტორის მიერ კურსში ჩადებული ალგორითმის შესაბა-

მისად. აბ პრძანებებით შედგენილი სასწავლო კურსი უნდა იქნეს შეტანილი ფაილში "კურსი.PRO". ნაწილები ერთმანეთისაგან გამოიყოფიან ჭდეებით და მათზე მიმართვა ხდება მენიუდან. SAM სისტემის ინტერპრეტატორი საშუალებას იძლევა ZJR ბრძანებებით ჩაწერილი კურსი შესრულდეს IBM PC - ის ტიპის პერსონალურ კომპიუტერზე ოპერაციულ სისტემა DOS-ში.

SAM სისტემის ფუნქციონირებისათვის უნდა შეიქმნას დირექტორია, სადაც იქნება ჩაწერილი როგორც ძირითადი პროგრამა და დამხმარე ფაილები, ასევე სასწავლო კურსი.

როგორც ვხედავთ ამს SAM მიეკუთვნება იმ სისტემათა რიცხვს, რომლებიც ასათვისებელ მასალას მოსწავლეს წარმოუდგენენ ეკრანზე ტექსტის სახით, კითხვაზე პასუხის გაცემა ხდება ძალიან შეზღუდული ფორმით, ხოლო უკუკავშირი (მოსწავლის მიერ შეკითხვის დასმა) საერთოდ არ განიხილება. რეალურია SAM სისტემის შესაძლებლობათა გაფართოება კითხვა-პასუხის პროცედურის გაუმჯობესებით. თუ აღნიშნულ სისტემას დავუმატებთ ქართული ფრაზების სინტაქსურ ანალიზს, რომელიც თავის მხრივ ითვალისწინებს ფრაზაში შემავალი სიტყვების მორფოლოგიურ ანალიზს. ეს საშუალებას მოგვცემს შევცვალოთ კითხვებისა და პასუხების წინადადებები მათი სინტაქსური სტუქტურებით. შევადგინოთ სტრუქტურათა ექვივალენტური გარდაქმნის წესები, რომლებიც გაითვალისწინებენ სტრუქტურაში შემავალი სიტყვების შეცვლას მათი სინონიმებით და სიტყვათა დასაშვებ გადაადგილებებს წინადადებაში. შემდეგ შეიძლება შედგეს სტრუქტურათა ექვივალენტობის ალგორითმი და მისი საშუალებით დავადგინოთ სწორი პასუხისა და მოსწავლის პასუხის ექვივალენტობა. ასეთი გზით საგრძნობლად გაიზრდება აზრობრივად სწორი პასუხის ტექსტურად წარმოდგენის ალტერნატივები

4.3 სწავლებადი სისტემების მიმოხილვა. ცოდნის ათვისე-

ბის ინსტრუმენტი შეიძლება გახდეს კომპიუტერი, რომელსაც შეიძლება ასწავლო. მისთვის არაა აუცილებელი გაიგოს ყველაფერი რაც არის ნათქვამი. ამის ნაცვლად მან უნდა გამოიცნოს, რაც მან ვერ გაიგო და შეძლოს კითხვის დასმა, რომელიც მიგვიყვანს უკეთეს გაგებამდე. მან შეიძლება ვერ აღმოაჩინოს ახალი ცოდნა ავტომატურად, მაგრამ უნდა ააგოს ცოდნის ბაზა ნახევრად ავტომატურად მასწავლებლის დახმარებით: ერთხელ შეძენილი ცოდნა რაიმე ფორმით მან უნდა გარდაქმნას სხვა ფორმაში. 80-იან წლებში შექმნილი სწავლებადი სისტემების Nanoklaus [28], Unix Consultant [29] და Kalipsos-ის [30] მოკლე განხილვა იძლევა სწავლებადი სისტემების აგების ტექნიკის გაცნობის კარგ შესაძლებლობას.

Nanoklaus არის კლასიკური მაგალითი ისეთი სისტემისა, რომლის სწავლება იოლია. როგორც შედეგი დიალოგისა Nanoklaus (NK) აგებს ცნებების ტიპიურ იერარქიას, შეზღუდვებს კვანტიფიკატორებზე და ფუნქციონალურ დამოკიდებულობებზე, აგრეთვე მარტივ სიტყვურ განსაზღვრებებს. როცა ახალი კვანძი შემოიტანება ტიპთა იერარქიაში NK ყოველთვის სვამს კითხვას იმის გასარკვევად, თუ როგორ არის დაკავშირებული ახალი ტიპი თავისი სუპერტიპის სხვა ქვეტიპთან. NK დიალოგით აგებს ცოდნის დიდ ნაწილს. ცოდნის ასეთი შეტანა დიალოგის საშუალებით შეიძლება გახდეს მოსაწყენი და იმისათვის რომ შემცირდეს კითხვები მასწავლებელმა სისტემამ ეს ინფორმაცია უნდა მიიღოს სხვა გზით მანქანური ლექსიკონიდან ან წინასწარ განსაზღვრულ ტიპთა იერარქიიდან. დიალოგი სასურველია ცოდნის ბაზის გამართვისა და მოქნილობისათვის, მცდარი ინფორმაციის შემოწმებისათვის და გამოყვანების საჩვენებლად.

Unix Consultant (UC) შექმნილი იყო მომხმარებლისა-თვის Unix-ის ბრძანებებისა და მათი ფორმატების სასწავლებლად. მას აქვს ორი რეჟიმი: კონსულტაციის რეჟიმი, როცა

იგი უხსნის Unix-ის მომხმარებლებს Unix-ის ბრძანებების გამოყენებას და ხწავლების რეჟიმი(UCT), როცა ექსპერტი ასწავლის ახალ ბრძანებებს UC-ს. UC ცოდნის წარმოსადგენად იყენებს ფრეიმების მსგავს ფორმებს PEARL ენაზე (სპეციალურ ქსელურ წარმოდგენის ენას Kodiak-ს) მისი შესაძლებლობები ძალზე შეზღუდულია იმისათვის, რომ იგი გამოყენებულ იქნეს ზოგადი ცოდნის წარმოდგენისათვის. კერძოდ, მას არ შეუძლია ქვანტიფიცირებული გამოსახულებების წარმოდგენა და მით უმეტეს ცოდნის ლოგიკურ გამოსახულებებში გადაყვანა, რაც მისი გამოყენების არეს ძალზე ზღუდავს. განსხვავებით Nanoklaus-ისა და UCT-საგან, Kalipsos დაპროექტებული იყო ნებისმიერი ცოდნის დიალოგის საშუალებით ასათვისებლად. იგი კითხულობს ფრანგულ ტექსტებს, მაგრამ არ შეუძლია ზოგიერთი ისეთი პრობლემის გადაწყვეტა, როგორიცაა რთული ლინგვისტური კონსტრუქციები და წინააღმდეგობრივი ინფორმაცია ცოდნის ბაზაში. როცა აწყდება სიძნელეებს ის სვამს კითხვას. დიალოგის შედეგს წარმოადგენს ტექსტი გადასახული კონცეპტუალურ გრაფებში. Cyc[31] წარმოადგენს განხილული სისტემებისაგან გარკვეული თვალსაზრისით განსხვავებულ პროექტს, რომელიც მიზნად ისახავს ენციკლოპედიის სტატიებიდან ცოდნის ათვისებას. ამ სისტემაში ცოდნის შეტანა ხდება ხელით, ნაცვლად იმისა, რომ მოხდეს დიალოგის საშუალებით. ცოდნა აქ კოდირებულია ორ განსხვავებულ, მაგრამ ამასთან ერთად დაკავშირებულ ენაზე: ფრეიმების ენა და შეზღუდვათა ენა. ფრეიმები ქმნიან მარტივ წარმოდგენას, რომლებიც მოხერხებულია დიდი მოცულობის ცოდნის წარმოსადგენად, მაგრამ ცოდნის წარმოდგენა შეზღუდულია, ვინაიდან ის უძლურია წარმოადგინოს რთული ლოგიკური დამოკიდებულობები, ამიტომაც აქვს მას შეზღუდვების ენა, რომელიც არის პრედიკატების ლოგიკის ენის გარკვეული

ვერსია LISP-ზე დაფუძნებულ ნოტაციაში. კონცეპტუალური გრაფი არის ერთ-ერთი ვერსია სემანტიკური ქსელებისა. განვიხილოთ კონცეპტუალური გრაფი წინადადებისათვის A cat chased a mouse. ჩაეწეროთ მისი კონცეპტუალური გრაფი წრფივი ფორმით:

(PAST)®[[CAT]¬(AGNT)¬[CHASE]®(PTNT)®[MOUSE]]

ეს კონცეპტუალური გრაფი შეიძლება გადაყვანილ იქნეს პრედიკატების აღრიცხვის ფორმულაში:

PAST (($x)($y)($z)(CAT(X) Ù CHASE(X) Ù MOUSE(Z) Ù AGNT(y,x) ⇀ Ù PTNT(y,z))).

ეს არის პირველი რიგის პრედიკატების ფორმულა. ფრეიმიდან გრაფში გადასახვა არის პირდაპირი. ფრეიმების სლოტები ხდებიან ორადგილიანი დამოკიდებულობები, შეზღუდვები სლოტებზე ხდებიან ცნებების ტიპების მონიშვნები და სლოტებში მოთავსებული მნიშვნელობები შედიან მითითების ველში, რომელიც მოსდევს ცნების ტიპის მონიშვნას.

სლოტები CYC ფრეიმებში და პრედიკატები CYC შეზღუდვებში გამიზნულია იმისათვის, რომ წარმოვადგინოთ მაღალი რიგის დამოკიდებულებები. კონცეპტუალური გრაფების უმეტესობა გამიზნულია უფრო პრიმიტიული დამოკიდებულების გამოსაყვანად, როგორიცაა AGNT(რაიმე მოქმედების აგენტი), PTNT(პაციენტი), RLST(შედეგი) STAT(მდგომარეობა), PART(შეიცავს როგორც ნაწილს), PTIM (დროითი წერტილი), Name (აქვს სახელი); POSS(ფლობს, ეკუთვნის), IN(მდებარეობს შიგნით). ცხადია, დამოკიდებულობათა ორივე სახეობას აქვს თავისი უპირატესობა და ნაკლი.

კონცეპტუალური გრაფების განსაზღვრებითი მექანიზმი ითვალისწინებს მაღალი რიგის დამოკიდებულებების განსაზღვრის ხერხს უფრო პრიმიტიული დამოკიდებულობებით. ორადგილიანი დამოკიდებულობა როგორიცაა OWNER (მფლობელი) შეიძლება იყოს განსაზღვრული OWN

ცნების ტერმინებში, რომელიც შეესატყვისება ზმნას OWN :

OWNER = (I X,Y)

[ENTITY:*C]¬(PTNT)®[OWN]¬(STATE)¬[PERSON:*Y]

ეს გრაფი ამბობს, რომ OWNER დამოკიდებულება აკავშირებს OWN-ის პაციენტს X არსებას ვინმე პიროვნებასთან Y, რომელიც არის OWN მდგომარეობაში.

CYC ფრეიმებსა და შეზღუდებს არა აქვთ სტანდარტული გადასახვა ინგლისურში, მაგრამ კონცეპტუალური გრაფები, რომლებიც შეიცავენ დაბალი დონის დამოკიდებულობებს, აკეთებენ პირდაპირ გადასახვას ინგლისურში.

ცოდნის წარმოდგენის ენა L_{LILOG}[32] შემუშავებულ იქნა როგორც საფუძველი LILOG პროექტისა, რომელიც მიძღვნილია ბუნებრივი ენის გაგებაში ძირითადი კვლევებისადმი. პროექტის მიზანია განავითაროს მეთოდები და იარაღები გერმანული ენის რეალური ფრაგმენტის ავტომატური დამუშავებისათვის [33]. L_{LILOG}-ს საფუძვლად უდევს მრავალსორტიანი პრედიკატების ლოგიკა სორტებზე რიგის დამოკიდებულებით. იგი იგება ჩვეულებრივად: თერმები მიუთითებენ ობიექტებს და ფორმულები აღნიშნავენ ობიექტებს შორის დამოკიდებულებებს, რადგან მრავალსორტიან ლოგიკასთან გვაქვს საქმე, ყოველ თერმს დამატებით უნდა მიეთითოს სორტი. მაგალითად, father(John) არის სწორად ჩაწერილი თერმი თუ John არის Person სორტის კონსტანტა და father არის ერთადგილიანი ფუნქციონალური სიმბოლო, რომელიც არგუმენტად იღებს Person სორტის რაიმე ობიექტს და გვაძლევს შედეგად Person სორტის ობიექტს. ფორმულების აგება იწყება ატომური ფორმულიდან

$P(t_1,t_2, ...t_n)$

$t=t\not\in$,

სადაც t და t¢-ს უნდა ჰქონდეს ერთი და იგივე სორტი. L_{LILOG}-ში დაშვებულია ატრიბუტების დამატება სორტებისათვის. სორტულ განსაზღვრებას

red -building = = building(colour:{red})

გამოჰყავს სორტი red -building სორტისაგან building და ატრიბუტისაგან colour სადაც red არის colour სორტის რაიმე კონსტანტა.

განვიხილოთ პრედიკატი see(John, the men), სადაც არაცხადად ნაგულისხმევია, რომ პირველი არგუმენტი მიგვითითებს სუბიექტს და მეორე არგუმენტი ობიექტს. L_{LILOG}-ში არგუმენტების როლი არის შემოტანილი ატრიბუტი - მნიშვნელობა ნოტაციაში და ვღებულობთ see პრედიკატისათვის ჩაწერას: see(subject = John, object= the man). თუ საჭიროება წარმოიშვა see პრედიკატი შეიძლება იყოს განხილული ყველა შესაძლო არგუმენტებით: see(subject = John, object= the man, tool= telescope, time= every day), რომელიც წარმოადგენს წინადადებას " John sees the man with telescope every day ".

შესაძლოა L_{LILOG}-ის გაფართოება მაღალი რიგის ენამდე, რომელიც უშვებს პრედიკატული ცვლადების ქვანტიფიკაციას. არსებობს მთელი რიგი ცოდნის წარმოდგენის ენებისა, რომლებიც აქ არ მოგვიყვანია, მაგრამ ისინი გარკვეული თვალსაზრისით ახლოს დგანან ჩვენს მიერ განხილულ სისტემებთან. ასეთებია: KL-ONE [34] და მისი სახეცვლილებები LL-two[35], KRYPTON/X [36], BACK [12], BABILON [37].

4.4 ქართული ტექსტის ანალიზი

4.4.1 **შესავალი.** სწავლების უმნიშვნელოვანეს ელემენტს წარმოადგენს, ბუნებრივ ენაზე მასწავლებელსა და მოსწავლეს შორის ორმხრივი კავშირის დამყარება. ეს

კავშირი შეიძლება განხორციელდეს ორივე მხრიდან კითხვებისა და პასუხების სახით როდესაც მასწავლებლის როლში გამოდის ეგმ-ის პროგრამა, ნარმოიშობა კითხვის შინაარსის ავტომატური გამოცნობისა და შესაბამისი პასუხის გენერირების პრობლემა. ამ პრობლემის გადაწყვეტის გარეშე შეუძლებელია სრულყოფილი ავტომატიზირებული მასწავლებელი სისტემის (ამს) აგება. ამიტომ უნდა შემუშავდეს ტექსტობრივი ინფორმაციის ავტომატური გაგების მექანიზმი, რაც აქტუალურია არა მარტო ამს-ის შექმნისას, არამედ აგრეთვე სხვა სისტემების შესაქმნელად, რომელთაც სჭირდებათ ბუნებრივი ტექსტის ავტომატური გაგება.

იმისათვის, რომ მასწავლებელმა სისტემამ დააგროვოს ცოდნა შესასწავლი საგნის შესახებ, გაიგოს მოსნავლის მიერ დასმული კითხვა და შეადგინოს შესატყვისი პასუხი, საჭიროა სისტემას შეეძლოს ტექსტის ავტომატური მორფოლოგიური, სინტაქსური და სემანტიკური ანალიზი.

მე-2 პარაგრაფში განხილულია ქართული სიტყვების მორფოლოგიური ანალიზისა და ლექსიკონის სტრუქტურის საკითხები. შემოთავაზებულია ორიგინალური მიდგომა. მე-3 პარაგრაფში აღწერილია ქართული ზმნის ფორმების მორფემებად დაშლის პროგრამა. მე-4 პარაგრაფში განხილულია ქართული ფრაზების სინტაქსური ანალიზის საკითხები და შემოთავაზებულია მათი გადაწყვეტის გზები. მე-5 პარაგრაფში აღწერილია სათანადო პროგრამა

***4.4.2* ქართული სიტყვების მორფოლოგიური ანალიზი.** ამ პარაგრაფში განხილულია ქართული სიტყვების მორფოლოგიური ანალიზისა და ლექსიკონის სტრუქტურის საკითხები და შემოთავაზებულია მათი გადაწყვეტის გზები[39].

ქართული სიტყვების კომპიუტერული მორფოლოგიური ანალიზისას ძირითადი ყურადღება უნდა დაეთმოს ზმნების წარმოდგენას ლექსიკონში და ზმნის კონკრეტული ფორმის ამოცნობას, ე.ი. მის დაშლას მორფემებად და პირიქით, ზმნის კონკრეტული ფორმის აგებას ზმნის ლექსიკური

ერთეულისა და იმ ცოდნის დახმარებით, რომელიც თან ახლავს ამ ლექსიკურ ერთეულს ლექსიკონში, რადგანაც დანარჩენი მეტყველების ნაწილების მორფოლოგიური ანალიზი ან სინთეზი არ წარმოადგენს დიდ სირთულეს და ცნობილია მისი გადაწყვეტის გზა [40]. არსებობს მორფოლოგიური ანალიზისადმი სხვა მიდგომებიც [41-43]. პირველ რიგში განვიხილოთ თუ რა სახის ცოდნა უნდა იყოს წარმოდგენილი ცოდნის ბაზაში ქართული ენის შესახებ. ეს ცოდნა შეიძლება დაიყოს შემდეგ ნაწილებად: ლექსიკოგრაფიული ცოდნა, ცოდნა სინტაქსზე, ცოდნა სემანტიკაზე და ცოდნა პრაგმატიკაზე [44,45].

თითოეული ნაწილისათვის უნდა იყოს განსაზღვრული როგორც ჩაწერის ფორმა, ასევე მისი შინაარსი. ჩვენ აქ განვიხილავთ მხოლოდ ლექსიკოგრაფიულ ცოდნას ქართული ზმნების წარმოდგენის მაგალითზე. ჩაწერის ფორმად იგულისხმება ფრეიმების ტექნიკა. თუ რა შინაარსის ინფორმაცია უნდა ჩაიწეროს ფრეიმებში, ამისათვის განვიხილოთ ქართული ზმნების კლასიფიკაცია ზმნის ფორმების წარმოქმნის თვალსაზრისით [46].

თუ ჩვენ ზმნის ფორმების მისაღებად საწყის ერთეულად ავიღებთ მხოლოდ ზმნის ძირს, მაშინ შეგვექმნება სიძნელეები ძირის შესატყვისი ლექსიკური მნიშვნელობების განცალკევების გზაზე. რადგან, კარგად ცნობილია, რომ თუ ზმნისწინს წინასწარ არ დავაფიქსირებთ, მაშინ ძალზე გაიზრდება ომონიმიური ფორმები და მათი გამოცნობის ალგორითმები ძალზე გაართულებს მორფოლოგიურ ანალიზს. ამიტომ, ჩვენ ვგულისხმობთ, რომ ზმნის ლექსიკური მნიშვნელობის დასაფიქსირებლად აღებულია ძირი და ზმნისწინი. თუმცა მათი დაფიქსირებაც კი სრულად ვერ გამორიცხავს ომონიმიის შემთხვევებს. მაგალითად, "ააგო" შეიძლება ნიშნავდეს ძეგლის აგებას ან ხანჯალზე აგებას. ამგვარად, ფრეიმში უნდა შევიტანოთ ზმნისწინი და ძირი, რომელიც ძირითადად დააფიქსირებს ზმნის ლექსიკურ მნიშვნელობას.

შემდეგი მნიშვნელოვანი ინფორმაციაა მორფემები ზმნის აქტიური და პასიური ფორმების მისაღებად და ზმნის ფორმის პირიანობის დასადგენად. ესენია ხმოვანი პრეფიქსები: ა, ი, ე, უ, ან მათი არდართვა - # და დონიანი ვნებითის მაჩვენებელი დ. აქ უნდა განვიხილოთ ორი შემთხვევა: როცა კონკრეტული მორფემა საერთოდ არ გამოიყენება მოცემული ლექსიკური მნიშვნელობისათვის და როცა კონკრეტულ ზმნის ფორმას არ გააჩნია განსახილველი მორფემა. ამისდამიხედვით ხდება ლექსიკური მნიშვნელობების დაყოფა კლასებად. რაც საშუალებას მოგვცემს ზუსტად დავადგინოთ ზმნის ფორმის პირიანობა კონკრეტული მორფების დართვა არდართვის მიხედვით. ზოგიერთი ძირი საერთოდ არ ირთავს ზმნისწინს ან ზმნისწინის როლში გამოდის სხვა სახის მორფემა, რომელიც საზოგადოდ გამოიყენება სხვა დანიშნულებით, ან ზმნისწინი ახლავს ძირს, მაგრამ მას დაკარგული აქვს ფუნქცია, რომლითაც იგი ჩვეულებრივ გამოიყენება. ასეთი ინფორმაცია აუცილებელია ზმნის ფორმის მორფოლოგიური კატეგორიის დასადგენად. გარდა ამისა, გვხვდება შემთხვევები, როცა ზმნის ერთიდაიგივე ლექსიკური მნიშვნელობისათვის გამოიყენება რამოდენიმე ძირი ან განსაკუთრებული სახით იწარმოება ზმნის ლექსიკური მნიშვნელობა, როგორიცაა მოძრაობის აღმნიშვნელი ზმნები. ასეთი სახის ინფორმაციაც უნდა იყოს წარმოდგენილი ფრეიმში. რადგანაც ფრეიმში წარმოდგენილი ცოდნა ზმნების შესახებ გამოიყენება, როგორც ზმნის ფორმის გამოსაცნობად ასევე ზმნის ფორმის ასაგებად, ამიტომ ზმნების ძირები უნდა დაიყოს კლასებად ზმნათა უღლებების თავისებურებათა გათვალისწინებით, ასევე პირის ნიშნები, თემის ნიშნები, მწკრივის ნიშნები უნდა კლასებად იმისდამიხედვით, თუ რომელი კლასის ძირებთან გამოიყენებიან ისინი. საერთოდ, როდესაც ზმნის რაიმე ფორმის მიღებისას ან მისი დაშლისას რაიმე წესით მორფემებად მიიღება ომონიმური შემთხვევა, უნდა შესწორდეს მორფემათა დადგენილი

კლასიფიკაცია ისეთნაირად, რომ შესაძლებელი იყოს ახალი წესის შედგენა ამ ომონიმიის ასაცილებლად. ზმნების ანალიზისათვის მნიშვნელოვანია ინფორმაცია იმის შესახებ, თუ რა რიგით (მარცხნიდან მარჯვნივ) ზმნის ფორმაში შეიძლება შეგვხვდნენ მორფემათა კლასების წარმომადგენლები და ინფორმაცია იმის შესახებ, როცა ერთი კლასის წარმომადგენლის ან კლასთა ჯგუფის წარმომადგენლების არსებობა ზმნის ფორმაში გამორიცხავს სხვა კლასის წარმომადგენლის არსებობას. მაგალითად, როცა ხმოვანი პრეფიქსის არსებობა ზმნის ფორმაში გამორიცხავს დონიანი ვნებითის ნიშნის არსებობას. ამგვარად, თითოეული წყვილისათვის ძირი და ზმნისწინი უნდა დადგინდეს რა მორფემებს ირთავს ეს წყვილი და რომელ კლასს ეკუთვნის იგი კონკრეტულ მორფემებთან მიმართებაში. როგორც აღვნიშნეთ კონკრეტული წყვილი ძირი და ზმნისწინი განსაზღვრავს როგორც მისგან წარმოებულ ზმნის ფორმათა ლექსიკურ მნიშვნელობას, ასევე იმასაც თუ როგორი წესით უნდა შევადგინოთ კონკრეტული ზმნის ფორმა მოცემული წყვილისაგან ან პირიქით, კონკრეტული ზმნის ფორმა რა მორფემებისაგან არის შედგენილი და როგორ დავადგინოთ მისი მორფოლოგიური კატეგორიები. ყველა წყვილს, რომელიც ერთნაირად აწარმოებს ზმნის ფორმებს ან პირიქით, მათი ზმნის ფორმები იშლებიან მორფემებად ერთიდაიგივე წესით, ვუწოდოთ უღლების სორტი. მაგალითად, "აგება" ზმნისთვის გვაქვს წყვილი (ა,გ). მისთვის ჩვენ შეგვიძლია ჩამოვაყალიბოთ წესი, თუ როგორ მივიღოთ მისი ნებისმიერი ზმნის ფორმა ამ წყვილისაგან და მორფემებისაგან, რომელსაც ეს წყვილი ირთავს (თუ რომელ მორფემებს ირთავენ და რომელი მორფემები არაა ურთიერთ თავსებადი მოცემულია ლექსიკონში). ყველა წყვილი, რომლისთვისაც იგივე წესი გამოიყენება რაც (ა,გ) წყვილისათვის, ქმნიან უღლების სორტს. ზმნის ფორმათა აღრიცხვის ფორმალიზმის აღწერისათვის განვიხილოთ მორფემათა ჯგუფები, რომლებიც გვხვდებიან საზოგადოდ

ზმნის ფორმებში და დავალაგოთ ისინი ზმნის ფორმაში მათი შეხვედრის რიგის მიხედვით (მარცხნიდან მარჯვნივ):

1.ზმნისწინი;
2.პირისნიშანი პრეფიქსი;
3.ხმოვანი პრეფიქსი;
4.ძირი;
5.დონიანი ვნებითი;
6.კონტაქტის ნიშანი;
7.თემის ნიშანი;
8.მწკრივის ნიშანი;
9.პირის ნიშანი;
10.რიცხვის ნიშანი.

ვუწოდოთ მათ სორტები და მოვიხსენიოთ ისინი მათი რიგითი ნომრის მიხედვით. მაგალითად, სორტი 1 არის მორფემათა სიმრავლე, რომელიც ცნობილია ზმნისწინების სახელწოდებით. ზმნის ფორმათა გარკვეული ქვეკლასისათვის დამახასიათებელია სორტების გარკვეული მიმდევრობა, რომელსაც ვუწოდოთ შედგენილი სორტი, მაგალითად, უღლების სორტისათვის (ა,გ), შედგენილი სორტი (1,2,3,4,5,6,7,8,9,10) განსაზღვრავს ზმნის ფორმათა გარკვეულ სიმრავლეს, როგორც აგება ზმნისათვის ასევე იმ ზმნებისათვის, რომლებიც იგივე წესით აწარმოებენ ზმნის ფორმებს, როგორც აგება ზმნა აწარმოებს. აგება ზმნისათვის შედგენილი სორტის (1,2,3,4,5,6,7,8,9,10) პირველი წევრი 1 მიგვითითებს , რომ აგება ზმნას აქვს შემდეგი თვისება: როცა ზმნისწინი ა არ გვხვდება ზმნის ფორმაში, მაშინ გვაქვს აწმყოს მწკრივის ზმნის ფორმები ან I და II სერიის არასრული ზმნის ფორმები ან თუ მესამე წევრი არის ა, მაშინ ზმნის ფორმები ორპირიანია. ამგვარად, ასეთნაირი წარმოდგენა საშუალებას გვაძლევს მარტივად ჩამოვაყალიბოთ წესები მთელი უღლების სორტისათვის, როგორც ანალიზისათვის ასევე სინთეზისათვის. მეორე მნიშვნელოვანი საკითხია თუ როგორ ჩავწეროთ ასეთი წესები. ბუნებრივ ენაზე მათი ზუსტი ჩაწერა თითქმის შეუძლებელია, რომ არაფერი ვთქვათ, თუ

რამდენად რთულია ბუნებრივ ენაზე ჩაწერილი ასეთი წესების აღქმა. ამისათვის მიზანშეწონილია სპეციალური გრაფების გამოყენება. ეს გრაფები წარმოადგენენ ხეების [40,47] შემოკლებულ ჩაწერას და ზუსტად აღრიცხავენ ზმნის ფორმებს. ჩვენს შემთხვევაში დაგვჭირდება გრაფის კვანძში წარმოდგენილი ინფორმაციის ჩაწერის მოდიფიკაცია. კერძოდ, როცა გვინდა შევამოწმოთ რაიმე თვისებას ა-ს აქვს თუ არა რაიმე კონკრეტული მნიშვნელობა v, ვწერთ შემდეგ გამოსახულებას f = v, ხოლო როცა გვინდა f-ს მივცეთ მნიშვნელობა მ ვწერთ f := v. როგორც თვისება, ასევე მნიშვნელობა შეიძლება იყოს ცვლადი ან კონსტანტა. მაგალითად, უღლების სორტისათვის ~f^u` I სერიის მწკრივის დადგენის წესს აქვს სახე:

→ 1 = + →7 = eb → 8 = # →mw:= 4 .....→ 8 = di → mw:= 5 ... → 8 = de → mw:= 6 → 8 = d→ mw:= 5Ú 6 → 7 = eb → 8 = # →mw := 1 → 8 = di → mw:= 2 → 8 =de → mw:= 3 → 8 =d ® mw:= 2Ú 3

რიცხვებით 1-6 გადანომრილია I სერიის მწკრივები. mw აღნიშნავს თვისებას მწკრივი, ხოლო ტოლობის მარცხენა მხარეში მოთავსებული რიცხვები არიან სორტები. ასეთი ჩაწერა გამოდგება ასევე ზმნის ფორმათა ზუსტი აღრიცხვისათვის, რაც საშუალებას მოგვცემს ზუსტად აღვრიცხოთ ქართულ ზმნათა დასაშვები ზმნის ფორმები [48] და ასეთი დაშლა გასაგები იქნება მათთვისაც ვინც არ იცის ქართული ენა. მაგალითად, თუ ფრანგული ზმნების უღლება წარმოიდგინება ცხრილის სახით [49], ქართული ენის შემთხვევაში იგი შეიძლება წარმოვადგინოთ განხილული გრაფის სახით.

4.4.3 ზმნის ფორმათა მორფემებად დაშლის პროგრამის აღწერა. ზმნის კონკრეტული ფორმის მორფოლოგიური ანალიზისათვის უნდა დადგინდეს ამ ფორმაში წარმოდგენილი ყველა მორფოლოგიური

კატეგორია. ლექსიკონში მიზანშეწონილია შეტანილ იქნეს ზმნის ძირი, ზმნისწინი და საერთოდ, ის აუცილებელი ინფორმაცია, რომელიც საჭიროა ამ ძირისაგან შედგენილი ფორმების მორფოლოგიურ კატეგორიათა დასადგენად. თავდაპირველად, ტექსტიდან აღებული ზმნის ფორმა უნდა დაიშალოს ნაწილებად — მორფემებად, შემდეგ კი მიღებული მორფემების საშუალებით დადგინდეს ზმნის ფორმის მორფოლოგიური კატეგორიები. თუ როგორ გაკეთდეს ეს დამოკიდებულია იმაზე, როგორია ზმნის ძირთან არსებული ინფორმაცია. ამის მიხედვით მორფოლოგიური ანალიზის ალგორითმიც იქნება სხვადასხვა. ჩვენ ვახდენთ ზმნათა კლასიფიკაციას უღლების სორტების მიხედვით, შემდეგ კი უღლების სორტის ყოველ ზმნას ვახასიათებთ შედგენილი სორტის მიხედვით. ასეთნაირი წარმოდგენა საშუალებას გვაძლევს ყოველი ზმნისათვის მივიღოთ ის აუცილებელი ინფორმაცია, რომელიც თან უნდა ახლდეს ზმნის ძირს ლექსიკონში ამ ძირისაგან შედგენილი ფორმების მორფოლოგიურ კატეგორიათა დასადგენად. როგორც უკვე აღვნიშნეთ, თავდაპირველად ზმნის ფორმა უნდა დაიშალოს მორფემებად და სწორედ ამ დანიშნულებით შეიქმნა ქვემოთ წარმოდგენილი პროგრამა. პროგრამა დაწერილია ტურბო პროლოგზე. ტურბოპროლოგს გააჩნია გამოყვანის საკუთარი ალგორითმი, რომლის მთავარ ინსტრუმენტს წარმოადგენს უნიფიკაციისა და უკუსვლის მეთოდები. წარმოდგენილი პროგრამა ეყრდნობა შემდეგ ალგორითმს: მას შემდეგ რაც შესავალი სიტყვა st_list პრედიკატით ჩაიწერება სიის სახით, იწყება მოცემული ზმნის ფორმის დაშლა მორფემებად. ე.ი. შესრულებას იწყებს s_dashla პრედიკატი და თავდაპირველად dashla_z პრედიკატით ხდება ზმნისწინის გამოყოფა, ხოლო სიტყვის დარჩენილი ნაწილი ჩაიწერება suf ცვლადში, ამასთან getbactrack პრედიკატი დაიმახსოვრებს შემდეგი ალტერნატივის მისამართს (ეს პრედიკატი სტანდარტულია და ცხადია, ის

სხვა შემთხვევაშიც იგივე დანიშნულებით იქნება გამოყენებული). შემდეგ dashla_p პრედკატი გამოყოფს პირის ნიშანს და სიტყვის დარჩენილი ნაწილი შეინახება suf1-ში. მომდევნო ეტაპზე dashla_r პრედიკატით გამოიყოფა რიცხვის ნიშანი და სუფიქსი (ამ შემთხვევაში სიტყვის დარჩენილი ნაწილი) ჩაიწერება suf2-ში, შემდეგ dashla_pn პრედიკატი გამოყოფს პირის ნიშან სუფიქსს, სიტყვის დარჩენილი ნაწილი ჩაიწერება suf3 –ში და ასე შემდეგ suf7–ში ჩაიწერება შესავალი სიტყვის დარჩენილი ნაწილი. ძირის გამოყოფის შემდეგ. მომდევნო ეტაპზე გამოიყოფა ხმოვანი პრეფიქსი. ცხადია, რომ ზმნის მორფემებად დაშლა შესაძლებელია წავიდეს არა სწორად. მაგალითად, თუ ზმნისწინად აღებული "ა" არ არის ზმნისწინი, არამედ ხმოვანი პრეფიქსია, დაშლა წავა არა სწორად, მაგრამ რომელიღაც ეტაპზე, მაგალითად, ზმნის ძირის გამოყოფისას აუცილებლად გაირკვევა რომ დაშლა არა სწორი გზით წავიდა და უკუსვლის მექანიზმის გამოყენებით ხდება ზმნის ფორმის ხელახალი დაშლა მორფემებად. შესავალი სიტყვის მორფემებად დაშლისათვის გამოყენებული ყველა ზემოთ ნახსენები პრედიკატი მუშაობს sheert(A,B,C) პრედიკატის საშუალებით, სადაც ვგულისხმობთ რომ A არის შესავალი სიტყვის პრეფიქსი, B შესავალი სიტყვის დარჩენილი ნაწილი ანუ სუფიქსი და მათი ერთმანეთთან შეერთება გვაძლევს C-ს, ანუ შესავალ სიტყვას. აქ A, B და C წარმოადგენენ ქართული ასოების სიებს. sheert პრედიკატი განსაზღვრულია ისე რომ თუ ცნობილია A,B და C ელემენტებიდან ორი, შესაძლებელი იყოს მესამეს პოვნა. ამის შემდეგ შესავალი სიტყვის დაშლა მორფემებად არ წარმოადგენს სირთულეს. პრედიკატი შეიძლება წარმოვადგინოთ როგორც ტოლობა A + B = C, სადაც A არის პრეფიქსი, B-სუფიქსი და C არის მათი კონკატენაციის შედეგი. ამ შემთხვევაში + არის

კონკატენაციის ოპერაცია და = არის დამოკიდებულება, რომელიც ჭეშმარიტია, როცა A და B- ს კონკატენაცია ემთხვევა C – ს. ამ ტოლობის შესამოწმებლად ვიყენებთ პროლოგის გამოყვანის მექანიზმს და კონკატენაცია ოპერაციის თვისებებს. ეს ტოლობა გადაიქცევა განტოლებად, როცა მისი რამოდენიმე კომპონენტი დაუკავშირებელი ცვლადია და პროლოგის გამოყვანის მექანიზმი ამ შემთხვევაში ეძებს შესაძლო ვარიანტებს განტოლების დასაკმაყოფილებლად. თუ ჩვენ ამ ტოლობას განვსაზღვრავთ როგორც პრედიკატს sheert(A,B,C.) და მივანვდით ორ არგუმენტს როგორც დაკავშირებულ ცვლადს, ხოლო მესამეს როგორც დაუკავშირებელ ცვლადს, პროლოგის გამოყვანის მექანიზმი იმუშავებს როგორც სიტყვის პრეფიქსად და სუფიქსად დაშლის ალგორითმი და ამოხსნის A + B = C განტოლებას. რადგანაც პროლოგის გამოყვანის მექანიზმი დაფუძნებულია უნიფიკაციის პრინციპზე, უკუსვლების რეალიზაციაზე. სინამდვილეში სიტყვის პრეფიქსად და სუფიქსად დასაშლელად ჩვენ ვიყენებთ კარგად ცნობილ უნიფიკაციის გრამატიკას და ამავე დროს შეგვიძლია მოვახდინოთ პრეფიქსებისა და სუფიქსების შესაძლო გადარჩევები (თუ გვექნება მათი სია) რომ დავაკმაყოფილოთ განტოლება. მაგალითად, თუ ავიღებთ ზმნის ფორმას აგებს შესასვლელ სიტყვად ე.ი. C=[ა,u,t,,,c] სიას, A-დ ყველა შესაძლო ზმნისწინებს ([ა], [ ], [ა,მ,j] და სხვა) და B-ს ავიღებთ უცნობად (დაუკავშირებელ ცვლადად), მაშინ შესაძლო ამოხსნები იქნება: 1)A=[ ], B=[ა,u,t,,,c] და C=[ა,u,t,,,c]; 2) A=[ა], B=[,u,t,,,c] და C=[ა,u,t,,,c]; B-ს შემდგომმა დაშლამ უნდა გამოავლინოს რომელი მათგანი არის სწორი. sheert (A,B,C) პრედიკატი შეიძლება ჩვენ განვაზოგადოთ და ზოგადად განვიხილოთ ტოლობა:

A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10 = C,

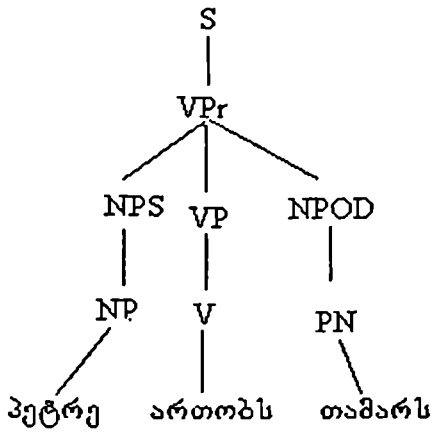
სადაც C დასაშლელი ზმნის ფორმაა, ხოლო A1,A2,A3.....A10 სორტები იმ რიგით დალაგებული, როგორც ეს მოცემულია შედგენილ სორტში. ამ განტოლების ამოხსნა მოგვცემს შესაძლო ვარიანტებს, ხოლო ძირთან არსებული ინფორმაციით უნდა ავირჩიოთ სწორი ვარიანტი. ეს პრინციპი გატარებულია წარმოდგენილ პროგრამაში ზმნის ფორმის მორფემებად დასაშლელად.

4.5 **ქართული წინადადების სინტაქსური ანალიზი.** წინადადების აზრის დასადგენად, მონტეგიუს მიდგომის მიხედვით პირველ რიგში ჩვენ უნდა ვიცოდეთ ამ წინადადების სინტაქსური სტრუქტურა და მას შევუსაბამოთ იგივე წინადადების სემანტიკური სტრუქტურა. ამისათვის კი საჭიროა გამოვიყენოთ რაიმე ფორმალიზმი, რომელიც დაეყრდნობა სინტაქსური ფორმის რაიმე თეორიას. არსებობს სხვადასხვა გრამატიკები, რომლებიც გამოიყენებიან ბუნებრივი ენის სინტაქსის აღსაწერად, როგორიცაა DCG გრამატიკა, უშუალო შემადგენელთა მეთოდი, GPSG[50], კატეგორიული გრამატიკები და სხვა. ჩვენ გამოვიყენებთ GPSG-ს და წინადადების სინტაქსურ სტრუქტურას წარმოვადგენთ ხეების საშუალებით. ქართული წინადადების სინტაქსს აღვწერთ თანდათანობით. ჯერ დავიწყებთ უმარტივესი შემთხვევებიდან და შემდეგ თანდათანობით განვიხილავთ უფრო რთულ შემთხვევებს. ავიღოთ მაგალითად:

პეტრე ართობს თამარს (1)

ეს წინადადება შედგება სამი შემადგენელი ნაწილისაგან: სუბიექტისაგან პეტრე, პირდაპირი ობიექტისაგან — თამარი და აწმყოს მწკრივში მყოფი ორპირიანი გარდამავალი ზმნისაგან — ართობს. ამგვარად, ეს

წინადადება შეიძლება დავშალოთ შემდეგ შემადგენელ ნაწილებად:



აქ ჩვენ გადავუხვევთ ტრადიციული წარმოდგენისაგან, როცა წინადადება იშლება ორ შემადგენელ ნაწილად სუბიექტის ჯგუფად და ზმნის ჯგუფად და შემდეგ ზმნის ჯგუფი იშლება ზმნად და პირდაპირ ობიექტად. რადგანაც ჩვენ მიგვაჩნია, რომ ქართული წინადადების სტრუქტურას სრულად განსაზღვრავს წინადადებაში მოცემული ზმნის ფორმა. ასეთი წარმოდგენის უპირატესობა უფრო ნათლად გამოჩნდება შემდგომში. ამ წარმოდგენაში წინადადების სინტაქსურ კატეგორიას აღნიშნავს S, რომელიც წარმოადგენს (1) წინადადებას. VPr არის ზმნა-ფრაზა, რომელიც გაიგივებულია S–თან, მაგრამ არის წარმოდგენის უფრო დაბალი დონე. იგი არის შუალედური სინტაქსური კატეგორია, რომელიც იშლება სუბიექტის ჯგუფად NPS, ზმნის ჯგუფად VP და პირდაპირი ობიექტის ჯგუფად NPOD. (1) წინადადებაში NPS – ს წარმოადგენს საკუთარი სახელი

PN, VP- ს — V ზმნა და NPOD- ს — PN. ხის ფოთლებს კი წარმოადგენენ მოცემულ სინტაქსურ კატეგორიათა კონკრეტული წარმომადგენლები: პეტრე, ართობს და თამარი. ჩვენ შემთხვევაში NPS და NPOD წარმოდგენილი არიან ს„კუთარი სახელებით, თუმცა სხვა შემთხვევაში ისინი შ¿იძლება წარმოდგენილი იყვნენ სხვა ტიპებით, როგორიცაა: პირის ნაცვალსახელები, ზოგადი არსებითი სახელები ან კიდევ დაიშალოს სხვა კატეგო-რიებად. (2) ანალიზი შეიძლება აღიწეროს ხომსკის წესით [51]:

$$S \rightarrow VPr \qquad (3)$$

რომელიც ამბობს, რომ წინადადება არის ზმნა - ფრაზა. ასევე

$$VPr \rightarrow NPS\ VP\ NPOD \qquad (4)$$

სადაც VPr წარმოადგენს NPS - ის, VP - სა და NPOD- ის კონკატენაციას. თავის მხრივ

$$NPS \rightarrow PN \qquad (5)$$

$$NPOD \rightarrow PN \qquad (6)$$

$$VP \rightarrow V \qquad (7)$$

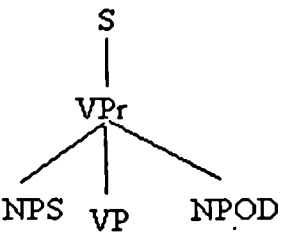
თუ ჩვენ გამოვიყენებთ (3)-(7) შეგვიძლია თანდათანობით გამოვიყვანოთ:

$$S \Rightarrow A \Rightarrow B \Rightarrow C \Rightarrow D \Rightarrow E \qquad (8)$$

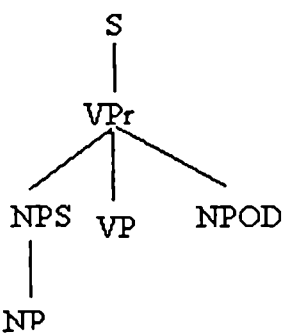
სადაც A,B,C,D და E ქვემოთ მოყვანილი ხეებია.

A:

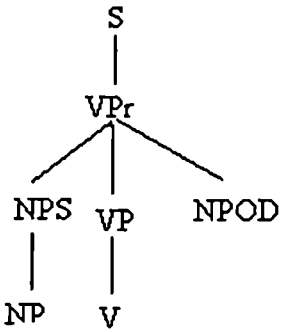
B:



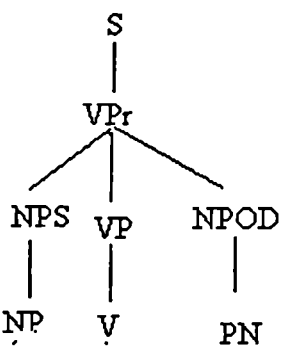
C:



D:



E:



იმისათვის, რომ დავამთავროთ (2)-ის გამოყვანა ჩვენ დაგვჭირდება ლექსიკური წესები, ე.ი. ცხადად წარმოვადგინოთ PN და V ტიპის სიტყვები. მათთვის ლექსიკური ჩასმის წესები შეიძლება წარმოვადგინოთ შემდეგნაირად:

PN → პეტრე, თამარს, ნინო, გიორგი...,მტკვარი,.. (9)

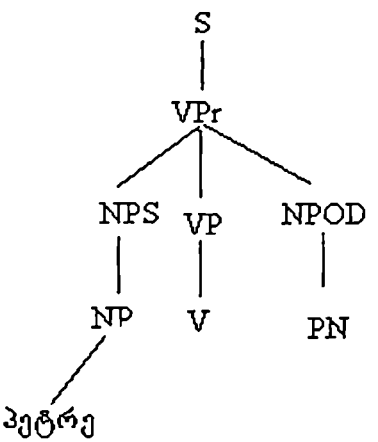
V → ართობს, გაართობს, ასახელებს,..., იცნობს,... (10)

აქ მძიმე გამოყენებულია ან მნიშვნელობით. ამ წესების გამოყენებით საბოლოოდ მივიღებთ:

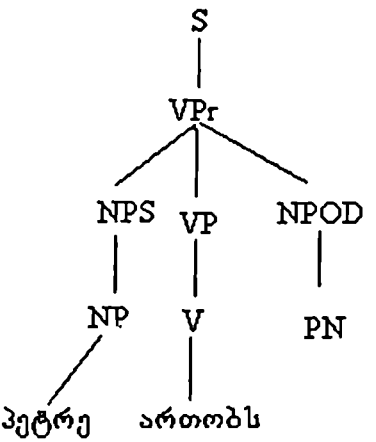
E⇒F⇒G⇒I

სადაც F,G და I ქვემოთ მოყვანილი ხეებია.

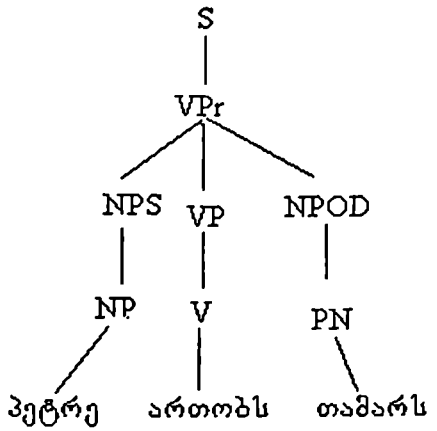
F:



G:



I:



I ხის ფოთლებში წერია ბუნებრივი ენის სიტყვები (ტერმინალური სიმბოლოები). მათი მარცხნიდან მარჯვნივ კონკატენაციით მივიღებთ (1) წინადადებას.

(3)-(10) წესების გამოყენებით შეიძლება მივიღოთ სხვა წინადადებებიც:

ნინო ასახელებს თამარს. (12)

მტკვარი იცნობს თამარს. (13)

აქედან (13) უაზროა. იმის გამო, რომ ქართულ წინადადებაში სუბიექტის, ზმნისა და პირდაპირი ობიექტის რიგი თავისუფალია, ამიტომ დასაშვებია (4) წესის შემდეგი ვარიანტებიც:

VPr→VP NPS NPOD (14)

VPr→VP NPOD NPS (15) VPr→NPOD VP NPS (16)

იმისათვის, რომ მოვიცვათ ქართული წინადადების მნიშვნელოვანი ნაწილი, საჭიროა დავუმატოთ ახალი წესები, ტიპები და სრულად აღვწეროთ ლექსიკური ჩასმის წესები.

პირველ რიგში განვიხილოთ ლექსიკური ჩასმის წესები. ახალი ტიპების დამატება მოგვცემს ახალი ლექსიკური ჩასმის წესებს და ეს გააფართოებს ჩვენს ენას ახალი წინადადებებით.

მაგალითად, უნდა შემოვიტანოთ პირის ნაცვალსახელის, განსაზღვრული ნაცვალსახელის, ზოგადი არსებითი სახელის, ზედსართავის, რიცხვითი სახელის, სხვადასხვა ზმნისზედების და სხვა ტიპები. გარდა ამისა, ჩვენ უნდა ვეცადოთ, შევამციროთ სიტყვების რაოდენობა თითოეული ლექსიკური ჩასმის წესში. ასე მაგალითად, (14) წესი შეიცავს თითოეული ზმნისათვის მის ყველა ფორმას, რაც ძალიან ზრდის ამ წესში სიტყვათა რაოდენობას. თუ ჩვენ ზმნის ტიპში შევიტანთ თითოეული ზმნისათვის მის უცვლელ ნაწილებს (ძირებს) ყველა ფორმაში, მაშინ საგრძნობლად შევამცირებთ ამ წესში სიტყვათა რაოდენობას. მაგრამ ასეთ შემთხვევაში დაგვჭირდება მორფოლოგიური წესები ძირისაგან ზმნის ნებისმიერი ფორმის მისაღებად (სიტყვათა ფორმის მისაღებად), ან წესები ზმნის ნებისმიერი ფორმის ძირად და სხვა ნაწილებად დასაშლელად (ზმნის პარადიგმიდან შესატყვისი მორფოლოგიური კატეგორიების დასადგენად). ასევე სახელებისათვის საჭიროა შევიტანოთ სახელის უცვლელი ნაწილები და სახელის ნებისმიერი ფორმა, მივიღოთ ბრუნვის ნიშნის, რიცხვის, თანდებულისა და ნაწილაკის დართვით. უნდა აიგოს მორფოლოგიური წესები სათანადო ტიპების

შემოტანით.

ამ საკითხს ჩვენ ვეხებით როცა განვიხილავთ ქართული სიტყვების მორფოლოგიის საკითხებს. ახლა კი, ჩვენ ვიგულისხმოთ, რომ ლექსიკური ჩასმის წესები შეიცვლება ამ წესებში შემავალი სიტყვების მიღების წესებით. ამისათვის, ჩვენ ვაგებთ სპეციალურ მორფოლოგიურ გრამატიკას, რომლის საშუალებითაც შესაძლებელი იქნება, როგორც სიტყვის ფორმის მიღება, ასევე სიტყვათა ფორმის დაშლა შემადგენელ ნაწილებად. იმისათვის, რომ ჩვენმა წესებმა გამოიყვანოს ქართული ენის მხოლოდ სწორი წინადადებები საჭიროა მათი დაზუსტება.

მაგალითად, (4)-ს უნდა მიეთითოს , რომ სუბიექტის გამომხატველი არსებითი სახელი უნდა იყოს სახელობით ბრუნვაში და პირდაპირი ობიექტის გამომსახველი სახელი უნდა იყოს მიცემით ბრუნვაში იმ შემთხვევაში, როცა ზმნა არის პირველ სერიაში და ორპირიანია. ასევე, სუბიექტი უნდა იყოს შეთანხმებული ზმნასთან პირსა და რიცხვში. იმისათვის, რომ ეს შეზღუდვები გამოვსახოთ წესის სახით ჩვენ დაგვჭირდება ყოველი მეტყველების ნაწილისათვის შემოვიტანოთ სათანადო თვისებები. არსებითი სახელებისათვის ეს არის ბრუნვა და რიცხვი, ხოლო ზმნისათვის — რიცხვი, პირი, გარდამავლობა, პირიანობა, დრო, კილო, გვარი, კონტაქტი, გეზი, ორიენტაცია და სხვა. ამჯერად, (4) წესის დასაზუსტებლად საკმარისია PN კატეგორიისათვის შემოვიტანოთ ბრუნვის და რიცხვის თვისებები. ყოველი თვისებისათვის უნდა განისაზღვროს თვისების მნიშვნელობათა არე. მაგალითად, ბრუნვის თვისების მნიშვნელობებია კონკრეტული ბრუნვები: სახელობითი, მოთხრობითი, მიცემითი, ნათესაობითი, მოქმედებითი, ვითარებითი და წოდებითი. თუ ბრუნვის თვისებას ავღნიშნავთ case-ით და მნიშვნელობებს გადავნომრავთ შესაბამისად 1-დან 7-მდე, მაშინ ჩანაწერი case=3 ნიშნავს, რომ ბრუნვა უნდა იყოს მიცემითი. აღვნიშნოთ გვარი voice-ით, ხოლო მისი მნიშვნელობები: მოქმედებითი, ვნებითი, საშუალო

(მედიოაქტივი, მედიოპასივი) გადავნომროთ 1-დან 4-მდე. გრამატიკული რიცხვი აღვნიშნოთ ტგქიუკ-ით, ხოლო მისი მნიშვნელობები მხოლობითი, მრავლობითი, მხოლოდ მხოლობითი, მხოლოდ მრავლობითი გადავნომროთ 1-დან 4-მდე. გარდამავლობა აღვნიშნოთ transitivity- ით, ხოლო მისი მნიშვნელობები: გარდამავალი და გარდაუვალი ' + ' - ით და ' -ით. პირი აღვნიშნოთ Person-ით და მისი მნიშვნელობები გადავნომროთ 1-დან 3-მდე. პირიანობა აღვნიშნოთ Personality-ით და მისი მნიშვნელობები: ერთპირიანი, ორპირიანი, სამპირიანი და ერთ ან ორპირიანი გადავ-ნომროთ რიცხვებით 1-დან 4-მდე. მწკრივი აღვნიშნოთ series-ით და მისი მნიშვნელობები გადავნომროთ 1-დან 11-მდე.

ამის შემდეგ (4) წესი გადაიწერება ასე:

$$\begin{array}{l} VPr_{[voice=1,\ personality=2,\ person=3,\ series<7, number=\alpha]} \rightarrow \\ NPS_{[case=1,\ number=\alpha]} \quad VP_{[voice=1,\ personalyty=2,\ person=3,\ series<7,\ number=\alpha]} \\ NPOD_{[case=3, number=\beta]} \end{array} \qquad (17)$$

ამ წესში number=α მოითხოვს, რომ NPS-სა და VP-ს ჰქონდეს ერთიდაიგივე გრამატიკული რიცხვი, სადაც α არის ცვლადი, რომელმაც შეიძლება მიიღოს მნიშვნელობები 1 ან 2, ხოლო number=β არაა შეზღუდული რაიმე სხვა პირობით, რაც იმას ნიშნავს, რომ NPOD შეიძლება იყოს ნებისმიერ რიცხვში. α და β თვისებათა ცვლადების შემოტანამ შესაძლებლობა მოგვცა 4 წესი ჩაგვეწერა ერთი წესით. ანალოგიურად შეიცვლება (14)—(16) წესებიც.

(17) წესში თუ ზმნას გადავიყვანთ მეორე სერიაში, ე. ი. მოვითხოვთ, რომ თუ series<9Üseries>6, მაშინ NPS-ში

შეიცვლება მხოლოდ case=2 და NPOD-ში case=1 და მივიღებთ ახალ წესს, რომელიც აღვნიშნოთ (18)-ით. ასევე მესამე სერიისათვის ზმნაში შეიცვლება series>8, NPS-ში—case=3 და NPOD-ში—case=1. ამ ცვლილებებით მიღებული წესები აღვნიშნოთ (19)- ით. (18) და (19) წესებს შეესაბამება წინადადებები:

პეტრემ გაართო თამარი (20)

და

პეტრეს გაურთვია თამარი (21)

აქვე უნდა შევნიშნოთ, რომ PN-ის შემთხვევაში β გვაძლევს მხოლოდ ერთ მნიშვნელობას მხოლოდ მხოლობითს.

ახლა განვიხილოთ შემთხვევები, როცა ზმნა მოცემულია პირველ ან მეორე პირში. ამ შემთხვევაში მოგვიხდება გადავაკეთოთ (17)-(19) წესები. მაგალითად, (17) წესში person=3 უნდა შეიცვალოს person=1Ú2 ეს კი გამოიწვევს NPS ჯგუფის კორექტირებას. NPS არის არსებით სახელიანი ფრაზა სუბიექტის როლში, სადაც არსებითი სახელის მონაწილეობა სავარაუდოა დიდი ალბათობით (17) წესში და ჩვენს შემთხვევაში იგი გამორიცხულია. ასევე სავარაუდოა დიდი ალბათობით, რომ NPS საერთოდ არ გვქონდეს. ე.ი. NPS-დან გამოიყვანებოდეს ცარიელი სტრიქონი, როცა person=1Ú2.

ამგვარად (17) წესები უნდა გადაკეთდეს შემდეგნაირად:

$$VPr_{[volce=1,personality=2,\ person=\delta,\ series<7,number=\alpha]} \ \textcircled{R} \ NPS1_{[case=1,\ number=\alpha]} \ VP_{[voice=1,\ personalyty=2,\ person=\delta,\ series<7,\ number=\alpha]} \ NPOD_{[case=3,number=\beta]\ \delta\in\{1,2\}} \quad (22)$$

ანალოგიურად გადაკეთდება (18) და (19) წესები. ოღონდ NPS1-ში case =1 და მიღებული წესები აღვნიშნოთ

(23) და (24) შესაბამისად. აქ NPS1-ს ექნება განსხვავებული გამოყვანის წესები, რომლებსაც ჩვენ შემდგომში დავაზუსტებთ. თუ გავითვალისწინებთ წევრების თავისუფალ რიგს, მაშინ ეს წესები მოგვცემენ 18 განსხვავებულ წესს.

საჭიროა ამ წესების შემდგომი დაზუსტება, რადგანაც ამ წესებით შეიძლება მივიღოთ წინადადებები: მე ვაშენებ სახლს და მე ვიშენებ სახლს, რომელთაც ექნებათ ერთნაირი სინტაქსური სტრუქტურა, მაგრამ შინაარსობრივად ეს წინადადებები განსხვავებულია და ამიტომ მათ უნდა ჰქონდეთ განსხვავებული სინტაქსური სტრუქტურა, რომელზე დაყრდნობითაც შემდგომში ჩვენ მივიღებთ განსხვავებულ სემანტიკურ სტრუქტურას. ამას მოითხოვს მონტეგიუს მიდგომა. ამ წინადადებებს რომ ჰქონდეს განსხვავებული სინტაქსური სტრუქტურა, ჩვენ უნდა განვასხვავოთ ზმნა-ფრაზაში ზმნის სათავისო და საარვისო ქცევები. ე. ი. უნდა დავუმატოთ ზმნა-ფრაზას ახალი თვისება ქცევა, რომელიც აღვნიშნოთ version-ით, ხოლო მისი მნიშვნელობები: საარვისო — ნულით, სათავისო — ერთით და სასხვისო — ორით. ამგვარად, VPr-სა და VP-ში თუ ჩავუმატებთ version=0 ან version=1 ჩვენი წესების რაოდენობა გაორკეცდება და გვექნება სულ 36 წესი. მაგალითად, (22) მოგვცემს ორ ახალ წესს:

$$\underset{\text{[voice=1,personality=2, version=}\gamma\text{, person=}\delta\text{, series<7, number=}\alpha\text{]}}{\text{VPr}} \rightarrow \quad (25)$$

$$\underset{\text{[case=1, number=}\alpha\text{]}}{\text{NPS1}} \quad \underset{\text{[voice=1, personality=2, version=}\gamma\text{ ,person=}\delta\text{, series<7, number=}\alpha\text{]}}{\text{VP}}$$

$$\underset{\text{[case=3, number=}\beta\text{]}}{\text{NPOD}}$$

სადაც $\gamma \in \{0,1\}, \delta \in \{1,2\}$

ჩვენ აქამდე განვიხილეთ VPr, სადაც იგულისხმებოდა

სუბიექტური წყობა. ობიექტური წყობის VPr-სათვის მთავარ წევრებს შორის დამოკიდებულება შეიცვლება და მათთვის მივიღებთ ახალ წესებს. ამ წესების მისაღებად უნდა გავითვალისწინოთ სუბიექტური პირის ნიშნები. აქამდე განხილული თვისება person სინამდვილეში არის თვისება სუბიექტური პირი და მას დაუპირისპირდება თვისება ობიექტური პირი, რომელიც აღვნიშნოთ personob-ით. ზმნის ჯგუფის თვისება number ყოველთვის გვიჩვენებდა აქამდე რეალური სუბიექტის რიცხვს, ე. ი. სუბიექტური წყობის დროს სუბიექტის რიცხვს და იგი ახლა აღვნიშნოთ numbers - ით, ხოლო NPOD -ის თვისება number გვიჩვენებდა ობიექტის რიცხვს და იგი აღვნიშნოთ numberob -ით. VPr-ში numberob არ ფიგურირებდა , რადგანაც იგი არ იყო განსაზღვრული სუბიექტური წყობის დროს და ამიტომ NPOD- ის numberob შეიძლება ყოფილიყო მხოლობითი ან მრავლობითი. ასეთ შემთხვევაში numberob-ს არ მოვიხსენიებთ VPr-ში. ამის შემდეგ, მაგალითად, (25) წესი გადაიწერება ასე:

$$
\begin{aligned}
&\text{VPr}_{[\text{voice}=1,\text{personality}=2,\text{version}=\gamma,\ \text{persons}=\delta,\ \text{series}<7,\ \text{numbers}=\alpha]} \rightarrow \\
&\quad \text{NPS}_{[\text{case}=3,\ \text{persons}=\delta,\ \text{numbers}=\alpha]} \\
&\qquad \text{VP}_{[\text{voice}=1,\ \text{personality}=2,\ \text{version}=\gamma,\ \text{persons}=\delta,\ \text{series}<7,\ \text{numbers}=\alpha]} \\
&\quad \text{NPOD}_{[\text{case}=3,\ \text{personob}=\delta 1,\ \text{numberob}=\beta]}
\end{aligned}
$$

სადაც $\gamma,\alpha,\beta \in \{0,1\}$, $\delta,\delta_1 \in \{1,2,3\}$ (26)

იმისათვის, რომ ეს წესი განზოგადდეს და გამოდგეს, როგორც სუბიექტური წყობის ზმნის ფორმებისათვის, ასევე ობიექტური წყობის ზმნის ფორმებისათვისაც, საჭიროა ცხადად მოვიხსენიოთ VPr-ში, როგორც სუბიექტის პირი და რიცხვი, ასევე ობიექტის პირი და რიცხვიც.

ამგვარად, (26) გადაიწერება ასე:

VPr [voice=1,personality=2,version=γ,persons=α, numbers=β, personob=α1,numberob=β1, series<7]

→

NPS [case=1, persons=α, numbers=β] VP [voice=1, personality=2, version=γ, persons=α, numbers=β, personob=α1, numberob=β1, series<7]

NPOD [case=3, personob=α1, numberob=β1]

სადაც αბα1∈ϭ1ბ2ბ3ℨბ βββ1ბγ∈ϭ0ბ1ℨ λφ α≠α1⊥ σε α≠3. (27)

(27) წესით ჩვენ წარმოვადგინეთ 336 განსხვავებული წესი.

ამავე დროს, თუ გავითვალისწინებთ (27)-ის მარჯვენა მხარეში შემავალი წევრების თავისუფალ რიგს, რაც მოგვცემს კიდევ 5 ახალ წესს, სულ გვექნება 2016 წესი. ანალოგიურად მეორე სერიის ზმნის ფორმირებისათვის გვექნება:

VPr [voice=1,personality=2,version=g,persons=α,numbers=β, personob=α1,numberob=β1, series>6Ůseries<9]

→

NPS [case=2,persons=α,numbers=β]

VP [voice=1,personality=2,version=γ,persons=α,numbers=β, personob=α1,numberob=β1, series>6Λseries<9]

NPOD (28)
[case=1,personob=α1,numberob=β1]
სადაც $\alpha,\alpha_1\in\{1,2,3\}$, $\beta,\beta_1\gamma\in\{0,1\}$ და $\alpha=\alpha_1\perp$ $\sigma\varepsilon$ $\alpha\neq3$.

რაც შეეხება მე-3 სერიის ზმნის ფორმებს აქ ჩვენ უნდა გავითვალისწინოთ რომ მესამე სერიაში არა გვაქვს ქცევის გაგება ე.ი. version=0 ყოველთვის და გვაქვს მხოლოდ ობიექტური წყობის მესამე პირი ე.ი. personob=3 ყოველთვის. შევნიშნოთ აქვე, რომ თუ personob=3, მაშინ numberob-ის მნიშვნელობა არის ნებისმიერი (1 ან 2). ამგვარად, გვექნება:

VPr
[voice=1, personality=2, version=0, persons=a, numbers=b, series>8, personob=3] →
NPS
[case=3, persons=α, numbers=β]
VP
[voice=1, personality=2, version=0, personos=α, numbers=β, personob=3,series>8]
NPOD
[case=1, personob=3]
$\alpha\in\{1,2,3\}$, $\beta\in\{1,2\}$ (29)

სამპირიანი ზმნის შემთხვევაში (29) წესის ანალოგი არ გვექნება და (27)-სა და (28)-ს უნდა დაემატოს ირიბი ობიექტი მიცემით ბრუნვაში და პირი არის ყოველთვის მესამე. ამგვარად,

VPr
[voice=1, personality=3, version=2, persons=a, numbers=b, personob=α1, numbers= β1, →
series<7]
NPS
[case=1, persons=α, numbe ı=β]
VP
[voice=1, personality=3, version=2, persons=α, numbers=β, personob=α1,

numbers= β1

series<7]

NPOD [case=3, personob=α1, numberob=β1] NPOID [case=3]

სადაც, $\alpha,\alpha_1 \in \{1,2,3\}, \beta,\beta_1 \in \{1,2\}$ (30)

VPr [voice=1, personality=3, version=2, persons=α numbers=β, personob=α1, numbers= β1,

→

series>6 Ù series<9]

NPS [case=2, persons=α, numbers=β]

VP [voice =1, personality=3, version=2, persons=α , numbers=β, personob=α1, numbers= β1,

series>6 Ù series<9] NPOD [case=1, personob=α1, numberob=β1] NPOID [case=3]

სადაც, $\alpha,\alpha_1 \in \{1,2,3\}, \beta,\beta_1 \in \{1,2\}$ (31)

ვნებითი გვარის ზმნის ფორმები გვაძლევენ შემდეგი სტრუქტურის წინადადებებს:

VPr [voice=2, personality=1, version=0, persons=α, numbers=β] →

NPS [case=1, persons=α, numbers=β] VP [voice=2, personality=1, version=0, persons=α,

numbers=β]

სადაც, $\alpha \in \{1,2,3\}, \beta \in \{1,2\}$ (32)

VPr [voice=2, personality=2, version=1, numbers=b, persons=α, personob=α1, numberob=β1]

NPS
[case=1, persons=α, numbers=β]
VP
[voice=2, personality=2, version=1, numbers=β, persons=α, personob=α1, numberob=β1]
NPOID
[case=3, personob=α1, numberob=β1] (33)

ერთპირიანი ზმნებისათვის გვარის მიხედვით სხვადასხვა სტრუქტურა გვაქვს. კერძოდ, მედიოაქტივისათვის გვექნება შემდეგი წესები:

VPr
[voice=3, personality=1, persons=α, numbers=β, series<7] →
NP
[case=1, persons=α, numbers=β]
VP
[voice=3, personality=1, persons=α, numbers=β, series<7] (34)

VPr
[voice=3, personality=1, persons=α, numbers=β, 6<series<9] →
NP
[case=2, persons=α, numbers=β]
VP
[voice=3, personality=1, persons=α, numbers=β, 6<series<9] . (35)

VPr
[voice=3, personality=1, persons=α, numbers=β, series>8] →
NP
[case=3, persons=α, numbers=β]
VP
[voice=3,personality=1,persons=α,numbers=β,series>8] (36)

ორპირიანი მედიოაქტიური ზმნებისათვის გვაქვს მხოლოდ I სერიის ფორმები:

VPr
[voice=3personality=2, persons=α, numbers=β, personob=α1, numberob=β1 series<7] →
NP
[case=1, persons=3, numbers=β]

VP
[voice=3,personality=2, persons=3, numbers=β, personob=α1, numberob=b1 , series<7]

NP
[case=3,personob=α1,numberob=β1] (37)

ანალოგიურად გვაქვს ერთპირიანი მედიოპასივის ფორმებისათვის:

VPr
[voice=4, personalyty=1,persons=α,numbers=β, ნებისმიერი მნკრიეი] →
NP
[case=1, persons=α, numbers=β]
VP
[voice=4, personalyty=1,persons=α,numbers=β, ნებისმიერი მნკრიეი] (38)

ორპირიანი ზმნებისათვის გვაქვს:

VPr
[voice=4, personalyty=2,persons=α, numbers=β, personob=α1,numberob=β1]
→
NP
[case=3,persons=α, numbers=β]
VP
[voice=4, personalyty=2,persons=α, numbers=β, personob=α1,numberob=β1]
NP
[case=1,personob=α1,numberob=β1] (39)

უპირო ზმნებისათვის გვაქვს შემდეგი სტრუქტურა:

VPr
[personality=0, persons=3, numbers=1] →
VP
[personality=0, persons=3, numbers=1] (40)

4.5.2 **ქართული წინადადების სინტაქსური ანალიზის პროგრამის აღწერა.** წინა პარაგრაფში აღწერილ მეთოდს ახორციელებს TURBO PROLOG–ზე დაწერილი ქვემოთ წარმოდგენილი პროგრამა. რადგანაც, სიტყვათა მორფოლოგიური ანალიზის პროგრამა არაა უშუალოდ

მიერთებული ამ პროგრამასთან, ამიტომ საჭიროა სიტყვათა მორფოლოგიური სტრუქტურის შესახებ ინფორმაცია მიეწოდოს ამ პროგრამას, რომ მან იმუშაოს სწორად. ამ ინფორმაციის მიწოდება ხდება მონაცემთა ბაზიდან, რისთვისაც ასეთი ბაზა სპეციალურად იქმნება პროგრამის მიერ. იმისათვის, რომ კონკრეტული წინადადებისათვის პროგრამამ მიიღოს შესატყვისი გამოყვანის ხე, საჭიროა წინასწარ მომზადდეს ბაზაში წინადადების ყოველი სიტყვისათვის მორფოლოგიური ინფორმაცია. ამისათვის, პროგრამას გააჩნია ბაზასთან მუშაობის საშუალებები, როგორიცაა ბაზის შექმნა, ინფორმაციის დამატება ბაზაში, ინფორმაციის ამოგდება ბაზიდან და ბაზის წაშლა. ასევე შესაძლებელია საჭირო ინფორმაცია წინასწარ მომზადდეს ფაილში და შემდეგ იგი ჩაიტვირთოს ბაზაში. ყველა მოქმედება სრულდება მენიუს საშუალებით. მენიუს გააჩნია შემდეგი ელემენტები: რაიმე წინადადების სინტაქსური ანალიზი, ბაზის ჩატვირთვა ფაილიდან, ბაზის წაშლა, ენის დათვალიერება განახლება, ბაზის შენახვა ფაილში, ბაზის ფაილის რედაქტირება, მოკლე აღწერა, ოპერაციულ სისტემაში გასვლა და პროგრამის დამთავრება. როცა ავირჩევთ სინტაქსურ ანალიზს, მაშინ პროგრამა ითხოვს გასაანალიზებელი წინადადების შეტანას და პასუხად გვაძლევს ამ წინადადების სინტაქსური ხის ფრჩხილებში ჩაწერას (ინვერსიულ პოლონურ ჩაწერას) და როცა გამოყვანის ხე ეტევა ეკრანზე, მაშინ გვაძლევს ამ ხის გრაფიკულ წარმოდგენასაც. პროგრამის მიზანია აჩვენოს წინა პარაგრაფში აღწერილი მეთოდის რეალიზაცია და არა ქართული წინადადების სრული სინტაქსური ანალიზი. სრული სინტაქსური ანალიზისათვის საჭიროა სინტაქსური ანალიზის წესები იყოს სრული. აღწერილი მიდგომისათვის ახალი წესის დამატება ან არსებულის მოდიფიკაცია არ წარმოადგენს სიძნელეს, რადგან ასეთ შემთხვევაში იცვლება კონკრეტული წესი და იგი არ გამოიწვევს პროგრამის შეცვლას, არამედ შეიცვლება სათანადო წესის წარმოდგენა პრედიკატის სახით. რაც შეეხება გარჩევის ხის

გრაფიკულ წარმოდგენას, იგი არა აუცილებელი იყოს პროგრამაში, რადგან იგი წარმოდგენილია გამოყვანის ხის საილუსტრაციოდ.

4.6 ბუნებრივი ენის ფრაზების ავტომატური სემანტიკური ანალიზისადმი ერთი მიდგომის შესახებ. მეცნიერული კვლევები, ერთი ენიდან მეორეზე თარგმანი, სწავლება, სახეთა გამოცნობა და მათთან დაკავშირებული სხვა პრობლემების გადაწყვეტა დღეისათვის ადამიანთა კომპეტენციას წარმოადგენს. მიუხედავად ამისა, ადამიანები მათ გადაწყვეტაში სულ უფრო და უფრო ხშირად იყენებენ კომპიუტერს და ამ მიმართულებით დღეისათვის კომპიუტერის შესაძლებლობების მცირე ნაწილია გამოყენებული. საყოველთაოდ ცნობილია, ბუნებრივი ენის როლი ამ საკითხების გადაწყვეტაში. კერძოდ, ბუნებრივი ენის ტექსტიდან შინაარსის ამოღებისა და მისი ფორმალიზაციის საკითხების გადაწყვეტა. იგულისხმება, რომ ეს პრობლემა გადაწყვეტადია და იგი ბუნებრივი ენის სემანტიკური ანალიზის სახელითაა ცნობილი. ქართული ტექსტების კომპიუტერული სემანტიკური ანალიზის მიმართულებით ცოტა რამაა გაკეთებული [52-57]. სემანტიკური ანალიზის სირთულე გამოწვეულია წინადადებათა თავისებურებით. განვიხილოთ ზოგიერთი მათგანი: 1. წინადადება შეიძლება იყოს ჭეშმარიტი ან მცდარი. ოთხკუთხედის შიგა კუთხების ჯამი უდრის 360-ს, რომლის ჭეშმარიტება ან მცდარობა შეიძლება დამტკიცდეს ლოგიკურად, მაგრამ წინადადება, რომელიც აღწერს რაიმე ისტორიულ მოვლენას მისი ლოგიკურად დამტკიცება შეუძლებელია, იგი უნდა იყოს მიღებული ისტორიკოსების მიერ როგორც ფაქტი ან უარყოფილი. 2. წინადადება შეიძლება იყოს უნივერსალური ან ტიპიური. მაგალითად, ყველა ადამიანი

მოკვდაეია, უნივერსალურია, ან ყველა მამაკაცს უყვარს ალკოჰოლი – იგი არაა უნივერსალური, მაგრამ ტიპიურია. 3. წინადადება შეიძლება იყოს სარწმუნო ან დაუჯერებელი. 4. წინადადება შეიძლება იყოს იშვიათი ან სარწმუნო მიახლოებით. 5. წინადადება შეიძლება იყოს ომონიმიური თავისი მნიშვნელობით. ასეთი მაგალითების ჩამოთვლა შეიძლება გაგრძელდეს[58], მაგრამ ჩამოთვლილი მაგალითებიც საკმარისია იმისათვის, რომ დავრწმუნდეთ თუ რა რთული ბუნება აქვს წინადადების შინაარსს და რომ იგი არ შეიძლება დახასიათდეს მხოლოდ მისი ჭეშმარიტებით ან მცდარობით. ზოგიერთი წინადადებისათვის აზრი არა აქვს მისი ჭეშმარიტების ან მცდარობის განხილვას. აქედან გამოდის, რომ წინადადების შინაარსის დასადგენად უნდა განხილულ იქნეს მისი შემადგენელი ნაწილები – სიტყვები. უნდა შესწავლილ იქნეს თვითეული სიტყვის შესაძლო მნიშვნელობები, და ის, თუ რა გავლენას ახდენს წინადადების სიტყვები ერთიმეორეზე, რომ მივიღოთ წინადადების შინაარსი.

ასევე აუცილებელია იმ რეალური სამყაროს ცოდნა, რომელშიც ხდება ბუნებრივი ენის გამოყენება. რეალური სამყაროს აღწერა ხდება ცოდნის ბაზების საშუალებით, სადაც რეალური ან შესაძლო სამყაროს წარმოდგენისათვის გამოიყენება სემანტიკური ქსელები. სემანტიკური ქსელების კლასიფიკაცია მათი გამოყენების მიხედვით მოცემულია [59]-ში. განსაკუთრებით ფართო გამოყენება ჰპოვა Schank [60]–ისა და Shapiro-ს [61] გრაფიკულმა წარმოდგენებმა, რომლებიც კონცეპტუა-ლური გრაფების სახელწოდებით არის ცნობილი ლიტერატურაში. ასევე ფართოდ გამოიყენება საუბრის შინაარსის წარმოსადგენად Shmidt-ის გრაფიკული წარმოდგენა, რომელიც

ცნობილია DDRS სახელწოდებით [62]. ჩვენს მიზანს წარმოადგენს განვსაზღვროთ წინადადების ცალკეული სიტყვების სემანტიკური მნიშვნელობა და წინადადების სინტაქსურ სტრუქტურაზე დაყრდნობით განვსაზღვროთ მთელი წინადადების სემანტიკური მნიშვნელობა. ამისათვის საჭიროა გვქონდეს ცალკეული სიტყვების სემანტიკური დახასიათება მათი სემანტიკური თვისებების ჩამოთვლითა და ცალკეული თვისებების მნიშვნელობების მოცემით. ამავე დროს უნდა იყოს გათვალისწინებული სამყაროსეული ცოდნა, რომელიც ჩადებულია ამ წინადადების სემანტიკურ მნიშვნელობაში. ამის შემდეგ წინადადების სინტაქსურად დაკავშირებულ სიტყვებს შეიძლება დაედოს სემანტიკური შეზღუდვები, რომლებიც გამოიხატება სიტყვათა სემანტიკური თვისებებითა და მათი მნიშვნელობით. მათი საშუალებით შეიძლება დადგინდეს სწორი სემანტიკური დაკავშირება სინტაქსურად დაკავშირებულ სიტყვებს შორის. იგივე შეიძლება ითქვას უფრო რთული სინტაქსური კავშირებისათვის როგორიცაა სახელთა ჯგუფებსა და ზმნის ჯგუფებს შორის სინტაქსური კავშირი. ასევე შეზღუდვები თავის მხრივ წარმოადგენენ ლოგიკურ გამოსახულებებს აგებულს სემანტიკურ თვისებებზე და მათ მნიშვნელობებზე. ამის შემდეგ შეიძლება იგივე ფორმალიზმი იყოს გამოყენებული წინადადების სემანტიკური ანალიზისას, რაც იყო ჩვენს მიერ გამოყენებული ქართული წინადადების მორფოლოგიური და სინტაქსური ანალიზისათვის[63,64]. ამისათვის მნიშვნელოვანია ცალკეული მეტყველების ნაწილებისათვის სემანტიკური თვისებების სწორად დადგენა, რისთვისაც ჩვენ ვიყენებთ მიკროკოსმოსისა და მიკროთეორიის ცნებებს, რომლებიც

გამოყენებულია ბუნებრივი ენის ტექსტის გადასაყვანად TMR ენაზე[65]. ამ თეორიის მიხედვით მიკროკოსმოსი შედგება მიკროთეორი-ებისაგან. მიკროთეორიას ადგენენ ცალკეული მეტყველების ნაწილებისათვის. უფრო ნათლად რომ წარმოვიდგინოთ თუ როგორ დგება ასეთი მიკროთეორია, განვიხილოთ იგი ზედსართავი სახელისათვის. იგი ეყრდნობა უპირველეს ყოვლისა ზედსართავების სინტაქსსა და სემანტიკას, რაც ფართოდაა მიღებული ლიტერატურაში. იგი ეყრდნობა შემდეგ პრინციპებს: 1. ატრიბუტული ზედსართავები აზუსტებენ არსებით სახელებს, რომლებთანაც ისინი არიან დაკავშირებული. 2. ყველა ზედსართავი არ აზუსტებს არსებით სახელებს განსაკუთრებით როცა ისინი გამოყენებულია პრედიკატულად. ზედსართავის ატრიბუტული და პრედიკატული გამოყენება არის ერთერთი იმ ხუთ თვისებათაგან, რომლებითაც დაკავშირებულია ზედსართავებთან ატრიბუტულობა | პრედკატულობა; მოდიფიცირება ზმნისზედებით; სტატიკურობა | დინამიურობა; ხარისხის ქონა | არ ქონა; მემკვიდრეობითობა | არამემკვიდრეობითობა. 3. ზედსართავების ტაქსონომია. 4. პრედიკატული | არაპრედიკატული ზედსართავები. 5. რელაციური ზედსართავები. 6. რელაციური | ხარისხობრივი ზედსართავები. 7. ზედსართავების რიგი. 8. ხარისხობრივი ზედსართავების შედარების ხარისხები და სკალები და სხვა. გარდა ამისა, მნიშვნელოვანია ზედსართავების სემანტიკა დაფუძნებული ონთოლოგიასა და ლექსოკოლოგიაზე. ონთოლოგიური მიდგომა გულისხმობს ზემოთ აღწერილი კლასიფიკაციის მიხედვით ყოველი სიტყვის დახასიათებას სათანადო თვისებებისა და მათი მნიშვნელობების

მოცემით [66]. ჩვენს შემთხვევაში თვისებები და შესატყვისი მნიშვნელობები მიეწერება სიტყვას ლექსიკონში. ასეთი თვისებები და მათი მნიშვნელობები გამოიყენება შეზღუდვების შესადგენად, რომლითაც დაზუსტდება სიტყვისა თუ მთელი სინტაქსური კონსტრუქციის სემანტიკური მნიშვნელობა. ანალოგიურად უნდა აიგოს მიკროთეორია მეტყველების სხვა ნაწილებისათვის (არსებითი სახელისათვის, ზმნებისათვის და სხვა მეტყველების ნაწილებისათვის). ასეთი წარმოდგენა საშუალებას გვაძლევს ჩვენს მიერ შექმნილი ინსტრუმენტული საშუალებები გამოვიყენოთ სემანტიკური ანალიზისათვის ამ ინსტრუმენტების ყოველგვარი მოდიფიკაციის გარეშე. რაც საშუალებას მისცემს სემანტიკური ანალიზის ალგორითმის შემდგენელს გამოიყენოს ჩვენს მიერ შექმნილი ინსტრუმენტული საშუალებები თავისი ალგორითმის შესამოწმებლად და კორექტირებისათვის ალგორითმის დაპროგრამების გარეშე[63,64].

4.7 განზოგადებული ბუნებრივენოვანი სისტემები

4.7.1 შესავალი. ამ პარაგრაფში განხილულია განზოგადებული ბუნებრივენოვანი სისტემის აგების კონკრეტული მიდგომა. აღწერილია: ცნობილი ბუნებრივენოვანი სისტემები, გაფართოებული გადასვლათა ქსელური გრამატიკები, ცოდნის ბაზის კონკრეტული სტრუქტურა და მათ საფუძველზე შემუშავებულია ქართულენოვანი მასწავლებელი სისტემის აგების კონკრეტული მიდგომა და მოცემულია ასეთი სისტემის სქემა.

ამ სქემაში ძირითადი ნაწილებია ტექსტის გარჩევის მექანიზმი და ლოგიკური გამოყვანის მექანიზმი. ტექსტის გარჩევის მექანიზმს საფუძვლად დაედება გადასვლათა ქსელური გრამატიკა, რომელიც უნდა მოერგოს ამ

კონკრეტულ შემთხვევას. რაც შეეხება ლოგიკური გამოყვანის მექანიზმს აქ შეიძლება გამოყენებულ იქნეს პროლოგის გამოყვანის მექანიზმი, მაგრამ იგი მოდიფიცირებული უნდა იყოს, რომ შეეძლოს კონცეპტუალური გრაფების გამოყენება. ცოდნის ბაზის რეალიზაციისათვის გამოდგება ნებისმიერი მონაცემთა ბაზების აგების სისტემა ან შეიძლება აიგოს სპეციალური სისტემა კონკრეტულად აღწერილ ბაზაზე მუშაობისათვის. მე-2 პარაგრაფში აღწერილია გადასვლათა ქსელური გრამატიკები და მათი შესაძლებლობები. მე-3 პარაგრაფში გადმოცემულია კონკრეტული მიდგომა მასწავლებელი სისტემის აგებისადმი და შედგენილია სათანადო სქემა. მე-4 პარაგრაფში აღწერილია ცოდნის ბაზის სტრუქტურა, რომელიც შეესაბამება წამოსადგენ ცოდნას და მის კონკრეტულ გამოყენებას.

4.7.2 **გადასვლათა ქსელური გრამატიკები.** ენის მოდელირებისათვის სასურველია, რომ ამ ენის გრამატიკა იყოს აგებული სანიმუშო წინადადებების სიმრავლეზე დაყრდნობით. გრამიტიკის აგებას სანიმუშო წინადადებების სიმრავლეზე დაყრდნობით ეწოდება გრამატიკული გამოყვანა. გრამატიკულ გამოყვანას ფართო გამოყენება აქვს ინფორმაციის დამუშავებაში, ხელოვნურ ინტელექტში და ერთი ენის მეორეში გადაყვანის დროს, რადგანაც არ არსებობს ურთიერთ ცალსახა დამოკიდებულება ენასა და გრამატიკას შორის სასურველია გრამატიკათა ექვივალენტობა განისაზღვროს მათ მიერ წარმოქმნილ ენათა ექვივალენტობით. ორ გრამატიკას ეწოდება ექვივალენტური თუ ისინი განსაზღვრავენ ერთი და იგივე ენას.

გამოყვანის პროცესში უნდა იყოს მოცემული იმ წინადადებათა სიმრავლე, რომლებიც უნდა ეკუთვნოდეს ენას. ამ სიმრავლეს ეწოდება ენის დადებითი ნიმუში. წინადადებათა იმ სიმრავლეს, რომლებიც არ უნდა ეკუთვნოდეს ენას, ეწოდება ენის უარყოფითი ნიმუში. L(G) ენის დადებით ნიმუშს ეწოდება სტრუქტურულად სრული,

თუ G გრამატიკის ყველა გადაწერის წესი გამოიყენება დადებითი ნიმუშის არაცარიელი ქვესიმრავლის გენერაციისათვის. გრამატიკული გამოყვანისას იგულისხმება, რომ

1. გამოსაყვანი გრამატიკის წესი მოცემულია;
2. ენის მოცემული ნიმუში სასრულია;
3. ენის დადებითი ნიმუში სტრუქტურულად სრულია;
4. გამოყვანილი G გრამატიკა არის ისეთი, რომ $S^+ \in L(G)$ და $S^- \notin L(G)$, სადაც S^+ და S^- ენის დადებითი და უარყოფითი ნიმუშებია შესაბამისად და ``L(G) არის L(G) -ს დამატება.

გადასვლათა ქსელური გრამატიკები შექმნილია ბუნებრივი ენის ანალიზის მოდელირებისათვის [67-70].

რაიმე ძირითადი გადასვლათა ქსელი (BTN) არის პირდაპირი გრაფი, მონიშნული მდგომარეობებითა და რკალებით, გამოყოფილი მდგომარეობებით, რასაც ეწოდება საწყისი მდგომარეობა და გამოყოფილი საბოლოო მდგომარეობათა სიმრავლით, რომელსაც ეწოდება საბოლოო მნიშვნელობათა სიმრავლე. იგი არსებითად გამოიყურება, როგორც არადეტერმინისტული სასრულ მდგომარეობიანი გადასვლათა დიაგრამა, იმ განსხვავებით რომ, რკალებზე მოთავსებული ჭდეები შეიძლება იყოს როგორც ტერმინალური სიმბოლოები, ასევე მდგომარეობათა სახელები.

რკალზე მოთავსებული მდგომარეობის სახელი ნიშნავს, რომ რკალის ბოლოში არსებული მდგომარეობის სახელი შეინახება სტეკში და მართვა გადაეცემა რკალზე მოთავსებული მდგომარეობის სახელიან მდგომარეობას. როცა საბოლოო მდგომარეობა მიიღწევა, სტეკიდან ამოიღება სახელი და ამ სახელიან მდგომარეობას გადაეცემა მართვა. თუ სტეკი ცარიელი აღმოჩნდა მდგომარეობის სახელის ამოღებისას და შესატანი სტრიქონი დამთავრდა, ეს იმას

ნიშნავს, რომ შესასვლელ სტრიქონს უშვებს გადასვლათა ქსელი.

BTN შეიძლება განვიხილოთ როგორც მანქანა მაღაზიური მეხსიერებით, რომელიც შედგება სასრულ მდგომარეობიანი მანქანების სასრული რაოდენობისაგან და მაღაზიური მეხსიერებისაგან. ფორმალურად BTN შეიძლება განისაზღვროს როგორც შვიდეული:

TN =(å, Q, A, Q_0, Q_f, q_0, Z_0),

სადაც å არის შესასვლელი სიმბოლოების სასრული სიმრავლე, Q არის მდგომარეობის სასრული სიმრავლე, Q_f არის სასრულ მდგომარეობიან მანქანების საბოლოო მდგომარობათა სიმრავლე, $q_0 \hat{I} Q_0$ არის TN საწყისი მდგომარეობა, Z_0 არის მაღაზიის ძირში მოთავსებული სპეციალური სიმბოლო, A არის რკალების სასრული სიმრავლე. ეს რკალები შეიძლება დავყოთ სამ კლასად:

1. CAT რკალი. გადასვლა ხდება მიმდინარე მდგომარეობიდან რკალის ბოლოში მოთავსებულ მდგომარეობაზე, თუ შესასვლელი სიმბოლო ეკუთვნის იმ კლასს, რომლითაც მონიშნულია რკალი. ამავე დროს ხდება ახალი შესასვლელი სიმბოლოს აღება.

2. PUSH რკალი რკალის ბოლოში მოთავსებული მდგომარეობის სახელი ინახება სტეკში და გადასვლა ხდება იმ მდგომარეობაზე, რომლის სახელითაც მონიშნულია ეს რკალი. ეს მდგომარეობა არის რაიმე სასრულ-მდგომარეობიანი მანქანის საწყისი მდგომარეობა.

3. POP რკალი. მიმდინარე მდგომარეობის გადაყვანა ხდება იმ მდგომარეობაში, რომელიც მოთავსებულია სტეკის თავში და ხდება სტეკიდან ამ მდგომარეობის ამოგდება.

შესასვლელ სტრიქონს უშვებს TN , როცა Z_0 ამოვარდება სტეკიდან. TN -ით განსაზღვრული ენა აღვნიშნოთ

L(TN). აღწერილი გადასვლათა ქსელი TN არის განზოგადოებული მაღაზიური ავტომატი და იგი კონტექსტისგან თავისუფალი გრამატიკის ექვივალენტურია.

TN შეიძლება გავხადოთ უფრო ძლიერ მანქანად, თუ განვაზოგადებთ რკალების მონიშვნას. მაგალითად, თუ ყოველ რკალს მოვნიშნავთ რაიმე პირობით, რომელიც უნდა დაკმაყოფილდეს რომ ამ მიმართულებით გაგრძელდეს მუშაობა. აგრეთვე თუ შევასრულებთ რეგისტრების დაყენების მოქმედებებს, როცა ამ რკალის მიმართულებით ვმოძრაობთ.

გაფართოებული ქსელური გრამატიკის (ATN)-ის სიძლიერე დამოკიდებულია იმ საშუალებებზე, რომლებიც დაშვებულია რკალზე პირობების ჩასაწერად. თუ ჩვენ ავიღებთ TN-ს მხოლოდ ერთი ქსელით, რომელიც არის მდგომარეობათა სიმრავლე CAT და POP რკალებით, მაშინ იგი განსაზღვრავს რეგულარულ სიმრავლეებს, რომლებიც როგორც ცნობილია წარმოიდგინება სასრული ავტომატებით ან რაც იგივეა, მარჯვნივწრფივი გრამატიკებით. თუ ავიღებთ BTN-ს, რომელიც არის სასრულმდგომარეობიანი მანქანების სიმრავლე სტეკით, მაშინ იგი უშვებს კონტექსტისგან თავისუფალ ენებს, ხოლო თუ ავიღებთ ATN -ს რეგისტრების დაყენების მოქმედებებითა და რკალებზე ნებისმიერი პირობებით, მაშინ მისმა სიმძლავრემ შეიძლება მიაღწიოს ტიურინგის მანქანის სიმძლავრეს [71] .

4.8 მასწავლებელი სისტემის აგებისადმი ერთი მიდგომის შესახებ. რადგანაც ცოდნის ათვისების პროცესის სრული ავტომატიზაცია ძალზე რთული პრობლემაა და ახლო მომავალში მისი გადაწყვეტა არაა მოსალოდნელი, ამიტომ რეალურად შესაძლებელია ამ პრობლემის გადაწყვეტა ნახევრად ავტომატურად ექსპერტისა და ბუნებრივ ენაზე დიალოგის გზით.

კომპიუტერი აგროვებს ცოდნას ქართული ტექსტის წაკითხვისას. იგი იძლევა echo ინფორმაციას იმის შესახებ, თუ როგორ გაიგო წინადადება. როცა იგი წააწყდება პრობლემებს ტექსტის გაგებაში, იგი სვამს კითხვებს და ექსპერტის დახმარებით დიალოგის რეჟიმში ხდება წამოჭრილი პრობლემების გადაწყვეტა. თუ როგორი წარმატებული იქნება ათვისების პროცესი ერთის მხრივ დამოკიდებულია იმაზე, თუ როგორია ცოდნის წარმოდგენა კომპიუტერში და შესაბამისად ცოდნის წარმოდგენის ენა , შეუძლია მას თუ არა ტექსტში მოცემული ცოდნის ზუსტად წარმოდგენა და მეორეს მხრივ, თუ როგორ ზუსტად არის აღწერილი ცოდნის ბაზაში ცნებები შესაძლო ატრიბუტებით და როგორია მათი იერარქია.

ცოდნის წარმოდგენის ენა უნდა იძლეოდეს ელემენტალური ცნებების საშუალებით უფრო მაღალი რიგის ცნებების გამოსახვის საშუალებას. აგრეთვე, უნდა იყოს ქართული ენის წინადადების აზრის კონცეპტუალური გრაფით გამოსახვით საშუალება და პირიქით. ქართულენოვანი ტექსტიდან ცოდნის მისაღებად მასწავლებელ სისტემას უნდა ჰქონდეს საკმაოდ სრული ცოდნა ქართული ენის შესახებ (ლექსიკური, მორფოლოგიური, სინტაქსრი და სემანტიკური ინფორმაცია). ქართული წინადადების აზრის დასადგენად ჩვენი აზრით მიზანშეწონილია დავეყრდნოთ აზრის კომპოზიციურობის პრინციპს და უნდა გვქონდეს შემუშავებული მექანიზმი იმის გამოსაცნობად როცა ეს პრინციპი არ მუშაობს. ასეთმა მასწავლებელმა სისტემამ უნდა მიმართოს ექსპერტს დახმარებისათვის თუ რა ფორმალიზმი გამოვიყენოთ ქართული წინადადების აზრის დასადგენად ქართული ენის ლინგვისტურ ცოდნაზე დაყრდნობით, ეს ცალკე განხილვის საკითხია.

ცნობილი სისტემების ანალიზისა და საკუთარი გამოცდილების შედეგად ჩვენ მივედით შემდეგ დასკვნამდე [72,73]:

1. სწავლებადი სისტემა უნდა იძლეოდეს echo ინფორ-

მაციას იმის შესახებ, თუ როგორ გაიგო მიწოდებული წინადადება.

2. წინადადების აზრის დადგენის პროცესში წარმოშობილი პრობლემების შემთხვევაში სწავლებადმა სისტემამ უნდა დაუსვას კითხვები ექსპერტს და დიალოგის საშუალებით გადაწყვიტოს წარმოშობილი პრობლემა.

3. სისტემას უნდა ჰქონდეს საკმაოდ სრული ლინგვისტური ცოდნა ბუნებრივი ენის შესახებ და ზოგად ცნებათა მოქნილი იერარქია.

4. ელემენტარული ცნებების წარმოსადგენად მიზანშეწონილია ფრეიმების ტექნიკის გამოყენება.

5. სასურველია წინადადებაში არსებული ცოდნა წარმოდგენილ იქნეს კონცეპტუალურ გრაფებში, რაც საშუალებას იძლევა წინადადებიდან მიღებული ცოდნა პირდაპირ წარმოდგენილ იქნეს კონცეპტუალური გრაფით და პირიქით, კონცეპტუალური გრაფით პირდაპირ შეიძლება წინადადების აგება.

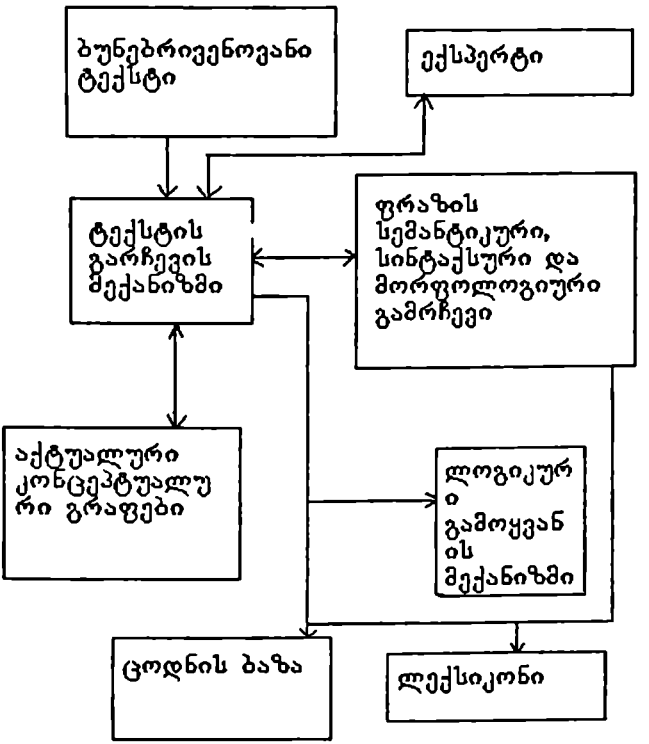
6. სისტემას უნდა გააჩნდეს ძლიერი ლოგიკური ენა, რომელიც საშუალებას მოგვცემს მაღალი რიგის ცნებები გამოისახოს ელემენტარული ცნებების საშუალებით და ბუნებრივი ენის რთული სემანტიკური კონსტრუქციები გამოისახოს ამ ენაზე. ამ ენას უნდა ჰქონდეს ძლიერი გამოყვანის მექანიზმი ლოგიკურ გამოსახულებათა ექვივალენტობის დასადგენად.

7. ფრეიმებითა და კონცეპტუალური გრაფებით უნდა აიგოს შესატყვისი ლოგიკური გამოსახულება და პირიქით, რაიმე ლოგიკური გამოსახულებიდან პირდაპირ აიგოს შესატყვისი ბუნებრივენოვანი წინადადება.

ლოგიკური ენის ძირითადი დანიშნულებაა გამოსახულებათა ექვივალენტობის დადგენა, ურთიერთ საწინააღმდეგო აზრის აღმოჩენა და მაღალი რიგის ცნებების გამოსახვა პრიმიტივების საშუალებით. ლოგიკურ ენად სასურველია აღებულ იქნეს ინტენსიონალური ლოგიკის რაიმე ვარიანტი. თუ რა სისრულით იქნება იგი აღებული დამოკიდებულია მასწავლებელი სისტემისადმი კონკრეტულ

მოთხოვნებზე.

განხილული მოთხოვნების საფუძველზე ნახ. 2-ზე წარმოდგენილია მასწავლებელი სისტემის არქიტექტურა.



ნახ. 2. მასწავლებელი სისტემის არქიტექტურა .

მასზე გამოსახულია ძირითადი ბლოკები და მათ შორის კავშირები. წარმოდგენილი ბლოკებიდან ქართული ენის შემთხვევაში ცალკე განხილვის საგანია: ლოგიკური ენის აგება და შესაბამისად ლოგიკური გამოყვანის მექანიზმი, ფრაზის სემანტიკური, სინტაქსური და მორფოლოგიური

გამრჩევი; ლექსიკონისა და ცოდნის ბაზის შიგთავსი. მათი დეტალური დამუშავების შემდეგ შესაძლებელია ერთიანი ალგორითმის შედგენა და მისი რეალიზაცია.

4.9 **ცოდნის ბაზის სტრუქტურა.** როცა ვგეგმავთ ცოდნის ბაზის შედგენას, პირველ რიგში უნდა გავარკვიოთ როგორი სახის ცოდნა უნდა იყოს წარმოდგენილი ცოდნის ბაზაში და როგორი უნდა იყოს მისი წარმოდგენის ხერხი. ცოდნის ბაზის სტრუქტურა არ უნდა იყოს დამოკიდებული იმაზე, თუ რისთვის ვიყენებთ ბაზაში წარმოდგენილ ცოდნას, მაგრამ იგი საჭირო ცოდნის სწრაფად პოვნის საშუალებას უნდა იძლეოდეს, იოლი უნდა იყოს მისი განახლება და არ უნდა მოითხოვდეს რთულ გარდაქმნებს მისი პრაქტიკული გამოყენებისათვის. ე.ი. უნდა იყოს ადაპტირებული კონკრეტული მეთოდით მისი გამოყენებისათვის. ბუნებრივია ამ მიზნისათვის ცოდნის პროცედურული წარმოდგენა არ გამოდგება, რადგან იგი გამიზნულია კონკრეტულ ვითარებაში გამოყენებისათვის, თუმცა იგივე ცოდნა შეიძლება საჭირო იყოს სხვა შემთხვევაში. ამიტომ ცოდნის ზოგადი გამოყენებისათვის უფრო მოხერხებულია ცოდნის დეკლარაციული წარმოდგენა. რადგან ჩვენს შემთხვევაში ცოდნის ბაზა გამოიყენება ბუნებრივენოვანი სისტემის შესადგენად და ჩვენ ვიყენებთ განზოგადოებულ ქსელურ გადასვლათა გრამატიკას, ბუნებრივია ცოდნა წარმოვადგინოთ ამ გრამატიკის საშუალებით. რადგან მისი გამოყენება გაადვილდება და მეორეს მხრივ ადვილია ასეთი ცოდნის ბაზის მოდიფიკაცია, რადგან მისი ცალკეული ნაწილების მოდიფიკაცია არ მოითხოვს სხვა ნაწილების შეცვლას. ბუნებრივენოვანი სისტემები იყენებენ დეკლარაციულ ცოდნას ტექსტის შინაარსის გასაგებად და პირიქით ბუნებრივენოვანი ტექსტის გენერირებისათვის. ზოგიერთი პროგრამები იყენებენ დეკლარაციული ცოდნის რაიმე ზოგად ინტერპრეტატორს, დაწერილს პროლოგზე. მაგალითად [74,75]. ასეთ შემთხვევაში ცოდნის აღწერის ენაა პრედიკატული აღრიცხვის ენა. ექსპერტისათვის შეიძლება უფრო მოხერხებული იყოს

ყოველი სახეობის ცოდნისათვის ერთი ზოგადი ფორმალიზმის გამოყენება, მაგრამ როცა ცოდნის ბაზა იზრდება, მისი გამოყენება ხდება არაეფექტური, რადგან არ შეიძლება გამოყენებულ იქნეს გარკვეული სახის დამახასიათებელი დეტალები. თუ ექსპერტმა იცის, როგორ მუშაობს ინტერპრეტატორი, მას შეუძლია წარმოადგინოს ცოდნა ისე, რომ იგი ეფექტურად იყოს გამოყენებული. მაგრამ ასეთ შემთხვევაში ცოდნის დეკლარაციული წარმოდგენა ჰკარგავს თავის უპირატესობებს, რადგან იგი იქნება ორიენტირებული კონკრეტულ პროცედურაზე (ინტერპრეტატორზე), რომელიც გამოიყენებს მას. ზოგიერთი რეალიზაციები იყენებენ დეკლარაციული ცოდნის წარმოდგენას, რომელიც მოხერხებულია ბუნებრივი ენის გაგებისათვის და აგებენ შესატყვის ინტერპრეტატორს. ასეთი ინტერპრეტატორი შეიძლება იყოს ეფექტური, რადგან მისი ავტორი შეიძლება კარგად იცნობდეს იმ პრობლემების სახეობებს, რომლებიც უნდა გადაიჭრას ბუნებრივი ენის დამუშავებისას [76], მაგრამ ბუნებრივი ენის წარმოდგენის შემთხვევაში იგი არაეფექტური გახდება, რადგან იგი მოგეზულია მხოლოდ ბუნებრივი ენის გაგებისათვის. ჩვენ წარმოვადგენთ ცოდნის ბაზის სტრუქტურას, რომელიც ერთნაირად გამოიყენება, როგორც ბუნებრივი ენის გაგებისათვის, ასევე წარმოდგენისათვის, მაგრამ იგი მოგეზულია მიდგომაზე, რომელიც აღწერილია (თავი 1-ის §3 -ში). ჩვენ ცოდნას ვყოფთ კლასებად: მორფოლოგიური, სინტაქსური, სემანტიკური და პრაგმატული. ლექსიკოგრაფიული ცოდნა. ცოდნა შედგება შემდეგი ძირითადი ნაწილებისაგან: სიტყვები, დაბოლოებათა მიმდევრობები, უღლებები. ბაზაში მოთავსებული სიტყვები შედგება შემდეგი ნაწილებისაგან:

1.სიტყვების სახელი, რომლებიც გამოიყენება ექსპერტთან დასაკავშირებლად;

2.მისი უღლების (ბრუნების) სახელი;

3.ძირების დალაგებული მიმდევრობა.

დაბოლოებათა მიმდევრობა შეიცავს კონკრეტულ

დაბოლოებათა მიმდევრობის სახელს და დაბოლოებათა დალაგებულ მიმდევრობას.

უღლებები. უღლებებს გააჩნიათ სპეციალური გრაფული წარმოდგენები, რომლებიც აღწერილია თავი 2-ის მე- 3 პარაგრაფში.

სინტაქსური ცოდნა. სინტაქსური ცოდნა შედგება ნაწილებისაგან: ზოგადი სინტაქსური წესისა და მისი ქვენაწილებისაგან. მთლიანად ზოგადი სინტაქსური წესი წარმოადგენს განზოგადოებულ გადასვლათა ქსელს (იხილეთ თავი 3 §2). სინტაქსური წესი აღწერილი გადასვლათა რაიმე ქსელის საშუალებებით აღწერს წინადადებათა შესაძლო სტრუქტურებს და არ წარმოადგენს რაიმე პროცედურას, რომელიც უნდა შესრულდეს, რომ მივიღოთ გარკვეული წინადადება. ე.ი. გადასვლათა ქსელები გამოიყენება წინადადებათა სინტაქსური სტრუქტურის შესახებ ცოდნის წარმოსადგენად. იგი აღწერს კონკრეტული სტრუქტურის წინადადებაში შესაძლო სინტაქსურ ელემენტებს.

სემანტიკური ცოდნა. რაიმე სიტყვის მნიშვნელობა შედგება ორი მთავარი ნაწილისაგან:

1. სანიმუშო ნაწილი, რომელიც გვიჩვენებს ეს მნიშვნელობა შეიძლება იყოს წარმოდგენილი რაიმე ტექსტში თუ არა.

2. აქტიური ნაწილი, რომელიც გვიჩვენებს, რომ ეს მნიშვნელობა უნდა იყოს გააქტიურებული მნიშვნელობის წარმოდგენის შექმნისას.

ამ ორი ნაწილის განსაზღვრებები დაკავშირებულია მათ გამოყენებასთან ანალიზისას. თუ ჩვენ ვაწარმოებთ რაიმე ტექსტს, მაშინ აქტიური ნაწილი გვიჩვენებს მნიშვნელობის წარმოდგენის რაიმე ნაწილის აღსაწერად რაიმე მნიშვნელობის არჩევის შესაძლებლობას, მაშინ როცა სანიმუშო ნაწილი გვიჩვენებს, თუ როგორ ვიყენებთ კონკრეტულ სიტყვას, რომელიც შეესატყვისება რაიმე მნიშვნელობას. ანალიზის დროს მნიშვნელობა აქვს მსგავსებას წესთან ნიმუში მოქმედება, ხოლო თუ ჩვენ ვქმნით

ტექსტს, მაშინ იგი არის წესი: მოქმედება ნიმუში. მაგრამ ეს არის დეკლარაციული ცოდნა და შესაძლებელია სხვადასხვა მნიშვნელობები იყოს განხილული რაიმე სიტყვისათვის და სემანტიკური ცოდნა არ მიუთითებს , თუ როგორ უნდა მოვახდინოთ მათ შორის არჩევანი. როცა ინტერპრეტატორი ვერ პოულობს სწორ მნიშვნელობას, მან უნდა მიმართოს სხვა სახის ცოდნას, რომ აღმოფხვრას ომონიმია.

სანიმუშო ნაწილი შეიცავს შემდეგ კომპონენტებს:

1. ერთნაირი მნიშვნელობის მქონე სიტყვების სიმრავლე, რომელთაც აქვს ერთნაირი მნიშვნელობა ცოდნის განსახილველ არეში.

2. კვანძის სახელი, სადაც შეიძლება ეს სიტყვები ვიპოვოთ.

3. სემანტიკური მარკერები. რაიმე პირობა შეიძლება აზუსტებდეს, რომ რაიმე კვანძში არსებობს სიტყვა, რომელსაც აქვს მნიშვნელობა კერძო სემანტიკური მარკერით. ეს სემანტიკური მარკერები უნდა განსაზღვროს ექსპერტმა.

4. რაიმე პირობა. თუ პირობა მცდარია, მნიშვნელობა არაა წარმოდგენილი კონკრეტულ წინადადებაში.

5. სამეულთა სიმრავლე, რომლებიც აღწერენ კონკრეტული სიტყვის გარემოცვას. რაიმე სამეულს აქვს შემდეგი კომპონენტები:

1. აუცილობლობა. მას აქვს სამი მნიშვნელობა (სავალდებულო, არჩევითი, განსაკუთრებული).

2.კვანძების სახელები, სადაც შეიძლება იყოს დამატებითი ინფორმაცია.

3. კვანძში კონკრეტული სიტუაციის აღწერა. არსებობს რაიმე სიმრავლე შესაძლებლობებისა და რომელი მათგანი უნდა იყოს დაკმაყოფილებული.

რაიმე მნიშვნელობის აქტიური ნაწილი. აქ განვიხილავთ ორი სახის მოქმედებას:

1. მოქმედება, რომელიც განსაზღვრავს რაიმე ობიექტს, რომელიც წარმოადგენს კონკრეტულ მნიშვნელობას სიტყვათა სიმრავლიდან.

2. მოქმედება, რომელიც აღწერს რაიმე სიტყვის არსებობა თუ როგორ აზუსტებს სხვა სიტყვების მნიშვნელობების წარმოდგენას. ჩვეულებრივ, აქტიური ნანილი განსაზღვრავს რაიმე ობიექტს, რომელიც მნიშვნელობის წარმოდგენის ნაწილია. იგი შეიძლება აუცილებელი იყოს რაიმე დამატებითი ობიექტის განსასაზღვრად. ყოველი ობიექტი წარმოდგენილია წყვილების სიმრავლით (ატრიბუტი-მნიშვნელობა).

სიტყვები როგორიცაა "ძალიან", "ბევრი", "საკმარისი" და ასე შემდეგ არ განსაზღვრავენ რაიმე ობიექტს, არამედ მიუთითებენ, რომ რაიმე ობიექტის რაიმე ატრიბუტის მნიშვნელობა უნდა დაზუსტდეს. მაგალითად რაიმე ობიექტის ატრიბუტის მნიშვნელობა თუ არის სუსტი, მაშინ სუსტი ძალიან სუსტი ან დიდი ძალიან დიდი და ა.შ.

პრაგმატული ცოდნა. პრაგმატული ცოდნის წარმოსადგენად გამოიყენება გრამატიკული წესები, რომელთაც აქვთ სახე A→B. სადაც A არის რაიმე პირობა და B არის მოქმედება ან მოქმედებათა ერთობლიობა, რომელიც უნდა შესრულდეს თუ A პირობა ჭეშმარიტია. მაშასადამე A არის საზოგადოდ რაიმე ლოგიკური გამოსახულება და B არის რაიმე ოპერაციების ერთობლიობა. ამგვარად, უნდა იყოს რაიმე გრამატიკით განსაზღვრული A-სა და B-ს ზოგადი სახე და იგი დამოკიდებულია კონკრეტულ რეალიზაციაზე.

5 თავი
გაფართოებული კონტექსტისაგან თავისუფალი გრამატიკის გამრჩევი

5.1. შესავალი. კონტექსტისაგან თავისუფალი გრამატიკის გაფართოებული გამრჩევი (სინტაქსური ანლიზატორი) წარმოადგენს კონტექსტისაგან თავისუფალი გრამატიკის ახალი, გაფართოებული ფორმალიზმის და მისი რეალიზაციის შედეგად მიღებული პროგრამული უზრუნველყოფის ერთობლიობას, რომელიც შეიძლება წარმატებით იქნეს გამოყენებული ბუნებრივი ენის სინტაქსური, მორფოლოგიური და გარკვეულ წილად სემანტიკური ანალიზის ჩასატარებლად. ზემოხსენებული ანალიზატორის შემუშავებას და შექმნას საფუძვლად დაედო ხანგრძლივი გამოკვლევები ქართული ენის მორფოლოგიური და სინტაქსური, კომპიუტერული ანალიზის სფეროში და გათვალისწინებულ იქნა ის შედეგები, რომლებიც მიღებულ იქნა ანალოგიურ სფეროში მოღვაწე უცხოელი სპეციალისტების მიერ. თუმცა ამთაეითვე უნდა აღინიშნოს ის, რომ შემოთავაზებული ფორმალიზმი არ წარმოადგენს უკვე არსებული, ბუნებრივი ენის დასამუშავებლად შექმნილი რაიმე ფორმალიზმის უბრალოდ გაფართოებას ან რომელიმე მსგავსი პროგრამული უზრუნველყოფის მოდიფიკაციას, არამედ არის დამოუკიდებლად არსებული, პრინციპულად ახალი და პრაქტიკული საჭიროებებიდან გამომდინარე მოთხოვნების გათვალისწინებით მიღებული პროდუქტი და ქართულის გარდა იგი შეგვიძლია მივუსადაგოთ თითქმის ნებისმიერ ბუნებრივ ენას. ამგვარად, მიღებული იქნა საკმარისად მოხერხებული ფორმალიზმი, რომლის მეშვეობითაც ბუნებრივი ენის გრამატიკის ჩაწერა არის უფრო მარტივი, თვალსაჩინო და მოცულობით ნაკლები ვიდრე სხვა ანალოგიურ მიდგომებში[77-

82]. ამ ფორმალიზმის შედეგად მიღებული პროგრამული უზრუნველყოფა წარმოადგენს მოხერხებულ ინსტრუმენტს ლინგვისტებისათვის, რათა მათ გამოსცადონ კომპიუტერზე თავიანთი მოდელი და გააუმჯობესონ იგი ისე, რომ უფრო მისაღები ალმოჩნდეს კომპიუტერული დამუშავებისათვის, ჩაატარონ ექსპერიმენტები დიალოგურ რეჟიმში და მათი გათვალისწინებით მოახდინონ გრამატიკის სრულყოფა. მრავალი დანიშნულებით შეგვიძლია გამოვიყენოთ მიღებული პროგრამული უზრუნველყოფა, მაგალითად ისეთ სისტემებში, როგორიცაა: ავტომატური საძიებო სისტემები ინტერნეტისთვის, ბუნებრივ (სალაპარაკო) ენაზე მომუშავე დიალოგური პროგრამები, ტექსტების მრავალმიზნობრივი ანალიზი და დამუშავება (შემდგომი ლექსიკონის მიღებისათვის), ენის სწავლებისათვის განკუთვნილი სავარჯიშო ტრენაჟორები, მანქანური თარგმანი და მრავალი სხვა. ინფორმაციული ტექნოლოგიების დღევანდელი ტემპით განვითარების პირობებში ძნელია იპოვო სფერო, სადაც არ იქნება მოთხოვნილება ამგვარ პროგრამულ პროდუქტებზე. ამ თავში დაწერილებით განვიხილავთ ჩვენს მიერ შემუშავებულ ახალ, გაფართოებულ ფორმალიზმს. ავღწერთ სინტაქსური გარჩევის პროგრამას (როგორც ანალიზატორის ბირთვს, ასევე მომხმარებლის ინტერფეისს). მოვიყვანთ შესაბამის მაგალითებს, დაწვრილებით ავღწერთ ლექსიკონის და გრამატიკის ფაილთა სტრუქტურას. განვიხილავთ განსხვავებებს, სხვა მსგავსი დანიშნულების სისტემებისაგან. ბოლოს კი განვიხილავთ ქართული ენის გრამატიკის ფრაგმენტს კომენტარებით.

5.2. ახალი, გაფართოებული ფორმალიზმი. გამრჩევის ყველაზე მნიშვნელოვანი ფაილი არის გრამატიკის ფაილი, სადაც მოიცემა იმ წესთა ერთობლიობა, რომლებიც ადგენენ ენას და რომელთა მეშვეობითაც ანალიზატორი ცდილობს ააგოს სინტაქსური გარჩევის ხე ე.ი. დაადგინოს როგორ კავშირში არიან შემავალი

ლექსიკური ერთეულები ერთი მეორესთან. ცხადია, იმისათვის, რომ განვსაზღვროთ რომელიმე ენის გრამატიკა, ჩაეწეროთ იგი ადამიანისათვის და მანქანისათვის გასაგები სახით, საჭიროა გარკვეული ფორმალიზმის შემოღება, რომლის ფარგლებშიც მოხდება ამგვარი ჩაწერები. რას წარმოადგენს ეს ახალი ფორმალიზმი, რას ემყარება იგი, რა ნაწილებისაგან შესდგება და რითი განსხვავდება უკვე არსებული სხვა ფორმალიზმებისაგან? ვცადოთ ამ კითხვებზე თანმიმდევრობით პასუხის გაცემა. თაეიდან აეხსნათ თუ რას ემყარება ეს ფორმალიზმი. უპირველესად იგი ემყარება კონტექსტისაგან თავისუფალი გრამატიკის ცნებას, და კონტექსტისაგან თავისუფალი გრამატიკის წესებს. რას წარმოადგენს ასეთი წესი? ეს 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.. ეს წესები აღწერენ იმ დასაშვებ ურთიერთდამოკიდებულებებს, რომლებშიც შეიძლება იყვნენ შემავალი ლექსიკური ერთეულები (უფრო დაწვრილებით კონტექსტისაგან თავისუფალი გრამატიკის შესახებ იხილეთ [1]). ამავე წესების გამოყენებით ჩვენ შეგვიძლია ვცადოთ ბუნებრივი ენის გრამატიკის ჩაწერა. მაგალითად, შეგვიძლია წინადადება მარტივად შემდეგი სახით განესაზღვროთ (ქართულში ეს შეესაბამება ერთპირიანი ზმნების შემთხვევას):

S -> NP VP;

NP არის სახელის ჯგუფი, VP ზმნის ჯგუფი. ავიღოთ

წინადადება: 'ბიჭი მირბის', აქ სახელის ჯგუფი ერთი სიტყვით (არსებითი სახელით) არის წარმოდგენილი, ისევე როგორც ზმნის ჯგუფი. თუმცა ქართლისათვის დასაშვებია აგრეთვე შებრუნებული წყობა 'მირბის ბიჭი'. ქართულში შეგვიძლია ნებისმიერი კომბინაციით დაეალაგოთ 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 ზემოთ მოყვანილი გრამატიკა უნდა შეიცავდეს აგრეთვე მეორე წესსაც:

S -> VP NP;

ერთპირიანი ზმნების შემთხვევაში გვაქვს სულ 2 ვარიანტი, ორპირიანი ზმნების შემთხვევაში ვარიანტების რაოდენობა გაიზრდება 6-მდე, ხოლო სამპირიანი ზმნების დროს იგი მიაღწევს 24-ს, რაც ძალიან ზრდის წესების რაოდენობას. ამ ნაკლოვანების აღმოფხვრა მოხდა სტანდარტული ფორმალიზმის გაფართოებით მასში სიმბოლოთა ურთიერთმდებარეობის მარეგულირებელი კონსტრუქციების შემოტანის გზით. ავსწნათ ეს მარტივი მაგალითის საშუალებით S -> A B C D : D < A B - C; შევნიშნოთ, რომ ორწერტილის შემდეგ წესს მოსდევს სიმბოლოთა ურთიერთმდებარეობის მარეგულირებელი კონსტრუქცია, ორწერტილის არსებობა უკვე მიუთითებს ანალიზატორს იმაზე, რომ უნდა მოხდეს ამ წესში სიმბოლოთა ყველანაირი კომბინაციის განხილვა, ხოლო დამატებითი დირექტივები განსაზღვრავენ თუ რა დამოკიდებულებებშია ერთმნეთთან ეს სიმბოლოები და გაფილტვრის შედეგად დარჩება მხოლოდ ის წყობები, რომლებიც დააკმაყოფილებენ მათ. ამგვარად ჩაწერილი წესი აღნიშნავს, რომ ანალიზატორის შიგნით ამ ერთი წესისაგან მოხდება სამი წესის წარმოქმნა, რომლებიც ერთიმეორესაგან 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. შემოღებული გვაქვს ორი ტიპის მარეგულირებელი კონსტრუქცია. პირველი არის დირექტივა "წინმსწრები" და იგი აღინიშნება '<' ნიშნით, ხოლო მეორე - "წინმდგომი" და მას ავღნიშნავთ დეფისით. უფრო დაწვრილებით განვიხილოთ ზემოთმოყვანილი მაგალითი. '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 შემდეგი კომბინაციები გვაქვს:

D A * * * D A *

D * A * * D * A

D * * A * * D A *-ის ადგილას იგულისხმება B ან C სიმბოლო.

მომდევნო დირექტივა წარმოადგენს "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 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 ადრე მოვიყვანეთ.

ამ მაგალითიდან ჩანს, თუ როგორ კომპაქტურად იწერება წესები. ცხადია ჩვენ შეგვიძლია ჩავწეროთ წესი,

რომელსაც ორწერტილის შემდეგ არ მოსდევს არც ერთი ურთიერთმდებარეობათა მარეგულირებელი კონსტრუქცია. ამ შემთხვევაში მოხდება ყველა კომბინაციის გათვალისწინება, რომელიც შეიძლება მივიღოთ ამ წესში სიმბოლოთა თავისუფალი გადანაცვლებებით. მაგალითად შემდეგი წესი:

S -> A B C : ;

წარმოქმნის შემდეგ კომბინაციებს:

A B C B C A

A C B C A B

B A C C B A

შესაბამისად, იგი იმუშავებს ყველა შემთხვევაში, როცა ერთმანეთის მიყოლებით ნებისმიერი მიმდევრობით შეგხვდება A B C სიმბოლოები, და ამ შემთხვევაში მოხდება მათი ჩანაცვლება S არატერმინალით.

განვიხილოთ ამ ფორმალიზმის კიდევ ერთი თავისებურება. თუკი წესის შიგნით რომელიმე სიმბოლო მეორდება საჭიროა მათი ერთმანეთისაგან გამოყოფა ინდექსების მეშვეობით. ინდექსი გვერდზე ეწერება სიმბოლოს 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'-გან, `S' განსხვავდება `S.2'-გან და `B.1' `B.2'-გან. 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 არ ავღნიშნავთ ცხადი სახით. ეს აღნიშვნები გვხმარება როგორც ურთიერთმდებარეობის მარეგულირებელ კონსტრუქციებში, აგრეთვე თვისებათა შემზღუდველებში (იხ. ქვემოთ).

შემდეგი გაუმჯობესება, რომელსაც ემყარება ეს ახალი

ფორმალიზმი, წარმოადგენს ე.წ. თვისებათა შემზღუდველების მექანიზმს. რაში მდგომარეობს ეს მექნიზმი? დავუბრუნდეთ საწყის მაგალითს: S -> NP VP საკმარისია თუ არა ეს წესი იმისათვის, რომ კორექტულად მოხდეს წინადადების გარჩევა. ბუნებრივი ენის თავისებურობებიდან გამომდინარე, აშკარაა, რომ მხოლოდ ეს წესი ვერ მოახერხებს წინადადების სწორ სინტაქსურ ანალიზს, იგი არის აუცილებელი, მაგრამ არასაკმარისი ჩვენი მიზნების განსახორციელებლად. სინამდვილეში ამ წესის მიერ განსაზღვრულ წინადადებათა კლასი, იმაზე გაცილებით უფრო ფართოა ვიდრე ამას მოითხოვს ბუნებრივი ენა. ამის მიზეზი არის ის, რომ ამ წესის მიერ არ ხდება ენის თავისებურობების გათვალისწინება, მისი უმნიშვნელოვანესი ატრიბუტები, ისეთები როგორიცაა: პირი, რიცხვი, ბრუნვა, ზმნის ფორმა. არამედ იგი უბრალოდ მიგვითითებს წინადადების შემადგენელ კომპონენტების ურთიერთგანლაგებაზე, მის სტრუქტურაზე. შედეგად ჩვენ ვიღებთ ისეთ გრამატიკას, რომელშიც დაშვებულია და კორექტულად ითელება არა მარტო "ბიჭი მირბის" წინადადება, არამედ მისი არასწორი ტრანსფორმაციებიც, მაგალითად: "ბიჭმა მირბის", "ბიჭი მირბიან", "მე მირბის" და ა.შ. რაც ნათლად გვიჩვენებს, თუ რამდენად მოუხერხებელია ასეთი გრამატიკა. ამიტომ, საჭიროა, რომ გრამატიკის წესები რამოდენადმე გავართულოდ, დავადოთ მათ გარკვეული შეზღუდვები, რათა გავცხრილოთ და ამოვყაროთ ის უსარგებლო კონსტრუქციები, რომლებიც ხელს უშლიან სწორ და ენისთვის მისაღებ წინადადებათა სიმრავლის ჩამოყალიბებას. სწორედ ამ მიზანს ემსახურება ჩვენს მიერ ნახსენები თვისებათა შემზღუდველები. მათი საშუალებით ჩვენ შეგვიძლია უფრო გავამკაცროთ გრამატიკა, უფრო მოხერხებული და მოქნილი გავხადოთ და უფრო მეტად დავხვეწოთ, რათა მან დიდი სიზუსტით მოიცვას ბუნებრივი ენის გრამატიკა. ამ მექანიზმის

დაწვრილებითი განხილვა დავიწყოთ იმის ახსნით, თუ რას წარმოადგენს თვისებათა სტრუქტურა – ცნება, რომელიც გადამწყვეტ როლს თამაშობს შემზღუდველებში. თვისებათა სტრუქტურა არის 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 სტრიქონის სახით, ან წარმოადგენს სხვა თვისებათა სტრუქტურას, ან კიდევ არის ცარიელი. ეს გახლავთ რეკურსიული განმარტება, შესაბამისად დასაშვებია, რომ თვისებათა რამოდენიმე სტრუქტურა ერთმანეთში იყოს საკმარისად დიდ სიღმეზე ჩალაგებული. თვისებათა სტრუქტურებს ვწერთ კვადრატულ ფრჩხილებში, ხოლო თვისების სახელს მისი მნიშვნელობისაგან ორწერტილით გამოვყოფთ. მოვიყვანოთ შესაბამისი მაგალითები:

1.
[ბრუნვა: სახ]
2. [რიცხვი: მხ
პირი: '1']
3.
[ობიექტი: [რიცხვი: მხ]
სუბიექტი: [რიცხვი: მრ
ბრუნვა: სახ]]
4.
[ცარიელი_თვისება: None]

პირველ მაგალითში მოცემული თვისებათა სტრუქტურა შეიცავს მხოლოდ ერთ თვისებას, რომლის სახელია "ბრუნვა" და მნიშვნელობად აქვს "სახ" (სახელობითი). შემდეგ მაგალითში გვაქვს უკვე ორი თვისება "რიცხვი" და "პირი" მათი მნიშვნელობებს კი შესაბამისად წარმოადგენენ "მხ" (მხოლობითი) და "1" (პირველი პირი), შევნიშნოთ, რომ ერთიანი ჩასმულია აპოს-

ტროფებში, რადგანაც ფორმალიზმის სტრუქტურიდან გამომდინარე თუკი თვისების სახელი ან მისი მნიშვნელობა არ იწყება სიმბოლოთი (ლათინური ან სტანდარტულ კოდირებაში მოცემული ქართული ასოებით) აუცილებელია ჩავსვათ იგი აპოსტროფებში. მომდევნო მაგალითი გვიჩვენებს ჩალაგებულ თვისებათა სტრუქტურას. მასში მოცემულია ორი ველი "ობიექტი" და "სუბიექტი", რომელთა მნიშვნელობებს წარმოადგენენ სხვა თვისებათა სტრუქტურები. ბოლო მაგალითში კი ნაჩვენებია, თუ როგორ აღვწეროთ თვისება ცარიელი მნიშვნელობით. ამ მიზნისათვის გამოიყენება სპეციალური მნიშვნელობა None (იგი რეზერვირებული სიტყვაა და არ შეიძლება სხვა ადგილას მისი გამოყენება, მაგალითად იდენტიფიკატორში).

ამგვარად, ჩვენ გავარკვიეთ თუ რას წარმოადგნენ თვისებათა სტრუქტურები და ის, თუ რა მნიშვნელობები შეიძლება მიიღონ მათ. განვსაზღვროთ ის ოპერაციები, რომლებიც შეიძლება 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 მათგანი ცალცალკე: ვთქვათ გვაქვს ორი თვისებათა სტრუქტურა a და b. შემდეგი გამოსახულება:

a := b

წარმოადგენს მინიჭების ოპერაციას. მისი მნიშვნელობა ისეთივეა როგორიც პროგრამირების ენაში ცვლადების მინიჭებისას. a თვისებათა სტრუქტურის შიგთავსი შეიცვლება b თვისებათა სტრუქტურით და ამის შედეგად ისინი გაუტოლდებიან ერთმანეთს. ოპერაცია ყოველთვის ლოგიკურად ჭეშმარიტ მნიშვნელობას აბრუნებს.

a = b

ამ ოპერაციით ხდება ორი თვისებათა სტრუქტურის

ერთმანეთთან ტოლობაზე შემოწმება. ორი თვისებათა სტრუქტურა ერთმანეთს ემთხვევა თუკი ისინი შეიცავენ მხოლოდ ერთიდაიგივე თვისებებს და ამ თვისებათა მნიშვნელობები ერთმანეთს ემთხვევა. ტოლობის დადგენის შემთხვევაში ოპერაცია დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას, ხოლო თუ თვისებათა სტრუქტურები განსხვავდებიან დაბრუნდება ლოგიკურად მცდარი მნიშვნელობა.

a <== b

უნიფიკაცია. ორი თვისებათა სტრუქტურა ერთმანეთთან უნიფიცირდება თუკი მათი ერთნაირი სახელის მქონე თვისებების მნიშვნელობები არ ეწინააღმდეგებიან ერთმანეთს. თვისებების მნიშვნელობები ერთმანეთს არ ეწინააღმდეგებიან, მაშინ როდესაც:

1.ერთერთი მნიშვნელობა არის ცარიელი (None)

2.ორივე მნიშვნელობა მარტივია (არ არიან სხვა თვისებათა სტრუქტურები) და ისინი ერთმანეთს ემთხვევა.

3.ორივე მნიშვნელობა რთულია (თავად არიან თვისებათა სტრუქტურები) და ისინი უნიფიცირდებიან ერთმანეთთან.

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

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. ანუ უნიფიკაციის წარმატებით შესრულების შემთხვევაში ბრუნდება ლოგიკურად ჭეშმარიტი მნიშვნელობა, ხოლო როდესაც თვისებათა სტრუქტურები არ უნიფიცირდებიან ერთმანეთთან ბრუნდება ლოგიკურად მცდარი მნიშვნელობა.

a == b

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 შესაძლებელია მეორე სახითაც, შესაბამისი ფუნქციის გამოძახების მეშვეობით. ყველა ჩვენს მიერ ჩამოთვლილ ოპერაციას აქვს თავისი სახელი, რომლითაც იგი რეგისტრირებულია სისტემაში. მაგალითად: მინიჭების ოპერაციას (:=) შეესაბამება სახელი 'assign', ტოლობის (=) სახელი 'equal', უნიფიკაციის (<==) 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 აბრუნებს ჭეშმარიტ მნიშვნელობას. მას რაიმე განსაკუთრებული პრაქტიკული დატვირთვა არ გააჩნია, გარდა იმისა, რომ იგი შეგვიძლია გამოვიყენოთ როგორც კონსტანტა ლოგიკური გამოსახულებების შედგენისას ან სადიაგნოსტიკო მიზნებისთვის.

გარდა, ზემოთ ჩამოთვლილისა, ასევე გვაქვს ორი დამატებითი ფუნქცია. მათ შეიძლება წინასწარ უცნობი რაოდენობის არგუმენტები გადაეცეთ. ამიტომ ისინი ჩაიწერებიან როგორც ფუნქციები, და მათი გამოძახებისას ფრჩხილებში ხდება ყველა იმ პარამეტრის გადაცემა, რომლებიც მონაწილეობენ ოპერაციის შესრულებაში. ეს ფუნქციებია: meq (multiple equation checking) და muc (multiple unification checking). მათი გამოძახების ფორმატია:

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 ფუქციათა 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 ფუნქციის ფარგლებში. იმის გამო, რომ ეს ფუნქციები მოხერხებულია და ხშირად გვიხდება მათი გამოყენება, შემოღებულ იქნა მათი ჩაწერის ოპერატორული ფორმა, რომლიც შემდეგნაირად გამოიყურება:

meq(e, a, b, c,...) <-> e = (a, b, c,...)

muc(e, a, b, c,...) <-> e == (a, b, c,...)

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

განვიხილოთ რაიმე წესი:

S -> A B C;

პირველ რიგში უნდა ავღნიშნოთ ის, რომ ნებისმიერ სიმბოლოს (ტერმინალურს ან არატერმინალურს) ჩვენს სისტემაში შეესაბამება თვისება, რომელის მნიშვნელობაც შეიძლება იყოს როგორც მარტივი (ვთქვათ რაიმე სტრიქონი), ასევე, და ეს უფრო ხშირი შემთხვევაა, რთული ანუ თვისებათა სტრუქტურა. შესაბამისად S, A, B და C სიმბოლოებს გააჩნიათ თვისებები, რომლებიც თავიდან გაჩუმებით ცარიელია, ან შეიცავენ რაიმე ცარიელ ან არაცარიელ მნიშვნელობას მარჯვენა მხარეში მდგომ იმ სიმბოლოთათვის რომლებიც მიღებულნი იქნენ წინა ეტაპზე რომელიმე წესის გამოყენებით. აღნიშნოთ კიდევ ის, რომ მარჯვენა მხარეში გამოყენებული სიმბოლოებისთვის ამ თვისების გამოყენება შეიძლება მხოლოდ წასაკითხად, მარცხენა მხარეში მდგარი ერთადერთი არატერმინალური სიმბოლოსთვის კი როგორც წასაკითხად ასევე (უფრო ხშირად) ჩასაწერად. ისმის კითხვა, თუ როგორ მოვახდინოთ ამ თვისებებით მანიპულირება, მათი შემოწმება, შეცვლა და ა.შ.. ამ მიზნისთვის გამოიყენება

თვისებათა შემზღუდველები. ისინი იწერებიან წესის ბოლოში, წერტილმძიმის (';') მაგივრად, ფიგურულ ფრჩხილებში. მაგალითად:

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

ფიგურულ ფრჩხილებში იწერება თვისებებზე განსაზღვრული ოპერაციების გამოყენებით მიღებული ლოგიკური გამოსახულება. ვინაიდან თვითოეული ასეთი ოპერაცია აბრუნებს ლოგიკურ მნიშვნელობას, ისინი შეგვიძლია გაეაერთიანოთ შემდეგი უმარტივესი ლოგიკური ოპერაციებით: '~' - ლოგიკური უარყოფა, ერთადგილიანი (უნარული) ოპერაცია, იღებს მისი არგუმენტის საწინააღმდეგო მნიშვნელობას; '&' - ლოგიკური ნამრავლი, "და" ორადგილიანი (ბინარული) ოპერაცია, იღებს ჭეშმარიტ მნიშვნელობას თუკი მისი ორივე ოპერანდი ჭეშმარიტია, წინააღმდეგ შემთხვევაში იღებს მცდარ მნიშვნელობას, '|' - ლოგიკური ჯამი, "ან" ორადგილიანი (ბინარული) ოპერაცია, იღებს ჭეშმარიტ მნიშვნელობას თუკი ერთი რომელიმე ოპერანდი მაინც ჭეშმარიტია, წინააღმდეგ შემთხვევაში იღებს მცდარ მნიშვნელობას. ასეთნაირად, მიღებული ლოგიკური გამოსახულება გამოთვლება (შესრულდება) ამ ლოგიკური შემზღუდველის მფლობელი წესის ნებისმიერი ამოქმედების შემთხვევაში. თუკი ეს ლოგიკური გამოსახულება დააბრუნებს ჭეშმარიტ მნიშვნელობას, სისტემა ჩათვლის რომ მოცემული გრამატიკული წესის არგუმენტები აკმაყოფილებენ მათზე დადებულ შეზღუდვებს და რომ ეს წესი ამ კონკრეტულ შემთხვევაში უნდა ჩაითვალოს როგორც დასაშვები და წინადადების გარჩევის პროცესი გაგრძელდეს ჩვეულებრივ. ხოლო თუკი ამ ლოგიკური გამოსახულების მიერ დაბრუნებული იქნა ლოგიკურად მცდარი მნიშვნელობა, სისტემა ჩათვლის რომ ამ კონკრეტულ შემთხვევაში ეს წესი არ არის დასაშვები და წინადადების გარჩევა არ გაგრძელდება მოცემულ მომენტში წესის გამოყენებით, არამედ განიხილება

სხვა დასაშვები წესი ან თუ ასეთი არ მოიძებნა მოხდება ანალიზატორის დაბრუნება ერთი ნაბიჯით უკან და გაგრძელდება გარჩევის შემდგომი მცდელობები. თვისებათა შემზღუდველების ძირითადი მიზანია, რომ გავამკაცროთ გრამატიკა, დავადოთ მას უფრო მეტი შეზღუდვები, ვიდრე ამის საშუალებას ჩვეულებრივი კონტექსტისაგან დამოუკიდებელი გრამატიკის წესი 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 შემზღუდველები არის ინფორმაციის გადატანა წესის მარცხენა მხარეში მოთავსებული არატემინალური სიმბოლოს თვისებაში. რათა გარჩევის უფრო გვიან ეტაპებზე მოხდეს ამ ინფორმაციის გამოყენება სხვადასხვა შემოწმებებისთვის და დამუშავებისთვის. განვიხილოთ ისევ ძველი მაგალითი, სადაც წინადადების განმსაზღვრავი წესია მოცემული და დავამატოთ მას რამოდენიმე თვისებათა შემზღუდველი:

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 აეხსნათ თუ როგორ იმუშავებს ეს წესი. იგი იმუშავებს როგორც კონტექსტისაგან თავისუფალი გრამატიკის ჩვეულებრივი წესი, სადაც აღმოაჩენს

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 მივიღებთ იმას რომ 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 სამკუთხა ფრჩხილების მეშვეობით. რას წარმოადგენს ეს ჩანაწერი. როგორც უკვე ვთქვით ყოველ სიმბოლოს წესის შიგნით შეესაბამება თავისი თვისება. მაგალითად სიმბოლო A-ს თვისება შემზღუდველებში შეგვიძლია ჩავწეროთ <A> სახით. ამის გარდა, ამავე სამკუთხა ფრჩხილებში შეგვიძლია მივუთითოთ გზა რომელიმე ჩვენთვის საინტერესო ქვეთვისებისათვის. მაგალითად: <A head num> ჩანაწერი აღნიშნავს head-ის თვისება num-ს რომელიც, 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 სამკუთხა 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 ასეთები აქვთ რა თქმა უნდა. 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. მაგალითად:

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

ეს ჩანაწერი (რომელიც უნდა მოთავსებული იყოს ლექსიკონის ფაილში) აღნიშნავს ლექსიკურ ერთეულს, სიტყვა "სახლს", ხოლო კვადრატულ ფრჩხილებში კი, რომლების მოსდევს ამ სიტყვას, ჩაწერილია ჩვენთვის კარგად ნაცნობი თვისებათა სტრუქტურა. კერძოდ, აღნიშნულია რომ სიტყვა "სახლს" წარმოადგენს არსებით სახელს, რომ იგი 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 შესაბამის თვისება მნიშვნელობად მიიღებს იმ თვისებათა სტრუქტურას, რომელიც მოცემული იყო ლექსიკონში. მაგალითად, მოცემულია ერთი წესისგან შემდგარი შემდეგი

გრამატიკა:

S -> N {

<S ბრუნვა> := <N ბრუნვა>

}

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

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

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

$a = [f1: v1]

ეს ნიშნავს რომ ცვლად a-ს მიენიჭა მნიშვნელობად თვისებათა სტრუქტურა [f1: v1]. ამის შემდგომ ეს ცვლადი შეგვიძლია გამოვიყენოთ როგორც ოპერანდი თვისებათა შემზღუდველებში შემდეგნაირად: <$a> ან თუ გვინდა კონკრეტული გზის მითითება <$a some path>. ცვლადების ძირითადი დანიშნულებაა ის, რომ ისინი ამარტივებენ წესების ჩაწერას, მაგალითად როცა რამოდენიმე ადგილას საჭიროა

მოზრდილი მოცულობის მქონე თვისებათა სტრუქტურის ჩაწერა, შეგვიძლია ეს თვისებათა სტრუქტურა ავღნიშნოთ რაიმე ცვლადით, და ყველა დანარჩენ ადგილებში გამოვიყენოთ ეს ცვლადი. გარდა ამისა, ცვლადები შეგვიძლია გამოვიყენოთ როგორც ინფორმაციის გადაცემის კიდევ ერთი მექანიზმი წესების მუშაობის პროცესში ან გამოვიყენოთ დიაგნოსტიკის მიზნით. გრამატიკის ფაილის გარდა, ცვლადები გვაქვს აგრეთვე ლექსიკონის ფაილშიც. ორივე ფაილს შეესაბამება ერთიდაიგივე ცვლადების ცხრილი, შესაბამისად ისინი ორივე ერთიდაიგივე ცვლადებთან მუშაობენ. 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 აყენებს a ცვლადის მნიშვნელობას შესაბამის თვისებათა სტრუქტურით. ამით დავასრულეთ ამ ახალი, გაფართოებული ფორმალიზმის მოკლე აღწერა. უფრო დეტალური და მკაცრი აღწერა იმისა, თუ როგორ იწერება გრამატიკის და ლექსიკონის ფაილი, რა კონსტრუქციებია მასში დასაშვები და როგორ გამოვიყენოთ ისინი ეფექტურად, მოცემული იქნება მომდევნო პარაგრაფებში.

5.3. პროგრამის აღწერა. ჩვენს მიერ ზემოთ განხილული ფორმალიზმის პრაქტიკული გამოყენებისათვის და მისი რეალიზაციისათვის საჭიროა პროგრამული უზრუნველყოფა. ეს პროგრამული უზრუნველყოფა წარმოადგენს სინტაქსურ ანალიზატორს რომელიც გამოიყენება ბუნებრივ ენაზე ჩაწერილ წინადადებათა გარჩევისათვის. მისთვის ძირითად შემავალ ინფორმაციას წარმოადგენს ჩვენს მიერ უკვე განხილულ ფორმალიზმში ჩაწერილი გრამატიკის ფაილი, რომელიც გრამატიკული წესების მეშვეობით ასახავს დასაშვებ წინადადებათა სიმრავლეს, აგრეთვე ლექსიკონის ფაილი,

რომელშიც მოცემულია ენის ლექსიკური მარაგი, ანუ მარტივად, რომ ვთქვათ ენაში შემავალი სიტყვები თავისი დამახასიათებელი თვისებებით და მათი მნიშვნელობებით, ისეთებით როგორიცაა: ბრუნვა, რიცხვი და ა.შ. ორივე ფაილის არსებობის შემთხვევაში გამრჩევი უკვე მზადაა თავისი ფუნქციების შესასრულებლად. ამისათვის, საკმარისია ავკრიფოთ ბრძანება "parse" და ის წინადადება, რომლის გარჩევაც გვსურს. გამრჩევი მონახავს წინადადების შემადგენელ სიტყვებს ლექსიკონში და ყოველი მათგანის არსებობის შემთხვევაში ლექსემებად დაყოფილ შემავალ წინადადებას გადასცემს დამუშავების მომდევნო ეტაპს, სადაც გრამატიკული წესების საფუძველზე მოხდება წინადადების ანალიზი, აიგება გარჩევის ხე (ან ხეები, თუკი ასეთი ერთზე მეტია) და მოხდება მისი გამოტანა მომხმარებლისათვის. 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 სიმბოლებმა გარჩევის პროცესში. ეს დამატებითი საშუალებაც გეაძლევს საკმაოდ მნიშვნელოვან ინფორმაციას წინადადების აგებულებაზე და გრამატიკული წესების მუშაობის პროცესზე.

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

Welcome to Enhanced Parser 0.1

Copyright (C) 2002 By VIAM (Vekua Institute of Applied Mathematics)

Author: David Mishelashvili <david@posta.ge>

Supervisor: Jemal Antidze <ja@viam.hepi.edu.ge>
For help type: \h
&>

დიალოგურ რეჟიმში მუშაობისათვის მიწვევის 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 ახორ-

ციელებს ეს ბრძანება. განვიხილით 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. არის 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 პარამეტრებით ხდება ჩვენთვის საინტერესო ბრძანების შესრულებაზე გაშვება.

\q - პროგრამიდან გამოსვლა.

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

\ll ლექსიკონის ჩატვირთვა ფაილიდან. ეს ბრძანება მოითხოვს ერთ პარამეტრს, ფაილის სახელს. მისი შესრულების შედეგად მოხდება ლექსიკონის ჩატვირთვა მითითებული ფაილიდან. ცხადია ასეთი ფაილი წინასწარ უნდა იყოს მომ–

ზადებული და მასში შესაბამისი ფორმატით უნდა ჩაიწეროს ლექსიკური ერთეულები (ანუ ბუნებრივი ენის სიტყვები). რაც უფრო დიდია ლექსიკონი, რაც უფრო მეტ ლექსიკურ ერთეულს შეიცავს იგი, მით უფრო დიდია იმ წინადადებათა სიმრავლე რომელთა დამუშავებაც შეგვიძლია გამრჩევის მეშვეობით. მაგალითად: \ll sample ან \ll sample.lex ორივე ვარიანტი თითქმის ანალოგიურია, ვინაიდან პირველ შემთხვევაში სისტემა, თუკი მან ვერ აღმოაჩინა sample ფაილი მიმდინარე კატალოგში, ფაილის სახელს ავტომატურად დაუმატებს `.lex' გაფართოვებას 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 კომპიუტერი ეკრანზე. თუკი ვერ მოხდა ასეთი სიტყვის მოძებნა გამოვა შესაბამისი შეტყობინება. მაგ:

&> \lf a

[cat: A

lex: a]

1 entry(ies) was(were) found.

\li ინფორმაცია ლექსიკონის შესახებ. ამ ბრძანებას გამოაქვს ინფორმაცია ჩატვირთული ლექსიკონის შესახებ. მაგალითად რამდენ სიტყვას შეიცავს მიმდინარე ლექსიკონი. 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. ეს ფაილი ცხადია წინასწარ უნდა იქნეს გამზადებული ჩვენს მიერ და მასში წესები უნდა ჩაწერილი იქნენ შესაბამის (ჩვენს მიერ ზემოთ განხილულ) ფორმალიზმში. ამ ფორმალიზმის უფრო დაწვრილებითი აღწერა განიხილული იქნება ქვემოთ, შესაბამის პარაგრაფში. მაგალითი: \gl sample ან \gl sample.grm პირველ შემთხვევაში, სისტემა თავად დაუმატებს sample ფაილის სახელს '.grm' გაფართოვებას თუკი ეს ფაილი ვერ აღმოაჩინა მიმდინარე კატალოგში.

\sl ენის გრამატიკის შემადგენელი სიმბოლოების სიის გამოტანა. იგულისხმება რომ ფაილიდან 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 მომხმარებლისათვის. მაგალითად:

```
&> \sl
1 = A
2 = B
3 = C
0 = S
4 symbol(s) total.
```

\rl გრამატიკის შემადგენელი წესების სიის გამოტანა. იგულისხმება რომ ფაილიდან.ჩატვირთულია რაიმე გრამატიკა. ამ ბრძანების შედეგად კომპიუტერის ეკრანზე გამოტანილი იქნება ყველა ის წესი, რომლებიც ამ გრამატიკას შეადგენენ. სიმბოლოები მოცემულია მათი უნიკალური ნომრების სახით, და აგრეთვე გლობალურად უნიკალური იდენტიფიკატორებით. 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 ვნახულობთ 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 მნიშვნელობაც უნდა იყოს ერთერთი შემდეგი ჩამონათვალიდან: '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 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)
2: A
[cat: A
lex: a]
3: S
(None)
4: B
[cat: B
lex: b]
5: C
[cat: C
lex: c]

\vl - გამოაქვს სისტემაში 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. ასუფთავებს სისტემაში არსებულ მიმდინარე ცვლადების ცხრილს.

\dp - გამრჩევის გამართვის რეჯიმი. ბრძანება მოითხოვს ერთ პარამეტრს რომლის მნიშვნელობაც შეიძლება იყოს `on' ან `off'. პირველ შემთხვევაში გამართვის რეჯიმი 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 ან გარჩევის ხეთა 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 დრობე ადრე შეწყვეტა და მომხმარებლისთვის გამოტანილი იქნება მხოლოდ ის ნაპოვნი ამონახსნები. თუკი ეს პარამეტრი ნულის ტოლია, მოხდება ყოველგვარი ლიმიტის მოხსნა გამოსატანი გარჩევის ხეების რაოდენობაზე.

\lb - მაქსიმალური ბუჩქის გამოტანა წარუმატებელი გარჩევის შემთხვევაში. 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 იგი ამ დროს 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 შემზღუდველების მექანიზმის კონტროლი. ბრძანება მოითხოვს ერთ პარამეტრს, რომლის მნიშვნელობაც შეიძლება იყოს 'on' ან 'off' და შესაბამისად ააქტიურებს ან ბლოკავს თვისებათა შემზღუდველების მექანიზმის მუშაობას. თუ ეს რეჟიმი ჩართულია (გაჩუმებით იგი ყოველთვის ჩართულია) ხდება წესებთან დაკავშირებული თვისებათა შემზღუდველების (თუ ასეთები არსებობენ) შემოწმება და მიღებული შედეგის გათვალისწინება გარჩევის პროცესში. თუ ეს რეჟიმი გამორთულია წინადადების ანალიზი მოხდება მხოლოდ კონტექსტისაგან თავისუფალი წესების საფუძველზე და ნებისმიერი თვისებათა შემზღუდველი იგნორირებული იქნება.

\fc - კატეგორიის ველის დაყენება. გაჩუმებით კატეგორიის ველს, ანუ ველს რომლის მიხედვითაც განისაზღვრება ლექსიკური ერთეულის კუთვნილება გრამატიკის რომელიმე ტერმინალური სიმბოლოსადმი, აქვს `cat' მნიშვნელობა. ჩვენ 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, მას ემატება თეისებათა შემზღუდველების 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 ადამიანის მიერ. თუკი არ მოხდება ამ წესის გადაყვანა მანქანისათვის მისაღებ ფორმაში, თუკი მასში არსებული ყველა სიმბოლო, სტრიქონული სახით მოცემული, არ გადავა შესაბამის რიცხვით იდენტიფიკატორებში, თუკი არ მოხდება წესის მარჯვენა და მარცხენა მხარის

მოთავსება შესაბამის სტრუქტურებში (მაგ. სიებში), თვისებათა შემზღუდველების 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 იქნას 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 ძალზე ამარტივებენ სხვადასხვა ფორმალიზმების ანალიზატორების გაკეთებას. მათი მეშვეობით საკმაოდ რთული დაპროგრამების ენების კომპილატორების გაკეთებაც კი მეტად გამარტივებულია. ამ ინსტრუმენტების მნიშვნელობის გათვალისწინებით გადავწყვიტეთ მოგვეყვანა მათი ქართულენოვანი დოკუმენტაცია. შესაბამის დანართში შეგიძლიათ იხილოთ მათი აღწერა.

2. გამრჩევის ბირთვი. ძირითადი, აქტიური ლოგიკა სწორედ პროგრამის ამ ნაწილშია თავმოყრილი. გამრჩევის ბირთვს ევალება მოახდინოს კონტექსტისაგან თავისუფალ გრამატიკაში ჩაწერილი ფრაზის სინტაქსური ანალიზი და გზადაგზა მოწმება იმისა აკმაყოფილებს თუ არა თითოეული წესი მასში არსებულ შეზღუდვებს. კონტექსტისაგან თავისუფალი გრამატიკის გარჩევა–სათვის გამოიყენება სტანდარტული ქვემოდან ზემოთ გარჩევის ალგორითმი (იხ. [1]). მოკლედ ავღწეროთ ამ ალგორითმის მუშაობის პრინციპი. გამრჩევი სათითაოდ განიხილავს სიმბოლოებს ერთამენთის მიყოლებით და ათავსებს მათ სტეკში ვიდრე სტეკის თავში მოთავსებული სიმბოლოთა მიმდევრობა არ დაემთხვევა რომელიმე წესის მარჯვენა მხარეში არსებულ სიმბოლოთა მიმდევრობას. დამთხვევის შემთხვევაში იგი

ანაცელებს სტეკში ამ სიმბოლოთა მიმდევრობას ამ წესის მარცხენა მხარეში მდგარ არატერმინალურ სიმბოლოთი. როცა საბოლოო ჯამში სტეკში მივიღეთ ერთადერთ სიმბოლოს და იგი დაემთხვევა საწყის სიმბოლოს, ანალიზატორი ჩათელის 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 შემშღუდველების შემოწმება. ეს ხდება დინამიურად, ინტერპრეტირების რეჟიმში. თუკი თვისებათა შემშღუდველებისაგან შემდგარი გამოსახულება დააბრუნებს ლოგიკურად ჭეშმარიტ მნიშვნელობას მაშინ წესის 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 ითქვას ის თუ რატომ მოხდა მომხმარებლის ინტერფეისის შემთხვევაში არჩევანის გაკეთება მაინცადამაინც ტექსტურ რეჟიმზე. უპირველეს ყოვლისა იმიტომ რომ იგი არის უფრო უნივერსალური და ადვილად გადატანადი სხვადასხვა პლატორმებზე. ვინაიდან დაპროგრამების ენა C++-ის სტანდარ–ტით არ არის გათვალისწინებული გრაფიკულ ბიბლიოთეკებთან მუშაობა, საჭირო ხდებოდა სხვა მწარმოებლების მიერ შემოთა–ვაზებული 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 მრავალი სახის სტრუქტურები (სტეკები, ვექტორები, სიები, ასოციატიური მასივები) კლასების სახით. იგი ძირითადად ორიენტირებულია UNIX და Microsoft Windows-ის ტიპის ოპერაციული სისტემებისთვის თუმცა შესაძლებელია გადატანა ნებისმიერ აპარატურულ თუ პროგრამულ პლატფორმებზე, სადაც მოიძებნება სტანდარტული C++-ის თანამედროვე კომპილატორი (UNIX-თვის GCC 2.95 - Gnu Project Compiler, Windows-თვის VC++ 6.0 - Microsoft Visual Studio).

5.4. მარტივი მაგალითები. ამ თავში განვიხილავთ რამოდენიმე მარტივ მაგალითს, რომლებიც ნათლად გვაჩვენებენ გამრჩევის მუშაობის პრინციპს, მის თავისებურობებს და შესაძლებლობას მოგვცემს გამოვიყენოთ იგი სხვა პრაქტიკული მაგალითებისათვისაც. მაგალითები დალაგებულია სირთულის მიხედვით. თავიდან განვიხილავთ ელემენტარული კონტექს–ტისაგან თავისუფალი გრამატიკის რეალიზაციას. შემდგომ უფრო გავართულებთ მას და დავამატებთ მას თვისებათა შემზღუდველებს.

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 [1])

ჩავწეროთ ეს გრამატიკა ფაილში ex1.grm.

გარდა ამისა, შევქმნათ კიდევ ერთი ფაილი ლექსიკონისათვის ex1.lex და მასში შევიტანოთ შემდეგი ლექსემები:

a [cat: a]

b [cat: b]

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

მას შემდეგ რაც გავამზადებთ ორივე ფაილს, გავუშვათ გამრჩევი და მივცეთ მას ბრძანება:

\\lex1

ამ ბრძანებით მოხდება სისტემაში ლექსიკონის ჩატვირთვა. იმაში დასარწმუნებლად, რომ ლექსიკონი ნამდვილად ჩაიტვირთა, შეგვიძლია მივმართოთ '\li' და '\lf' ბრძანებებს:

&> \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 ცალია ასეთი, რომლებიც ჩვენ ლექსკონის ფაილში განვსაზღვრეთ a და b). მეორე ბრძანება კი მოძებნის კონკრეტულ სიტყვას ლექსიკონში, ამ შემთხვევაში a-ს, და გამოგვიტანს მთელ მის თვისებებს.

ახლა ჩავტვირთოთ გრამატიკის ფაილი შემდეგი ბრძანებით:

\gl ex1

თუკი გრამატიკა შეცდომების გარეშე იქნა ჩაწერილი ბრძანება ჩვეულებრივ შესრულდება. წინააღმდეგ შემთხვევაში გამოვა შეტყობინება თუ რომელ სტრიქონშია შეცდომა და შესაბამისად მოგვიწევს მისი გასწორება. იმისათვის, რომ დავრწმუნდეთ მართლა ჩაიტვირთა თუ არა გრამატიკა 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 გამოვა იმ სიმბოლოთა სია, რომლებიც ადგენენ გრამატიკას. მეორე შემთხვევაში კი ამ გრამატიკის წესები, ისეთნაირად ჩაწერილნი, როგორც არიან გამრჩევის შიდა მანქანურ წარმოდგენაში. მას შემდეგ, რაც ჩაიტვირთება ორივე,

ლექსიკონის და გრამატიკის ფაილი უკვე შეგვიძლია ჩავატაროთ ფრაზის სინტაქსური ანალიზი. ამისათვის, მივცეთ 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, რომელიც ამ შემთხვევაში ცარიელია. ჩვენს მიერ მოცემული ფრაზის გარჩევა წარმატებით დასრულდა. თუკი გარჩევაზე შევიტანთ 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 იმას ნიშნავს რომ ჩვენს მიერ შეტანილი ფრაზის წარმატებული გარჩევა ვერ მოხერხდა. პროგრამასთან მუშაობის დამთავრების შემდეგ ავკრიფოთ სისტემიდან გამოსვლის ბრძანება '\q':

&> \q

Quit.

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

მაგალითი 2.

განვიხილოთ ბუნებრივი ენის მარტივი გრამატიკა:

წინადადება -> არსებითი_სახელი ზმნა;

რომელიც აღწერს წინადადებას როგორც არსებითი სახელი–სა და ზმნის მიმდევრობას. შევნიშნოთ რომ ჩვენი გამრჩევი მუშაობს როგორც ინგლისურ ასევე ქართულ იდენტიფი–კატორებთან. შესაბამისად გრამატიკის და ლექსიკონის ფაილი შეგვიძლია მთლიანად ქართულ ენაზე დავწეროთ. ვთქვათ მოცემულია 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 შევამჩნევთ ორ სიახლეს. პირველი ის, რომ ჩაწერების შესამოკლებლად შემოვიტანეთ ცვლადები. ჩანაწერი: ას = [კატ: არსებითი_სახელი] ნიშნავს რომ ჩვენ განვსაზღვრეთ ცვლადი "ას" როგორც თვისებათა სტრუქტურა, რომელსაც აქვს ერთი ველი სახელად "კატ" და მისი მნიშვნელობაა: "არსებითი_სახელი". შემდგომში ჩვენ ეს ცვლადი შეგვიძლია გამოვიყენოთ სიტყვების

ჩაწერებში. თვისებათა კონსტრუქტორი, ანუ '[' 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 აყენებს სიტემაში არსებულ კატეგორიის და ლექსიკური ერთეულის აღმნიშვნელ სახელებს. გაჩუმებით მათ მნიშვნელობებს შესაბამისად cat და lex წარმოადგენენ. ამის შემდეგ სისტემას შემდეგი გარჩევის ბრძანება მივცეთ:

```
&> 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 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)
```

როგორც ვხედავთ მოცემული წინადადება ამ მარტივი

გრამატიკისათვის დასაშვები აღმოჩნდა, თუმცა ბუნებრივი ენისათვის იგი არაა მისაღები. აქედან გამომდინარეობს, რომ საჭიროა ამ გრამატიკის სრულყოფა, რათა მან შესძლოს გაარჩიოს სუბიექტის და პრედიკატის რიცხვები, ხოლო შემდეგ კი შეათანხმოს ისინი ერთმანეთთან. სწორედ ამ მიზანს ემსახურება ჩვენს მიერ უკვე ნახსენები თვისებათა შემმღუდველები. ლექსიკონში ამ სიტყვების თვისება "რიცხვი" მოცემულია, რაც იმას ნიშნავს, რომ სინტაქსურ ანალიზატორს გარჩევის ეტაპზე ექნება ყველა ის თვისება (მად შორის რიცხვიც), რომელიც საჭიროა ჩვენს მიერ დასმული ამოცანის გადასაწყვეტად. ერთადერთი რაც დაგვრჩა გასაკეთებელი ეს არის ის, რომ გრამატიკის წესში მიეუთითოთ ზმნის და არსებითი სახელის რიცხვში შეთანხმების პირობა. იგი ასეთნაირად შეგვიძლია ჩავწეროთ:

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

წერტილმძიმის მაგიერად ამ წესში გვაქვს ფიგურულ ფრჩხილებში მოთავსებული გამოსახულება, თვისებათა შემმღუდველი. 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 ჩათვლის მას კორექტულად. სამაგიეროდ "ბიჭები მირბიან" წინადადებას სავსებით სამართლიანად 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) მირბიან:3 (ზმნა)
1: წინადადება
(None)
```

ამგვარად, მოხერხდა ამ კონკრეტული პრობლემის მოგვარება თვისებათა შემზღუდველების საშუალებით. იგივე-ნაირად შეგვიძლია გავაუმჯობესოთ გრამატიკა დავუმატებთ რა მას ანალოგიურ შეთანხმებებს ბრუნვაში, რიცხვში და ა.შ.. ასეთნაირად ჩაწერილი შეზღუდვები შეგვიძლია გავაერთიანოთ ერთმანეთთან ლოგიკური ოპერაციების მეშვეობით. თუკი ეს ლოგიკური გამოსახულება გამოთვლის შედეგად იძლევა ლოგიკურად ჭეშმარიტ მნიშვნელობას, მაშინ წესი ჩაითვლება როგორც მისაღები, წინააღმდეგ შემთხვევაში მოხდება წესის უკუგდება. გარდა შეზღუდვების დადებისა, თვისებათა შემზღუდველთა მექანიზმს აქვს კიდევ მეორე დანიშნულება. კერძოთ, თვისებათა გადატანა გარჩევის უფრო მაღალ დონეებზე. ჩვენს შემთხვევაში შეგვიძლია გრამატიკის საწყის სიმბოლო წინადადებას მივანიჭოთ რაიმე თვისება ან თვისებები. 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 გავუშვებთ გარჩევაზე იგივე წინადადებას 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 სიმბოლო "წინადადებას" დაემატა ორი ახალი თვისება და ისინი შეივსნენ პრედიკატში და სუბიექტში არსებული ინფორმაციით რიცხვის შესახებ. თვისებათა გადატანა შესაძლებელია მინიჭების ოპერატორის (':=') გამოყენებით. აგრეთვე უნიფიკაციის ოპერატორის ('<==') გამოყენებით. ეს ოპერატორი ჯერ ახდენს თვისებათა უნიფიკაციას, ხოლო შემდეგ შედეგი გადააქვს მარცხენა მხარეში.

5.5. ლექსიკონის ფაილის სტრუქტურა. ლექსიკონის ფაილი წარმოადგენს გამრჩევის ერთერთ უმთავრეს ფაილს. მასში მოცემულია ლექსიკური ერთეულები. ბუნებრივი ენისათვის ისინი წარმოადგენენ ჩვეულებრივ სიტყვებს. გარდა თავად ლექსიკური ერთეულისა, ლექსიკონში აგრეთვე მოცემულია მისი დამახასიათებელი თვისებები. ბუნებრივი ენისათვის ესენია: 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 არსებობდნენ ლექსიკონის ფაილში არის 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". ამის შემდეგ ჩვენ უკვე შეგვიძლია ნებისმიერ ადგილას სადაც საჭირო იქნება იმის აღნიშვნა, რომ ლექსიკური ერთეული, სიტყვა, დგას სახელობით ბრუნვაში, გამოვიყენოთ ეს ცვლადი:

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

იგი მიეთითება თვისებათა კონსტრუქტორის დასაწყისში და იწერება მრგვალ ფრჩხილებში. ამგვარ, მრგვალ ფრჩხილებში

მოთავსებულ გამოსახულებას ეწოდება კონსტრუქტორის საინი–ციალიზაციო ნაწილი. თუკი საჭიროა, რომ რამოდენიმე ცვლადი გამოვიყენოთ ინიციალიზაციისათვის, ისინი უნდა ჩამოვთვალოთ და ერთმანეთისაგან 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 თვალსაჩინოდ გვიჩვენებს, თუ როგორ ამოკლებს ჩაწერას ასეთი ცვლადების შემოღება. პრაქტიკულად, სულ რამოდენიმე, ყველაზე გავრცელებული შემოკლების (ცვლადის) შემოღებით, მთელი ლექსიკური ერთეულები, იშვიათი გამონაკლისების გარდა, შეგვიძლია მხოლოდ ამ შემოკლებების მეშვეობით ჩავწეროთ.

ლექსიკონის ფაილში შესაძლებელია გვქონდეს ერთიდაიგივე ლექსიკური მნიშვნელობის მქონე სიტყვები. ანუ სიტყვები, რომლებიც ერთნაირად იწერება და მხოლოდ თვისებებით გან–სხვავდებიან. მაგალითად: "ვაშლი" როგორც არსებითი სახელი და "ვაშლი" როგოც ზმნა. ჩვენი ლექსიკონის ფაილში ისინი შემდეგნაირად ჩაიწერებიან:

ვაშლი [
კატ: არსებითი_სახელი
ბრუნვა: სახ
რიცხვი: მხ]
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 ჩვენ სისტემაში ჩავტვირთავთ ამგვარად განსაზღვრულ ლექსიკონს და ამის შემდგომ მივცემთ ბრძანებას '\lf ვაშლი', ჩვენ 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 მათგა-ნისათვის, თუკი მოხერხდება გარჩევის ხის აგება, გამოიტანს ამონახსნს კომპიუტერის ეკრანზე.

ახლა მოვიყვანოთ იმ ფორმალიზმის მკაცრი აღწერა, რომელშიც ხდება ლექსიკონის ფაილის ჩაწერა. რათა ზუსტად გადმოვცეთ ის სინტაქსური კონსტრუქციები, რომლებიც შეადგენენ ამ ფაილის სტრუქტურას. ეს ჩაწერები მოცემულია Bison პროგრამის შემავალი ფაილის ფორმალიზმში. Bison წარმოადგენს მეტად მოხერხებულ პროგრამას, რომელიც საშუალებს გვაძლევს მარტივად გავაკეთოთ სხვადასხვა ტიპის ფორმალიზმების გამრჩევები, მაგ: კონფიგურაციის ფაილების გამრჩევი, დაპროგრამების ენის' კომპილატორი და ა.შ.. ჩვენ შემთხვევაში ასეთ ფორმალიზმს წარმოადგენს ლექსიკონის ფაილის ფორმალიზმი (იხილეთ დანართი 1).

ასეთნაირად ჩაწერილი ფორმალიზმის გრამატიკა შეგვიძლია პირდაპირ გადავცეთ Bison პროგრამას, რომელიც დაგვიბრუნებს დაპროგრამების ენა C-ზე დაწერილ პროგრამულ მოდულს და იგი წარმატებით შეიძლება გამოყენებულ იქნეს ასეთ ფორმალიზ-მებში ჩაწერილი ფაილების ანალიზში.

5.6. გრამატიკის ფაილის სტრუქტურა. გამრჩევის უმთავრეს შემაეალ ფაილს წარმოადგენს გრამატიკის ფაილი. ამ ფაილში იწერება იმ წესთა ერთობლიობა რომლებიც ადგენენ ენას. ფაილის ჩატვირთვის გამრჩევი ანალიზს უკეთებს მას და მასში არსებული წესები გადაჰყავს თავის შიგა წარმოდგენაში. ჩვენ მიმდევრობით აეღწეროთ ის კონსტრუქციები, რომლებისგანაც შედგება გრამატიკის ფაილი. ამისათვის, ჯერ გამოვიყენოთ ბეკუს-ნაურის ფორმალიზმი, ხოლო შემდეგ მოვიყვანოთ Bison-ისთვის გამზადებული ფაილი. ქვემოთ მოცემულია გრამატიკის ფაილის სრული აღწერა კომენტარებით:

<წესი> ::= <თავი> <ტანი> <ბოლო>

<წესი> ::= <თავი> <ტანი> ':'

<მდებარეობათა_მარეგულირებლები> <ბოლო>

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

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

<არატერმინალური_სიმბოლო> ::= <იდენტიფიკატორი>

წესის "თავი" არის მარცხენა მხარეში მდგომი არატერმინალური სიმბოლო, რომელსაც მოსდევს ისარი ('->'). ხოლო არატერმინალური სიმბოლოა ჩვეულებრივი 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 "წინმდგომი" როდესაც ერთი სიმბოლო უშუალოდ მეორეს წინ დგას და ეს ასეთნაირად ჩაიწერება: A - B. მდებარეობათა მარეგულირებელი კონსტრუქციების სიმრავლე შესაძლოა ცარიელიც იყოს, ამ შემთხვევაში განიხილება წესში სიმბოლოთა ყველანაირი მიმდევრობა, ყველა კომბინაცია რომელიც სიმბოლოების გადანაცვლებებით მიიღება.

<ბოლო> ::= ';'

<ბოლო> ::= '{' <შემზღუდველები> '}'

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

<შემზღუდველები> ::= <შემზღუდველი_ოპერაცია>

<შემზღუდველები> ::= '+' < შემზღუდველი_ოპერაცია>

<შემზღუდველები> ::= '-' < შემზღუდველი_ოპერაცია>

<შემზღუდველები> ::= <შემზღუდველები> '&'

<შემზღუდველები>

<შემზღუდველები> ::= <შემზღუდველები> '|'

<შემზღუდველები>

<შემზღუდველები> ::= '~' <შემზღუდველები>

<შემზღუდველები> ::= '(' <შემზღუდველები> ')'

თვისებათა შემზღუდველები წარმოადგენს ლოგიკურ გამოსახულებას. იგი შედგება შემზღუდველი ოპერაციებისაგან, რომელთაც წინ შესაძლოა ჰქონდეთ '+' ან '-' ნიშანი, რაც ნიშნავს იმას, რომ გაჩუმებით ან 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 შეკრება. ოპერაციები დალაგებულია პრიორიტეტების მიხედვით. ამ პრიორიტეტების შესაცვლელად შეგვიძლია გამოვიყენოთ მრგვალი ფრჩხილები.

<შემზღუდველი_ოპერაცია> ::= <ოპერაცია>

<შემზღუდველი_ოპერაცია> ::= <ფუნქციის_გამოძახება>

შემზღუდველი ოპერაცია წარმოიდგინება ორი, ოპერაციის და ფუნქციის გამოძახების სახით. ისინი ერთიმეორეს ეკვივალენტურებია.

<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 ხდება სტანდარტული სახით. ფუნქციის სახელს მოსდევს მრგვალ ფრჩხილებში ჩაწერილი არგუმენტების მიმდევრობა.

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

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

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

ეს არის არგუმენტების სია, რომელიც განისაზღვრება როგორც არგუმენტების ჩვეულებრივი მიმდევრობა ერთმანეთისაგან მძიმეებით გამოყოფილი.

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

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

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

თვისებათა სახელების მიმდევრობა წარმოადგენს ერთიმეორეს მიყოლებით ჩაწერილ თვისებათა სახელებს.

<თვისებათა_კონსტრუქტორი> ::= '[' <თვსიებები> ']'

<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, და ამ ცვლადებიდან მოხდება თვისებების გადმოტანა და დაჯამება მთავარ თვისებათა სტრუქტურაში.

<ინიციალიზაციის_ნაწილი> ::=

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

თვისებათა კონსტრუქტორის ინიციალიზაციის ნაწილი წარმოადგენს ერთმანეთის 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 (ეს

უკვე 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.

იგივე ფორმალიზმი ჩაწერილი Bison-ის ფაილის სახით მოცემულია დანართი 2-ში.

ასეთნაირად გამზადებული ფაილი შეგვიძლია პირდაპირ გადავცეთ Bison-ს და იგი მოგვცემს ამ ფორმალიზმში ჩაწერილი ფაილის გამრჩევს.

5.7. ქართული ენის გრამატიკის ფრაგმენტი. ამ პარაგრაფში განხილულია ქართული ენის გრამატიკის ფრაგმენტი, რომელიც ამუშავებს სამპირიანი ზმნებით შედგენილ წინადადებებს და ენაში არსებულ სხვა კონსტრუქციებს. იგი მოცემულია დანართი 3–ში მცირეოდენი ლექსიკური მარაგით, რომელშიც შედის ენაში არსებული ძირითადი მეტყველების ნაწილები.

```
%{
#include "feature.h"
#define YYSTYPE DICT_YYSTYPE
#define YYPARSE_PARAM DICT_YYPARSE_PARAM
#define YYLEX_PARAM DICT_YYLEX_PARAM
#include "dict.h"
%}
%pure_parser
%token SID
%token OP_UNIFY
%token NONE
%%
input: /* empty */
| input def { }
| input entry { }
| input show_var { }
| input find_word { }
| input unify { }
;
entry: word fsconstr { $$.feat = $2.feat; ENV.lex().addWord($1.str, $2.feat); }
| word error { ENV.error(@1.first_line) << "syntax error in lexical entry definition" << endl; yyerrok; }

word: SID { $$.str = $1.str; }
; pairs:
| pairs name ':' fv { $$.feat = $1.feat; $$.feat.set($2.str, $4.fval); }
```

```
;
fv: NONE { $$.fval.empty(); }
| value { $$.fval.simple($1.str); }
| fsconstr { $$.fval.complex($1.feat); }
;
value: SID { $$.str = $1.str; }

name: SID { $$.str = $1.str; }
;
fsconstr: '[' pairs ']' { $$.feat = $2.feat; }
| '[' '(' init ')' pairs ']' { $$.feat = $3.feat; $$.feat.merge($5.feat); }
| '[' error ']' { ENV.error(@1.first_line) << "syntax error in
feature constructor." << endl; yyerrok; }
| '[' '(' error ')' pairs ']' { ENV.error(@2.first_line) << "syntax
error in feature constructor initialization part." << endl; yyerrok; }
| '[' '(' error ')' error ']' { ENV.error(@2.first_line) << "syntax
error in feature constructor and initialiazation part." << endl;
yyerrok; } ;
init: var { if (ENV.vars().exists($1.str))
if (ENV.vars().value($1.str).isComplex())
$$.feat = ENV.vars().value($1.str).complex();
else
ENV.warn(@1.first_line) << "feature variable \"" << $1.str
<< "\" is not complex (ignoring)."
<< endl;
else
ENV.warn(@1.first_line) << "feature variable \"" << $1.str
<< "\" not defined."
<< endl;
}
| init ',' var { $$.feat = $1.feat;
```

```
    if (ENV.vars().exists($3.str))
    if (ENV.vars().value($3.str).isComplex())
    $$.feat.merge(ENV.vars().value($3.str).complex());
    else
    ENV.warn(@3.first_line) << "feature variable \"" << $3.str
    << "\" is not complex (ignoring)."
    << endl;
    else
    ENV.warn(@3.first_line) << "feature variable \"" << $3.str
    << "\" not defined."
    << endl;
    }

    var: SID { $$.str = $1.str; }
    ;
    def: var '=' fv { ENV.vars().set($1.str, $3.fval); $$.fval = $3.fval;
}
    | var '=' def { ENV.vars().set($1.str, $3.fval); $$.fval = $3.fval; }
    | var '=' error { ENV.error(@1.first_line) << "syntax error in
variable definition." << endl; yyerrok; }
    ;
    show_var: '$' var { if (ENV.vars().exists($2.str))
    ENV.out() << ENV.vars().value($2.str) << endl;
    else
    ENV.out() << "feature variable \""
    << $2.str << "\" not defined."
    << endl;
    }
    ;
    find_word: '?' word {
    FeatureStructList ls = ENV.lex().findall($2.str);
```

```
    FeatureStructList::iterator i;
    if (ls.size() == 0)
    {
    ENV.out() << "lexical entry \"" << $2.str << "\" was not found."
<< endl;
    }
    else
    {
    for (i = ls.begin(); i != ls.end(); i++)
    ENV.out() << *i << endl << endl;
    ENV.out() << "found " << ls.size() << " entry(s)." << endl;
    }
    }

    unify: var OP_UNIFY var {
    if (ENV.vars().exists($1.str))
    {
    if (ENV.vars().exists($3.str))
    {
    if (ENV.vars().value($1.str).unify(ENV.vars().value($3.str)))
    ENV.out() << "unification successful." << endl;
    else
    ENV.out() << "unification failed." << endl;
    }
    else
    ENV.out() << "feature variable \""
    << $3.str << "\" not defined."
    << endl;
    }
    else
    ENV.out() << "feature variable \""
```

```
<< $1.str << "\" not defined."
<< endl;
}
%%
```

დანართი 2

```
%{
#include "rule.h"
#define YYPARSE_PARAM RULE_YYPARSE_PARAM
#define YYLEX_PARAM RULE_YYLEX_PARAM
#define ENV (*((RuleYyEnv* )RULE_YYPARSE_PARAM))
%}
%pure_parser
%union {
IdType *name;
IntType ival;
UnsIntType uival;
FeatureStruct *fs;
FeatureValue *fv;
FeaturePath *fnseq;
SymRec *sym;
OperArg *arg;
OperArgPtrList *args;
struct {
SymbolList *rhs;
SymRecList *srl;
} body;
Symbol *symbol;
struct { SymRecPairList *follow, *prec; } perm;
BoolExp *bexp;
CnstrOper *cop;
RuleIdList *ridl;
}
%{
```

```
    int rule_yylex(YYSTYPE *lvalp, YYLTYPE *llocp, void
*parm);
    %}
    %token <name> ID
    %token <uival> UNSINT
    %token NONE
    %token LARROW
    %token ASSIGN
    %token UNICHECK
    %token UNIFY
    %type <fs> fsconstr
    %type <fs> init
    %type <fs> pairs
    %type <name> fn
    %type <fv> fv;
    %type <fv> var;
    %type <fv> vardec;
    %type <name> varn;
    %type <fnseq> fnseq;
    %type <sym> sym;
    %type <sym> new_sym;
    %type <arg> path;
    %type <arg> arg;
    %type <args> args;
    %type <body> body;
    %type <name> nonterm;
    %type <symbol> head;
    %type <perm> perm;
    %type <bexp> cnstrnt;
    %type <cop> cop;
    %type <cop> op;
```

```
%type <cop> fnctcall;
%type <bexp> tail;
%left '|'
%left '&'
%left '~'
%%
input: { }
| input rule { }
| input vardec { }
;
vardec: var '=' fv { $$ = $1; *$$ = *$3; delete $3; }
| var error { ENV.error(@1.first_line) << "syntax error in variable definition." << endl; YYABORT; }
rule: head body {
$<ridl>$ = new RuleIdList();
ENV.srl($2.srl);
if (find($2.srl->begin(), $2.srl->end(), SymRec(*$1)) != $2.srl->end()) { ENV.error(@1.first_line) << "ambiguity between LHS symbol and some symbol of RHS"
<< endl;
}
else {
ENV.srl()->push_back(SymRec(*$1));
$<ridl>$->push_back(ENV.rules().add(*$1, *($2.rhs)));
}
}
tail {
if ($4 != NULL)
{
for (RuleIdList::iterator i = $<ridl>3->begin(); i != $<ridl>3->end(); i++)
```

```
    ENV.cnstr().add(*i, $4);
    }
    delete $1;
    delete $2.rhs;
    delete $2.srl; delete $<ridl>3;
    }
    | head body ':' perm {
    $<ridl>$ = new RuleIdList();
    ENV.srl($2.srl);
    if (find($2.srl->begin(), $2.srl->end(), SymRec(*$1)) != $2.srl-
>end())
    {
    ENV.error(@1.first_line) << "ambiguity between LHS symbol
and some symbol of RHS"
    << endl;
    }
    else
    {
    ENV.srl()->push_back(SymRec(*$1));
    PermElemSet pes;
    PairSet prec, follow;
    for (PermElem k = 0; k < $2.rhs->size(); k++)
    pes.insert(k);
    for (SymRecPairList::iterator i = $4.prec->begin(); i != $4.prec-
>end();
    i++)
    {
    int first, second;
    first = find_symbol(*($2.rhs), i->first);
    if (first == -1)
    {
```

```
    ENV.error() << "symbol not found: " <<
ENV.symbols().namebyid(i->first.sym)
    << "." << i->first.ord << endl;
    }
    second = find_symbol(*($2.rhs), i->second);
    if (second == -1)
    {
    ENV.error() << "symbol not found: " <<
ENV.symbols().namebyid(i->second.sym)
    << "." << i->second.ord << endl;
    }
    if (first != -1 && second != -1)
    prec.insert(Pair(first, second));
    }
    for (SymRecPairList::iterator i1 = $4.follow->begin(); i1 !=
$4.follow->end();
    i1++)
    {
    int first, second;
    first = find_symbol(*($2.rhs), i1->first);
    if (first == -1)
    {
    ENV.error() << "symbol not found: " <<
ENV.symbols().namebyid(i1->first.sym)
    << "." << i1->first.ord << endl;
    }
    second = find_symbol(*($2.rhs), i1->second);
    if (second == -1)
    {
    ENV.error() << "symbol not found: " <<
ENV.symbols().namebyid(i1->second.sym)
```

```
<< "." << i1->second.ord << endl; }
if (first != -1 && second != -1)
follow.insert(Pair(first, second));
}
PermList pl;
pl = construct_perm(pes, prec, follow);
if (pl.size() == 0)
{
ENV.warn(@1.first_line) << "permutation constraints have
generated empty rule set." << endl;
}
for (PermList::iterator i2 = pl.begin(); i2 != pl.end(); i2++)
{
SymbolList rhs;
for (PermElemList::iterator j = i2->begin(); j != i2->end(); j++)
rhs.push_back((*($2.rhs))[*j]);
$<ridl>$->push_back(ENV.rules().add(*$1, rhs));
} }
}
tail {
if ($6 != NULL)
{
for (RuleIdList::iterator i = $<ridl>5->begin(); i != $<ridl>5-
>end(); i++)
ENV.cnstr().add(*i, $6);
}
delete $1;
delete $2.rhs;
delete $2.srl;
delete $4.follow;
delete $4.prec;
```

```
delete $<ridl>5;
}
;
head: nonterm LARROW { $$ = new Symbol(ENV.symbols().idbylhs(*$1)); delete $1; }
| nonterm error { ENV.error(@1.first_line) << "syntax error (1) in the grammar rule lhs definition." << endl; YYABORT; }
| error LARROW { ENV.error(@1.first_line) << "syntax error (2) in the grammar rule lhs definition." << endl; YYABORT; }
nonterm: ID { $$ = $1; }
;
body: { $$.rhs = new SymbolList();
$$.srl = new SymRecList(); }
| body new_sym { $$.rhs = $1.rhs; $$.srl = $1.srl;
if (find($$.srl->begin(), $$.srl->end(), *$2) != $$.srl->end())
{
ENV.error(@2.first_line) << "ambiguity in RHS symbols (ignoring)" << endl;
}
else
{
$$.rhs->push_back($2->sym);
$$.srl->push_back(*$2);
} delete $2;
}
| body error { ENV.error(@1.first_line) << "syntax error in the grammar rule rhs definition." << endl; YYABORT; }

perm: { $$.follow = new SymRecPairList(); $$.prec = new SymRecPairList(); }
| perm sym '<' sym { $$ = $1; SymRecPair p; p.first = *$2;
```

```
p.second = *$4;
    $$.prec->push_back(p); delete $2; delete $4; }
    | perm sym '-' sym { $$ = $1; SymRecPair p; p.first = *$2;
p.second = *$4;
    $$.follow->push_back(p); delete $2; delete $4; }
    | perm error { ENV.error(@1.first_line) << "syntax error in the
permutation constructor definition." << endl; YYABORT; }
    ;
    tail: ';' { $$ = NULL; }
    | '{' cnstrnt '}' { $$ = $2; }
    | '{' error '}' { ENV.error(@1.first_line) << "syntax error in the
feature constraint definition." << endl; YYABORT; }
    ;
    cnstrnt: cop { $$ = $1; }
    | '+' cop { $2->defvalue(true); $$ = $2; }
    | '-' cop { $2->defvalue(false); $$ = $2; }
    | cnstrnt '&' cnstrnt { $$ = new BoolAndOp($1, $3); }
    | error '&' { ENV.error(@2.first_line) << "syntax error in
constraint boolean expression." << endl; YYABORT; }
    | cnstrnt '|' cnstrnt { $$ = new BoolOrOp($1, $3); }
    | error '|' { ENV.error(@2.first_line) << "syntax error in
constraint boolean expression." << endl; YYABORT; }
    | '~' cnstrnt { $$ = new BoolNegOp($2); }
    | '(' cnstrnt ')' { $$ = $2; }

    cop: op { $$ = $1; }
    | fnctcall { $$ = $1; }

    op: arg ASSIGN arg { OperArgPtrList args; args.push_back($1);
args.push_back($3); $$ = new CnstrOper(ocAssign, args); }
    | arg '=' arg { OperArgPtrList args; args.push_back($1);
```

```
args.push_back($3); $$ = new CnstrOper(ocEqual, args); }
    | arg UNICHECK arg { OperArgPtrList args; args.push_back($1); args.push_back($3); $$ = new CnstrOper(ocUniCheck, args); }
    | arg UNIFY arg { OperArgPtrList args; args.push_back($1); args.push_back($3); $$ = new CnstrOper(ocUnify, args); }
    | arg '=' '(' args ')' { $4->push_front($1); $$ = new CnstrOper(ocMultiEqual, *$4); delete $4; }
    | arg UNICHECK '(' args ')' { $4->push_front($1); $$ = new CnstrOper(ocMultiUniCheck, *$4); delete $4; }
    ;
    fnctcall: ID '(' args ')' { OperCode code;
    if (find_cop(*$1, code)) {
    try {
    $$ = new CnstrOper(code, *$3);
    }
    catch (ArgCountMismatch& e) {
    ENV.error(@1.first_line) << "fatal: argument count does not match required argument count for this function."
    << endl;
    ENV.abort();
    }
    delete $3; }
    else {
    ENV.error(@1.first_line) << "fatal: unknown operation \"" << *$1 << "\"." << endl;
    ENV.abort();
    }
    }
    | ID '(' error ')' { ENV.error(@2.first_line) << "syntax error in function arguments." << endl; YYABORT; }
```

```
arg: var { $$ = new VarFeatureArg($1); }
| path { $$ = $1; }
| fv { $$ = new ConstFeatureArg(*$1); delete $1; }
args: { $$ = new OperArgPtrList(); }
| arg { $$ = new OperArgPtrList(); $$->push_back($1); }
| args ',' arg { $$ = $1; $$->push_back($3); }

path: '<' sym fnseq '>' { if (ENV.find_symrec(*$2)) $$ = new
OperSymArg($2->sym.num(), *$3);
else {
ENV.abort();
}
delete $2;
delete $3;
}
| '<' '$' varn fnseq '>' { $$ = new OperVarArg(*$3, *$4); delete
$3; delete $4; }
;
fnseq: { S$ = new FeaturePath(); }
| fnseq fn { $$ = $1; $$->push_back(*$2); delete $2; }
;
fsconstr: '[' pairs ']' { $$ = $2; }
| '[' error ']' { ENV.error(@1.first_line) << "syntax error in
feature constructor." << endl; YYABORT; }
| '[' '(' init ')' pairs ']' { $$ = $3; $$->merge(*$5); delete $5; }
| '[' '(' error ')' pairs ']' { ENV.error(@2.first_line) << "syntax
error in feature constructor initialization part." << endl; YYABORT;
}
| '[' '(' error ')' error ']' { ENV.error(@2.first_line) << "syntax
error in feature constructor and it's initialization part." << endl;
```

```
 YYABORT; }
      ;
      init: { $$ = new FeatureStruct(); }
      | init varn { $$ = $1;
      if (ENV.vars().exists(*$2))
      if (ENV.vars().value(*$2).isComplex())
      $$->merge(ENV.vars().value(*$2).complex());
      else
      ENV.warn(@2.first_line) << "feature variable \"" << *$2
      << "\" is not complex (ignoring)."
      << endl;
      delete $2; }

      pairs: { $$ = new FeatureStruct(); }
      | pairs fn ':' fv { $$ = $1; (*$$)[*$2] = *$4; delete $2; delete $4;
}
      ;
      fn: ID { $$ = $1; } ;
      fv: NONE { $$ = new FeatureValue(); }
      | ID { $$ = new FeatureValue(*$1); delete $1; }
      | fsconstr { $$ = new FeatureValue(*$1); delete $1; }

      var: '$' varn { $$ = &(ENV.vars().value(*$2)); }

      varn: ID { $$ = $1; }

      new_sym: ID { $$ = new SymRec(); $$->sym =
ENV.symbols().idbyname(*$1); $$->ord = 1; delete $1; }
      | ID UNSINT { $$ = new SymRec(); $$->sym =
ENV.symbols().idbyname(*$1); $$->ord = $3;
      delete $1; }
```

```
    | ID '.' error { ENV.error(@2.first_line) << "syntax error in
symbol definition." << endl; YYABORT; }
    sym: ID { $$ = new SymRec();
    if (!ENV.symbols().symbyname(*$1, $$->sym)) {
    ENV.error(@1.first_line) << "Undeclared symbol \"" << *$1 <<
"\"." << endl;
    ENV.abort();
    }; $$->ord = 1; delete $1;
    }
    | ID '.' UNSINT { $$ = new SymRec();
    if (!ENV.symbols().symbyname(*$1, $$->sym)) {
    ENV.error(@1.first_line) << "Undeclared symbol \"" << *$1 <<
"\"." << $3 << endl;
    ENV.abort();
    }
    $$->ord = $3;
    delete $1; }
    | ID '.' error { ENV.error(@2.first_line) << "syntax error in
symbol definition." << endl; YYABORT; }
    ;
    %%
```

დანართი 3

გრამატიკის ფაილი:

```
# ზჯგ3პ - ზმნის ჯგუფი, მესამე პირი
ზჯგ3პ -> სპნს.1 სპნს.2 ზჯგ სპნს.3 : {
<ზჯგ პირიანობა> == '3' &
<ზჯგ სერია> == '1' &
<სპნს.1 ბრუნვა> == 'სახ' &
<სპნს.1 პირი> == <ზჯგ პირი> &
<სპნს.1 რიცხვი> == <ზჯგ რიცხვი> &
<სპნს.2 ბრუნვა> == 'მიც' &
<სპნს.2 პირი> == <ზჯგ პირდაპირი_ობიექტის_პირი> &
<სპნს.3 ბრუნვა> == 'მიც' &
<სპნს.3 პირი> == <ზჯგ ირიბი_ობიექტის_პირი> &
<სპნს.3 რიცხვი> == <ზჯგ ირიბი_ობიექტის_რიცხვი> &
<ზჯგ3პ სუბიექტი> := <სპნს.1> &
<ზჯგ3პ პრედიკატი> := <ზჯგ> &
<ზჯგ3პ ობიექტი_1> := <სპნს.2> &
<ზჯგ3პ ობიექტი_2> := <სპნს.3>
}
ზჯგ3პ -> სპნს.1 სპნს.2 ზჯგ სპნს.3 : {
<ზჯგ პირიანობა> == '3' &
<ზჯგ სერია> == '2' &
<სპნს.1 ბრუნვა> == 'მოთხრ' &
<სპნს.1 პირი> == <ზჯგ პირი> &
<სპნს.1 რიცხვი> == <ზჯგ რიცხვი> &
<სპნს.2 ბრუნვა> == 'სახ' &
<სპნს.2 პირი> == <ზჯგ პირდაპირი_ობიექტის_პირი> &
<სპნს.3 ბრუნვა> == 'მიც' &
```

```
    <სპნს.3 პირი> == <ზჯგ ირიბი_ობიექტის_პირი> &
    <სპნს.3 რიცხვი> == <ზჯგ ირიბი_ობიექტის_რიცხვი> &
    <ზჯგ3პ სუბიექტი> := <სპნს.1> &
    <ზჯგ3პ პრედიკატი> := <ზჯგ> &
    <ზჯგ3პ ობიექტი_1> := <სპნს.2> &
    <ზჯგ3პ ობიექტი_2> := <სპნს.3>
    }
    # სპნს - სახელი ან პირის ნაცვალსახელი
    სპნს -> სჯგმ { <სპნს> := <სჯგმ> }
    სპნს -> პნს { <სპნს> := <პნს> }
    # ზჯგ - ზმნის ჯგუფი
    ზჯგ -> ზ { <ზჯგ> := <ზ> }
    ზჯგ -> ზზ ზჯგ.2 : { <ზჯგ> := <ზჯგ.2> }
    ზჯგ -> ზს ზჯგ.2 : { <ზს ბრუნვა> = 'ვით' & <ზჯგ> := <ზჯგ.2>
}
    # სჯგმ - სახელი ჯგუფი მართვა
    სჯგმ -> სჯგმ.2 სჯგმ.3 { <სჯგმ.2 ბრუნვა> == 'ნათ' & <სჯგმ>
:= <სჯგმ.3> }
    სჯგმ -> სჯგ { <სჯგმ> := <სჯგ> }
    # სჯგ - სახელის ჯგუფი
    სჯგ -> ატ სჯგ.2
    {
    ( <სჯგ.2 ბრუნვა> = ('სახ', 'ნათ', 'მოქმ') & <ატ ბრუნვა> =
'სახ_ნათ_მოქმ' |
    <სჯგ.2 ბრუნვა> = ('მიც', 'ვით') & <ატ ბრუნვა> = 'მიც_ვით' |
    <სჯგ.2 ბრუნვა> == <ატ ბრუნვა>
    ) &
    <სჯგ> := <სჯგ.2>
    }
    სჯგ -> ას { <სჯგ> := <ას> }
    # ატ - ატრიბუტი
```

ატ -> მს { <ატ ბრუნვა> := <მს ბრუნვა> }
ატ -> რს { <ატ ბრუნვა> := <რს ბრუნვა> }
ატ -> ნს { <ატ ბრუნვა> := <ნს ბრუნვა> }
ატ -> მიმ { <ატ ბრუნვა> := <მიმ ბრუნვა> }
ლექსიკონის ფაილი:
მს = [კატ: მს]
ას = [კატ: ას]
მიმ = [კატ: მიმ]
მმ = [კატ: მმ]
მ = [კატ: მ]
პნს = [კატ: პნს]
მრ = [რიცხვი: მრ]
მხ = [რიცხვი: მხ]
სახ = [ბრუნვა: სახ]
მოთხრ = [ბრუნვა: მოთხრ]
მიც = [ბრუნვა: მიც]
ნათ = [ბრუნვა: ნათ]
ვით = [ბრუნვა: ვით]
მოქმ = [ბრუნვა: მოქმ]
წოდ = [ბრუნვა: წოდ]
მიც_ვით = [ბრუნვა: მიც_ვით]
სახ_ნათ_მოქმ = [ბრუნვა: სახ_ნათ_მოქმ]
პ1 = [პირი: 1]
პ2 = [პირი: 2]
პ3 = [პირი: 3]
პნ1 = [პირიანობა: 1]
პნ2 = [პირიანობა: 2]
პნ3 = [პირიანობა: 3]
ს1 = [სერია: 1]
ს2 = [სერია: 2]
ს3 = [სერია: 3]

პოპ1 = [პირდაპირი_ობიექტის_პირი: 1]
პოპ2 = [პირდაპირი_ობიექტის_პირი: 2]
პოპ3 = [პირდაპირი_ობიექტის_პირი: 3]
იოპ1 = [ირიბი_ობიექტის_პირი: 1]
იოპ2 = [ირიბი_ობიექტის_პირი: 2]
იოპ3 = [ირიბი_ობიექტის_პირი: 3]
იორმხ = [ირიბი_ობიექტის_რიცხვი: მხ]
იორმრ = [ირიბი_ობიექტის_რიცხვი: მრ]
დრა = [დრო: აწმყო]
დრწ = [დრო: წარსული]
დრმ = [დრო: მომავალი]
ცნობილ [(მს, მიც_ვით)]
ცნობილი [(მს, სახ_ნათ_მოქმ)]
ცნობილად [(მს, ვით)]
ქართველ [(მს, მიც_ვით)]
ქართველი [(მს, სახ_ნათ_მოქმ)]
მშენებლებს [(ას, მიც, მრ, პ3)]
მშენებლები [(ას, სახ, მრ, პ3)]
მშენებელი [(ას, სახ, მხ, პ3)]
მშენებელმა [(ას, მოთხრ, მხ, პ3)]
გაკეთებული [(მიმ, სახ_ნათ_მოქმ)]
მათემატიკური [(მს, სახ_ნათ_მოქმ)]
ანალიზის [(ას, ნათ, მხ)]
მკაცრი [(მს, სახ_ნათ_მოქმ)]
კურსის [(ას, ნათ, მხ, პ3)]
აგება [(ას, სახ, მხ, პ3)]
წინ [(მმ)]
უკან [(მმ)]
სახლი [(ას, სახ, მხ, პ3)]
სახლს [(ას, მიც, მხ, პ3)]
მეგობარს [(ას, მიც, მხ, პ3)]

მე [(პნს, მხ, პ1)]
ჩვენ [(პნს, მრ, პ1)]
შენ [(პნს, მხ, პ2)]
თქვენ [(პნს, მრ, პ2)]
ის [(პნს, სახ, მხ, პ3)]
ისინი [(პნს, სახ, მრ, პ3)]
მას [(პნს, მიც, მხ, პ3)]
მათ [(პნს, მიც, მრ, პ3)]
უშენებს [(ზ, პ3, ს1, მხ, პოპ3, იოპ3, იორმხ, პნ3, დრა)]
გვიშენებს [(ზ, პ3, ს1, მხ, პოპ3, იოპ1, იორმრ, პნ3, დრა)]
ვუშენებ [(ზ, პ1, ს1, მხ, პოპ3, იოპ3, იორმხ, პნ3, დრა)]
აუშენა [(ზ, პ3, ს2, მხ, პოპ3, იოპ3, იორმხ, პნ3, დრწ)]
ავუშენე [(ზ, პ1, ს2, მხ, პოპ3, იოპ3, იორმხ, პნ3, დრწ)]

გამოყენებული ლიტერატურა:

[1]. A.Aho, J.Ullman. The Theory of Parsing, Translation and Compiling, vol. 1, Prentice-Hall, N. J., 1972

[2]. R.Montague. The Proper Treatment of Quantification in Ordinary English, in K.J.J.HintikkaJ.M.E.Moravcsik and P.Suppes(eds), 'Approach to Natural Language', Synthese Library 49, Reidel, Dordrecht, 1973.

[3]. G.Gazdar, E.Klein, G.Pullum, I.Sag. Generalized Phrase Structure Grammar, Oxford University Press, Oxford, 1985

[4]. G.Erbach. A Bottom-up Algorithm for Parsing and Generation, CLAUS report, number 5, Saarland University, 1991

[5]. M.Kay. Algorithm Schemata and Data Structure in Syntactic Processing, Report SLS-12-80, Palo Alto, 1980

[6]. A.Joshi. Tree-Adjoining Grammars, in The encyclopedia Language and Linguistics, R.E.Asher(ed.), Pergamon Press, Oxford, UK, 1994

[7]. A.Aho. Nested Stack Automata, Journal of the ACM, Vol. 16, Issue 3, 1969

[8]. R.Sharp. CAT2: An Experimental EUROTRA Alternative, Machine Translation, Volume 6, Number 3, Springer Netherlands, 1991

[9]. D.Arnold. Eurotra: a European Perspective on MT in: Proceedings of the IEEE, Vol 74, No 7, Special Issue on Natural Language Processing, 1986, pp. 979—992

[10]. R.Jonson, M.Rosner. A Rich Environment for Experimentation with Unification Grammars, in Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics

[11]. R.Kasper. A Unification Method for Disjunctive Feature

Descriptions, in Proceedings of the 25 Annual Meeting of the Association for Computational Linguistics

[12]. R.Kaplan, J.Bresnan. A Formal System for Grammatical representation , in Formal Issues in Lexical functional Grammar, CSLI Lecture Notes, N47

[13]. M.Kay. Nonconcatenative finite-state morphology, European Chapter Meeting of the ACL Proceedings of the third conference on European chapter of the Association for Computational Linguistics, Copenhagen, Denmark, 1987

[14]. G.Pollard, I.Sag. Head Driven Phrase Structure Grammar, CSLI and University Chicago Press, Stanford/Chicago, 1994

[15]. H.Camp, U.Reyle. From Discourse to Logic, Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Kluwer, Dordrecht,1993

[16]. B.Partee. Montague Grammar and Transformational Grammar, Linguistic Inquiry 6, 1975.

[17]. R.Thomason(ed.), Formal Philosophy. Selected Papers of Richard Montague, Yale University Press, New Haven, 1974

[18]. J.J.Katz, J.A.Fodor.The structure of a semantic theory (1963)

[19]. E.Keenan, L. Faltz. Logical Types for Natural Language, UCLA Occasional Papers in Linguistics, 1978

[20]. T.M.V.Jansen. Foundations and Applications of Montague Grammar, Part 2: Application to Natural Language, CWI Tract 28, 1988

[21]. H. Kamp. A Theory of Truth and Semantic Representation, inJ. Groenendijk, Th. Janssen, and M. Stokhof, editors, Formal Methods in the Study of Language, Mathematisch Centrum,Amsterdam, 1981

[22]. I.Heim. The Semantics of Definite and Indefinite Noun Phrases, PhD Thesis, MIT, 1982.

[23]. C.Morris. Foundations of the Theory of Signs, in International

Encyclopedia of Unified Science 1, 1938

[24]. A.Colmerauer. Prolog in 10 figures, Communication of the ACM, Volume 28, Issue 12, 1985

[25]. S.Vadera, F.Maziane. From English to Formal Specification, The Computer Journal, 37, 9, 1994

[26]. Автоматизированная обучающая система ЧОК. М., 1982.

[27]. Дж.Г.Антидзе,Н.Н.Джгаркава, Э.А.Иорданашвили, Т.Г.Саженюк, Б.Д.Хасия, К.Пурцеладзе, М.М.Чабашвили. Инструментальное средство для составления автоматизированных обучающих курсов, Отчет контракта ИПМ ТГУ, 1991.

[28]. N. Haas, G.G. Hendrix. Learning by being told, in R.S. Michalaski, J.G.Garbonell, T.M. Mitchell, eds., Machine learning,Tioga Publislino, Co.,1983..

[29]. Wilensky, Chin D.N., Luria M., Mayfield J., D.Wu. The Berkeley Unix Consultant Project, Computational lingustics, 14 : 4,1982.

[30]. J.Fargueiras, M.C. Landau, A.Duguord, L.Catach. Conceptual graph for semantics and knowledge processing, IBM Journal of Research and Development, 30:1, 1986.

[31]. J.F. Sowa. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley,1984.

[32]. C.Beirle, J. Dorre, U. Pletat, C. Rollinger, P. Schmitt, R.Sfuder. The Knowledge Representation Language L_{LILOG} CSL 88,1989

[33]. O. Herzog, et al.. L_{LILOG} Linguistic and Logic Methods for the Computational understanding of German, L_{LILOG} Report lb, IBM Germany, 1986.

[34]. J.G.Schmolze, T.A.Lipkis. Classification in the KL-ONE Knowledge Representation System, in: Proc. IJCAI, 1983.

[35]. M.Vilain. The Restricted Language Architecture of a Hybrid

Representation System, in: Proc. IJCAI,1985.

[36]. R.J. Brachmann, V.P.Gilbert, H.J.Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON, in: Proc. IJCAI, 1985.

[37]. K.von Luck . Semantic Networks with Number Restricted Roles or Another Story about Clyde, in: Proc. GWAI, 1986.

[38]. F.Di Primio, G.Bewka. Babylon: Kernel System of an Integrated Environment for Expert System Development and Operation, in: Proc. 5th International Workshop on Expert Systems and their Applications, Avignon, 1985.

[39]. J. Antidze. N.Gulua. Analysis of Georgian text for construction of teaching system. Reports of Enlarged Session of the Seminar of I.Vekua Institute of Applied Mathemetics Vol.13, #4,1998.

[40]. Дж. Антидзе. Экспериментальный алгоритм машинного перевода с грузинского языка на русский, диссертация на соискание ученой степени кандидата физ.-мат. наук,ТГУ,1966.

[41]. G.Chikoidze.Net Representation of Inversible Morfologic Processor, Second Tbilisi Simposium "Language, logic and Computation, Tbilisi, 1997.

[42]. L.Margvelani, I..Samsonadze, N. Javashvili. Computer Aid for Georgian Morphology Teaching, Second Tbilisi Symposium "Language, Logic and Computation", Tbilisi,1997.

[43]. L.Chkaidze, T.Kvantaliani, T Kvinikadze. Electronic Dictionary of Georgian Verb Stems. Second Tbilisi Symposium "Language, Logic and Computation", Tbilisi,1997.

[44]. EDR Electronic Dictionary, Technical Guide, Japan Electronic Dictionary Research Institute, Ltd,Tokio,1993.

[45]. M.Gross. la construction de dictionaires electroniques, Analles des telecommunications, Tome 44, N1-2,1989.

[46] ა.შანიძე. ქართული ენის გრამატიკა, I მორფოლოგია, თბილისი, 1955.

[47]. ჯ. ანთიძე. ლექსიკონის აგებულება ქართული ენიდან მანქანური თარგმნისათვის, საქართველოს მეცნიერებათა აკადემიის მოამბე, XXXI:2,1963.

[48]. გ. გოგოლაშვილი, ც. კვანტალიანი, დ. შენგელია. ქართული ზმნის ძირების ლექსიკონი, თბილისი, 1989.

[49]. J.et J.P.Caput.Dictionaire des verbs francais, Paris, 1969.

[50]. G. Gazdar, K. Pullum, I. Sag. Generalized Phrase Stucture ,Oxford,1985.

[51]. N. Chomsky. Semantic Structures, The Hague : Mouton,1957

[52]. N.Gulua. Formalized description of Georgian texts, the software for texts processing and its application for construction of a teaching system, PHD dissertation, Tbilisi, 1999.

[53]. J.Antidze, N. Gulua, On selection of Georgian texts computer analysis formalism, Communications of the Georgian Academy of Sciences, 162, N2, 2000

[54]. 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.

[55]. N. Gulua, On the construction of Georgian interactive teaching system, Reports of enlarged session of the seminar of IAM TSU, vol.13, N4, 1998

[56]. J.Antidze, N.Gulua, On the method for construction of a teaching systems, Communications of Sukhumi branch of Tbilisi State University, N1, 1998.

[57]. N.Gulua. On Computer Syntactical Analysis of Georgian Texts, Communications of the Georgian Academy of Sciences, N3, 1999.

[58]. D. Kaiser. La semantique des langues naturelles et les logiques, Annales des Telecommunications, Tome 44, N1-2,1989.

[59]. John F.Sowa. Semantic Networks, http://www.sowa.com

[60]. Shank, Roger C.. Conceptual Information Processing, Amsterdam,1975.

[61]. Shapiro, Stuart C, W. Rapoport. The SNePS family, In Semantic Networks in Artificial Intelligence, Oxford,1992.

[62]. H.Kamp, U. Reyle. From Discourse to Logic, Dordrecht, 1993.

[63]. J.Antidze, d. Mishelashvili. Software Tools for Natural language Texts processing, VI international Tbilisi Symposium: Language, Logic and Computation, Batumi, 2005

[64]. J.Antidze, d. Mishelashvili. Software Tools for morphological and Syntactic Analyses of Natural Language Texts, Proceedings of the conference III _ Georgian Language and Computer Technologies, Tbilisi, 2005.

[65]. Reale, Stephen, Sergei Nirenburg and Gavi Maheseh. Semantic Analysis in the Microcosms Machine Translation Project, in SNLP_ 95, Bangkok.

[66]. V. Raskin, S. Nirenburg. Lexical Semantic of Adjectives – A Micro-theory of Adjectival Meaning, Computing Research Laboratory, New Mexico State University, Las Cruces (http://www.crl.nmsu.edu).

[67]. J.Thorne,P.Bratley and H.Dewar. The syntactic analysis of English by machine, Machine Intelligence 3, New York, 1968.

[68]. D.Robrow and B.Fraser. An Augmented State Transition Network Analysis Procedure, Proceeding International Joint Conference an Artificial Intelligence, Washington, 1969.

[69]..W.A.Woods. Transition network grammars for natural language analysis, CACM, 13, 1970.

[70]..W.A.Woods. AN Experimental Parsing System for Transition Network Grammars, Massachusetts, 1972.

[71]..S.M.Chou and K.S. FU.. Transition Network for pattern Recognition, Tech.Rept.TR-EE 75-39, Purdue university,

1975.

[72]. N.Gulua. On the construction of Georgian interactive teaching system. Reports of Enlarged Session of the Seminar of I.Vekua Institute of Applied Mathemetics Vol.13, #4, 1998.

[73]. ჯ.ანთიძე, ნ.გულუა. მასწავლებელი სისტემის აგებისადმი ერთი მიდგომის შესახებ. თბილისის სახელმწიფო უნივერსიტეტის სოხუმის ფილიალის მოამბე, 1, 1998.

[74].F.Pereira and Warren. Definitive clauses Grammars for language analysis, Artificial Intelligence, 13(3), 1980.

[75]. M.MeCord. Using slots and Modifiers in Logic Grammars for Natural Language, Artificial Intelligence 18(3), 1982.

[76]. J.Pitrat. Un langage pour decrire les connaissances de facon declarative, publication 30, Universite Paris VI, 1982.

[77]. J.Antidze, D.Mishelashvili. Morphological and Syntactic Analysis of Natural Language Texts, Internet Academy, Georgian Electronic Scientific Jurnals: Computer Sciences and Telecommunications, 1(12), 2007, IISN 1512-1232, (in English). http://gesj.internet-academy.org.ge/gesj_articles/1345.pdf

[78]. J.Antidze, D.Mishelashvili. Software Tools for Morphological and Syntactic Analysis of Natural Language Texts, Report of VI Tbilisi International Symposium "Language, Logics and Computation", Batumi, September, 2006, http://fpv.science.tsu.ge/pub1.mht http://www.viam.science.tsu.ge/sysprg/pub_sia_2007.mht

[79]. J.Antidze, D.Mishelashvili. Software Tools for Morphological and Syntactic Analysis of Some Natural

Language's texts, Report of Symposium – Natural Language Processing, Georgian Language and Computer Technologs, Institute of Linguistics of Georgian Academy of Sciences, Tbilisi, 2005. http//www.ice.ge/conferenciebi/Conf_Fs.html

[80]. J.Antidze, D.Mishelashvili. Morphological Analyzer of a Natural Language, Symposium – Natural Language Processing, Georgian Language and Computer Technologies, Tbilisi, 2004. http://fpv.science.tsu.ge/pub13.mht http//www.ice.ge/conferenciebi/Conf_Fs.html

[81]. J.Antidze, D.Mishelashvili. nstrumental tools for Computer Processing of Georgian Texts, Symposium – Natural Language Processing, Georgian Language and Computer Technologies, Tbilisi, 2003. http://fpv.science.tsu.ge/pub14.mht http//www.ice.ge/conferenciebi/Conf_Fs.html_

[82]. J.Antidze. Program for Georgian verbs decomposition in morphemes, Report of I Georgian mathematicians' congress, Tbilisi, 1994 (in Georgian).

სარჩევი

მათ ეწოდებათ ტერმინალური სიმბოლოები. ყოველი განსამარტავი გრამატიკული კატეგორია ავღნიშნოთ რაიმე სიმბოლოთი. მაგალითად წინადადება S-ით, სახელის ჯგუფი - P-თი და ზმნის ჯგუფი - VP-თი. მაშინ წესი

S→VP

აღნიშნავს რომ წინადადება არის ზმნის ჯგუფი, ხოლო წესი VP→ np& vp1& nP (1)

აღნიშნავს, რომ ზმნის ჯგუფი შედგება სუბიექტის ჯგუფისაგან, ორადგილიანი პრედიკატის და პირდაპირი ობიექტის ჯგუფისაგან სიღრმისეულ დონეზე, ხოლო ზედაპირულ დონეზე გამოსახულია სახელის ორი ჯგუფით და ზმნის ჯგუფით, რომელთაც გააჩნიათ თავიაანთი დამახასიათებელი ფორმალური ნიშნები, რომლებიც განსაზღვრავენ გრამატიკულად სწორად შედგენილ წინადადებას, ხოლო სიღრმისეულ დონეზე ჯგუფები უნდა იყვნენ ერთიმეორესთან თავსებადნი, რომ მოგვცეს აზრობრივად სწორი წინადადება. თავსებადობა განისაზღვრება ჯგუფის სემანტიკური ნიშნებით და გამოისახება სემანტიკური წესის საშუალებით. ამგვარად, წინადადების გრამატიკულად სწორად გაფორმება გამოისახება სინტაქსური წესებით, ხოლო მის აზრობრივ სისწორეს განსაზღვრავს შესატყვისი სემანტიკური წესი. ჩვენ აქ არ შევეხებით სემანტიკურ წესებს და დავკმაყოფილდებით მხოლოდ სინტაქსური წესების განხილვით. დავუბრუნდეთ (1) წინადადებას. იმისათვის, რომ გამოვსახოთ სინტაქსურად სწორი წინადადების კერძო სახეობა, დავაზუსტოთ იგი იმ ნიშნებით, რომლებიც მოითხოვება ორპირიანი გარდამავალი ზმნის ფორმებისათვის. ეს ნიშნები კარგადაა ცნობილი ქართული გრამატიკის სახელმძღვანელოებში. კონკრეტულად, თუ ზმნის ფორმა პირველი სერიიდანაა, მაშინ პირველი სახელის ჯგუფის მთავარი სახელი უნდა იყოს სახელობით ბრუნვაში (შეესაბამება სუბიექტის ჯგუფს სიღრმისეულ დონეზე), ხოლო მეორე სახელის ჯგუფის მთავარი სახელი