Математички факултет Универзитета у Београду

МАСТЕР РАД

Алгоритам унификације, уопштења и неке примене

Филип Радуловић

Београд 2014.

Ментор: проф. др. Александар Јовановић

Чланови комисије: проф. др. Александар Перовић
проф. др. Жарко Мијајловић

САДРЖАЈ

1. УВОД 4
2. ИСТОРИЈА 5
3. ОСНОВНИ ПОЈМОВИ 7
4. УЈЕДНАЧАВАЊЕ 11
5. АЛГОРИТАМ 14
6. ИМПЛЕМЕНТАЦИЈА АЛГОРИТМА 16
7. КОД ПРОГРАМА 17
8. ПРИМЕРИ УНИФИКАЦИЈЕ 24
Пример 1 24
Пример 2. 27
Пример 3. 29
Пример 4. 31
9. УОПШТЕЊА И ПРИМЕНЕ 33
10. УНИФИКАЦИЈА И ПРОЛОГ 37
11. ЗАНИМЉИВОСТИ 40
12. ЗАКЉУЧАК 42
13. ЛИТЕРАТУРА 43

1. УВОД

Унификација је алгоритамски процес мењања два терма са циљем да се ти терми учине једнаким (идентичним).

Један од најбитнијих алгоритама у рачунарству јесте алгоритам унификације (уједначавања). За два дата терма провериће се да ли су они уједначиви.

Алгоритам који је пред нама испитује да ли су два дата терма уједначива, а као одговор можемо добити да су два терма t_1 и t_2 неуједначива или уједначива (у том случају добијамо и списак замена ако терми нису једнаки). Постоје разни алгоритми унификације, али је у раду описан општи алгоритам унификације, тј. алгоритам првог несклада. У поступку уједначавања јавља се потреба за применом више замена, односно да се на терм над којим је примењена једна замена, примени и нека друга замена.

Увек ћемо користити најнужније замене, тј. уједначења. То значи да алгоритам унификације који је пред нама, показује да у случају уједначивости два терма, постоји и најопштији уједначивач – минималан скуп замена.

Џон Алан Робинсон је први истраживао најопштији уједначивач и тиме дао велики допринос аутоматском резоновању– део рачунарства који се бави разумевањем различитих аспеката резоновања. Аутоматско резоновање је област рачунарства која проучава резоновање у контексту развоја софтвера, помоћу којег рачунари могу аутоматски, или скоро потпуно аутоматски да резонују.

Алгоритам уједначавања има највише примене у области аутоматског резоновања, која је подобласт вештачке интелигенције. Најчешће се користи за аутоматско доказивање теорема, интерактивно доказивање теорема и аутоматску проверу доказа.

У раду се може видети код написан у програмском језику C++, са малим унапређењима и побољшањима.

2. ИСТОРИЈА

Човек је одувек тежио да сва досадашња сазнања имплементира у један систем, који ће омогућити лакшу обраду и претрагу података. Ако се осврнемо кроз историју, Чарлс Бебиџ (1791-1871), енглески математичар, први је конструисао машину у нади да она неће правити људске грешке при дугачком рачунању, тј. покушао је да аутоматизује процес израчунавања више математичких операција. То успева развојем диференцијалне машине 1822. године.

Алан Тјуринг (1912 - 1954) енглески математичар, дао је идеју за машину (Тјурингова машина) која је могла да решава проблеме низом елементарних операција, а истовремено има довољно меморије да складишти све потребне инструкције и алгоритме. У тешким временима Другог светског рата направио је концепт алгоритма који и ми данас користимо. Израдио је тест којим се демонстрира вештачка интелигенција рачунара, објављен 1950. у раду: "Рачунарске машине и интелигенција". Тјуринг је машину описао 1936. године, као машину једноставне конструкције која упркос својој једноставности може да се прилагоди да симулира логику било ког рачунара који би могао да се конструише. Иако једноставне конструкције, Тјурингова машина никад није имала намену као практична рачунарска технологија, већ као модел за теоријско разматрање о границама механичког рачунања.

Џон фон Нојман (1903-1957), мађарско-амерички математичар, је први осмислио архитектуру рачунара који се данас користе. Према његовој архитектури постоји јасна разлика између опипљивог дела рачунара и „неопипљивог" дела рачунара, познатије као хардвер и софтвер. У Америци 1944. године са групом научника фон Нојман конструише први дигитални електронски рачунар ЕНИЈАК (ENIAC- Electronic Numerical Integrator And Computer). Тај рачунар који је радио на основу унапред задатог програма, могао је да решава разне рачунарске проблеме. На жалост, он се у највећој мери употребљавао за решавање проблема израде хидрогенске бомбе.

Након тога, научници су хтели да развију софтвер, па да уз помоћ рачунара решавају проблеме и задатке на начин који се може описати као интелигентан. Подобласт рачунарства која се бави имплементацијом оваквог софтвера назива се вештачка интелигенција.

Скуп научника 1956. године у Дармуту (Америка), који је организовао Џон Мекарти, са Клодом Шеноном, Марвином Минским на коме су били и други научници који су се бавили темама интелигенције и теорије аутомата, дефинисали су појам вештачке интелигенције. Према Марвину Минском, вештачка интелигенција је подобласт рачунарства која се бави конструисањем рачунарских система са особинама које би код људских бића биле окарактерисане као интелигентне.

Први значајни успеси вештачке интелигенције су програм за играње шаха, играње дама, и аутоматско доказивање теорема. Програмски језик PROLOG (PROgramming in

LOGic) је најзначајнији програмски језик за развој и примену вештачке интелигенције, намењен за доказивање теорема, експертне системе, NLP (natural language processing)...

Програмски језик пролог (PROLOG), који је специјализован за логичко програмирање повезан са вештачком интелигенцијом, развијен раних 1970. година, (1972.) ослања се у доброј мери на алгоритам унификације. У PROLOG-у, унификација је најчешће коришћена операција над подацима, а често се користи као механизам за описивање односа међу структурним објектима. Иако најпознатији, поред PROLOG-а, могу се користити и следећи програмски језици за вештачку интелигенцију: XSB Prolog, F-logika, LISP, Haskell, Godel, JLog/JScriptLog…

Први који се бавио унификацијом је амерички математичар Џон Алан Робинсон, рођен 1928. Он је први показао да изједначиви терми имају најопштији уједначивач и описао алгоритам за уједначивање два терма. Алгоритам који је он описао назива се „наиван“ алгоритам. Он је уједно и најједноставнији. Његов алгоритам по записивању два терма поставља индексе (маркере) на почетак сваког од њих. После тога

1. заједно померамо индексе (маркере) по један симбол истовремено све док оба не дођу до краја терма (једнаки су) или док не дођу до два различита симбола.
2. Ако оба симбола нису променљиве, онда су терми неуједначиви. Ако је један симбол променљива а други симбол је подтерм, тада постоје две могућности:
    а) ако се променљива једног терма јавља у подтерму другог терма, тада су неуједначиви, и
    б) замени се променљива са подтермом, тј. правимо нови терм.

Када је Робинсон посетио 1961. године Argonne National Laboratory, постао је заинтересован за развијање унификације и њену примену у аутоматском доказивању теорема. Резолуција и унификација су од тада уграђени у многим системима за аутоматско доказивање теорема и основа су за механизам закључивања који се користи у логичком програмирању и програмском језику PROLOG. Први алгоритам унификације написао је Робинсон 1965. године.

Овај алгоритам је једноставан јер идући с лева на десно, систематски налази неслагања у термима покушавајући да их изједначи, унификује. Робинсон је направио велики напредак у технологији аутоматизованог резоновања, која је и данас главна област примене алгоритма унификације. Робинсон је 1996. године добио награду за допринос овој технологији.

Код нас се алгоритмом унификације највише бавио Славиша Прешић (1933-2008), српски математичар, објавивши рад у часопису „Рачунарство“ 1990. године у коме је описао општи алгоритам унификације и приложио код програма написан у BASIC-у (видети [5]).

Стуб темељац програмског језика PROLOG, алгоритам унификације, сматра се једним од најзначајнијих алгоритама.

3. ОСНОВНИ ПОЈМОВИ

Исказна слова су симболи p, q, r, s, p_1, q_1, r_1, …p_n, q_n, r_n (иначе, исказна слова могу бити било који изабрани знаци, само треба да се разликују од знакова логичких операција, помоћних знакова заграда и знакова константа). Исказне формуле се дефинишу индуктивно, помоћу следећих правила:

- Исказна слова су исказне формуле.
- Ако су A и B исказне формуле, онда су исказне формуле и: $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$.
- Исказне формуле се могу формирати применом претходна два правила, коначан број пута.

Често се усвајају допунска правила о неписању неких заграда, конкретно, спољашње заграде се не пишу. Ако су A и B исказне формуле, онда формуле $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$ можемо да запишемо редом $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$. Нека сада имамо n различитих исказних слова, где ћемо да заменимо свако слово са једним од симбола $\top$ $\bot$ (тачно или нетачно) и на тај начин добијамо вредности тих слова. На пример, када n исказних слова p, q, r,…, x, y, z заменимо тим симболима, добијамо уређену n торку тих симбола. Ако имамо формулу формирану од од исказних слова p_1, p_2,…, p_n вредност сваког од ових слова одговара вредности дате формуле, која може да буде тачна или нетачна. Ако две формуле A и B, чије су вредности a и b, онда су $\neg a$, $a \wedge b$, $a \vee b$, $a \Rightarrow b$ и $a \Leftrightarrow b$ вредности формула редом $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ и $(A \Leftrightarrow B)$.

Терм језика првог реда

Нека су a_1, a_2,…, a_n елементи неког скупа. Реч чија су слова елементи тог скупа, у ознаци a_1a_2,…,a_n, је пресликавање $\begin{pmatrix} 1,2 & \cdots & n \\ a1, a2 & \cdots & an \end{pmatrix}$ а број n је дужина ове речи a_1a_2,…,a_n. Константама ћемо сматрати тачно одређене симболе, којих има пребројиво много као што су a_1, a_2,…, a_n,.. док за променљиве узимамо унапред одабрано пребројиво много симбола као што су: x, y, z, x_1, y_1, z_1,… Операцијска слова (или функцијска слова) су симболи f, g, h… где је сваком операцијском слову додељен природан број који назвамо дужина слова (операција дужине n скупа A је пресликавање скупа A^n у скуп A). И операцијских слова имамо пребројиво много.

Сада може да се уведе дефиниција терма:

- Променљиве и знаци константа су терми.
- Ако су t_1, t_2,..., t_n терми,а f је операцијско слово дужине n, онда је и реч f(t_1, t_2,..., t_n) терм.
- Све термове добијамо једино коначном применом претходна два правила.

Применом прва два правила можемо да добијемо следеће терме:

x, y, f(x), f(y), f(z), g(x), f(g(x)), f(g(y), h(z), g(f(x), h(f(y)))

Користили смо у датим примерима и дефиницији терма и знаке заграда (), које ћемо назвати помоћним знацима. Овом дефиницијом одређени су најкраћи терми (променљиве и константе су терми дужине 1), а потом конструишемо дуже терме помоћу краћих.

Из ове дефиниције терма се види да уз свако операцијско слово f који има дужину n, ми имамо и пар заграда и n-1 зарез. Може се доказати (видети [1]) да се сваки терм приказује на јединствен начин помоћу својих подтермова. Терм можемо да рачунамо тек када га интерпретирамо у некој структури. Сваком терму одговара једно коначно дрво. Ради лакше визуализације, у неким примерима ћемо терме да прикажемо помоћу дрвета терма (енг. tree).

Дефиниција дрвета

Нека је дат скуп A. Дрво је бинарна релација на скупу A ако задовољава следеће услове:

1. представља парцијално уређење,
2. сваки тотално уређени подскуп је добро уређен.

Овде радимо са коначним дрветима и оно нам одговара за представљање терма.

Хијерархијска структура дрвета је одређена чворовима дрвета и везама између чворова на следећи начин: сваки чвор, осим једног, има тачно једног претходника, док сваки чвор има одређен број следбеника. Јединствен чвор који нема претходника зове се корен дрвета (енг. root) и представља врх хијерархије. Чворови који немају следбеника називамо листовима (енг. leaf) дрвета (или спољашњи чворови), док су сви остали чворови такозвани унутрашњи чворови, и представљају поддрво (подтерм).

У листовима дрвета ће бити променљиве и константе, а у унутрашњим чворовима операцијска слова. Ако је операцијско слово n-арано (операцијско слово дужине n), имаћемо n грана. Основна карактеристика дрвета је да нема циклуса, због чега између свака два његова чвора постоји јединствен пут. Код дрвета немамо циклусе, док су код графа могући циклуси. Граф је скуп чворова и скуп релација, које се називају линије графа, који описују везе између чворова.

Формални систем

Формални систем се састоји од скупа речи и у овом скупу речи издвојен је подскуп формула. У овом подскупу формула издвојен је подскуп аксиома. У скупу формула имамо релације (скуп релација) које називамо правила извођења.

Дефиниција 1. Коначан низ формула A_1, A_2,..., A_n формалног система се назива извођење (дедукција, доказ) ако свака од ових формула испуњава услов:

- A_i $(1 \leq i \leq n)$ је аксиома, или
- A_i је директна последица неких претходних формула низа по извесном правилу извођења.

Дефиниција 2. Формулу A_n формалног система називамо теорема, и записујемо $\vdash A_n$, ако постоји најмање један низ A_1, A_2,..., A_n који је извођење у формалном систему. Тај низ се назива извођење теореме A_n.

Дефиниција 3. Нека је дат неки скуп формула неког формалног система, и нека је B одређена формула тог система. Формулу B назваћемо последицом датог скупа формула, ако постоји коначан низ формула A_1, A_2,..., A_n , где је A_n једнака B, чија свака формула A_i $(1 \leq i \leq n)$ испуњава услов:

- A_i је аксиома или
- A_i је из датог скупа формула формалног система или
- A_i је директна последица неких претходних формула низа по извесном правилу извођења теорије формалног система.

Исказни рачун $\mathcal{L}$

Исказни рачун $\mathcal{L}$, или само рачун $\mathcal{L}$ представља формални систем у коме се формуле (исказне формуле) граде од променљивих и других логичких исказа, користећи логичке операције у складу са правилима тих операција (другим речима, радимо са исказима које формирамо користећи операције и променљиве). Слово $\mathcal{L}$ потиче од пољског математичара Јан Лукашијевича 1878-1965. Од кога потиче систем аксиома овог рачуна.

Основни симболи рачуна $\mathcal{L}$ су: p, q, r, (,), ¬, ⇒,… где симболе p, q, r називамо исказним словима. Исказне формуле уводимо на следећи начин:

- Исказна слова су исказне формуле.
- Ако су A и B исказне формуле, онда су исказне формуле и: $\neg A$, $(A \Rightarrow B)$.
- Исказне формуле се могу добити применом претходна два правила, коначан број пута.

С тим да не морамо да пишемо спољашње заграде. Аксиоме исказног рачуна су формуле:

1) $A \Rightarrow (B \Rightarrow A)$
2) $(\mathrm{A} \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
3) $(\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$

где су овде A, B, C било које исказне формуле. Тако да су и следеће формуле аксиоме: $r \Rightarrow (r \Rightarrow r)$, $\neg q \Rightarrow (p \Rightarrow \neg q)$, $r \Rightarrow (\neg q \Rightarrow \neg p)) \Rightarrow ((r \Rightarrow \neg q) \Rightarrow (r \Rightarrow \neg p))$. Једино правило извођења исказног рачуна је:

modus ponens: $\dfrac{A,\ A \Rightarrow B}{B}$.

Према овоме, извођење у исказном рачуну је сваки коначан низ формула A_1, A_2,..., A_n чија свака формула A_i ($1 \leq i \leq n$) задовољава један од следећа два услова:

- A_i је аксиома
- A_i произилази по правилу modus ponens из неке две претходне формуле тог низа.

Ако је A изказна формула, са $A(p_1,\ldots,p_n)$ означавамо да су сви елементарни искази који се јављају у формули A, неки од исказа $p_1,\ldots,p_n$. Нека су $A(p_1,\ldots,p_n)$ и A_1,..., A_n формуле. Формула која се из формуле A добија заменом свих јављања променљивих $p_1,\ldots,p_n$ редом формулама A_1,..., A_n , је инстанца формуле A.

4. УЈЕДНАЧАВАЊЕ

Нека су t_1 и t_2 два терма, која су састављена од променљивих (које ћемо да записујемо: p, q, r,..,x, y, z) и функцијских знакова датих дужина (обележимо их са f, g, h…).

Са k_1, k_2,…, k_n обележимо неке друге терме. Имајући у виду запис променљивих, дефинишимо функцију (пресликавање, замену), која ће променљиве p, q,..., z да пресликава на терме k_1, k_2,…, k_n

$$\boldsymbol{\mathcal{F}}: \quad \begin{array}{l} p \longrightarrow k_1 \\ q \longrightarrow k_2 \\ \vdots \\ z \longrightarrow k_n \end{array}$$

за које ћемо имати следећи услов: променљиве које учествују у термима t_1 и t_2 , редом p, q,..., z не сме да буде у терму, редом k_1, k_2,…, k_n осим ако је i-ти терм k_i идентички једнак i-тој променљивој p, q, r,..,x, y, z где је i=1,…,n. Дакле, променљива x не сме да буде у терму k_1 осим ако је k_1 = x, променљива y не сме да буде у терму k_2 осим ако је k_2 = y, и исто тако за остале променљиве.

Примера ради, $\boldsymbol{\mathcal{F}}$ сме да садржи пресликавање облика x ⟶ f(y, z), z ⟶ z, јер испуњава дати услов, међутим, x ⟶ f(x, z), не испуњава наш услов, па због тога $\boldsymbol{\mathcal{F}}$ не сме да садржи овако нешто (да се не би заглавило у рекурзији у алгоритму). Приликом пресликавања $\boldsymbol{\mathcal{F}}$, терми t_1, t_2 се пресликавају на два нова терма $t_1\boldsymbol{\mathcal{F}}$, $t_2\boldsymbol{\mathcal{F}}$. Дефинишимо појам уједначивости два терма:

Дефиниција 1. Терми t_1, t_2 су уједначиви, ако постоји бар једно пресликавање $\boldsymbol{\mathcal{F}}$ при коме су нови терми $t_1\boldsymbol{\mathcal{F}}$, $t_2\boldsymbol{\mathcal{F}}$ једнаки (поклапају се).

На пример, терми f(p), f(q) су уједначиви, ако је $\boldsymbol{\mathcal{F}}$ дефинисана овако:

$$\boldsymbol{\mathcal{F}}: p \longrightarrow q, \quad q \longrightarrow q.$$

Такође, терми f(z), f(g(y)) су уједначиви, јер их са пресликавањем

$$\boldsymbol{\mathcal{F}}: z \longrightarrow g(y), \quad y \longrightarrow y$$

доводимо до поклапања. Сада погледајмо два терма која нису уједначива: f(x), h(z) због тога што им функцијски знаци различити, они нису уједначиви.

У тражењу функције $\boldsymbol{\mathcal{F}}$, увек се тежи налажењу само најнужнијих поклапања, тј. уједначења. Пресликавање $\boldsymbol{\mathcal{F}}$ називаћемо уједначивач.

Из описа алгоритма уједначивања, за терме k_1, k_2,…, k_n може се претпоставити да су састављени само од основних делова термова t_1, t_2 дакле, од њихових функцијских знакова и знакова константи. Уз овакву претпоставку, за неке термове t_1, t_2 може се десити да имају више пресликавања $\boldsymbol{\mathcal{F}}$, које их доводи до поклапања. На пример, за термове t_1: g(x,y) и t_2: g(x,z) налазимо следећа три пресликавања која их доводе до поклапања:

$$\boldsymbol{\mathcal{F}}_1 : x \longrightarrow y, \quad y \longrightarrow z, \quad z \longrightarrow z$$
$$\boldsymbol{\mathcal{F}}_2 : x \longrightarrow z, \quad y \longrightarrow z, \quad z \longrightarrow z$$
$$\boldsymbol{\mathcal{F}}_3 : x \longrightarrow x, \quad y \longrightarrow x, \quad x \longrightarrow x$$

И свако од ова три пресликавања која могу да „поклопе“ терме t_1 и, t_2 назваћемо уједначивач. Погледајмо сада опште пресликавање $\boldsymbol{\mathcal{F}}$

$$\boldsymbol{\mathcal{F}} : \begin{array}{l} p \longrightarrow k_1 \\ q \longrightarrow k_2 \\ \vdots \\ z \longrightarrow k_n \end{array}$$

које ћемо записати овако $\boldsymbol{\mathcal{F}}((p, q,,.., z)) = (k_1, k_2,\ldots, k_n)$**.** Ово пресликавање делује на једну n-торку променљивих и производи неку другу n-торку термова (у овом нашем случају n-торку променљивих). Ако испустимо један пар заграда овог општег пресликавања, из претходног примера можемо да добијемо

$$\boldsymbol{\mathcal{F}}_1(x, y, z) = (x, z, z)$$
$$\boldsymbol{\mathcal{F}}_2(x, y, z) = (z, z, z)$$
$$\boldsymbol{\mathcal{F}}_3(x, y, z) = (x, x, x)$$

и приметити да се пресликавања $\boldsymbol{\mathcal{F}}_2$, $\boldsymbol{\mathcal{F}}_3$ могу изразити и преко $\boldsymbol{\mathcal{F}}_1$ на следећи начин:

$$\boldsymbol{\mathcal{F}}_2(x, y, z) = \boldsymbol{\mathcal{F}}_1(\phi(x, y, z))$$
$$\boldsymbol{\mathcal{F}}_3(x, y, z) = \boldsymbol{\mathcal{F}}_1(\pi(x, y, z))$$

где су ϕ и π пресликавања (замене, супституције) дефинисане на следећи начин:

$$\phi(x, y, z) = (z, z, z)$$
$$\pi(x, y, z) = (x, x, x).$$

За $\mathcal{F}_2$ и $\mathcal{F}_3$ рећи ћемо да су *примерци* од $\mathcal{F}_1$, тј. имамо следећу дефиницију:

Дефиниција 2. Нека су

$$\alpha(p,q,..,z) = (k_1,k_2,\ldots,k_n)$$
$$\beta(p,q,..,z) = (l_1, l_2 ,\ldots,l_n)$$

два пресликавања (замене, супституције). Кажемо да је β примерак од α уколико важи једнакост облика

$$\beta(p,q,\ldots,z\ ) = \alpha(\phi\ (p,q,..,z\ ))$$

где је ϕ пресликавање:

$$\phi : \quad \begin{array}{l} p \longrightarrow k_1 \\ q \longrightarrow k_2 \\ \quad \vdots \\ z \longrightarrow k_n \end{array}$$

Сада, ако се два дата терма могу уједначити, видели смо да се то може урадити на неколико начина, тј. можемо наћи неколико пресликавања (уједначивача) $\mathcal{F}$ ($\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$) која ће да доведу до поклапања полазне термове. Увек треба тражити најнужнија уједначења, тј. замене која доводе до поклапања. Због тога, постоји и такозвани *најопштији уједначивач*, у скраћеној ознаци MGU (назив потиче од енглеског израза *the most general unifier*). Неформално речено, уједначивач је оно шта се дешава за време процеса који доводи до поклапања, скуп свих неопходних замена да би полазни терми постали једнаки. Од тог најопштијег уједначивача, сви остали уједначивачи настају као његови примерци. Ако смо користили само неопходне замене, и у једном кораку смо могли да користимо две различите замене, до краја процеса уједначења могли бисмо у општем случају да добијемо два различита уједначивача. Оба ова уједначивача била би равноправна, зато што би могли да нађемо неку пермутацију променљивих, која би један од ова два уједначивача учинила идентичним другом уједначивачу, и било који од њих би могли да прогласимо за најопштији уједначивач, у ознаци MGU.

Теорема (теорема унификације): Ако се два терма могу уједначити, онда они имају најопштији уједначивач (MGU). Доказ ове теореме базира се на алгоритму унификације (видети на пример [9] или [12]).

5. АЛГОРИТАМ

Имајући у виду да термове градимо уз помоћ функцијских знакова, заграда, променљивих, добро би било да терм посматрамо као стринг. Нека је дат терм t, на пример

```
a(b(p,q),c(x,y),a(b(x,y),c(p,x)))
```

који посматран као стринг, на позицијама од 1, 2,... има знаке од којих ћемо да правимо разлику да ли су ти знаци променљиве, функцијски симболи или заграде.

Ако у њему пронађемо једно појављивање неког функцијског знака, потражићемо подтерм коме је тај функцијски знак намењен, тј. главни функцијски знак тог подтерма. У датом терму t, можемо да уочимо функцијски знак b, онда ће њему да одговара следећи подтерм

```
b(p,q).
```

Ово смо урадили на следећи начин: функцијски знак b је треће слово (налази се на трећој позицији) у стрингу, тј. у терму t. Почев од четвртог места па надаље у овом стрингу, бројимо леве и десне заграде. Први пут када нам број левих заграда буде једнак броју десних заграда, ми ћемо стати са бројањем. То ће бити на осмом месту. Тражени подтерм у терму, је „подстринг“, почевши од трећег па до осмог знака у стрингу.

Уопштено, у неком задатом терму t (који ћемо у имплементацији писати као стринг), подтерм који почиње на k-том месту са својим функцијским знаком, налазићемо на следећи начин: почевши од следећег места, тј. место k+1, бројаћемо леве и десне заграде. Први пут кад број левих и број десних заграда буде једнак, зауставићемо се. То је рецимо место p у терму. Тада је тај подтерм део терма t, који почиње на k-том, и завршава на p-том месту (позицији).

У нашем алгоритму нека су t_1, t_2 два дата терма са променљивима p, q, r,..,x, y, z. С обзиром да их посматрамо као стрингове, ми ћемо да их поредимо редом (место по место), и тражићемо прво место на коме се разликују, тј. не слажу (полазни терми, као стрингови имају на тој позицији различите знаке). Зато се овај алгоритам назива алгоритам првог несклада. Пред нама су две могућности:

1. Не постоји место неслагања, тј. полазни терми су међусобно једнаки. Тада се алгоритам завршава, и исписује се порука „терми су подударни“. Најопштији уједначивач не исписује се, јер нисмо имали никакву потребу за заменама да би дошло до поклапања већ подударних термова.
2. Терми t_1, t_2 нису једнаки и њихово прво место неслагања је место (позиција) под бројем r. Ако на том месту код оба терма стоји функцијски знак, алгоритам се завршава поруком „терми су неуједначиви“. Ако је једном од ова два терма на

том месту променљива, алгоритам се наставља, и постоји могућност да се дати терми могу уједначити одговарајућим заменама. Даље постоје две могућности:

- r -то место терма t_1 је променљива, на пример q . Сада, бројећи заграде у терму t_2 одредићемо подтерм који почиње на месту r. Тај подтерм означавамо са k_q и могао би да буде терм или променљива или знак константе (следеће место, r+1 није лева заграда). У случају да је k_q терм (није променљива или знак константе) проверавамо да ли се променљива q налази међу променљивима подтерма k_q, и ако се налази алгоритам се завршава поруком „терми су неуједначиви“. Ако се та променљива не налази у подтерму k_q, алгоритам се наставља. Памти се замена

  $$q \longrightarrow k_q$$

  и одмах у оба терма сва појављивања променљиве q заменићемо са подтермом k_q. Сада смо добили нека друга два терма, t'_1, t'_2 који имају једну променљиву мање него полазни терми t_1, t_2 (променљива q се не налази у новим термима t'_1 и t'_2). Сада, у овим новим термима настављамо да тражимо прво место неслагања...
- r -то место терма t_1 није променљива, али r -то место терма t_2 јесте променљива q. Ово се разматра исто као на претходно описан начин.

Сваки пут када се у оба терма промењива замени са подтермом, број променљивих је мањи, тако да се завршетком алгоритма може десити један од следећа два случаја:

1. Алгоритам стаје и исписује се порука „терми су неуједначиви“.

2. Алгоритам се завршава, и он је заменама типа $q \longrightarrow k_q$ уједначио полазне терме. Све замене које су се користиле да би се два полазна терма уједначила су се памтиле и користе се да би се исписао најопштији уједначивач. Променљиве које не мењамо назовимо слободним променљивима, и ако хоћемо можемо их заменити произвољним термима. Свака замена се памти по принципу: променљива=стринг. Нека смо имали неки број замена. Ако у i- тој замени имамо терм, који има променљиву која такође учествује у заменама (али није i-та замена, нека буде k-та замена), онда ће се та променљива у том терму који учествује у i-тој замени, променити, тј. заменити са оним што се мења у k-тој замени. Ако имамо укупно n променљивих, а укупно k замена, (имамо k-n слободних променљивих) при том k<n, програм треба да реши систем једначина по променљивима које се замењују. Програм то решава као систем једначина састављен од тих замена методом замене, тј. програм схвата те једначине као наредбе придруживања. То се може урадити са две „for“ петље, да низ наредби придруживања (тј. замене) прођу два пута.

6. ИМПЛЕМЕНТАЦИЈА АЛГОРИТМА

Код програма описаног у овом поглављу, написан је на програмском језику C++.

Подразумева се да су стрингови које уносимо заиста терми, што подразумева писање свих одговарајућих заграда, које су најбитније, јер их алгоритам пребројава.

Променљиве означавамо малим или великим словима p, P, q, Q,…, x, X, y, Y, z, Z. Треба водити рачуна да ли је променљива означена великим или малим словом, пошто се прави разлика између њих. При уносу термова можемо, али и не морамо користити зарез између променљивих, тј. уместо f(x, y, z) можемо писати f(x y z). Дакле „белине“ или зарезе у терму, свеједно је. Програм на крају исписује и извршене замене. Док је за променљиве битно како их означавамо, није нам битно са чиме ћемо да обележимо функцијске знаке и знаке константи, јер је алгоритам једино осетљив на променљиве и заграде. Програм ће радити било да смо терме писали са „префиксним“ функцијским знаком (тј. функцијски знак написан на самом почетку терма), или „инфиксно“ где смо функцијске знаке писали негде у „средини“. Овакав запис омогућује директну примена унификације на исказне формуле. С’тим да уместо негације у исказним формулама користићемо n, дакле уместо ¬A пишемо n(A). За импликацију користићемо симбол i уместо $\Longrightarrow$, тако да $A \Longrightarrow B$ пишемо i(A,B).

Да би проверили да су два унета стринга (терма) подударна, не треба да проверавамо стрингове позицију по позицију. То може да се провери на почетку и са уграђеним функцијама за рад са стринговима.

Постоје разни алгоритми уједначавања: линеаран алгоритам унификације, наиван алгоритам (који је написао Џон Алан Робинсон)… Алгоритам који је овде написан је такозвани алгоритам првог несклада (тражи се прво место неслагања у термовима, тј. стринговима).

7. КОД ПРОГРАМА

```
#include<iostream>
#include<string>
#include<cstdlib>

using namespace std;

int f1(char w[], int k, char v);
int f2(char a[], char b[], char c[], int p);
main()
{
     char t1[100], t2[100], w[100], v, r[100][100], a[100], b[100], c[100], tt[100];
     int i, k, bb=-1, t, j, j2, p, n, kk=0;
     int poj[300], l, br, m[100], f[20][20];
     char red[20], par[20][20][100], var[300][100], su[100];

     for(i=0;i<20;i++)
       for(j=0;j<20;j++)
         f[i][j]=0;

     cout<<"unesi prvi term"<<endl;
     cin.getline(t1, 100);

    cout<<"unesi drugi term"<<endl;
     cin.getline(t2, 100);

     if(strcmp(t1,t2)==0)
     {
       cout<<"Oni su podudarni"<<endl;
       system("PAUSE");
       return 0;
     }

     for(i=0;t1[i]!='\0';i++)
     {
      if(t1[i]!=t2[i])
      {
       if((t1[i]>111 && t1[i]<123) || (t1[i]>79 && t1[i]<91))
       {
        strcpy(w,t2);
```

```
k=i;
v=t1[i];

t=f1(w,k,v);

bb++;
r[bb][0]=t1[i];
r[bb][1]='=';

for(j=k,j2=2;j<=t;j++,j2++)
{
 r[bb][j2]=t2[j];
 tt[j2-2]=t2[j];
}
r[bb][j2]='\0';
tt[j2-2]='\0';

strcpy(a,t1);
strcpy(b,t2);
strcpy(c,tt);
p=k;

i=f2(a,b,c,p);

strcpy(t1,a);
strcpy(t2,b);
}

else if((t2[i]>111 && t2[i]<123) || (t2[i]>79 && t2[i]<91))
{
 strcpy(w,t1);
 k=i;
 v=t2[i];

 t=f1(w,k,v);

 bb++;
 r[bb][0]=t2[i];
 r[bb][1]='=';

 for(j=k,j2=2;j<=t;j++,j2++)
 {
  r[bb][j2]=t1[j];
  tt[j2-2]=t1[j];
```

```
    }
    r[bb][j2]='\0';
    tt[j2-2]='\0';

    strcpy(a,t2);
    strcpy(b,t1);
    strcpy(c,tt);
    p=k;

    i=f2(a,b,c,p);

    strcpy(t1,b);
    strcpy(t2,a);
   }
   else {
       cout<<"Nisu ujednachivi"<<endl;
       system("PAUSE");
       exit(1);
       }
  }
}

cout<<"Ujednachivi su; rezultat je:" <<t2<<endl;

for(i=0;i<300;i++)
  poj[i]=-1;

n=bb+1;
for(i=0;i<n;i++)
{
  poj[r[i][0]]=i;
  red[i]=r[i][0];
}

for(i=0;i<n;i++)
{
  l=2;
  br=-1;

  for(j=2; r[i][j]!='\0'; j++)
  {
    if(r[i][j] < 80 || (r[i][j]>90 && r[i][j]<112) || r[i][j]>122)
      continue;
```

```
   if(poj[r[i][j]]==-1) continue;

   if(j!=l)
   {
     br++;
     for(k=l;k<j;k++)
       par[i][br][k-l]=r[i][k];
     par[i][br][j-l]='\0';
   }
    br++;
    par[i][br][0]=r[i][j];
    par[i][br][1]='\0';
    f[i][br]=r[i][j];
    l=j+1;
  }

  if(j!=l)
    {
      br++;
      for(k=l;k<j;k++)
        par[i][br][k-l]=r[i][k];
      par[i][br][j-l]='\0';
    }

  m[i]=br;
}

for(i=0;i<300;i++)
{
  var[i][0]=i;
  var[i][1]='\0';
}

do{
for(i=0;i<n;i++)
{
  su[0]='\0';
  for(j=0;j<=m[i];j++)
  {
   if(f[i][j]==0)
     strcat(su,par[i][j]);
   else strcat(su,var[f[i][j]]);
  }
  strcpy(var[red[i]],su);
```

```
    }

    kk++;
    }while(kk<n+1);


    cout<<"Obavljene su zamene"<<endl;
    for(i=0;i<n;i++)
      cout<<red[i] <<"----->"<<var[red[i]]<<endl;

    system("PAUSE");
}

int f1(char w[], int k, char v)
{
    int br, i;
    char lt=w[k], lt1;

    if(lt=='(')
    {
      br=-1;
      for(i=k+1;w[i]!='\0';i++)
      {
        lt=w[i];
        if(lt==v)
        {
          cout<<"Nisu ujednachivi"<<endl;
          system("PAUSE");
          exit(1);
        }
        if(lt=='(') br--;
        if(lt==')') br++;
        if(br==0) return i;
      }
    }
    else
    {
     if((lt>111 && lt<123) || (lt>79 && lt<91))
       return k;
     lt1=w[k+1];
     if(lt1!='(') return k;
     br=0;
     for(i=k+1;w[i]!='\0';i++)
      {
```

```
        lt=w[i];
        if(lt==v)
        {
          cout<<"Nisu ujednachivi"<<endl;
          system("PAUSE");
          exit(1);
        }
        if(lt=='(') br--;
        if(lt==')') br++;
        if(br==0) return i;
      }
    }
}
int f2(char a[], char b[], char c[], int p)
{
   int bl=0, i, d;
   char su1[100]="", su2[100]="";

   for(i=0;a[i]!='\0';i++)
   {
     if((a[i]==a[p]) && (i<=p))
     {
       strcat(su1,c);
       bl+=strlen(c);
     }
     else if((a[i]==a[p]) && (i>p))
             strcat(su1,c);
         else
         {
           d=strlen(su1);
           su1[d]=a[i];
           su1[d+1]='\0';

           if(i<p) bl++;
         }
   }

   for(i=0;b[i]!='\0';i++)
   {
     if(b[i]==a[p])
       strcat(su2,c);
     else
     {
           d=strlen(su2);
```

```
            su2[d]=b[i];
            su2[d+1]='\0';
          }
        }

        strcpy(a,su1);
        strcpy(b,su2);
        return bl;
}
```

8. ПРИМЕРИ УНИФИКАЦИЈЕ

Пример 1.

Нека су t_1 и t_2 дати терми:

t_1: f(x, g(z))
t_2: f(g(y), g(y)),

где су x, y, z промењиве, а f и g су функцијски знаци. Уочавамо да је прво место неслагања одређено позицијом 3. У првом терму се на том месту појављује промењива x, а у другом терму се појављује функцијски знак. Алгоритам унификације, у другом терму у односу на први, на позицији 3 уочава подтерм g(y). У складу са тим уочава се замена:

$$x \rightarrow g(y).$$

После обављања ове замене полазни терми изгледају:

t_1: f(g(y), g(z))
t_2: f(g(y), g(y)).

Сад се проверава следеће место неслагања ова два терма. Након провере добијамо да је место на коме се разликују одређено позицијом 10. У оба терма се на том месту појављује промењива, код првог терма z, а код другог y. Алгоритам унификације промењиву првог терма замењује промењивом другог терма, тј. имамо замену

$$z \rightarrow y$$

(напоменимо да је логички могућа и обрнута замена: $y \rightarrow z$). Након обављања ове замене, дати терми прелазе у

t_1: f(g(y), g(y))
t_2: f(g(y), g(y)).

Ови терми су подударни, па самим тим полазни терми су уједначиви. Скуп ових замена јесте уједначивач:

$$x \rightarrow g(y)$$
$$z \rightarrow y.$$

Сада треба испитати да ли је овај уједначивач истовремено и најопштији уједначивач. То питање се поставља због постојања две замене. У другом кораку напоменули смо да је логички могућа и обрнута замена, односно имамо и другу замену $y \rightarrow z$. Да смо применили другу замену, стигли би до другог уједначивача:

$$x \rightarrow g(y)$$
$$y \rightarrow z$$

па би после друге замене полазни терми изгледали овако:

t_1: f(g(z), g(z))
t_2: f(g(z), g(z))

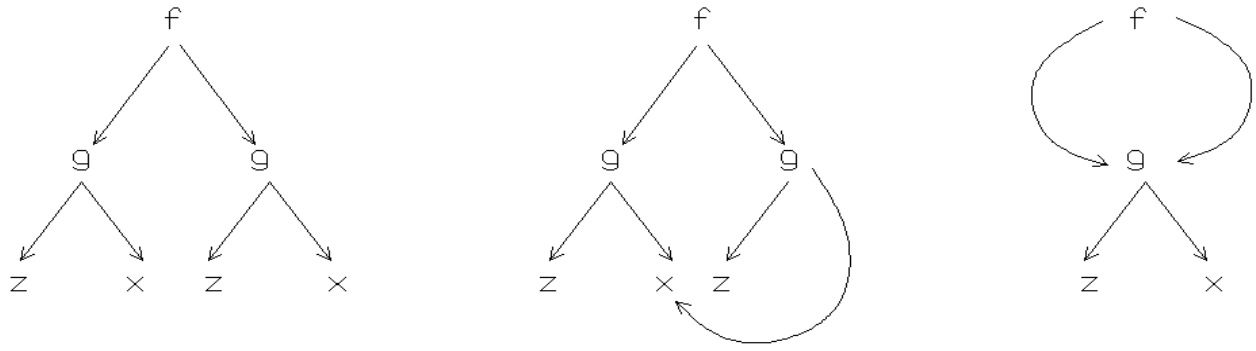
Оба ова уједначивача су равноправна и сваки од њих се може узети као најопштији уједначивач, тј. MGU. Оба уједначивача су примерци један другом, што се проверава ако се уочи ова пермутација променљивих $y \rightarrow z$, $z \rightarrow y$.

Овај пример се тестира написаним програмом који је дат у поглављу 7, и добијамо следеће:

```
unesi prvi term
f(x,g(z))
unesi drugi term
f(g(y),g(y))
Ujednachivi su; rezultat je:f(g(y),g(y))
Obavljene su zamene
x----->g(y)
z----->y
Press any key to continue . . .
```

Како сваком терму одговара једно дрво, због лакше визуализације користимо дрво за приказ терма. У листовима дрвета (чворови који немају следбеника) ће бити приказане променљиве, а у унутрашњим чворовима операцијска слова. Симболе у чворовима повезујемо са линковима на доле тј. везама на доле. На овај начин представљамо хијерархијски ред у терму.

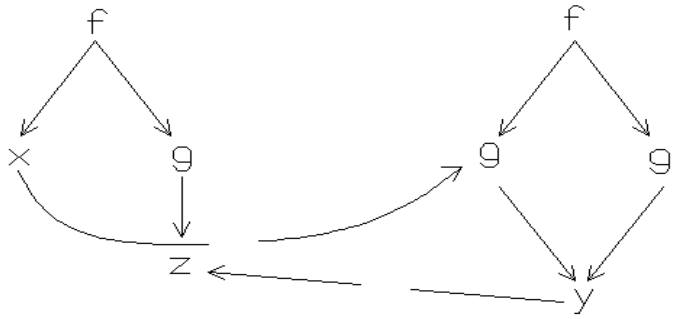
У таквом једном дрвету, сваки чвор има природну интерпретацију као подтерм и ми ћемо говорити о чворовима и подтермовима као да је то једно исто. (нпр. чвор f(x,y) је онај означен са f који има лукове (везе, линкове) ка чворовима, тј. листовима x и y). На неки начин, на дрвету се лакше уочава како је терм подељен на подтермове. На пример, можемо приказати терм f(g(z,x), g(z,x)) било којим од следећих дрвећа са слике:



Нека имамо два терма који се уједначавају. Замене могу бити представљене директно у вези са чворовима два дрвета, и оно шта мењемо је повезано линком (који се зове лук замене). У датом примеру 1, два терма f(x,g(z)) и f(g(y),g(y)), и њихов уједначивач,

$$x \rightarrow g(y)$$
$$y \rightarrow z$$

могу се представити графички, тј. ми ћемо замене из уједначивача представити стрелицом, на пример замену $x \rightarrow g(y)$ приказаћемо стрелицом која води од x до $g(y)$. Са овим начином представљања замена добијамо скицу коју називамо граф.

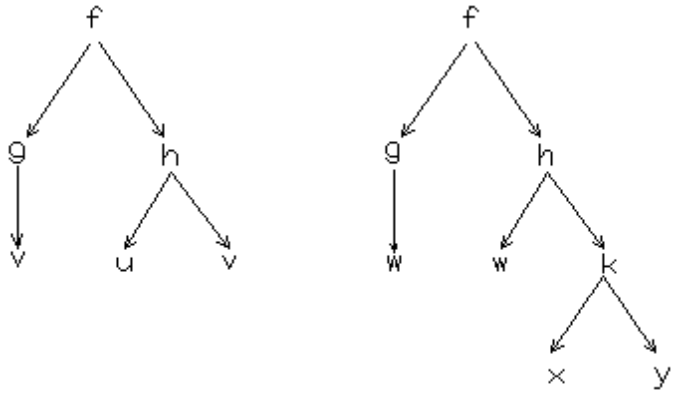


Пример 2.

Нека су сада дати терми

```
t1:        f(g(v), h(u, v))
t2:        f(g(w), h(w, k(x,y))).
```

Ради лакше визуализације, можемо да их прикажемо као дрвеће:



Уочавамо да на позицији 5 се налази прво место неслагања. У оба терма се на тој позицији налази променљива. У t_1 је v , а у t_2 је w. Наш алгоритам ће променљиву првог терма заменити промњнљивом другог терма, дакле

```
v   →   w
```

(и у овом примеру напоменимо да је логички могућа и обрнута замена, и да је могуће имати више уједначивача).

Кад обавимо ову замену, терми t_1 и t_2 прелазе у следећа два терма

```
t1:        f(g(w), h(u, w))
t2:        f(g(w), h(w, k(x,y))).
```

Сада опет тражимо прво неслагање, које налазимо на позицији 9. У оба терма се налази променљива. Од могуће две замене, описан алгоритам узима замену

```
u   →   w.
```

Кад обавимо и ову замену, терми t_1 и t_2 су:

```
t1:        f(g(w), h(w, w))
t2:        f(g(w), h(w, k(x,y))).
```

Тражећи следеће неслагање, налазимо да у првом терму на позицији 12 се налази променљива, а да у другом терму се појављује функцијски знак. У складу са тим уочава се замена:

```
w   →   k(x,y)
```

где после њене примене добијамо подударне термове:

```
t1:     f(g(w), h(w, k(x,y)))
t2:     f(g(w), h(w, k(x,y)))
```

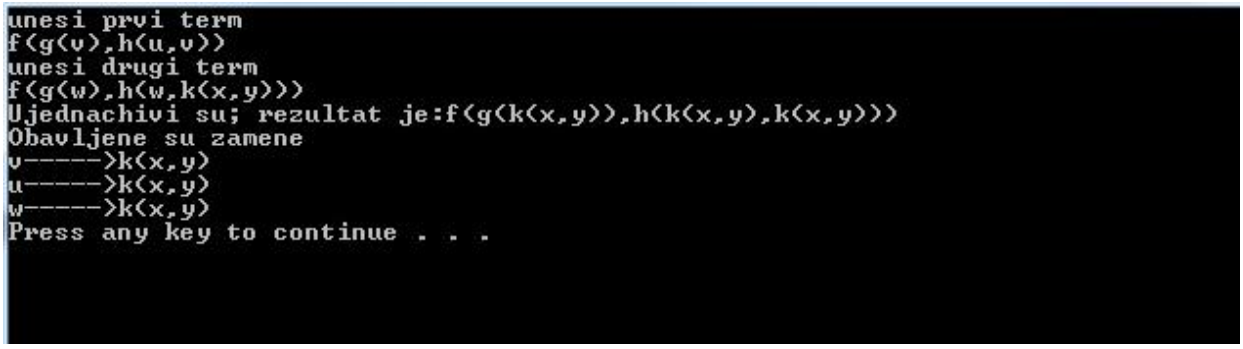
замене које смо обавили су:

```
v   →   w
u   →   w
w   →   k(x,y).
```

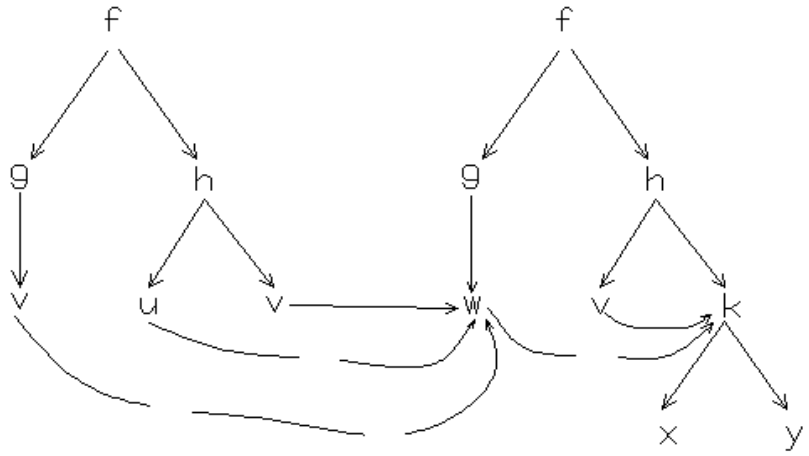
Међутим, када смо користили трећу замену у трећем кораку, тада се променљива `w` заменила са `k(x,y)` на којој год се позицији налазила после друге замене. Тако да је уједначивач:

```
v   →   k(x,y)
u   →   k(x,y)
w   →   k(x,y).
```

Ако бисмо нашли неку пермутацију променљивих, нпр. w→u и u→w, добили би још један уједначивач. Оба ова уједначивача су примерци један другоме. Тестирано програмом дат у поглављу 7, добијамо:



што можемо да прикажемо преко графа:

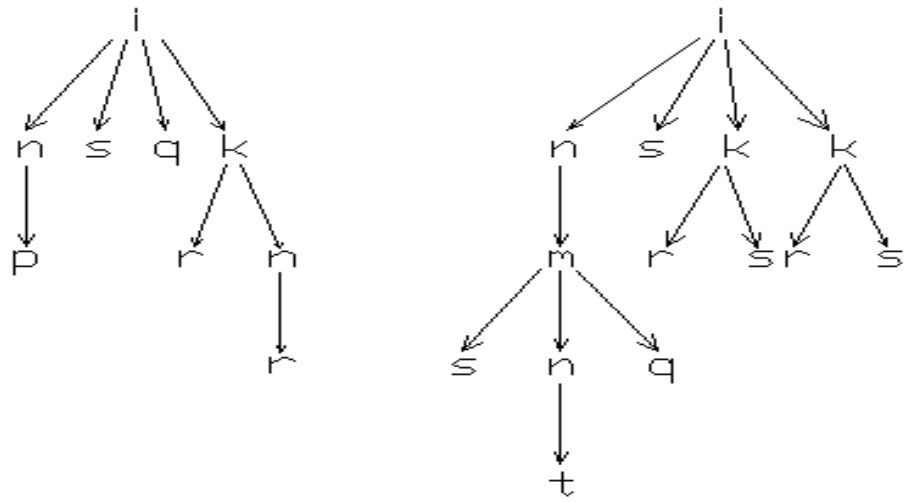


Пример 3.

Погледајмо сада терме t_1 и t_2:

```
t1 :                i( n(p), s, q, k(r, n(r)) )
t2 :                i( n(m(s, n(t), q)), s, k(r, s), k(r, s) ) )
```

Приказани терми преко дрвета изгледају:



У ова два терма примећује се да је прво место неслагања на позицији број 5. У првом терму на петом месту се појављује променљива p, а у другом терму се појављује функцијски знак. Алгоритам унификације, у другом терму у односу на први, на месту 5 уочава подтерм m(s, n(t), q). У складу са тим уочава се замена:

```
p  →  m(s, n(t), q).
```

После обављања ове замене полазни терми изгледају:

```
t1 :              i( n(m(s, n(t), q)), s, q, k(r, n(r)) )
t2 :              i( n(m(s, n(t), q)), s, k(r, s), k(r, s) ) ).
```

Сада кренимо да проверавамо следеће место неслагања ова два терма. Након провере добијамо да је место на коме се разликују одређено позицијом 20. У првом терму на том месту је променљива q, а у другом терму променљивој одговара подтерм k(r, s). Сада треба да урадимо следећу замену:

```
q  →  k(r, s).
```

После обављања ове замене добијамо следећа два терма:

```
t1 :      i( n(m(s, n(t), k(r, s))), s, k(r, s), k(r, n(r)) )
t2 :      i( n(m(s, n(t), k(r, s))), s, k(r, s), k(r, s) ) ).
```

Поновном провером ова два терма закључујемо да је следеће место неслагања на позициији 36. На том месту се у другом терму налази променљива s, којој у првом терму одговара подтерм n(r). Замена коју радимо је

```
S  →  n(r).
```

После треће замене добијамо два подударна терма:

```
t1:  i(n(m(n(r), n(t), k(r,n(r)))), n(r), k(r,n(r)), k(r,n(r)))
t2:  i(n(m(n(r), n(t), k(r,n(r)))), n(r), k(r,n(r)), k(r,n(r)))
```

Сада се може видети да су терми t_1 и t_2 уједначиви. Замене које смо извели и које доводе до подудадрности полазних термова су:

```
p  →  m(s, n(t), q)
q  →  k(r, s)
S  →  n(r).
```

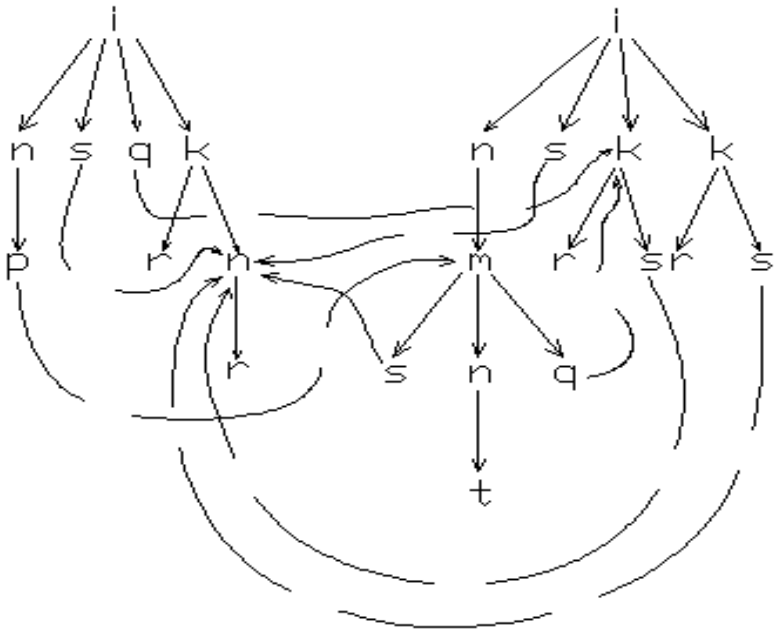
У другој замени мењали смо променљиву q са k(r,s). Где год се у заменама јављала променљива q, мењали смо је са k(r,s). Слично, када смо користили трећу замену у трећем кораку, тада се променљива s заменила са n(r) на којој год се позицији налазила после треће замене. Тако да је уједначивач

```
p  →  m(s, n(t), k(r, n(r)))
q  →  k(r, n(r))
S  →  n(r).
```

Ове замене су и минималне замене које треба извести да би дошло до подударности термова, па самим тим и одређују најопштији уједначивач, MGU. Проверено датим програмом, добијамо следећи резултат:

```
unesi prvi term
i(n(p),s,q,k(r,n(r)))
unesi drugi term
i(n(m(s,n(t),q)),s,k(r,s),k(r,s)))
Ujednachivi su; rezultat je:i(n(m(n(r),n(t),k(r,n(r)))),n(r),k(r,n(r)),k(r,n(r))
))
Obavljene su zamene
p----->m(n(r),n(t),k(r,n(r)))
q----->k(r,n(r))
s----->n(r)
Press any key to continue . . .
```

Ове замене можемо да представимо следећим графом:

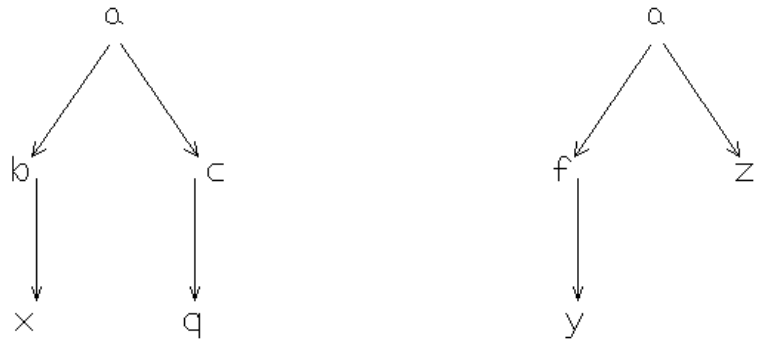


Пример 4.

Погледајмо сада терме t_1 и t_2 који нису уједначиви:

```
t1 :                a( b(x), c(q) )
t2 :                a( f(y), z )
```

Приказани терми преко дрвета изгледају:



Оба терма на трећој позицији немају променљиву, тако да не постоје никакве замене које би могле да уједначе ова два терма. Дакле, терми су неуједначиви. Тестирано програмом, добијамо следећи резултат:

```
Unesi prvi term:
a(b(x),c(q))
Unesi drugi term:
a(f(y),z)

 termi nisu ujednacivi
Press any key to continue . . .
```

9. УОПШТЕЊА И ПРИМЕНЕ

Осим за термове, алгоритам унификације и написан програм могу се тестирати и на формулама исказне логике (директна примена унификације на исказне формуле), ако исказне формуле схватимо као терме. Онда можемо да упоређујемо облике формула. Ако су формуле истог облика, нпр. аксиоме 1 онда ће оне бити унификоване са формулом $p \Longrightarrow (q \Longrightarrow p)$, слично за остале аксиоме. На овај начин може да се провери да ли је појединачна формула облика аксиома или није. Ради лакшег записа уместо негације користићемо n, дакле уместо $\neg p$ користимо n(p). За импликацију користићемо симбол i уместо $\Longrightarrow$, тако да $p \Longrightarrow q$ пишемо i(p,q).

Пример 1.

Ако узмемо формуле

$$x \Longrightarrow (y \Longrightarrow x)$$
$$\neg q \Longrightarrow (\neg p \Longrightarrow \neg q)$$

можемо проверити да ли су уједначиве. Ако буду уједначиве, то ће значити да је друга формула инстанца аксиоме 1 исказне логике. Овде су нам p, q, x, y исказна слова, а у коду програма су нам променљиве. У свим примерима из исказне логике, исказна слова из једне сматраћемо независним од исказних слова друге формуле. Из тог разлога приликом тестирања нашег програма за формуле из исказне логике смо користили различита исказна слова у првој и другој формули. Прву формулу смо записали као

i(x, i(y, x))

а другу формулу

i(n(q), i(n(p), n(q))).

Након пропуштања примера кроз програм добије се да су формуле (терми) уједначиве, ако се изврше следеће замене

$$x \longrightarrow n(q)$$
$$y \longrightarrow n(p).$$

```
unesi prvi term
i(x,i(y,x))
unesi drugi term
i(n(q),i(n(p),n(q)))
Ujednachivi su; rezultat je:i(n(q),i(n(p),n(q)))
Obavljene su zamene
x----->n(q)
y----->n(p)
Press any key to continue . . .
```

Дакле, друга формула се може уједначити са првом аксиомом исказног рачуна, тј. инстанца је аксиоме 1.
Пример 2.

Нека сада имамо формуле

$$(\neg y \Longrightarrow \neg x) \Longrightarrow (x \Longrightarrow y)$$
$$(\neg\neg q \Longrightarrow \neg p) \Longrightarrow (p \Longrightarrow \neg q)$$

за које се питамо да ли су уједначиве. У складу са договором писања из примера 1, користимо различита исказна слова у првој и другој формули p, q, x, y. Сада програм треба да тестира да ли су следеће формуле (терми) уједначиве

$$i(\, i(\, n(y),\, n(x)\,),\, i(x,y)\,)$$
$$i(\, i(\, n(n(q)),\, n(p)\,),\, i(p,\, n(q)\,)\,).$$

Као одговор добијамо да су терми уједначиви, што значи да су ове две формуле уједначиве ако се изврше замене

$$y \longrightarrow n(q)$$
$$x \longrightarrow p.$$

```
unesi prvi term
i(i(n(y),n(x)),i(x,y))
unesi drugi term
i(i(n(n(q)),n(p)),i(p,n(q)))
Ujednachivi su; rezultat je:i(i(n(n(q)),n(p)),i(p,n(q)))
Obavljene su zamene
y----->n(q)
x----->p
Press any key to continue . . .
```

Сада видимо да се друга формула може уједначити са трећом аксиомом исказног рачуна, другим речима, инстанца је аксиоме 3.

Пример 3.

Слично, ако посматрамо исказне формуле

$$(\neg y \Longrightarrow (\neg\neg z \Longrightarrow x)) \Longrightarrow ((\neg y \Longrightarrow \neg\neg z) \Longrightarrow (\neg y \Longrightarrow x))$$
$$(p \Longrightarrow (q \Longrightarrow r)) \Longrightarrow ((p \Longrightarrow q) \Longrightarrow (p \Longrightarrow r))$$

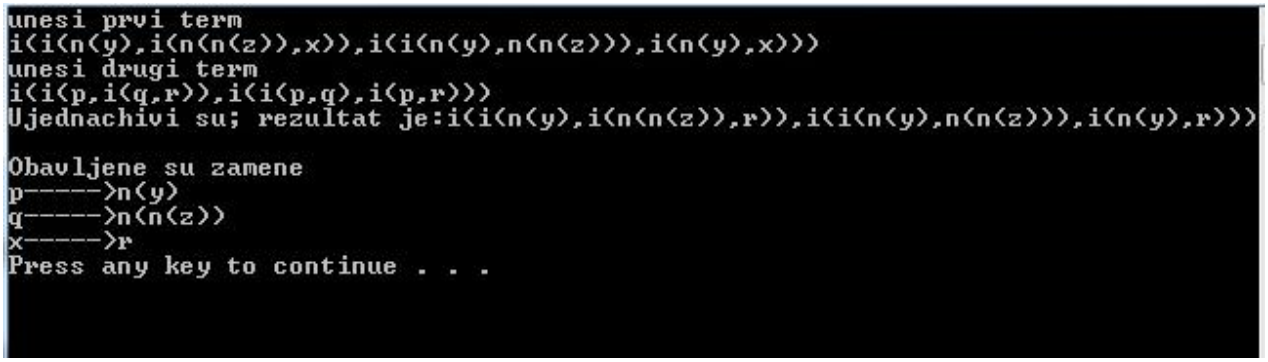
можемо утврдити да су оне уједначиве. Приметимо да је друга формула добијена директно из друге аксиоме исказног рачуна.
Два терма која програм треба да провери да ли су уједначива су:

i(i(n(y), i(n(n(z)), x)), i(i(n(y), n(n(z))), i(n(y), x)))
i(i(p, i(q, r)), i(i(p, q), i(p, r))).

Након тестирања, програм даје одговор да су уједначиви, и да су замене

$p \longrightarrow n(y)$
$q \longrightarrow n(n(z))$
$x \longrightarrow r.$

Обе формуле су инстанце друге аксиоме.



Пример 4.

Погледајмо две формуле исказне логике које се не могу уједначити

$$x \Longrightarrow (y \Longrightarrow x)$$
$$(q \Longrightarrow p) \Longrightarrow p$$

које ћемо у програму да запишемо као

i(x, i(y, x))
i(i(q, p), p)

Ми имамо могућност да уз малу дораду за MODUS PONENS проверимо да ли се низ формула састоји од аксиома или из формула изведених из претходних правилом извођења (MODUS PONENS). На овај начин можемо да овај алгоритам проширимо до употребе за аутоматизовану проверу да ли је низ формула доказ или није у исказној логици. На сличан начин применом алгоритма унификације може се изградити алгоритам за проверу доказа у различитим формалним системима, који су по својој природи слични исказној логици.

Алгоритам унификације се не може директно применити на упоређивање синтаксних формула предикатске логике првог реда, па се претходна примена на изградњу алгоритама за проверу доказа у исказним формалним системима не може директно спровести на формалне системе – формалне теорије у логици првог реда. Међутим, уопштењем алгоритма унификације може се остварити алгоритам наведеног типа у предикатском случају (погледати решења понуђена у [12]). Могућа су уопштења алгоритма унификације у различитим смеровима која обезбеђују разноврсне значајне примене, пре свега увођење сличности у синтаксним структурама.

Унификација има велики број реализација. Неке од најважнијих примена су трансформације укључене у молекуларну биологију, генетику, програмирање, лингвистику, музику...

Алгоритам унификације се може модификовати за упоређивање структуралне сличности молекуларно биолошких секвенци – генетских секвенци. Модификован алгоритам се користи и у процедурама доказивања удаљености врста у еволуцији, мерењу сличности у секвенцама молекуларне биологије (и структурама), праћењу мутација и разумним (еволуциским) прорачунима удаљености у еволутивним системима, истраживања унутрашњих мутација у вирусологији и имунологији.

10. УНИФИКАЦИЈА И ПРОЛОГ

Назив програмског језика PROLOG настаје као скраћеница од енглеских речи, (PROgramming in LOGic) 1972. године, а назив наговештава да је реч о програмском језику који је намењен логичком програмирању. Његов творац, француски научник Ален Калмеро, је прву верзију представио у Марсељу, док је посебно значајан шести IFIP конгрес 1974. године, на коме енглески научник Роберт Ковалски упознаје ширу научну јавност са идејом логичког програмирања и постојањем PROLOG-а. Првенствено намењен за примену у вештачкој интелигенцији, за њега се нераскидиво веже и назив „језик вештачке интелигенције“. Он се примењује за аутоматско доказивање теорема, експертне системе, NLP (natural language proccessing) обраду природних језика да би рачунар могао да комуницира. Када су Јапанци осамдесетих година објавили да ће њихови рачунари бити засновани на PROLOG-у, тада он постаје најпознатији језик вештачке интелигенције.

Програмер у процедуралним програмским језицима алгоритмом даје инструкције рачунару како да реши задати проблем, док у PROLOG-у, који је представник дескриптивних програмских језика, програмер описује шта програм треба да ради. Поступак за налажења решења заснива се на принципу резолуције (процедура закључивања која је заснована на само једном ефикасном правилу извођења) за аутоматско доказивање теорема коју је први описао Робинсон 1965. године.

Примена метода резолуције састоји се у синхронизацији између задатог циља са базом података (чињеницама). Ако PROLOG може да испуни циљ у складу са базом чињеница и правила, PROLOG ће дати позитиван одговор “YES”, а ако не може да испуни циљ, онда је одговор “NO”.

У програмском језику PROLOG, синтаксно исправан низ знакова назива се терм. Терм може да буде структура или прости терм, а прости терм може да буде константа или променљива. Константа може да буде атом или број. Атом који је максималне дужине 255 знакова, може се конструисати на неколико начина, а најчешће као низ слова и цифара, или као произвољан низ карактера између апострофа. Променљиве конструишемо као низ слова, цифара и знака _ а почињу великим словом или знаком _. Анонимна променљива _ се користи када нам вредност променљиве није важна. При конструкцији структуре, њено име је атом, а аргументи су терми. На пример:

imeStrukture(argument1, argument2, ..., argument3)
datum(10, 03,2010)
datum(maj, 2010)
student(filip,radulovic, datum(29,06,1984)).

У овом програмском језику, унификација (генерални тип доделе) је најчешће коришћена операција над подацима, а често се користи као механизам за описивање односа међу структурним објектима (и иначе, унификације је често коришћена операција у логичном програмирању и у многим апликацијама вештачке интелигенције). Могу да се унификују различите структуре података, и то:

- атом ће се увек унификовати са атомом ако су им вредности исте.
- Терм ће се унификовати са термом (а структура са структуром, листа са листом) ако су функтори исти, а аргументи се могу унификовати.
- Атом или структура се могу унификовати са променљивом.
- Променљива се може унификовати увек са променљивом.

Унификација два типа података треба да нам да и скуп замена преко којих се променљивим компонентама додељују вредности. Унификација се у PROLOG-у задаје оператором = , на пример:

```
?-X=1.
X=1
yes

?- f(a,b)=f(a,b).
yes

?- a=b.
no

?- f(X,Y)=f(a,b)
X=a
Y=b
yes

?-f(X,b)=f(a,Y).
X=a
Y=b
yes

?-datum(D,M,2014)=datum(10, mart,G).
D=10
M=mart
G=2010
yes

?-datum(D,M,2014)=datum(mart,G).
No
```

?-X=Y, X=1.
X=1
Y=1
yes

?-X=3, X=1.
no

?-oprema(racunar, A, tastatura, B)= oprema(C, monitor, tastatura, mis).
A=monitor
B=mis
C=racunar
yes

Оператор „=“ успева, терм1=терм2, ако се терм1 и терм2 могу унификовати.

Програмски језик PROLOG има могућност да се добију одговори на упите из базе знања, која је састављена од чињеница и правила. Ако би имали базу знања главних градова свих држава света, пример PROLOG програма који одређује главни град државе на основу имена може да се напише овако:

```
glavni(srbija, beograd).
glavni(makedonija, skoplje).
glavni(rusija, moskva).
glavni(francuska, pariz).


Glavnigrad : - nl, nl,
          write('unesi ime drzave, a potom i tacku: '),
          read(R),
          glavni(R,G),nl,
          write('glavni grad drzave je: '),
          write(G).
```

Овај програм можемо да позовемо помоћу предиката Glavnigrad, а да би успео, неопхпдно је да се изврши унификација по чињеницама glavni(drzava, grad). После покретања програма, наш пример би овако изгледао:

```
?- glavnigrad
   unesi ime drzave, a potom i tacku: srbija
   glavni grad drzave je:  beograd
```

11. ЗАНИМЉИВОСТИ

Вештачка интелигенција данас (или у скорије време) је присутна у скоро свим сферама живота, тако да се у септембру 2002. године у продаји појављује играчка „Cindy Smart“, чији хардвер и софтвер јој омогућавају препознавање говора, од преко 700 речи и 77 фраза, уз могућност синтезе говора, читање и изговарање тачног времена. Док чита, препознаје „ружне“ речи и одбија да их изговори. Гледајући на папиру записане најједноставније математичке задатке изговарала је решења која је сама рачунала. Њена цена тада је била око 150 долара.

У мају 1999. године програм вештачке интелигенције познат као Remote Agent управљао је летелицом Deep Space 1 на њеном свемирском путовању 100 000 000km далеко од Земље. Развијен од стране америчке агенције NASA (National Aeronautics and Space Administration) представљао је први контролни систем вештачке интелигенције за управљање свемирским летелицама без интервенције човека.

У мају 1997. године, Deep Blue рачунар компаније IBM усавршен за играње шаха, побеђује тадашњег светског првака у шаху, Гарија Каспарова. Deep Blue је могао да процени преко 200 милиона позиција у секунди. Иако 1997. године на 259. месту пописа најбржих рачунара, Deep Blue је анализирао 700 000 двобоја шаховских велемајстора и сам проценио најбоље потезе за сваку позицију. Савршенија верзија овог рачунара, Deep Fritz, у Немачкој 2006. године понавља успех свог претходника победивши тадашњег светског првака у шаху, Владимира Крамника, са резилтатом 4:2. После овог пораза неки сматрају да људи више немају шансе против „вештачких велемајстора шаха“.

Иако је шах игра са ограниченим скупом правила, ова победа се сматра прекретницом у достигнућима вештачке интелигенције. Следеће шта се поставља пред ВИ јесте сналажење у физичком свету, где свако правило има много изузетака, а на путу до циља има доста непредвидивих ситуација. Једна од таквих области примене ВИ јесте вожња аутомобилом.

Америчка агенција Дарпа (DARPA - Defense Advanced Research Projects Agency) организовала је 2005. године такмичење у употреби аутономних возила, која би прелазила пут дуг 240km кроз планинске и пустињске путеве Калифорније и Неваде, и те године је 5 аутомобила стигло до циља. Многи аутомобили данашњице имају уграђене технологије развијене кроз DARPA као помоћ возачу. Прва саобраћајна дозвола за аутономно возило издата је у Невади 2012. године.

Следећи проблем који се поставља пред ВИ јесте одговарање на питања постављена на природном језику. IBM-ов систем вештачке интелигенције WATSON је способан да одговара на питања постављена на природном језику. Прву награду од милион долара WATSON добија победом на познатом TV квизу Jeopardy 2011. године у коме су му

питања постављана на природном језику. За време квиза, WATSON је имао приступ 200 милиона страница разног текста, и потпун приступ Википедији, али није имао приступ интернету. Користи преко 100 различитих техника да препозна говор. Watson користи IBM-ов софтвер DeePQA написан на неколико језика: C++, Java и Prolog.

Волфрам алфа је систем који свако од нас може да тестира на сајту http://www.wolframalpha.com, који даје директне одговоре на постављена питања (а не даје странице и документа које можда садрже одговор, као неки интернет претраживачи). Одговарајући на постављена питања или радећи математичке или друге задатке, израчунавајући резултат Волфрам алфа користи разне алгоритме (преко 50 000 алгоритама) да нађе резултат из своје базе падатака, што га разликује од претраживача који дају само адресе страница које су индексиране.

Најнапреднији робот данашњице, Хондин робот ASIMO, представља један од највећих успеха роботике и вештачке интелигенције. Може да детектује објекте и предмете који се крећу, може да их прати, распознаје ствари у својој околини, реагује на обраћање човека (окреће се ка њему), препознаје лице.

12. ЗАКЉУЧАК

Алгоритам који смо описали испитује да ли су два дата терма уједначива, у случају да јесу, добијамо и списак замена. Откако је Џон Алан Робинсон написао први алгоритам унификације, до данас су написани и разни други алгоритми унификације. У раду је описан општи алгоритам унификације, тј. алгоритам првог несклада, који у случају уједначивости два терма, исписује најопштији уједначивач - минималан скуп замена. Алгоритам унификације је основна полуга програмског језика PROLOG, језика вештачке интелигенције.

Имплементацијом овог алгоритма у програмском језику C++, за разлику од имплементације у BASIC-у, добијамо много читљивији код, јер смо избацили GO TO наредбе[1]. Амерички научник и програмер Доналд Кнут, који се још назива и „оцем алгоритама“, упутио је критике на коришћење ове наредбе, и представио њене алтернативе у свом делу „Structured Programming with go to Statementc“, 1968.

Написан код је користан за унапређење дескриптивних програма за логичко програмирање, унапређење програма вештачке интелигенције, а има и велику улогу за аутоматско доказивање теорема, интерактивно доказивање теорема и аутоматску проверу доказа, молекуларној биологији, генетици, лингвистици, музици.

С обзиром да је вештачка интелигенција област рачунарства која се брзо развија и од које се много очекује, алгоритам унификације заузима своју позицију као стуб темељац ове области.

[1] Неки програмери, програме написане са овом наредбом жаргонски називају „шпагети програми“.

13. ЛИТЕРАТУРА

[1] Марица Прешић, Славиша Прешић: Увод у математичку логику теорија и задаци, Математички институт, Београд, 1979.

[2] Славиша Б. Прешић: ПРОЛОГ релацијски језик, Математички факултет, Београд, ИСБН 44071948, 1996.

[3] Душан Тошић, Радивој Протић: ПРОЛОГ кроз примере, Техничка књига, Београд, ИСБН 8632503065, 1991.

[4] Зоран Огњановић, Ненад Крцавац: Увод у теоријско рачунарство, Београд – Крагујевац, 2004.

[5] Славиша Прешић: Алгоритам уједначавања, РАЧУНАРСТВО, свеска 1, број 1, 1990, 35-46.

[6] Миодраг Живковић: Алгоритми, Математички факултет, Београд, 2000.

[7] Krystof Hoder, Andrei Voronkov: Comparing Unification Algorithms in First-Order Theorem Proving, University of Manchester, Manchester, UK, 2009.

[8] Peter Norvig: Correcting A Widespread Error in Unification Algorithms, Computer Science Division, University of California, Berkeley, USA, 1991.

[9] Franz Baader, Wayne Snyder: Unification theory, Handbook of automated reasoning, Alan Robinson, Andrei Voronkov, Elsevier Science Publishers, 1999, 3-85.

[10] Mark Stickel: A complete unification algorithm for associative – commutative functions, in "Proceedings of the 4th International Joint Conference on Artifical Intelligence", Tblisi, USSR, 1975, 71-82.

[11] Дарко Вељан: Комбинаторика са теоријом графова, школска књига, Загреб,1989.

[12] J .A . Robinson: A machine-oriented logic based on the resolution principle, J . Assoc. Comput. Mach, 1965, 23-41.

[13] Славиша Прешић: Елементи математичке логике, Колортон, Београд, 2007.

[14] P.Szabo: Theory of First Order Unification, Universitat Karlsruhe, 1982.

[15] M. Fay: First order unification in an equational theory , Proceedings 4th Workshop on Automated Deduction, Texas, 1979.

[16] J. Lassez, M. Maher and K. Marriott: Unification Revisited, in: J. Minker, Foundations of deductive Databases and Logic Programming, Morgan Kaufman, 1988, 587-625.

[17] J. Cheney: Relating nominal and higher-order pattern unification, Proceedings of UNIF 2005, 104–119.

[18] M. Paterson, M. Wegman: Linear unification, J. Of Computer and System sciences, 1978, 158-167.

[19] С. Вујошевић, Ж.К. Вукићевић: Увод у логику, Универзитет Црне Горе, 2009.