ლამბდა აღრიცხვის გამოყენება კლასიკურ და კვანტურ კომპიუტინგში

გურამ კაშმაძე
კომპიუტერულ მეცნიერებათა დეპარტამენტი. ივ. ჯავახიშვილის სახ. თბილისის სახელმწიფო უნივერსიტეტი. თბილისი. უნივერსიტეტის ქ. 13. gkashmadze@yahoo.com

ანოტაცია

სტატიაში გადმოცემულია მოვლენები, რომლებმაც განაპირობეს ლამბდა აღრიცხვის შექმნა. აღწერილია ლამბდა აღრიცხვის სინტაქსი და სემანტიკა. მოცემულია დაპროგრამების ენებში ლამბდა გამოსახულებების გამოყენების თავისებურებები. დახასიათებულია ტოპოლოგიური, დიფერენციალური, ალბათური და კვანტური ლამბდა აღრიცხვები.

საკვანძო სიტყვები: *ლამბდა აღრიცხვა, სინტაქსი, სემანტიკა, ტერმი, რედექსი, სახელით გამოძახება, მნიშვნელობით გამოძახება.*

Abstract

In this paper we discuss events that led to the lambda calculus. Syntax and semantics of the lambda calculus are described. Applications of lambda expressions to the programming languages are given. Topological, differential, probabilistic and quantum lambda calculi are characterized.

Keywords: *lambda calculus, syntax, semantics, term, redex, call-by-name, call-by-value.*

Аннотация

В статье приведены причины обусловившие создание лямбда исчисления. Показаны особенности применения лямбда выражений в языках программирования . Охарактеризованы топологическое, дифференциальное, вероятностное и квантовое лямбда исчисления.

Ключевые слова: *лямба исчисление, синтаксис, семантика, редекс, вызов по имени, вызов по значению*

შესავალი

ალგორითმი რაიმე მიზნის მიღწევისათვის საჭირო პროცედურაა. ალგორითმის უძველესი მაგალითებია ორი რიცხვის უდიდესი საერთო გამყოფის განსაზღვრის მეთოდი, რომელიც ევკლიდეს სახელს უკავშირდება; აგრეთვე მარტივი რიცხვების ძიების ერატოსთენეს მეთოდი (ერატოსთენეს ცხაური). საყოველთაოდ ცნობილი წესები რიცხვების ქვეშმიწერით შეკრებისა თუ გამრავლებისა აგრეთვე ალგორითმების მაგალითებია. ტერმინი „ალგორითმი” მომდინარეობს მეცხრე საუკუნის სპარსი მათემატიკოსის, ასტრონომისა და გეოგრაფის, მუჰამად ბენ მუსა ალ-ხვარაზმის სახელის ლათინური ვერსიიდან Algorithmi.

ა. მალცევი[1] ალგორითმს შემდეგი თვისებებით ახასიათებს:

- ალგორითმი დისკრეტულ დროში, ნაბიჯ-ნაბიჯ მიმდინარე პროცესია, რომლის დასაწყისში მოცემული გვაქვს მონაცემთა სასრული სისტემა და ყოველ მომდევნო

ნაბიჯზე მონაცემთა სისტემა წინა ბიჯზე არსებული სისტემიდან მიიღება გარკვეული წესების მიხედვით (ალგორითმის დისკრეტულობა).

- ყოველ ნაბიჯზე აგებული სიდიდეების სისტემა ცალსახად განისაზღვრება ადრე აგებული მონაცემებით (ალგორითმის დეტერმინირებულობა).
- წესები, რომლებიც განსაზღვრავენ სიდიდეთა ახალ სისტემას, მარტივი, ლოკალური და ყველასთვის ერთნაირად გასაგები უნდა იყოს (ალგორითმის ბიჯების ელემენტარულობა).
- წინასწარ უნდა იყოს დათქმული - რა ჩაითვალოს ალგორითმის მუშაობის შედეგად იმ შემთხვევაში, როდესაც მოცემული წესები რომელიმე ნაბიჯზე საძიებელ სიდიდეებს არ გვაძლევენ (ალგორითმის მოგეზულობა შედეგის მიღებისკენ).
- ალგორითმის საწყისი მონაცემების არჩევა შესაძლებელი უნდა იყოს პოტენციურად უსასრულო სიმრავლიდან (ალგორითმის მასობრიობა).

დონალდ კნუტის [2] მიხედვით ალგორითმის დამახასიათებელი ნიშნებია:
1) სასრულობა; 2) განსაზღვრულობა; 3) შესავალი; 4) გამოსავალი; 5) ეფექტურობა.
ალგორითმის ეფექტურობას დონალდ კნუტი ასე განმარტავს: ალგორითმით გათვალისწინებული ოპერაციები საკმარისად ელემენტარული უნდა იყოს, რათა ადამიანს გააჩნდეს პრინციპული შესაძლებლობა ფანქრისა და ქაღალდის გამოყენებით დროის სასრულ შუალედში ზუსტად შეასრულოს ისინი.

ალგორითმის ცნება, რომელიც ზემოთ ჩამოთვლილი თვისებებით განისაზღვრება, არაზუსტი, ინტუიციური ცნებაა, რადგან ამ თვისებების აღწერა შეიცავს ისეთ სიტყვებს, რომელთა ზუსტი მნიშვნელობა გაურკვეველია. ამის გამო ალგორითმის ინტუიციური ცნების გამოყენება მათემატიკურ მტკიცებებში შეუძლებელია. უზუსტობის მიუხედავად, ალგორითმის ინტუიციური ცნება პრაქტიკულად გვაძლევს საშუალებას დავადგინოთ ჩვენს ხელთ არსებული პროცედურა არის თუ არა ალგორითმი.

მეოცე საუკუნის პირველ ნახევარში გამოიკვეთა პრობლემები, რომელთა ამოხსნა ვერ ხერხდებობდა გამოჩენილი მათემატიკოსების ხანგრძლივი მცდელობის მიუხედავად. ბუნებრივად წარმოიშვა ეჭვი, რომ ამოხსნის ალგორითმი არ არსებობს. დამტკიცებისთვის კი საჭირო იყო ალგორითმის ცნების დაზუსტება. ამ მიმართულებით მუშაობდნენ ალონზო ჩორჩი[3][4], ალან ტიურინგი[5][6], ემილ პოსტი[7], სტივენ კლინი[8].

§1 ლამბდა-აღრიცხვა

ალონზო ჩორჩმა 1932-36 წწ. ჩამოაყალიბა ლამბდა აღრიცხვა და დოკუმენტურად გააფორმა იგი 1941 წელს გამოქვეყნებულ ნაშრომში[9]. ლამბდა აღრიცხვა არის გამოთვლების უნივერსალური მოდელი - ფორმალური სისტემა მათემატიკურ ლოგიკაში, რომელიც მათემატიკას ფუნქციონალურ საფუძვლად უნდა დასდებოდა. თუმცა მათემატიკის საფუძვლად მაინც სიმრავლეთა თეორია არჩიეს. დღევანდელი გადასახედიდან ლამბდა აღრიცხვა მეოცე საუკუნის ლოგიკის ერთერთ უდიდეს აღმოჩენად ითვლება[10].

ლამბდა-აღრიცხვა შეიცავს ლამბდა ტერმების ენას და ტერმების გარდაქმნის წესებს.

ლამბდა ტერმების სიმრავლე ინდუქციის მეთოდით განისაზღვრება:

1. ცალკე აღებული ნებისმიერი x ცვლადი არის ლამბდა ტერმი // პირდაპირი პუნქტი.

ლამბდა აღრიცხვაში ცვლადი გაიგება როგორც ზოგადი ფუნქცია.

2. თუ t ლამბდა ტერმია, ხოლო x ცვლადია, მაშინ $(\lambda x.t)$ არის ლამბდა ტერმი, რომელსაც ლამბდა განზოგადოება (lambda abstraction) ეწოდება. // ირიბი პუნქტი.

განისაზღვრება უსახელო ფუნქცია ერთადერთი x ფორმალური არგუმენტითა და t ტანით. "განზოგადოებულია" t გამოსახულებით აღწერილი გამოთვლების წესი, რომელიც შეიძლება ჩატარდეს ნებისმიერ x შესავალ სიდიდეზე. ესაა ფუნქციის დეკლარირება - განსაზღვრა გამოძახების გარეშე.

მრავალცვლადიანი ფუნქცია შესაძლებელია ტრანსფორმირებულ იქნას ერთცვლადიანი ფუნქციების შემცველ გამოსახულებად. ტრანსფორმაციის პროცესს კარინგი ეწოდება ჰასკელ კარის (Haskell Curry) გვარის მიხედვით.

3. თუ t და u ლამბდა ტერმებია, მაშინ (tu) არის ლამბდა ტერმი, რომელსაც გამოყენება (application) ეწოდება. // ირიბი პუნქტი.

იგულისხმება, რომ t არის ფუნქცია ან ოპერატორი, რომლის არგუმენტს მიენიჭა u მნიშვნელობა. ჩვეულებრივ, ეს გამოსახულება ჩაიწერება $t\,(u)$ სახით. სხვაგვარად რომ ვთქვათ, ესაა t ტერმით განსაზღვრული ფუნქციის გამოძახება მისი პარამეტრის u მნიშვნელობისთვის.

4. სხვა სახის ლამბდა ტერმები არ გვაქვს. // ჩამკეტი პუნქტი.

ტერმს, რომელსაც არ აქვს გამოყენების სახე, მნიშვნელობა ეწოდება. ტერმებს ხშირად გამოსახულებებად მოიხსენიებენ.

ლამბდა აღრიცხვის უფრჩხილო გამოსახულებების სწორი ინტერპრეტაციისათვის საჭიროა გავითვალისწინოთ:

- გამოყენება უფრო ძლიერი ქმედებაა, ვიდრე განზოგადოება. ამიტომ $\lambda x.xy$ ნიშნავს $\lambda x.(xy)$ გამოსახულებას, და არა $(\lambda x.x)y$-ს.
- გამოყენება ჯგუფდება მარცხნივ. $(x\,y\,z)$ ნიშნავს $((xy)z)$ გამოსახულებას.
- განზოგადოება ჯგუფდება მარჯვნივ. $\lambda x.\,\lambda y.\,\lambda z.E$ ნიშნავს $\lambda x.(\lambda y.(\lambda z.E))$ გამოსახულებას.

ინდუქციურად განისაზღვრება E გამოსახულების თავისუფალი ცვლადების სიმრავლე $FV(E)$ და ბმული ცვლადების სიმრავლე $BV(E)$:

$FV(x) = \{x\}$; $FV((MN)) = FV(M) \cup FV(N)$; $FV((\lambda x.M)) = FV(M) \setminus \{x\}$

$BV(x) = \emptyset$; $BV((MN)) = BV(M) \cup BV(N)$; $BV((\lambda x.M)) = BV(M) \cup \{x\}$.

E გამოსახულება ჩაკეტილია მაშინ და მხოლოდ მაშინ, როდესაც $FV(E) = \emptyset$, საწინააღმდეგო შემთხვევაში გამოსახულება ღიაა. ჩაკეტილ გამოსახულებას კომბინატორი ეწოდება.

ტერმების გარდაქმნის წესები:

1. α-გადარქმევა (α-conversion) $(\lambda x.f[x]) \to (\lambda y.f[y])$

ბმული ცვლადის სახელის შეცვლა - გამოიყენება სახელების კოლიზიის თავიდან აცილებისთვის.

2. β-გარდაქმნა (β-reduction) $((\lambda x.s)t) \to (s[x := t])$, აბსტრაქციის ტანში x ცვლადის ყოველი თავისუფალი შემოსვლის შეცვლა t გამოსახულებით.

მაგალითი: $(\lambda x.\,x\,(\lambda x.\,x)\,)y$ გვაძლევს $y(\lambda x.\,x)$ გამოსახულებას.

ყურადღება უნდა მივაქციოთ, რომ β-გარდაქმნის გამოყენებამ არ გამოიწვიოს სახელების კოლიზია, t გამოსახულებაში ცვლადის თავისუფალი შემოსვლის დაბმა. თუ x-ს გააჩნია თავისუფალი შემოსვლა t გამოსახულებაში, მაშინ β-გარდაქმნა ასე უნდა ჩავატაროთ: $((\lambda x.s)t) \rightarrow ((\lambda z.s')t) \rightarrow (s'[z := t])$, სადაც $s' = s[x := z]$, z არის ცვლადი, რომელსაც არ გააჩნია თავისუფალი შემოსვლები s და t გამოსახულებებში.

გამოსახულებას, რომელშიც შესაძლებელია β-გარდაქმნის ჩატარება, გარდაქმნადი გამოსახულება ანუ რედექსი (redex – reducible expression) ეწოდება. საწინააღმდეგო შემთხვევაში ამბობენ, რომ გამოსახულებას აქვს ნორმალური ფორმა.

ლამბდა გამოსახულებებს აქვთ თავსართიანი (და არა შუასართიანი x + y , ან ბოლოსართიანი x^2) ფორმა. ამიტომ $x + y$ იწერება $+xy$ სახით, ხოლო x^2 იწერება $*xx$ სახით.

წმინდა ლამბდა აღრიცხვაში, რომელსაც ჩვენ განვიხილავთ, ყველაფერი ფუნქციას წარმოადგენს (ანალოგიური ვითარებაა სიმრავლეთა თეორიაში - ყველაფერი სიმრავლეს წარმოადგენს); ფუნქციის არგუმენტი თვითონ არის ფუნქცია და ფუნქციის მიერ დაბრუნებული მნიშვნელობა აგრეთვე რაღაც ფუნქციაა. ამ აღრიცხვას არ გააჩნია ჩაშენებული კონსტანტები, მიმართებები. „გამოთვლების“ ყოველი ნაბიჯი არის გამოყენების გადაწერა ახალ ფორმაში: გამოყენების მარცხენა მხარეში მდგომი განზოგადოების ტანში ჩაისმება გამოყენების მარჯვენა მხარეში მდგომი გამოსახულება შესაბამისი ცვლადის თავისუფალი შემოსვლების მაგივრად.

ლამბდა გამოსახულებები შეგვიძლია გავიგოთ, როგორც ალგორითმები ან პროგრამები, ხოლო გარდაქმნის წესების გამოყენება - როგორც ამ პროგრამების შესრულება.

რიცხვები ლამბდა აღრიცხვაში განიხილება როგორც სპეციალური სახის გამოსახულებების შემამოკლებელი სიმბოლოები (Church numerals):

$0 := \lambda f.\lambda x.x$; $1 := \lambda f.\lambda x.fx$; $2 := \lambda f.\lambda x.f(fx)$; $3 := \lambda f.\lambda x.f(f(fx))$; ...

თუ n -ის ქვეშ ვიგულისხმებთ მოცემული რიცხვის შესაბამის ლამბდა გამოსახულებას, მაშინ მომდევნო რიცხვი განისაზღვრება ასეთი სქემით:

$$\lambda n.\ \lambda f.\lambda x.(f((n\ f)\ x))$$

ამრიგად, ყოველი რიცხვი f ფუნქციის გამოძახებათა რაოდენობით განისაზღვრება.

მსგავსი მდგომარეობაა ნ. ბურბაკის ფორმალურ სისტემაში, სადაც მათემატიკას საფუძვლად უდევს სიმრავლის ცნება.

$0 := \mathrm{Card}(\emptyset) = \emptyset$; $1 := \mathrm{Card}(\{\emptyset\})$; $2 := \mathrm{Card}(\{\emptyset, \{\emptyset\}\})$; $3 := \mathrm{Card}(\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\})$, ...

X სიმრავლის კარდინალური რიცხვი - $\mathrm{Card}(X)$ განისაზღვრება X სიმრავლის ეკვივალენტობის კლასის წარმომადგენლის სახით. სიმრავლეთა ეკვივალენტობა კი ნიშნავს მათ შორის ბიექციური თანადობის არსებობას. ბურბაკისეული ფორმალური გამოსახულება, რომლის შემამოკლებელი სიმბოლოა „1“, წარმოუდგენლად ბევრ სიმბოლოს შეიცავს [11][12].

ლოგიკური მნიშვნელობები (Church Booleans) განისაზღვრება ორპარამეტრიანი ფუნქციის სახით. ფუნქცია, რომელიც აბრუნებს პირველ არგუმენტს, წარმოადგენს „ჭეშმარიტ“ მნიშვნელობას. ფუნქცია, რომელიც აბრუნებს მეორე არგუმენტს, წარმოადგენს „მცდარ“ მნიშვნელობას. „ჭეშმარიტი“ := $\lambda x.\lambda y.x$ „მცდარი“ := $\lambda x.\lambda y.y$

ლამბდა აღრიცხვაში მიღებულია რამდენიმე სტანდარტული აღნიშვნა. მათ შორის $I := \lambda x.\ x$ - იგივური ფუნქცია; $(\lambda x.\ x)E = E$ ნებისმიერი E გამოსახულებისთვის.

$Y := \lambda\ f.\ (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$ - უძრავი წერტილის უმარტივესი ოპერატორი - Y კომბინატორი -პარადოქსული კომბინატორი. იგი შემდეგი თვისებით ხასიათდება: ნებისმიერი g ფუნქციისთვის Yg გამოსახულება უძრავი წერტილია. მართლაც

$$Yg = (\lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x)))g = (\lambda x.g(x\ x))(\lambda x.g(x\ x)) = (\lambda y.g(y\ y))(\lambda x.g(x\ x)) =$$
$$= g((\lambda x.g(x\ x))(\lambda x.g(x\ x))) = g(Yg)$$

გარდაქმნებში გამოყენებულია Y კომბინატორის განსაზღვრება; f-ის შეცვლა g ფუნქციით β -რედუქციის თანახმად; x ბმული ცვლადის შეცვლა y ცვლადით (α კონვერსია); y ცვლადის შეცვლა $\lambda x.g(x\ x)$ გამოსახულებით (β-რედუქცია).

ლამბდა აღრიცხვა ოპერირებს უსახელო ფუნქციებით, ამიტომ რეკურსიული სქემის ჩვეულებრივად ორგანიზება არ ხერხდება. ამ მიზნით გამოიყენება Y კომბინატორი. იგი მდგომარეობის არმქონე ოპერატორია - მხედველობაში არ იღებს და არ აფიქსირებს წარსულ მოვლენებს, გამოთვლების მდგომარეობას. თუ მას გამოვიყენებთ ასევე უმდგომარეობო ფუნქციისათვის, Y ოპერატორი დააბრუნებს ამ ფუნქციის რეკურსიულ ვერსიას.

$$Yg = g\ (Yg) = g\ (g\ (Yg)) = \ldots\ = g\ (\ldots g\ (\mathrm{Y}g)\ldots)\ =\ \ldots$$

მოვიტანოთ Y კომბინატორის მეშვეობით ფაქტორიალის გამომთვლელი ფუნქციის მაგალითი. $\lambda f.\ \lambda n.\ ((=\ n\ 1)\ ?\ 1\ :\ (*\ n\ (f\ (-\ n\ 1))))$ გამოსახულება აღვნიშნოთ F სიმბოლოთი. მაშინ (YF)k გამოსახულება მოგვცემს k! მნიშვნელობას ნებისმიერი ნატურალური k პარამეტრისთვის.

ჩორჩის თეზისი.

ა.ჩორჩმა ნატურალურ არგუმენტიანი, ალგორითმულად ანუ ეფექტურად გამოთვლადი ფუნქცია გააიგივა რეკურსიულ (ლამბდა აღრიცხვის ფარგლებში განსაზღვრებად) ფუნქციასთან. ეს მოსაზრება ცნობილია ჩორჩის თეზისის სახელწოდებით. იგი ერთმანეთან აკავშირებს განსხვავებული თვისებების მქონე ორ ცნებას. ერთია ალგორითმულად გამოთვლადი ფუნქციის ინტუიციური ცნება, რომელიც უხსოვარი დროიდან დაგროვილი გამოცდილების საფუძველზე ჩამოყალიბდა. მეორე კი - ზუსტად განსაზღვრული ცნება რეკურსიული ფუნქციისა. ჩორჩის თეზისი ჰიპოთეზაა, რომლის დამტკიცება შეუძლებელია.

ალონზო ჩორჩის მიზანს არ შეადგენდა დაპროგრამების ენის შექმნა. ალგორითმის ცნების დაზუსტებისთვის, გამოთვლადი ფუნქციის ფორმალური აღწერისა და მათემატიკის დაფუძნებისთვის შემუშავებული ლამბდა აღრიცხვის გამომსახველობითი ძალა იმდენად მნიშვნელოვანი აღმოჩნდა, რომ გასული საუკუნის 60-იანი წლებიდან დაიწყო მისი გამოყენება კომპიუტერულ მეცნიერებებში. იგი საფუძვლად დაედო ფუნქციონალური დაპროგრამების ენებს, მათ შორის -ჰასკელს [13], რომლის პირველი ვერსია “Haskell 1.0” გავრცელდა 1990 წელს. ასეთი ენებისთვის დამახასიათებელია ის, რომ ფუნქცია წარმოდგენილია როგორც სიდიდე, რომელიც რიცხვებისა თუ ლოგიკური მნიშვნელობების მსგავსად გამოიყენება: ფუნქცია შეიძლება იყოს სხვა ფუნქციის არგუმენტი ან დასაბრუნებელი მნიშვნელობა.

ლამბდა აღრიცხვა მნიშვნელოვან როლს თამაშობს მტკიცებულებების, დაპროგრამების ენების, კომპილერების თეორიებში.

ფუნქციონალური დაპროგრამების პარადიგმა აკადემიურ წრეებში წარმოიშვა გასული საუკუნის 50-იან წლებში. ძნელია მიანიჭო უპირატესობა რომელიმეს ფუნქციონალური დაპროგრამებისა და ობიექტზე ორიენტირებული დაპროგრამების

პარადიგმებს შორის. ორივეს თავის უპირატესობასთან ერთად სისუსტეც გააჩნია. ეს ორი პარადიგმა ერთმანეთის ანარეკლია ჩინური ფილოსოფიის ინ და იან ცნებების მსგავსად [14]. ფუნქციონალური დაპროგრამების ფლობა ბევრად აუმჯობესებს სხვა კონტექსტში შედგენილი კოდის ხარისხს. უკანასკნელ ხანს ამ პარადიგმის ელემენტები ობიექტზე ორიენტირებული დაპროგრამების ენებშიც იქნა შეტანილი.

ჩორჩის ლამბდა აღრიცხვის საფუძველზე შეიქმნა დაპროგრამების მცირე ენა A++[15]. იგი გამიზნულია დაპროგრამების მეთოდების სწავლებისათვის და არა რთული პრაქტიკული პრობლემების ამოსახსნელად. A++ სიმბოლოების უკან იგულისხმება ***A****bstraction* **+** *reference* **+** *synthesis*. A++ ენა განაზოგადებს ლამბდა აღრიცხვის ძირითად ოპერაციებს და ემყარება სამივე მთავარ პროგრამისტულ პარადიგმას: ფუნციონალურ, ობიექტზე ორიენტირებულ და იმპერატიულ დაპროგრამებას.

§2 მნიშვნელობით გამოძახების ლამბდა აღრიცხვა

ხშირად ლამბდა გამოსახულებაში β -გარდაქმნა სხვადასხვა გზით არის შესაძლებელი. ლამბდა აღრიცხვა გარდაქმნების რიგს არ განსაზღვრავს. დაპროგრამების რეალურ ენებში სარგებლობენ რედუქციის სტრატეგიებით ანუ წესებით, რომლებიც გარდაქმნების მიმდევრობას ადგენენ.

ნორმალური რიგის სტრატეგია გულისხმობს ყოველ ნაბიჯზე მარცხნიდან პირველი გარე რედექსის დამუშავებას (გარე რედექსი არის რედექსი, რომელიც არ არის სხვა რედექსის ქვეტერმი). ამ სტრატეგიის გამოყენებით განზოგადოების ტანში არგუმენტი უნდა ჩაისვას ამ არგუმენტის რედუცირებამდე.

გამოყენებითი რიგის სტრატეგია კი მოითხოვს პირველ რიგში მარცხნიდან პირველი შინაგანი რედექსის დამუშავებას (შინაგანი რედექსი არის რედექსი, რომელიც სხვა რედექსს არ შეიცავს ქვეტერმის სახით). ეს ნიშნავს ფუნქციის არგუმენტის რედუცირებას ამ ფუნქციის რედუცირებამდე.

სახელით გამოძახების (call-by-name) სტრატეგია ნორმალური რიგის სტრატეგიაა იმ განსხვავებით, რომ აკრძალულია განზოგადოების შიგნით რედუქციის ჩატარება.

მნიშვნელობით გამოძახების (call-by-value, CBV) სტრატეგით მხოლოდ გარე რედექსები უნდა დამუშავდეს, ამასთან მათი მარჯვენა მხარეები დაყვანილი უნდა იყოს მნიშვნელობამდე (არ უნდა წარმოადგენდნენ გამოყენებას). ეს სტრატეგია დამახასიათებელია „ხარბი“ ენებისთვის.

საჭიროების მიხედვით გამოძახების (call-by-need) სტრატეგია „ზარმაცი“ ენების პარადიგმაა. მოითხოვს გამოსახულების შეფასებას მხოლოდ მაშინ, როდესაც მისი გამოყენება არის საჭირო. ასეთია, კერძოდ, ჰასკელი.

მნიშვნელობით გამოძახების ლამბდა აღრიცხვა (λ_v -აღრიცხვა) შემოღებულ იქნა გორდონ პლოტკინის მიერ [16] აღნიშნული მეთოდის ფორმალური აღწერის მიზნით. λ_v -აღრიცხვის სინტაქსი კლასიკური ლამბდა აღრიცხვის (λ -აღრიცხვის) სინტაქსისგან განსხვავდება რედუქციის წესით. λ -აღრიცხვის β -რედუქციის მაგივრად λ_v -აღრიცხვა სარგებლობს მისი შეზღუდული ვარიანტით - β_v-რედუქციით: $((\lambda x.s)t) \rightarrow \beta_v\ (s[x := t])$. გარდაქმნა დასაშვებია მხოლოდ მაშინ, როდესაც t წარმოადგენს მნიშვნელობას.

მნიშვნელობით გამოძახების ლამბდა აღრიცხვასთან დაკავშირებული საკითხები მრავალ ნაშრომში იქნა შესწავლილი. ი. ფორსტერმა და გ. სმოლკამ მნიშვნელობით

გამოძახების ლამბდა აღრიცხვა განიხილეს როგორც Coq სისტემაში გამოთვლების მოდელი [17]. Coq წარმოადგენს ფორმალურ მტკიცებათა ხელშემწყობ სისტემას, რომელიც ფრანგმა მკვლევარებმა შეიმუშავეს გასული საუკუნის 80-იანი წლების ბოლოს. იგი მოკლედ არის აღწერილი [18] სტატიაში.

[19] ნაშრომში მნიშვნელობით გამოძახების ლამბდა აღრიცხვის სუსტი ვარიანტი შესწავლილია გამოთვლების სირთულის მოდელის თვალსაზრისით. დროის და სივრცის ბუნებრივი ზომების როლში გამოყენებულია β -გარდაქმნების რაოდენობა და გამოთვლებში უგრძესი ტერმის ზომა.

§3 ლამბდა-გამოსახულებები C# ენაში

C# ენაში განსაზღვრული დელეგატი არის მეთოდზე მიმთითებელი[20][21]. იგი განისაზღვრება კლასის გარეთ „delegate“ საკვანძო სიტყვის გამოყენებით და მსგავსია C/C++ ენების პოინტერისა, მაგრამ ამასთან ინახავს ტიპს. დელეგატი შეგვიძლია წარმოვიდგინოთ როგორც ერთობლიობა მეთოდებისა, რომელთაც ერთნაირი სიგნატურა (არგუმენტების ტიპების მიმდევრობა და დასაბრუნებელი ტიპი) გააჩნიათ. იგი ცვლადისათვის მეთოდის მინიჭების, აგრეთვე პარამეტრის სახით მეთოდის გადაცემის საშუალებას გვაძლევს.

დელეგატის გამოცხადების სინტაქსი ასეთია:

[წვდომის მოდიფიკატორი] delegate [დასაბრუნებელი ტიპი] [სახელი] ([პარამეტრების სია])

დელეგატს მიმართავენ, როდესაც წინასწარ უცნობია, რომელი მეთოდის გამოყენება იქნება საჭირო და ინფორმაცია ამ მეთოდის შესახებ მიიღება პროგრამის შესრულების პროცესში.

დელეგატის გამოცხადების შემდეგ მისი ობიექტი იქმნება new საკვანძო სიტყვის გამოყენებით შემდეგი სინტაქსის მიხედვით:

[დელეგატის სახელი] [ობიექის სახელი] =

new [დელეგატის სახელი] (გამოძახებული მეთოდის სახელი)

C# 2.0 ვერსიაში 2005 წელს შემოიღეს ანონიმური მეთოდი. შესაძლებელია მისი კოდის „in-line“ ჩაწერა იმ ადგილზე, სადაც delegate ტიპის გამოსახულება არის მოსალოდნელი. ეს ფუნქციონალური დაპროგრამების გამომსახველი საშუალებების გამოყენების შესაძლებლობას იძლევა. ანონიმური მეთოდის სინტაქსის ნაკლად ითვლება მისი შედარებითი სირთულე - სიტყვამრავლობა.

C# 3.0 ვერსიას 2007 წელს დაემატა ლამბდა გამოსახულება, რომელიც წარმოიდგინება დელეგატის ობიექტის სახით. C#-ის კომპილერს შეუძლია მისი გარდაქმნა გამოსახულებათა ხეში, რითაც აღიწერება ლამბდა გამოსახულების ლოგიკური სტრუქტურა.

C#-ის ლამბდა გამოსახულება შეიცავს => ისარს (ლამბდა_გამოცხადების ოპერატორი), რომლის მარცხნივ მოთავსებულია ([პარამეტრების სია]), ხოლო მარჯვნივ - {[შესასრულებელი კოდი]}. ერთი პარამეტრის შემთხვევაში ფრჩხილები არ გამოიყენება:

პარამეტრი => ... ; თუ პარამეტრი არ გვაქვს, მარცხენა მხარე ასეთ სახეს იღებს: ( ) => ...

თუ შესასრულებელი კოდი ერთი გამოსახულებაა, მაშინ მარჯვენა მხარეს ასეთი სახე აქვს: ... => გამოსახულება.

ხშირ შემთხვევაში კომპილერს შეუძლია კონტექსტის მიხედვით განსაზღვროს პარამეტრების ტიპები. ამიტომ მათი აღწერა არ არის აუცილებელი.

ლამბდა გამოსახულებების გამოყენებით შესაძლებელი გახდა ანონიმური მეთოდის სიტაქსის დახვეწა და გამარტივება.

მაგალითი:

```
public delegate int MyPoint(int arg1, int arg2);
MyPoint SquaredRadius = delegate (int arg1, int arg2){
                                    Return x*x+y*y;};
```

ზემოთ ნაჩვენები ფრაგმენტი ლამბდა გამოსახულების გამოყენებით ასე შეიძლება ჩაიწეროს:

```
public delegate int MyPoint(int arg1, int arg2);
MyPoint SquaredRadius =(x,y)=>x*x+y*y;
```

§4. ლამბდა-გამოსახულებები C++ ენაში

ლამბდა-გამოსახულებები დაემატა C++ ენის 2011 წლის სტანდარტს, რამაც საფუძვლიანად შეუცვალა ფორმა ამ ენას. იგი უფრო გამძლავრდა - შეიძინა ფუნქციონალური დაპროგრამების ენის შესაძლებლობანი. გაჩნდა საშუალება განისაზღვროს მარტივი, უსახელო ფუნქცია ზუსტად იქ, სადაც მისი გამოყენებაა საჭირო [22].

C++ ენის ლამბდა გამოსახულების უმარტივესი ფორმა სქემატურად შეგვიძლია წარმოვადგინოთ ფრჩხილთა წყვილების სამეულის სახით: [ ]( ){ }. ნებისმიერი ლამბდა გამოსახულება უნდა იწყებოდეს [ ] წყვილით.

ჩვეულებრივი ფუნქციებისაგან განსხვავებით, ლამბდა-ფუნქციას წვდომა აქვს გარე ცვლადებთან, რომლებიც მის მომცველ ბლოკს განეკუთვნებიან. ასეთი ცვლადების წვდომის ფორმა ფიქსირდება [ ] ნაკვეთში. ეს ფორმა სამი სახის შეიძლება იყოს:

- [&] - წვდომა სახელის გამოყენებით (capture by reference)
- [=] - წვდომა მნიშვნელობის გამოყენებით (capture by value) .
- ორმაგი წვდომა. ზოგი ცვლადის წვდომა წარმოებს სახელით, ზოგისა კი მნიშვნელობით.

ლოკალური ცვლადების მნიშვნელობათა წვდომის უნარი შესაძლებელია გამოყენებულ იქნას ლამბდა გამოსახულებაში მისი პირველი გამოძახების მომენტისთვის ამ ცვლადების მდგომარეობის დასაფიქსირებლად.

ლამბდა-ფუნქციის პარამეტრებს ათავსებენ ( ) ნაკვეთში. თუ ფუნქციას პარამეტრები არ გააჩნია, ( ) წყვილი არაა საჭირო და ფუნქცია წარმოიდგინება [ ]{ } ნაკვეთებით. ლამბდა-ფუნქციის შესრულებისთვის საჭირო ბრძანებები ჩაიწერება { } ნაკვეთში. ესაა ფუნქციის ტანი.

C++ პროგრამის დამუშავების პროცესში ლამბდა გამოსახულების აღმოჩენისთანავე კომპილერი ქმნის შესაბამის კლასს და განსაზღვრავს ლამბდა-ფუნქციის დასაბრუნებელი მნიშვნელობის ტიპს. ამიტომ წინასწარ ცხადი სახით მისი აღწერა არაა აუცილებელი. რთულ შემთხვევაში, როდესაც შეუძლებელია კომპილერმა წინასწარ განსაზღვროს დასაბრუნებელი ტიპი, მას მიუთითებენ ისრის გამოყენებით პარამეტრებსა და ფუნქციის ტანს შორის.

ამრიგად, C++ ლამბდა-ფუნქცია შეგვიძლია ასე აღვწეროთ:

[გარე ცვლადების წვდომა] (პარამეტრები) → დასაბრუნებელი ტიპი {ფუნქციის ტანი}
შესაძლებელია ლამბდა ობიექტის შენახვა ცვლადში შემდეგი სქემის გამოყენებით:
auto ცვლადის_სახელი { [...] (...) {...} }

ლამბდა გამოსახულებების შემოღებამ C++11-ში გაამარტივა სტანდარული თარგების ბიბლიოთეკის ალგორითმების გამოყენება, რომლებიც ფუნქტორებით ოპერირებენ. ასეთებია, კერძოდ std::for_each( ), std::transform( ).

§5. ტოპოლოგიური ლამბდა აღრიცხვა

ჩორჩის თეზისის თანახმად, ნებისმიერი გამოთვლადი ფუნქცია ლამბდა გამოსახულებით აღიწერება. ამის გამო გამოთვლების პროცესის ტოპოლოგიური დახასიათება შესაძლებელია ლამბდა აღრიცხვის ტოპოლოგიის შესწავლის გზით.

დეინა სკოტის (Dana Stewart Scott) ტოპოლოგია[23].

$(S, \leq)$ იყოს ნაწილობრივ დალაგებული სიმრავლე. მის არაცარიელ X ქვესიმრავლეს მიმართული ეწოდება, თუ $\forall x,y\in X\ \exists z\in X\ x \leq z\ \&\ y \leq z$. $(S, \leq)$ ნაწილობრივ დალაგებულ სიმრავლეს სრული ეწოდება, თუ სამართლიანია პირობები:

- S-ის ყოველ მიმართულ ქვესიმრავლეს გააჩნია უმცირესი ზედა ზღვარი;
- S-ს გააჩნია ფსკერი, ანუ ელემენტი $\perp \in S$ ისეთი, რომ $\forall x\in S\ \perp \leq x$.

$(S, \leq)$ ნაწილობრივ დალაგებული სიმრავლე სრულია მაშინ და მხოლოდ მაშინ, როდესაც $\leq$ დალაგების მიმართ მონოტონურ ყოველ $f: S \to S$ ფუნქციას გააჩნია ერთი მაინც უძრავი წერტილი:
$(\forall x,y\in S\ (x \leq y \Rightarrow f(x) \leq f(y))) \Rightarrow (\exists z\in S\ (f(z) = z))$
ნაწილობრივ დალაგებულ სრულ სიმრავლეზე განისაზღვრება სკოტის ტოპოლოგია, აიგება ლამბდა აღრიცხვის არაწინააღმდეგობრივი მოდელი და დენოტაციური სემანტიკა.
სკოტის მიხედვით $O \subseteq S$ არის ღია სიმრავლე, თუ:

- $x \in O\ \&\ x \leq y \Rightarrow y \in O$.
- ნებისმიერი მიმართული X სიმრავლისთვის

$X \subseteq S\ \&\ \text{supremum}(X) \in O \Rightarrow X \cap O \neq \emptyset$.

ადვილი შესამოწმებელია, რომ ღია სიმრავლის ასეთი განმარტება უზრუნველყოფს ტოპოლიგიის ძირითადი თვისებების შესრულებას:

- $\emptyset$ და S ღია სიმრავლეებია;
- ღია სიმრავლეების ერთობლიობა ჩაკეტილია ნებისმიერი გაერთიანებებისა და სასრული თანაკვეთების მიმართ.

მტკიცდება, რომ $f: S \to S'$ გადასახვა უწყვეტია მაშინ და მხოლოდ მაშინ, როდესაც
$f(\text{supremum}(X)) = \text{supremum}\, f(X) = \text{supremum}\ \{f(x) \mid x \in X\}$

თუ A, B სრული ნაწილობრივ დალაგებული სიმრავლეებია, მაშინ მათი დეკარტული ნამრავლიც ასეთი იქნება დალაგების მიმართებისა და უმცირესი ზედა ზღვრის კოორდინატული განსაზღვრის შემთხვევაში:
$(x_1, x_2) \leq (y_1, y_2)$ მაშინ და მხოლოდ მაშინ, როდესაც $x_1 \leq y_1\ \&\ x_2 \leq y_2$
$\vee(\{x_i\}, \{y_i\}) = (\vee\{x_i\}, \vee\{y_i\})$

$f: A \to B$ სახის ფუნქციათა B^A სიმრავლე ნაწილობრივ დალაგებული და სრულია, თუ $f \leq g$ მაშინ და მხოლოდ მაშინ, როდესაც $\forall_{x\in A}(f(x) \leq g(x))$, $f, g \in B^A$

ლამბდა-გამოყენება უწყვეტია $B^A \times A \to B$ სიმრავლეზე სკოტის ტოპოლოგიის მიხედვით (წყვილი $(f, x) \in B^A \times A \to B$ წარმოადგენს $f(x)$ ფუნქციას). უწყვეტია აგრეთვე ლამბდა-განზოგადოება.

ლამბდა აღრიცხვის ტოპოლოგიური მეთოდებით შესწავლის მეორე გზა მოცემულ იქნა ალბერტ ვისერის მიერ[24].

§6. დიფერენციალური ლამბდა აღრიცხვა

დიფერენციალური ლამბდა აღრიცხვა[25] არის პარადიგმა ფუნქციონალური დაპროგრამების ენისა, რომელიც აღჭურვილია გაწარმოების სინტაქსური ოპერატორით და საშუალებას გვაძლევს პროგრამა წრფივი სახით გამოვიყენოთ არგუმენტებისთვის. იგი მგრძნობიარეა რესურსების მიმართ. პროგრამისტს ეძლევა პრიმიტივი, რომლითაც შესაძლებელი ხდება პროგრამის შესრულების პროცესში რესურსების ცხადად მართვა. ლამბდა აღრიცხვა ითვლება ხიდად ლოგიკასა და კომპიუტერულ მეცნიერებებს შორის, დიფერენციალურ ლაბდა აღრიცხვას კი ახასიათებენ როგორც ხიდს მათემატიკურ ანალიზსა და ლამბდა აღრიცხვას შორის.

უნდა აღინიშნოს, რომ დიფერენციალურ ლამბდა აღრიცხვაში მნიშვნელოვან როლს თამაშობს წრფივობის ცნება. აქ მისი განმარტება განსხვავდება საყოველთაოდ ცნობილი ალგებრული განმარტებისგან. დიფერენციალური ლამბდა ტერმი წრფივია, თუ იგი არგუმენტს ზუსტად ერთხელ გამოიყენებს. დიფერენციალური ლამბდა აღრიცხვა მიიღება ლამბდა აღრიცხვის გაფართოებით. მას ემატება ახალი სახის ტერმები და წრფივი რედუქცია.

დიფერენციალური ლამბდა ტერმების სიმრავლე აღიწერება შემდეგი სქემით:

$x \mid \lambda x.t \mid s\ t \mid 0 \mid \sigma s + \tau t \mid Ds{\cdot}t \quad (\sigma, \tau \in R)$

ტერმების წრფივი კომბინაციის დამატება ჩვეულებრივი ლამბდა ტერმების სიმრავლისათვის განპირობებულია ნამრავლის გაწარმოების წესით $(f{\cdot}g)' = f'{\cdot}g + f{\cdot}g'$.

$Ds \cdot t$ დიფერენციალური ლამბდა ტერმის ინტერპრეტაცია ეფუძნება დიფერენცირების შესახებ ზოგად მოსაზრებებს. თუ $f : E \to F$ დიფერენცირებადი ფუნქციაა, მისი წარმოებული $\hat{f}'$ მოიაზრება როგორც ფუნქცია, რომელიც E სიმრავლეს გადასახავს $E \multimap F$ სიმრავლეში (სადაც $E \multimap F$ აღნიშნავს $E \to F$ სახის წრფივი ფუნქციების სივრცეს).

$\hat{f}' : E \to (E \multimap F)$. $\hat{f}'(x) \cdot a$ არის წარმოებულის წრფივი გამოყენება a-სთვის, ანუ f ფუნქციის წარმოებული x წერტილში a-ს გასწვრივ, $a \in E$. იგი აღინიშნება $Df \cdot a$ სიმბოლოებით. ეს გამოსახულება წრფივად არის დამოკიდებული a-ზე. $Df \cdot a : E \to F$.

ლამბდა აღრიცხვის β-რედუქციას ემატება დიფერენციალური β_D-რედუქცია:

$D(\lambda x.s) \cdot t \to \lambda x.((\partial s/\partial x) \cdot t)$

ჩვეულებრივი ლაბდა-გამოყენება არ არის წრფივი

$(\lambda x.s)(t+u) \neq (\lambda x.s)t + (\lambda x.s)u$

დიფერენციალური გამოყენება $D(\lambda x.s) \cdot t$ წრფივია.

დიფერენციალურ ლამბდა აღრიცხვაში მტკიცდება დიფერენციალური აღრიცხვის ძირითადი ფორმულები; აიგება ტეილორის მწკრივი.

§7. ალბათური ლამბდა აღრიცხვა

ალბათური მოდელები ფართოდ გამოიყენება კომპიუტერულ მეცნიერებათა სხვადასხვა მიმართულებებში შემთხვევითობის შემცველი ამოცანების აღწერისა და

დამუშავებისთვის (კრიპტოგრაფია, ტელეკომუნიკაციური ქსელები, მანქანური სწავლება, ლინგვისტიკა, რობოტიქსი). ასეთ ამოცანებზეა მორგებული ალბათური ლამბდა აღრიცხვა, რომელიც დიფერენციალური ლამბდა აღრიცხვის მსგავსად, მიიღება ჩვეულებრივი ლამბდა აღრიცხვის სპეციალური გაფართოებით. სინტაქსს ემატება ალბათური ამორჩევის ოპერატორი $\oplus_p$.

ალბათური ლამბდა ტერმები აღიწერება ასეთი სქემით:

$$x \mid \lambda x.\ S \mid S\ T \mid S \oplus_p T \qquad p \in [0,1]$$

ჩვეულებრივი ლამბდა აღრიცხვისაგან განსხვავებით აქ რედუქციის წესები გარკვეული ალბათობით მოქმედებენ:

$(\lambda x.S)T \ \rightarrow_1 \ S[x := T]$

$S \oplus_p T \ \rightarrow_p \ S$

$S \oplus_p T \ \rightarrow_{1-p} \ T$

პირველი რედუქცია მოქმედებს ალბათობით 1, მეორე რედუქცია S ტერმს გვაძლევს p ალბათობით, მესამე კი - T ტერმს $1-$ p ალბათობით.

ტერმის გარდაქმნის პროცესში რამდენიმე რედუქციის მიმდევრობით გამოყენების შედეგად მიღებული ტერმის ალბათობა ცალკეული ნაბიჯების ალბათობათა ნამრავლის ტოლია: $S \rightarrow_p T \rightarrow_q U$ რედუქციათა შედეგის ალბათობაა pq.

[26] სტატიაში დამუშავებულია ალბათური ლამბდა აღრიცხვის ოპერაციული სემანტიკა, როგორც საფუძველი ალბათური დაპროგრამების უნივერსალური ენებისთვის CHURCH, ANGLICAN, VENTURE.

[27] ნაშრომის ავტორები აღწერენ ალბათური ლამბდა აღრიცხვის ორ ვარიანტს $\Lambda^{cbv}{}_{\oplus}$ და $\Lambda^{cbv}{}_{\oplus}$ ალბათური ამორჩევის ოპერატორებით, რომლებიც მნიშვნელობით გამოძახებისა და სახელით გამოძახების სტრატეგიებს ეთანადებიან.

§8. კვანტური ლამბდა აღრიცხვა

უკანასკნელ ხანს კვანტური გამოთვლების ტექნიკური საშუალებების განვითარებამ გაზარდა ინტერესი ამ მიმართულებით თეორიული საკითხების დამუშავებისადმი.

კვანტური ალგორითმებისა და გამოთვლების ფორმალური აღწერისათვის ჩორჩის ლამბდა აღრიცხვის ანალოგიით შემოღებულ იქნა კვანტური ლამბდა აღრიცხვა [28] [29] [30] [31].

კვანტური ლამბდა აღრიცხვა ტიპების მქონე თეორიაა. ქვემოთ მოცემულია ტიპების სინტაქსის ერთ-ერთი ვარიანტი:

$qbit \mid !A \mid T \mid (A \multimap B) \mid (A \oplus B) \mid A \otimes B \mid$

qbit - კვანტური ბიტის ტიპი. კვანტური ბიტი დირაკის აღნიშვნებით შეგვიძლია წარმოვადგინოთ, როგორც $|0\rangle$ და $|1\rangle$ ორთოგონალური მდგომარეობების სუპერპოზიცია $\alpha|0\rangle + \beta|1\rangle$, α, β კომპლექსური რიცხვებია და $\alpha^2 + \beta^2 = 1$.

თუ A რაიმე ტიპია, მაშინ $!A$ არის ახალი ტიპი, რომლისთვისაც დასაშვებია A ტიპის ტერმის ასლის გამოყენება.

T არის სინგლტონი ანუ ტიპი, რომელიც ერთადერთ ობიექტს განსაზღვრავს.

$A \multimap B$ აღნიშნავს $A \rightarrow B$ სახის ფუნქციათა ტიპს.

$A \oplus B$ განსაზღვრავს თანაუკვეთ გაერთიანებას.

$A \otimes B$ არის წყვილთა სიმრავლე.

კლასიკური ბიტი შესაძლებელია მოცემულ იქნას წარმოებული ტიპის სახით:

$bit = T \oplus T.$

ტერმები აღიწერება შემდეგი სქემით:

$M, N, P ::= c \mid x \mid \lambda x.M \mid M\,N \mid \langle M, N \rangle \mid * \mid let\ \langle x, y \rangle = M\ in\ N \mid inj_l(M) \mid inj_r(M) \mid$
$match\ P\ with\ (x \mapsto M \mid y \mapsto N) \mid let\ rec\ f\ x = M\ in\ N.$

c არის ტერმი-კონსტანტა, რომელიც შეიძლება იყოს:

- *new* - მდგომარეობის მომზადების ფუნქცია. *new*-ს შესავალია კლასიკური ბიტი, ხოლო გამოსავალი კვანტური ბიტი. $new\ 0 = |0\rangle$, $new\ 1 = |1\rangle$.
- *meas* - გაზომვის ფუნქცია. იგი შესავალზე იღებს კვანტურ ბიტს, ზომავს მას სტანდარტულ $\{|0\rangle, |1\rangle\}$ ბაზისში და გამოსავალზე გვაძლევს კლასიკურ ბიტს.
- U კონსტანტა, რომლის მნიშვნელობა მიეკუთვნება სტანდარტული უნიტარული კვანტური ლოგიკური ელემენტების უნივერსალურ სიმრავლეს.

x ცვლადების მოცემული უსასრულო სიმრავლის ელემენტია.

$\lambda x.M$ და $M\,N$ ტერმების შინაარსი იგივეა, რაც ჩვეულებრივ ლამბდა აღრიცხვაში.

$\langle M, N \rangle$ აღნიშნავს წყვილს.

$*$ არის ცარიელი ერთობლიობა, „n-ეული“ რომლისთვისაც $n = 0$.

$let\ \langle x, y \rangle = M\ in\ N$ გამოსახულება აღნიშნავს პროგრამას, რომლის შესრულების ბიჯებია:

- $M \mapsto \langle V, W \rangle$ - M გამოსახულების შეფასების შედეგად აიგება $\langle V, W \rangle$ წყვილი;
- შესრულდება მინიჭების ოპერაციები $x = V$, $y = W$;
- შესრულდება N გამოსახულებით გათვალისწინებული მოქმედებანი.

$inj_l(M)$ და $inj_r(M)$ ფუნქციები აღნიშნავენ თანაუკვეთ გაერთიანებაში მონაცემების ჩართვას მარცხნიდან და მარჯვნიდან (შესაბამისად).

$match\ P\ with\ (x \mapsto M \mid y \mapsto N)$ გამოსახულება შეაფასებს P-ს თანაუკვეთ გაერთიანებაში ჩართვის თვალსაზრისით და შემდეგ შეასრულებს $x \mapsto M$ გადასახვას, თუ P-ს აქვს $inj_l(x)$ ფორმა, ან $y \mapsto N$ გადასახვას, თუ P-ს აქვს $inj_r(y)$ ფორმა.

$let\ rec\ f\ x = M\ in\ N$ ტერმი განსაზღვრავს $f(x) = M$ რეკურსიულ ფუნქციას და შემდეგ შეასრულებს N-ს.

ზემოთ აღწერილი სინტაქსი კვანტური ალგორითმების შემსრულებელი ტერმების შედგენის საშუალებას გვაძლევს.

ლიტერატურა

[1] Мальцев А. И., *Алгоритмы и рекурсивные функции.* 2-е изд. Москва. Наука. 1986.

[2] Knuth D. E., *The Art of Computer Programming. Vol.1:Fundamental Algorithms.* Third ed. Reading, Massachusetts. Addison-Wesley. 1997.

[3] Church A., “An Unoslvable Problem of Elementary Number Theory”, *American Journal of Mathematics,* 1936, vol. 58, no.2. pp 345-363.

[4] Church A., “A Note on the Entscheidungsproblem”, *The Journal of Symbolic Logic,* 1936, vol. 1, no. 1. pp. 40-41.

[5] Turing A. M., ”On computable numbers with an application to the entscheidungsproblem”, *Proc. London Math. Soc.* , 1936, Ser. 2 **42** . pp 230 -265.

[6] Turing A. M., “Computability and λ-definability”, *The Journal of Symbolic Logic,* 1937, vol. 2, i. 4. pp. 153-163.

[7] Post E. L., “Finite Combinatory Processes-Formulation I”, *The Journal of Symbolic Logic,* 1936, vol. 1, no. 3. pp 103 -105.

[8] Kleene S. C., "General recursive functions of natural numbers", *Mathematische Annalen,* 1936, v.112, i.1, 727-742.
[9] Church A., "The Calculi Of Lambda Conversion", *Annals Of Mathematical Studies,* 1941, v.6.
[10] Hyland J.M.E., "Classical lambda calculus in modern dress", *Mathematical Structures in Computer Science*, 2017, v. 27, i.5, pp. 762-781.
[11] Grimm J., "Implemenation of Bourbaki's Elements of Mathematics in Coq: Part One, Theory of Sets", *Journal of Formalized Reasoning*, 2010, vol. 3, no. 1. pp 79 – 126.
[12] Bourbaki N., *Theory of Sets.*Berlin Heidelberg: Springer-Verlag, 2004.
[13] Kurt W., *Get Programming with Haskell.* Shelter Island. NY: Manning Publications, 2018.
[14] Garson E. "Apply Functional Programming Principles". In *Henney K. ed.: 97 Things Every Programmer Should Know,* Sebastopol CA: O'Reilly Media, Inc. 2010.
[15] Loczewski G. P., *A++ : The smallest Programming Language in he World*. Hamburg: tredition GmbH, 2nd augmented ed. 2018.
[16] Plotkin G. D., "Call-by-Name, Call-by-Value and the lambda-Calculus", *Theoretical Computer Science,* 1975, 1(2), 125-159.
[17] Forster Y., Smolka G., "Call-by-Value Lambda Calculus as a Model of Computation in Coq", *Journal of Automated Reasoning,* 2018, https://doi.org/10.1007/s10817-018-9484-2 .
[18] Böhne S., Kretz C., "Learning how to Prove: From the Coq Proof Assistant to Texbook Style", 2018, arXiv.1803.01466v1[cs.LO].
[19] Forster Y., Kunze F., Roth M., "The Weak Call-by-Value λ-Calculus is Reasonable for Both Time and Space", 2019, arXiv: 1902.07515v1[cs.CC].
[20] Olsson M. *C# 7 Quick Syntax Reference.* New York: Apress, 2nd ed. 2018.
[21] Skeet J. *C# in Depth.* Shelter Island. NY: Manning Publications, 4th ed. 2018.
[22] Horton I., Van Weert P., *Beginning C++17.* New York: Apress, 2018.
[23] Andradi H., Ho W. K., "A Topological Scott Convergence Theorem", 2019, *Logical Methods in Computer Science,* v 15, 1, pp.1-29.
[24] Berline C., Manzonetto G., Salibra A., "The Visser topology of Lambda-calculus", 2007, *arXiv:math/0701684[math.LO.]*
[25] Ehrhard T., Regnier L., "The differential lambda-calculus", 2003, *Theoretical Computer Science,* v 309, I 1-3, pp 1-41.
[26] Borgström J., Dal Lago U., Gordon A., Szymczak M., "A lambda-calculus foundation for universal probabilistic programming", 2016, *International Conference on Functional Programming,* Nara, Japan. pp. 33 – 36.
[27] Faggian C., Ronchi della Rocca S., "Lambda Calculus and Probabilistic Compuation (Exended Version)", 2019, *arXiv: 1901.02853v5[cs.LO]*
[28] Van Tonder A., "A Lambda Calculus for Quantum Computation", 2004, *arXiv:quant.ph/0307150 v5*
[29] Selinger P., Valiron B., "Quantum Lambda Calculus". In *S.Gay & I. Mackie, editors: Semantic Technicues in Quantum Computation*, 2010. Cambridge University Press. pp. 135-172.
[30] Atzemoglou P., "The dagger lambda calculus", 2014, *arXiv:1406.1633v2[cs.LO]*.
[31] Ross N. J., "Algebraic and Logical Methods in Quantum Computation", 2017, *arXiv:1510.02198 v2[quant-ph]*.