Matematički dokaz pomoću računala

Mihael, Lukaš

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Mihael Lukaš

MATEMATIČKI DOKAZ POMOĆU RAČUNALA

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Lukaš

Matični broj: 42790/14–R

Studij: Poslovni sustavi

MATEMATIČKI DOKAZ POMOĆU RAČUNALA

ZAVRŠNI RAD

Mentor :

Izv. prof. dr. sc. Zlatko Erjavec

Varaždin, rujan 2018.

Mihael Lukaš

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Razvoj računala ima veliku važnost u dokazivanju matematičkih tvrdnji. Kombinacijom numeričkih, formalnih, heurističkih i simboličkih metoda na računalu se stvaraju novi matematički dokazi koji prije nisu bili mogući. S umjetnom inteligencijom računala, uz malu pomoć čovjeka, automatizirano dokazuju matematičke tvrdnje. Zajedno s ovim metodama, koristi se verifikacija pomoću računala koja pomaže pri provjeravanju istinitosti nekog dokaza. Navedene metode i verifikacija mogu biti veoma kompleksne, pa čak i do tolike mjere da naprave zastoj kod superračunala. Zato je potrebno znati iskoristiti matematičke metode da bi olakšali rad računala, jer svako računalo ima svoje granice. Matematički dokazi kao što su Flyspeck projekt i rješenje Booleovog problema bojanja Pitagorinih trojki bez superračunala ne bi bili ostvarivi. Oni zato predstavljaju najveće matematičke dokaze stvorene pomoću računala.

Ključne riječi: računala; numeričke metode; formalne metode; heurističke metode; simboličke metode; umjetna inteligencija; verifikacija pomoću računala

Sadržaj

1. Uvod

Početkom razvoja računala promijenio se način dokazivanja matematičkih tvrdnji. U prošlosti su se logičkim zaključivanjem dokazivali kraći matematički dokazi, ali nastankom računala, pogotovo kakvih danas poznajemo, stvorili su se novi načini dokazivanja matematičkih tvrdnji.

Svaki proces, algoritam i naredba unutar računala je bazirana na logici. Logika predstavlja vezu između korisnika i računala, jer inače ne bi postojala mogućnost njihove međusobne interakcije. Prilikom izrade matematičkih dokaza pomoću računala, koriste se matematičke metode koje se implementiraju u računalo. Metode kao što su numeričke, formalne, heurističke i simboličke, zajedničkom kombinacijom stvaraju i ubrzavaju dokazivanje tvrdnji koje bi inače veoma dugo trajale. Najbrže se razvija umjetna inteligencija koja se nalazi unutar heurističke metode. To nije umjetna inteligencija gdje bi računalo zamijenilo čovjeka, nego kako bi računalo zajedno s čovjekom donosilo zaključke. Prolog je prvi programski jezik koji je imao takve sposobnosti. Svako računalo ima softverska ili hardverska ograničenja, ali napretkom računalne tehnologije bliži se vrijeme kada će računalo samostalno u potpunosti donositi matematičke dokaze. Postoje slični programi kao što su automatizirano i interaktivno dokazivanje teorema, ali za rad još uvijek zahtijevaju čovjekovo vodstvo.

Matematičke metode se ne moraju samo koristiti u dokazivanju matematičkih tvrdnji, nego su korisne tijekom donošenja svakodnevnih odluka. Moguće je da pogrešno donesemo odluku, ali to je normalno i često se događa. Važno je znati riješiti i prilagoditi se pogreškama u kratkom vremenskom razdoblju. Zato postoji verifikacija matematičkih dokaza koja se koristi sve više i u budućnosti će se vjerojatno koristiti na svakom dokazu. Verifikacija pomoću računala možda traje dugo, ali omogućava pronalaženje najsitnije pogreške unutar dokaza. Da bi se verifikacija pomoću računala obavila u kratkom vremenu, koriste se već poznate sheme kao što je provjeravanje modela koji ubrzavaju i olakšavaju rad računala. Potrebno je poznavati ograničenja računala, jer ponekad niti superračunalo nije dovoljno brzo da napravi verifikaciju. Da se prilikom rješavanja Booleovog problema bojanja Pitagorinih trojki nije koristila izmijenjena heuristička metoda, superračunalu bi trebalo mnogo vremena da dokaže sve kombinacije matematičkih tvrdnji. Zbog navedenih stvari je važno pravilno iskoristiti odgovarajuću matematičku metodu kao i obradu verifikacije pomoću računala.

Moje zanimanje za računala i matematiku seže iz osnovne škole tijekom programiranja u QBasicu. Većina zadataka je bila matematičke prirode, gdje se zahtijevalo logičko razmišljanje i računanje jednostavnih linearnih jednadžbi. Nakon toga, u matematičkoj gimnaziji je učenje programskog jezika C rezultiralo većim zanimanjem u stvaranju aplikacija, a pogotovo na fakultetu učenjem C#-a. Za bilo koji programski jezik je potrebno poznavati matematiku pa otud dolazi moja motivacija za pisanjem ovog završnog rada.

2. Matematička logika i dokazi

Logikom se koristimo svakodnevno; služi nam za shvaćanje pojedinih tvrdnji i određivanja jesu li one istinite ili lažne. Prvim logičarom može se nazvati Aristotel. Iako nije znao pravila logike, znao je dobro zaključivati, a još bolje koristiti svoje riječi. Tako je donosio stavove koji se mogu prepoznati kao počeci logike, makar su je tek prvi put spomenuli njegovi sljedbenici zvani "stoici". Zaslužan za početke računalnih teorija je Ramon Llull, koji je djelovao u 13. stoljeću. Njegovo stvaralaštvo je poslije imalo utjecaja na Gottfrieda Wilhelma von Leibniza [1].

2.1. Matematički sudovi i operacije sa sudovima

Prema B. Divjak: *Rečenice u kojima se nešto tvrdi i mogu se podijeliti s obzirom na istinitost na istinite ili lažne nazivaju se sudovi.*

Ponekad sud može biti trivijalan i bez velikog razmišljanja moguće je odrediti je li istinit ili lažan. Kako bismo odredili je li istinit ili lažan, koristimo određena pravila i operacije nad sudovima s kojima dolazimo do konačnog rješenja. Kompleksan sud možemo podijeliti na manje dijelove koji tvore jednostavniji sud, a za prikaz svih podataka i interakciju između njih koristimo semantičku tablicu. Ona poprima sve moguće kombinacije koje neki kompleksan sud može poprimiti [2].

Tablica 1: Operacije nad sudovima

Naziv	**Prikaz**	**Čitanje**	**Opis kada je tvrdnja istinita**
negacija	$\neg$A	ne A	svaki lažan sud
konjunkcija	A $\wedge$ B	A i B	ako su oba suda istinita
disjunkcija	A $\vee$ B	A ili B	ako je istinit bilo koji sud
implikacija	A $\Rightarrow$ B	ako A onda B	bilo koji, osim ako je prvi sud istinit, a drugi lažan
ekvivalencija	A $\Leftrightarrow$ B	A je ako i samo ako je B	ako su sudovi jednake istinitosti

(Izvor: Divjak i Hunjak, 2004)

Iako se programski jezici razlikuju, logički operatori se pretežno identično prikazuju. U sljedećoj tablici se može vidjeti primjer logičkih operatora u programskom jeziku C#:

Tablica 2: Logički operatori u C#

Prikaz	**Čitanje**	**C#**
¬A	**NOT** A	!A
A ∧ B	A **AND** B	A && B
A ∨ B	A **OR** B	A || B
A ⇒ B	**IF** A **THEN** B	!A || B
A ⇔ B	**IF** A **AND ONLY IF** B	A == B

(Izvor: Microsoft, 2015)

Kao što se vidi u tablici, implikacija u programskom jeziku C# nema svoj poseban simbol. [3] To je zbog toga što se implikacija može prikazati pomoću osnovnih operacija: negacije, konjunkcije i disjunkcije, kao što je prikazano 1854. godine u knjizi Georgea Boolea "An Investigation of the Laws of Thought" [4].

2.2. Matematički dokazi

Da bi neka tvrdnja u matematici postala teorem, potrebno ju je dokazati. Dokaz predstavlja neosporan argument za tvrdnju, a jedini način da se dokaže jest utvrditi istinitost. U tome pomažu sljedeće vrste dokaza s kojima dokazujemo takve tvrdnje, odnosno hipoteze [5]:

1. **Direktni dokaz** se stvara slaganjem konačnog niza implikacija. On se izvodi tako da se korak po korak, korištenjem teorema, dokaže neka hipoteza. A da bi se dokazala, moguće je podijeliti problem na manje dijelove s kojima je lakše manipulirati i dokazati neku hipotezu [2].

$$A \Rightarrow B$$
$$A \Rightarrow A_1 \Rightarrow A_2 \Rightarrow ... \Rightarrow B$$

   Niz ekvivalentnih tvrdnji se pojavljuje često, gdje ekvivalentan predstavlja sve tvrdnje koje bi trebale vrijediti za taj dokaz. Ako postoje tvrdnje A, B i C i one su ekvivalentne, tada je potrebno napraviti dokaz na sljedeći način [2]:

$$A \Rightarrow B \Rightarrow C \Rightarrow A \text{ ili}$$
$$A \Rightarrow C \Rightarrow B \Rightarrow A$$

2. **Indirektni dokaz** ili dokaz po kontrapoziciji koji se koristi kada je teško moguće dokazati hipotezu direktnim dokazom. Tada se koristi obrat suprotnog teorema:

$$A \Rightarrow B, \text{ tada vrijedi } \overline{B} \Rightarrow \overline{A}$$

Slično je i dokazivanje protuprimjerom. Ako postoji neki sud A koji je istinit, tada s protuprimjerom tražimo neki primjer gdje je taj sud neistinit. Ako pronađemo takvu vrijednost, tada se dokazalo da sud A nije istinit. Dovoljno je da nađemo samo jedan primjer i odmah se može zaključiti da hipoteza ne vrijedi [2].

3. **Matematička indukcija** je često korišten pristup u dokazivanju, čiji je način zaključivanja od posebnog prema općem. Matematička indukcija se sastoji od tri koraka, a to su baza, pretpostavka i korak indukcije. Vrlo često se indukcija pojašnjava na primjeru domino efekta. Ako želimo saznati hoće li se srušiti n-ti domino, prvo je potrebno gurnuti prvi domino tako da se pokrene lančani efekt. Taj dio u matematičkoj indukciji zovemo baza indukcije. Nadalje pretpostavljamo da će se $(n-1)$ domino srušiti i ako dokažemo da će se tada srušiti n-ti domino, dokazali smo da će se svaki domino srušiti [6].

2.2.1. Normalne forme

Normalne forme koriste se radi pojednostavljenja logičkih izraza. Ako imamo neki kompleksan sud, potrebno ga je svesti na jednostavne operacije (negacija, konjunkcija i disjunkcija), a taj postupak zovemo minimizacija. Postoje tri vrste normalnih formi, a to su [2]:

- konjunktivna normalna forma (KNF)
- disjunktivna normalna forma (DNF)
- negativna normalna forma (NNF)

Da bismo bili u mogućnosti odrediti ove forme, prvo su nam potrebna bazična rješenja. Definicije bazične konjunkcije i disjunkcije su sljedeće:

Bazična konjunkcija zadane funkcije F je svaka konjunkcija $k_i(x, \overline{x}, y, \overline{y}, z, \overline{z}...)$ sudova ili njihovih negacija koja ima svojstvo $F(...) = 1$ kada je $k_i(...) = 1$, gdje F predstavlja $F(x, y, z, ...)$ funkciju algebre sudova.

Bazična disjunkcija zadane funkcije F je svaka disjunkcija $d_i(x, \overline{x}, y, \overline{y}, z, \overline{z}...)$ sudova ili njihovih negacija koja ima svojstvo $F(.) = 0$ kada je $d_i(.) = 0$ gdje F predstavlja $F(x, y, z, ...)$ funkciju algebre sudova [2].

Nadalje se mogu definirati KNF i DNF:

- **Konjunktivna normalna forma** neke funkcije sudova jest konjunkcija svih njezinih bazičnih disjunkcija.
- **Disjunktivna normalna forma** neke funkcije sudova jest disjunkcija svih njezinih bazičnih konjunkcija [2].
- **Negativna normalna forma** jest kada u minimiziranoj formuli postoji negacija unutar bazične konjunkcije ili disjunkcije. NNF vrijedi samo ako je sud negiran [7].

Možda se nakon ovih definicija čini nejasno, ali je lakše shvatljivo na primjeru:

Neka je zadano $F(x, y, z) = ((x \wedge y) \vee z) \Rightarrow ((\overline{y} \vee x) \wedge z)$, tada će semantička tablica izgledati ovako:

Tablica 3: Bazične konjunkcije i disjunkcije

x	y	z	F	bazične konj. i disj.
1	1	1	1	$x \wedge y \wedge z$
1	1	0	0	$\overline{x} \vee \overline{y} \vee z$
1	0	1	1	$x \wedge \overline{y} \wedge z$
1	0	0	1	$x \wedge \overline{y} \wedge \overline{z}$
0	1	1	0	$x \vee \overline{y} \vee \overline{z}$
0	1	0	1	$\overline{x} \wedge y \wedge \overline{z}$
0	0	1	0	$x \vee y \vee \overline{z}$
0	0	0	1	$\overline{x} \wedge \overline{y} \wedge \overline{z}$

Konjunktivna normalna forma je:

$$KNF(F) = (\overline{x} \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee \overline{z})$$

dok je disjunktivna normalna forma:

$$DNF(F) = (x \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge z) \vee (x \wedge \overline{y} \wedge \overline{z}) \vee (\overline{x} \wedge y \wedge \overline{z}) \vee (\overline{x} \wedge \overline{y} \wedge \overline{z})$$

Prikazane KNF i DNF su također i negativne normalne forme jer sadrže negacije unutar sebe. Sljedeći primjer nije NNF jer je negirana zagrada, a u NNF-u samo sudovi smiju biti negirani [7]:

$$\neg(\overline{x} \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee \overline{z})$$

3. Povijesni koncept logičkog stroja i računala

Počeci logike sežu u Aristotelovo razdoblje, ali prvi logički stroj spominje se u djelu "Ars generalis ultima" Ramona Llulla. On je bio katalonski pisac, filozof i logičar koji je svojim stvaralaštvom stvorio temelje računalne znanosti. Živio je u multikulturalnoj sredini, okružen muslimanima i židovima, a pošto je bio jako posvećen kršćanstvu, htio je napraviti uređaj koji bi preobratio sve druge vjere u kršćansku. I zbog toga je izgradio prvi logički stroj koji se sastojao od četiri figure, koje su prikazivale atribute boga i razliku između duša samo da bi svojim logičkim razmišljanjem i mogućnostima pridobio ljude. Međutim, nije važno što je njegov logički stroj pokazivao ili objašnjavao, nego kako je funkcionirao. Figure, koje su se nalazile na stroju, mogle su se micati tako da se dobiju različite kombinacije koje bi dovele do drugačijeg logičkog zaključka. To je bio prvi "uređaj" koji se može povezati s današnjim računalima. Zato što baš kao i računalo, koristi logičke izraze, a kao izlazne stavke donosi zaključke. Zbog toga mnogi ljudi iz računalnih znanosti vole nazivati Ramona Llulla pionirom računalne znanosti [8].

Njegovo spomenuto djelo "Ars generalis ultima" prethodilo je novom radu zvanom "Ars combinatoria" u kojem je prikazivao relacijsku strukturu između grafova i matrica. Tim djelima su bili nadahnuti mnogi znanstvenici, a jedan od najvažniji Gottfried Wilhelm Leibniz. Već u svojim dvadesetim godinama, Leibniz je temeljito analizirao djelo "Ars combinatoria" te napisao prvu knjigu vezano za Llullove radove. Kao i Llull, htio je stvoriti stroj koji bi mogao obavljati matematičke operacije, a već 1674. godine ga je opisao. Nekoliko godina kasnije, dobio je ideju o prvom univerzalnom jeziku kojeg bi svi ljudi mogli koristiti (i jednostavno naučiti). Time je htio postići da svatko iz bilo kojeg dijela svijeta može raspravljati o određenom problemu. To je prethodilo novom problemu zvanom "Entscheidungsproblem" ili "problemom odluke" [9].

Problem odluke je problem koji je postavio David Hilbert 1928. godine, a glasio je:

"Postoji li algoritam koji za svaki sud može odrediti je li istinit ili lažan."

Prvo je poznati logičar Kurt Gödel 1931. godine objavio svoj "Teorem nepotpunosti" u kojemu navodi da će u svakoj istinitoj tvrdnji, koja je dovoljno sadržajna da omogući izražavanje i dokaze osnovnih aritmetičkih operacija, biti moguće izraditi tvrdnju tako da se ni ona sama niti njezina negacija ne mogu dokazati iz danih aksioma i zato sustav mora biti nepotpun. Što bi ukratko značilo da se svakoj tvrdnji može pronaći neki sud koji će osporiti navedenu tvrdnju. Tom je definicijom započelo rješavanje "problema odluke", iako to nije bilo dovoljno za obaranje Hilbertovog pretpostavke [10].

Do 1936. nije postojao potpun odgovor na problem, ali tada su dva znanstvenika odvojeno dokazala da tvrdnja ne vrijedi. U radu Alana Turinga "On computable numbers, with an application to the Entscheidungsproblem" je navedeno da rezultati njegovih istraživanja i izračuna pokazuju da "problem odluke" nema rješenja. Isto to je dokazao i Alonzo Church, ali kako Turing piše: "na sasvim drugačiji način" [11]. Time je napravljena Church-Turingova teza koja navodi da sve što se fizički može izračunati, može se izračunati i uz pomoć Turingovog stroja. To znači da ne postoji jače ili bolje računalo od Turingovog stroja. Ne smije se zanemariti pretpostavka da Turingov stroj ne služi za brzo i efikasno rješavanje nekog problema, nego

za dokazivanje da je neki algoritam moguće napraviti računalom. Ako se algoritam ne može provesti Turingovim strojem, tada ga niti jedno računalo neće moći provesti [12]. No, sada se postavlja pitanje, kakve koristi će od toga biti u praksi? To je važno u računalnoj znanosti, ako, na primjer, želimo alat koji će nam biti u mogućnosti ispraviti sve pogreške tijekom programiranja. Nažalost, to nije moguće, ali zato to možemo iskoristiti tako da smanjimo pogreške i time suzimo područje u kojem bi pogreške mogle nastati [13].

Iz svega što je Turing napravio možemo zaključiti da je imao značajan utjecaj na razvoj računalne znanosti. No, osim Turingovog stroja, osmislio je i test koji nosi naziv Turingov test. Pomoću testa se raspoznaje ima li računalo preduvjete za umjetnu inteligenciju ili ne. Turing je time stvorio temelje i za stvari koje u njegovo vrijeme nisu bile tehnički provedive [14].

Nakon tih otkrića započelo je doba računala. Računala su se jako brzo razvijala, a vjerojatno će se u budućnosti još brže razvijati. Kako su se računala razvijala, tako su nastajali i programski jezici koji služe za pisanje algoritama. Prvi programski jezici osmišljeni su tijekom 50-ih godina 20. stoljeća. Poznatiji programski jezik bio je FORTRAN američke tvrtke IBM. FORTRAN se mogao normalno koristiti na računalu, a ne samo na papiru kao što je bio slučaj s nekim drugim. Nakon toga, pojavili su se i drugi programski jezici od kojih su najbitniji i najviše korišteni C, C++ i Java. Tijekom sljedećih poglavlja, bit će objašnjeno koji se programski jezici koriste unutar pojedine metode za dokazivanje matematičkih tvrdnji pomoću računala [15].

4. Izrada matematičkih dokaza pomoću računala

Računala su stvorena da bi automatizirala rad čovjeka. Ovisno o kakvom se računalu radi, moguće je obraditi velik broj podataka u malim vremenskim intervalima. Računala u prošlosti nisu bila brza, ali kako se tehnologija razvija, sve je teže držati korak s njom. Zbog brzog razvoja, matematika je dobila napredniji način dokazivanja tvrdnji prilikom kojih se koriste mnoge metode. Za daljnje razumijevanje sljedećih primjera, važno je definirati pojam matematičkog dokaza pomoću računala.

Matematički dokaz pomoću računala jest dokaz teorema koji ne bi mogao biti dokazan samostalno od strane čovjeka [16].

Navedena definicija predstavlja domenu gdje čovjek nije u mogućnosti izračunati neku matematičku tvrdnju, nego koristi pomoć računala radi brzine i olakšavanja izračuna. Da bi računalo razumjelo dokaze, potrebno je stvoriti algoritam razumljiv računalu. Kompleksni dokazi se rastavljaju na manje dokaze koje je brže i jednostavnije izračunati na računalu. Ako vrijedi svaki poddokaz unutar dokaza, tada i on cijeli vrijedi. Tim postupkom se opisuje verifikacija dokaza koja uz pomoć računala automatizirano dokazuje matematičku tvrdnju. Princip dokazivanja matematičkih tvrdnji pomoću računala ima tri faze [17]:

- Rastavljanje problema na konačan broj manjih potproblema koji se sastoje od manjeg broja varijabli.
- Analiza svakog potproblema prethodno prikazanog pomoću jednadžbi i / ili nejednadžbi.
- Izvršavanje specijaliziranog algoritma koji pronalazi pogreške.

Zadnja faza predstavlja zahtjevan zadatak za matematičare, jer moraju stvoriti algoritam (program), koji će pokazati je li matematička tvrdnja dokaziva [17].

Tijekom navedenih faza koriste se matematičke metode koje su neizbježne u stvaranju dokaza. Uz pomoć njih uspijeva se dokazati neka tvrdnja, odrediti vrijednosti i važnosti matematičkih izraza kako bismo olakšali pronalaženje dokaza.

4.1. Numeričke metode

Numeričke metode su matematičke metode koje se koriste za približno rješavanje kompleksnih problema tako da se rješenje sastoji od osnovnih operacija kao što su množenje, dijeljenje, zbrajanje i oduzimanje [18]. Za rješavanje kompleksnih problema, potrebno je napraviti kvalitetne i brze algoritme, te njihovu analizu i implementaciju. Uz pomoć računala, moguće je napraviti dokaze čiji kompleksni izračuni nisu bili mogući u prošlosti [19].

4.1.1. Područja korištenja numeričkih metoda

Numeričke metode su specijalizirane za rješavanje manjih problemskih područja, kao što su sustavi linearnih i nelinearnih jednadžbi, što predstavlja prvo područje korištenja nume-

ričkih metoda. [19]. Na primjer, poznati matematičar Carl Friedrich Gauss u 18. stoljeću je osmislio algoritam eliminacije, odnosno metodu rješavanja sustava linearnih jednadžbi. Jednostavno je izračunati nekoliko linearnih jednadžbi tim postupkom, ali kada bi broj linearnih jednadžbi porastao, nezamislivo bi bilo izraditi rješenja. No, suvremenim računalima je moguće izračunati nekoliko stotina linearnih jednadžbi u kratkom vremena. Zbog toga računala imaju veliku važnost jer omogućuju matematičarima da manje misle o računanju, a da više misle o stvaranju novih ideja i algoritama kako bi uspješno dokazali neku hipotezu [20].

Sljedeće područje jest aproksimacijska teorija. U aproksimacijskoj teoriji se računaju približne vrijednosti zbog nemogućnosti određivanja točne vrijednosti. Funkcijama, koje se ne sastoje od osnovnih aritmetičkih operacija, potrebno je odrediti približne vrijednosti s detaljnih numeričkim analizama. Primjer toga je Taylorov red koji nam pomaže pri izračunu vrijednosti kao što su e^x ili $ln(x)$, odnosno funkcijama koje teže u beskonačnost. Iako je pomoću Taylorovog reda moguće izračunati približnu vrijednost do točnosti koju želimo, on predstavlja neefikasan način određivanja približne vrijednosti. Ako ne postoji nijedan drugi način određivanja približne vrijednosti, tada se koristi ovaj način [19].

Derivacije i integrali predstavljaju treće područje. To je područje najzastupljenije u računalnoj znanosti i inženjerstvu. Derivacije i integrali često dovode do slučajeva kada nije moguće odrediti točan rezultat. Kod derivacija je problem određivanja početnih i graničnih vrijednosti koje u mnogim slučajevima teže u beskonačnost. Zbog toga je potrebno odrediti približne vrijednosti da bi se riješio problem [19]. U trećem području nalaze se i parcijalne derivacije kod kojih je nepoznato rješenje funkcija više varijabli. Zajedno s integralima, ovo područje se često koristi u inženjerstvu, gdje je potrebno računanje korištenjem numeričkih analiza (kvantna mehanika, valne jednadžbe, fluidna mehanika i slično) [21].

Primjer korištenja numeričkih metoda u inženjerstvu je projektiranje i analiza kontrolnih sustava u zrakoplovu. Automatizirani sustavi, koji se koriste u zrakoplovu, ne bi smjeli ni u jednom trenutku napraviti pogrešku. Zato ovise o matematičkim algoritmima i računanjima koja moraju biti u potpunosti neosporna. F.K. Kappetijn i H.L. Jonkers, autori rada "Aircraft Control System Design and Analysis Program Package", napravili su paket programa koji nudi mogućnost modeliranja, optimizacije i vrednovanja kontrola u vremenski diskretnim sustavima pomoću numeričkih metoda [22].

4.1.2. Pogreške u numeričkom računanju

Tijekom predavanja predmeta "Programsko inženjerstvo", profesor Strahonja je govorio kako programski kod ovisi o tome kako je stvoren. Naveo je primjer kako se navigacijski sustav u automobilu marke Mercedes sastoji od 20 milijuna linija koda, a zrakoplovni sustav od otprilike 1,7 milijuna linija koda. Na prvu loptu, izgleda vrlo neobično i pomislili bi kako je navigacijski sustav u automobilu napredniji od cijelog zrakoplovnog sustava, no to nije istina, jer nije bitna veličina napisanog algoritma, nego njegova kvaliteta. Kvarom navigacije u automobilu, nećete biti u životnoj opasnosti ili imati ogromne troškove popravka, ali ako se to dogodi u avionu, štete su ogromne. Važnije je razmišljati o tome kako kvalitetno napraviti i uz pomoć numeričkih metoda raspoznati kojeg su reda veličina pogreške, nego samo otprilike raditi algoritme koji

vjerojatno neće biti u mogućnosti obavljati svoje funkcije [23].

Pogreške su svuda oko nas i događaju se često. To je svakodnevica i zbog toga je potrebno kvalitetno poznavati kakve pogreške mogu biti i gdje se obično pojavljuju. Prva pogreška koja se često javlja jest pogreška odsijecanja. Pogreška odsijecanja se javlja tijekom korištenja približne vrijednosti (a ne točne vrijednosti). Primjer toga jest korištenje ovih dvaju izraza:

$$\sqrt{1+x} \approx 1 + \frac{x}{2}$$

Sljedeća pogreška jest pogreška zaokruživanja koja proizlazi iz zaokruživanja vrijednosti. Umjesto korištenja prave vrijednosti, koristi se zaokružena vrijednost. Primjer toga je zaokruživanje konstante π na 3.14 što može dovesti do bitne pogreške [23].

Većina pogrešaka se pojavljuje kod izračuna koje je napravio čovjek. Pogotovo ako se izračun mora napraviti u kratkom vremenu, to dovodi do trenutka nepažnje i konačno do pogreške. Primjer takve pogreške je situacija kada je NASA-in orbiter trebao ići na Mars. Izgubio se u svemiru 1998. godine, a isprva nije bilo jasno zašto. Nakon nekog vremena pronašli su propust inženjera koji nije napravio jednostavnu pretvorbu iz engleske mjerne jedinice u metričku mjernu jedinicu. Zbog ove pogreške, troškovi NASA-e su iznosili 125 milijuna američkih dolara [24].

Zbog navedenih primjera je potrebno znati kolika je mogućnost pogreške. Uz pomoć numeričkih analiza se računaju relativne pogreške, što olakšava pronalaženje pogrešaka. Relativne pogreške se računaju tako da se stvarna pogreška podijeli sa stvarnom vrijednošću kao što je prikazano na sljedećem primjeru:

Tijekom istraživanja brzine svjetlosti shvaćeno je da postoji pogreška od 13 m/s.

$$stvarna\,pogreska = 13\,m/s$$

$$stvarna\,vrijednost = 299\,792\,458\,m/s$$

$$relativna\,pogreska = \frac{stvarna\,pogreska}{stvarna\,vrijednost} = \frac{13\,m/s}{299\,792\,458\,m/s} = 4.33 \cdot 10^{-8} m/s$$

$$relativna\,pogreska = 0.00000433\%$$

Iz primjera se može zaključiti da je postotak relativne pogreške malen i nije potrebno poduzimati nikakve korake da bi se poboljšao rezultat. Ako je pogreška velika, mogla bi utjecati na stvaranje novih pogrešaka u budućnosti. Zato je potrebno poboljšati algoritam i provesti nova istraživanja kako bi se dovelo do smanjenja pogreške [23].

4.1.3. Programski jezici za numeričke metode

Programski jezik FORTRAN tvrtke IBM je proceduralni jezik, a svojom jednostavnošću i brzinom, predstavlja jedan od kvalitetnijih jezika koji se koriste za rješavanje numeričkih problema. Ukoliko je korisnik sposoban za izradu algoritama, tada će moći napraviti algoritam na bilo kojem programskom jeziku. Ako je potrebno napisati neki program na strojnoj razini,

koristi se programski jezik C. C je jedan od najpoznatijih programskih jezika, a osmislio ga je D. Ritchie. Makar numeričke metode ovise o procesorskim snagama, važno je znati iskoristiti memorijski prostor računala koji se u programskom jeziku C može poprilično dobro iskoristiti. Time se postiže brzina i olakšava rad računala, pogotovo ako je zahtjevan dokaz ili problem [25].

Ako korisnik nema želju za pisanjem izričito formalnog programa, tada postoje jezici kao što su Python, Matlab i R. S ovim programskim jezicima otvara se mogućnost kreiranja brzog dokaza ili protuprimjera, a osim toga njihova interakcija s korisnikom je vrlo dobro obrađena [25]. U matematičkim dokazima važno je da napisani kod sadrži što je moguće manje linija, a da obavlja više zadataka. Potrebno je da programer stvara efikasan programski jezik koji će svojom brzinom imati značajan utjecaj na izvođenje dokaza. Napretkom računala, brzina neće biti jedini faktor, nego će imati utjecaja na način pisanja koda i kvalitetu interakcije alata s korisnikom. Vrijeme potrebno za učenje temeljnih koncepata programa ne smije biti dugotrajno jer će učenjem svih koncepata rezultirati zahtjevnim i mukotrpnim radom. Važno je temeljito istražiti svaki programski jezik prije samog korištenja.

4.2. Formalne metode

Formalne metode su metode u računalnoj znanosti koje se sastoje od skupa pravila za traženje, dokazivanje i verificiranje tvrdnji. Da bi sustav funkcionirao potrebno je provesti testiranja kako ne bi došlo do pogreške. Zbog toga se koristi verifikacija koja ima mogućnost pronalaska najmanjih pogrešaka. Kao što se dokazuje istinitost teorema, traženjem njegovih protuprimjera, tako je potrebno provjeravati sve moguće kvarove na softveru ili hardveru [26]. Na primjer, tvrtka Intel i AMD koriste interaktivno dokazivanje teorema za provjeravanje procesora računala, a osim toga, formalne metode se koriste u automobilskoj industriji za Toyotine hibridne automobile. Poznavanje ovakvih metoda je važno zbog poboljšanja kvalitete proizvoda. Korištenje formalnih metoda još uvijek zahtijeva da programer donosi strategije rješavanja problema, pri čemu formalne metode samo pružaju okvir u kojima se programi mogu stvarati.

Glavni dio koji će u ovom poglavlju biti obrađen jest automatizirano razmišljanje, povezano s umjetnom inteligencijom i dokazivanjem matematičkih tvrdnji. U formalnim metodama se većinom koristi "satisfiability solver" (u nastavku SAT) koji će biti opisan u primjeru Booleovog problema bojanja Pitagorinih trojki. Korištenje SAT-a je u inženjerstvu i računalnoj znanosti (pogotovo kod verifikacije problema) [27].

Prije objašnjenja primjera, za početak ćemo u jednostavnim koracima objasniti teorem četiri boje. Teorem su 1976. godine dokazali K. Appel i W. Haken. Teorem glasi [28]:

"Dovoljne su četiri boje da se oboji planarna karta".

To znači da je moguće obojati bilo koju kartu u četiri boje s pravilom da dvije boje ne budu jedna do druge. Rješenje problema je započelo 1852. godine kada je F. Guthrie htio dokazati navedenu tvrdnju, ali nije bio u mogućnosti obraditi sve protuprimjere. Obraditi

svaki protuprimjer je bio mukotrpan posao, jer nije postojao nikakav stroj koji bi mu pomogao. Tijekom godina su se javljali mnogi dokazi ovog teorema, ali uvijek bi se ispostavila njegova neistinitost. Tako je 1922. godine Franklin dokazao da je moguće napraviti dokaz na kartama s najviše 25 regija. Ali, 1976. godine, Appel i Haken su uspješno dokazali da je teorem istinit. No, matematičari ne prihvaćaju taj dokaz jer se sastoji od iscrpne analize mnogih diskretnih slučajeva pomoću računala koje je nemoguće provjeriti ručno. 2005. godine N. Robertson i još trojica matematičara isprva su pokušali dokazati Appel-Hakenov teorem, ali su ubrzo odustali zbog kompleksnosti. Zato su napravili svoj algoritam pomoću formalnih jezika koji se može verificirati na računalu. U svome radu navode da je svaki dio moguće provjeriti ručno, iako nikada nisu provjeravani zbog kompleksnosti. Time su dokazali da Appel-Hakenov teorem vrijedi [28].

4.2.1. Automatizirano dokazivanje teorema

Automatizirano dokazivanje teorema je računalni program koji samostalno dokazuje unesenu tvrdnju. Prvi takav program je nastao 50-ih godina 20. stoljeća, a zvao se Presburgov algoritam za JOHNNIAC računalo. Napravio ga je Martin Davis i tada je uz pomoć računala dokazao da je suma dva parna broja parni broj [29]. Zbog zahtjevnosti algoritama, postoje mnogi teoremi koje još uvijek nije moguće dokazati niti superračunalom.

Automatizirano dokazivanje teorema se također koristi u rješavanju mnogih problema, čak i izvan matematike. Primjer su bankarski sustavi i internet, gdje je potrebno kriptirati podatke tako da se stvaraju kompleksni protokoli kojim bi se odbili "cyber" napadi i sačuvala privatnost. Ali u matematici ima veliku korist što se tiče geometrijskih dokazivanja kao i umjetne inteligencije (više o inteligenciji u poglavlju Heurističke metode u rješavanju problema) [30].

Postoji mnogo programa za automatizirano dokazivanje teorema, ali još uvijek ne postoji "savršen" program. Matematičari bi htjeli da funkcionira na razini umjetne inteligencije gdje bi se samo tvrdnja ubacila u računalo, nakon čega bi računalo pokazalo je li dokaz istinit ili lažan [30]. Automatizirano dokazivanje teorema bi trebalo funkcionirati na način da se kao ulaz koristi teorem, koji je napisan u računalno shvatljivom jeziku. Nakon što program obradi cijelu tvrdnju i pokaže je li tvrdnja istinita ili lažna. Ako je istinita, potrebno je prikazati njegov dokaz, a ako je lažan, potrebno je prikazati njegov protuprimjer [31]. Primjer toga je "Robbinsova pretpostavka":

- asocijacija: $a \vee (b \vee c) = (a \vee b) \vee c$
- komutativnost: $a \vee b = b \vee a$
- Robbinsova pretpostavka $\neg(\neg(a \vee b) \vee \neg(a \vee \neg b)) = a$

Korištenjem automatiziranog dokazivanja teorema, 1996. godine je W. McCune dokazao Robbinsovu pretpostavku koristeći program EQP [27]. EQP (engl. Equational Theorem Prover) i OTTER (engl. Organized Techniques for Theorem-proving and Effective Reasoning) su programi automatiziranog dokazivanja teorema za logiku prvoga reda. Stvoreni su u programskom jeziku C tvrtke Argonne, a većinom se koriste na UNIX sustavu [32]. Iako je ovaj

primjer rađen u EQP-u, OTTER je šire korišten unutar automatiziranog dokazivanja teorema. Većina matematičkih dokazivanja je napravljena u OTTER-u, a čak postoji 705 primjera koji služe matematičarima da efektivno nauče koristiti program [33]. Osim navedenih programa, važno je spomenuti program Mizar koji sadrži ljudsko-orijentiran jezik. Cilj Mizara je spojiti što više matematičara zbog njegove jednostavnosti korištenja i sporazumijevanja [34].

4.2.2. Interaktivno dokazivanje teorema

Uvijek je bilo potrebno raditi s računalima i robotima, posebice u industrijskoj grani gdje je većinom čovjeka zamijenio stroj. Možda stroj zamjenjuje ljude, koji obavljaju ponavljajuće stvari, ali još uvijek se ljudi moraju brinuti za strojeve. Za razliku od automatiziranog dokazivanja teorema, gdje program sam pronalazi i stvara dokaze, interaktivno dokazivanje teorema pruža interakciju između stroja i čovjeka tako da zajedno sudjeluju u stvaranju formalnog dokaza. Ovo predstavlja jedini način formaliziranja netrivijalnih matematičkih teorema. Da bi bilo moguće napraviti interakciju između računala i korisnika, potrebno je napraviti jezik koji će obje strane razumjeti. Osnova stvaranja programa koji služe za interaktivno dokazivanje teorema je razvoj SAM-a (poluautomatizirana matematika) čija se serija sastoji od pet računalnih programa koji funkcioniraju na automatiziranom zaključivanju u logici. U nastavku ćemo opisati programe koji se koriste za interaktivno dokazivanje teorema i koji su često korišteni za mnoga dokazivanja i verificiranja matematičkih tvrdnji [35].

AUTOMATH je formalni jezik iz 1967. godine, koji služi za iskazivanje detaljnog matematičkog dokaza na računalu. Ovo je najstariji interaktivni dokazivač teorema i trenutno postoji moderna verzija ovog jezika F. Wiedijka koja je napisana u programskom jeziku C. AUTOMATH se sastoji od točno definirane abecede i pravila, a njegova prednost je da može sadržavati cijele matematičke dokaze. Svaki dio matematičkog dokaza se mora pisati do najsitnijeg detalja što uvelike pomaže u daljnjem dokazivanju. Zbog toga su dokazi opširniji i cijeli proces stvaranja dokaza traje veoma dugo. Ne sadrži automatizaciju dokaza, a zbog jednostavnosti izrade, predstavlja veoma brz jezik. Nakon što se svaki dokaz opiše u AUTOMATH jeziku, stavlja se u računalo koje provjerava ne samo gramatičku točnost, nego i matematičke vrijednosti [36].

HOL Light je jedan od korištenijih programa interaktivnog dokazivanja teorema. Isprva je postojala verzija zvana HOL (engl. Higher Order Logic, odnosno logika višeg reda) koja je nastala 80-ih godina 20. stoljeća, a funkcionira na Churchovom sustavu λ-računa [35]. HOL Light je napravljen temeljem svih dotadašnjih verzija, a od HOL-a se razlikuje po tome što koristi jednostavniju logiku koja pomaže računalu u lakšem procesiranju unesenih podataka [37].

Osim navedenih programa koji služe za interaktivno dokazivanje teorema, postoje programi kao što su Isabelle (nalazi se unutar HOL, također ga je moguće pronaći pod nazivom Isabelle/HOL), Coq, ACL2, PVS ili EHDM, a detalje o njima može se naći u [35].

4.2.3. Rješavanje Booleovog problema bojanja Pitagorinih trojki

Uz pomoć računala, ljudi su u mogućnosti dokazati nešto što prije nije bilo moguće. Izradom programa za obradu zahtjevnih algoritama, dobivaju se rješenja koja je nemoguće

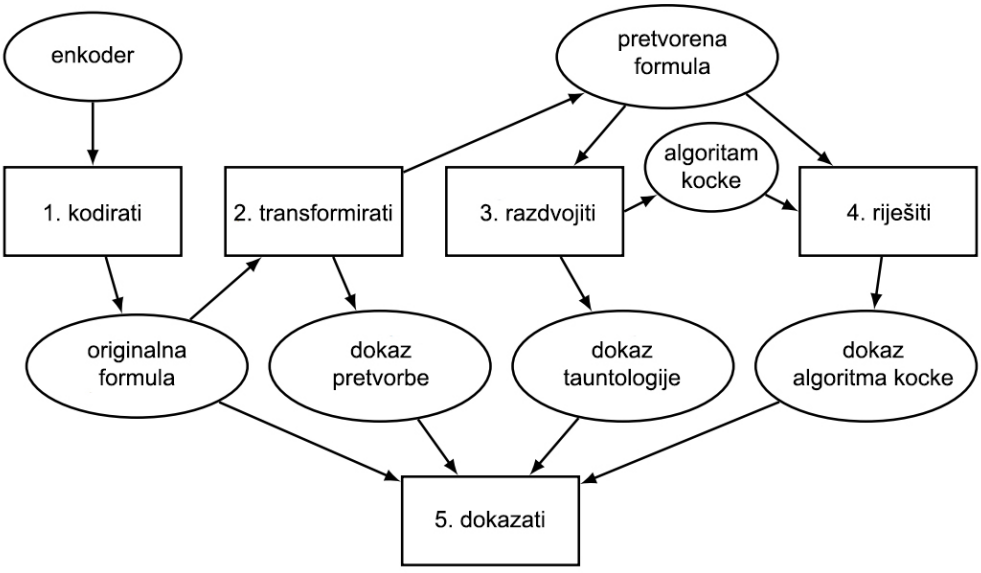
provjeriti ljudskom rukom. 2016. godine su znanstvenici, M. Heule, O. Kullmann i V. Marek izradili rješenje Booleovog problema bojanja Pitagorinih trojki. Priča o problemu je započela tijekom 80-ih godina 20. stoljeća, kada je Ronald Graham za rješenje problema ponudio sto američkih dolara. Problem glasi [38]:

"Mogu li se prirodni brojevi obojati plavnom i crvenom bojom na način da ne postoji jednobojna Pitagorina trojka brojeva tj. a, b, c *koji zadovoljavaju* $a^2 + b^2 = c^2$ *nisu obojeni istom bojom."*

Uz pomoć SAT-a je navedeni problem riješen. SAT je alat baziran na DPLL algoritmima, koji mogu rješavati desetak tisuća varijabli i oko milijun pravila (klauzula). Koristi se za planiranje, raspoređivanje problema i provjeru ograničenja modela [27]. Rješenje se može dobiti provjerom svake kombinacije (engl. brute-force), ali takav način je spor i neefikasan. Tijekom obrađivanja SAT-a, koji koristi normalne forme i Tseytine transformacije, se raspoznaje je li zadatak NP problem [39].

NP problem su problemi za koje nije poznato učinkovito rješenje. Rješenje NP problema se može pogoditi i verificirati prilikom korištenja računala. Najgori mogući slučaj predstavlja kombinaciju svih mogućih stanja, što u manjim zadacima nije problem. Ali u većim primjerima, kao što je primjer Booleovog problema bojanja Pitagorinih trojki, vrlo je važno koristiti odgovarajuće alate da bi dokazi bili mogući [40].

Booleov problem bojanja Pitagorinih trojki rješavan je pomoću SAT tehnologije, korištenjem metode "Cube-and-Conquer". Na sljedećoj slici je prikazan način rješavanja problema:



Slika 1: Model rješavanja problema - faze su prikazane u kvadratima, dok su ulazni i izlazni parametri prikazani u elipsama. (Izvor: Heule, Kullmann i Marek, 2016)

Prva faza je faza kodiranja. U toj fazi potrebno je "pripremiti" problem tako da ga SAT može prepoznati i koristiti. Nakon toga počinje faza pretvaranja kojoj je cilj složiti podatke tako da se efikasnije mogu koristiti u kasnijim fazama. U rješavanju problema su korištene dvije

transformacije pomoću kojih se uspješno smanji broj varijabli i pravila (klauzula). Da bi se očuvao integritet dokaza bilo bi potrebno dokazati istinitost svake transformacije. Treća faza, faza razdvajanja, je bila ključna u rješavanju problema. U njoj je problem razdvojen na manje dijelove kako bi se olakšalo njegovo rješavanje. Veliku važnost je imala heuristika, u kojoj je izračunavanjem uspješno smanjen broj kombinacija (više u poglavlju heurističke metode). Sljedeća faza je faza rješavanja. Ta faza sadrži algoritme "kocke" i transformirane formule koje stvaraju podatke potrebne za dokazivanje tvrdnje. U fazi dokazivanja je potrebno dokazati svaku tvrdnju koja se koristila u problemu. Tek nakon što je svaka tvrdnja dokazana, dokaz je završio.

Nakon što je superračunalo obradilo izračune, generiralo je podatke veličine 200 terabajta. Rezultat je pokazao da tvrdnje za brojeve od 1 do 7824 su istinite, ali da nakon broja 7824, nije moguće dokazati tvrdnju [38]. Time su Heule, Kullmann i Marek uz pomoć računala uspjeli dokazati jedan od najvećih dokaza u povijesti, a pritom su osvojili nagradu od sto američkih dolara.

Tijekom rješavanja problema, korišteno je superračunalo zvano "Stampede cluster", koje se sastoji od 6400 manjih računala koji su povezani u jednu cjelinu. Svako manje računalo sadrži Intelov 16-jezgreni procesor Xeon Phi, dok cijelo računalo ima ukupnu memoriju od 260 terabajta i prostor diska veličine 14 petabajta (14,000 TB). Bez ovakve jačine i razvijenosti računala, matematički dokaz ne bi bio moguć [41].

4.3. Heurističke metode

Heuristika služi za lakše donošenje odluka kod uspoređivanja različitih alternativa i njihovih atributa. Primjer jest kupnja automobila. Kada bi gledali svaki automobil koji se prodaje, donošenje odluka trajalo bi veoma dugo. S heuristikom možemo odrediti prioritete (npr. cijena, potrošnja, brzina), ograničenja (npr. 10 tisuća eura u cijeni) i dodavati mogućnosti koje imaju utjecaja na odabir. S tim mogućnostima se u kraćem vremenu može pronaći željeni automobil [42]. Heuristika nam, također, služi kao pomoć u dokazivanju matematičkih tvrdnji, gdje zajedno s njezinim metodama i umjetnom inteligencijom smanjuje broj mogućih slučajeva. Iako još uvijek moramo pričekati na računala koja će sama stvarati dokaze, uz trenutni razvoj to možemo uskoro očekivati. U sljedećim poglavljima pokazat ćemo primjere i metode u kojima se heuristika koristi.

4.3.1. Umjetna inteligencija

Umjetnu inteligenciju smo spomenuli tijekom objašnjavanja Turingovog testa. Umjetna inteligencija otvara mogućnost da računalo oponaša čovjekove osobine. Da bi računalo prošlo Turingov test, trebalo bi zadovoljavati sljedeće:

1. Uspješno komunicirati s čovjekom na engleskom jeziku.
2. Zapamtiti informacije od čovjeka, koristiti ih u daljnjoj komunikaciji i stvarati nove zaključke.

3. Prilagoditi se novim okolnostima i otkrivanju ekstrapolacije.

Još uvijek nijedno računalo nije uspjelo proći Turingov test, ali znanstvenici ukazuju na to da će se to dogoditi u sljedećih dvadeset godina [43]. Osim maštanja o računalima, koje će moći "razmišljati" kao ljudi, veliku važnost u stvaranju umjetne inteligencije imao je programski jezik Prolog.

Prolog je prvi programski jezik baziran na logičkim razmišljanjima. Nastao je 1972. godine, a izradio ga je francuski računalni znanstvenik A. Colmerauer. Jezik funkcionira tako da mu se zada set pravila (klauzula) prema kojima tada program donosi zaključak. Pomoću njega je bilo moguće ostvariti da stroj "razmišlja" pridržavajući se zadanih pravila [44]. Prolog se mnogo koristi i u dokazivanju matematičkih tvrdnji, jer s logičkim načinom razmišljanja, lakše se stvaraju pravila koja je inače komplicirano napraviti u nekom drugom jeziku. Osim toga, Prolog sadrži mogućnost pretraživanja pravila i mogućnost gledanja unatrag što olakšava traženje dokaza ili protuprimjera. Njegova abeceda logičkih varijabli je identična kao i u matematičkom dokazivanju, te se ne mora gubiti vrijeme na prevođenje izraza. Prolog sadrži i druge prednosti koje ga postavljaju ispred nekih drugih programskih jezika, a postoji i slična definicija kao i kod Turingovog stroja. Ako se teorem može implementirati u računalo, tada se može implementirati i u Prologu [45].

Problem velikog broja mogućnosti u umjetnoj inteligenciji predstavlja problem "kombinatorne eksplozije". One se javljaju kada je poznat matematički algoritam, ali ga je zbog velikog broja varijabli zahtjevno riješiti u konačnom vremenu [20]. Primjer toga jest šah. Računalo bi tijekom svakog odigranog poteza trebalo izračunati sve moguće kombinacije tako da odredi najbolji mogući potez. Pretpostavimo da postoji 20 odabira po svakom potezu i da računalo gleda 15 koraka unaprijed. Tada bi moralo ispitati $3.3 \cdot 10^{19}$ poteza, za što bi računalu trebalo preko 10 tisuća godina. Takvim pristupom nije moguće riješiti problem. Zato je potrebno dobro isplanirati i koristiti matematičke metode čijom se kombinacijom može stvoriti efikasan algoritam koji je u mogućnosti izračunati optimalno rješenje [46].

4.3.2. Metode unutar heuristike i umjetne inteligencije

Da bi se problem mogao riješiti heurističkim metodama, koriste se neke od sljedećih metoda:

1. Monte Carlo,
2. ekspertni sustavi i
3. Fuzzy logika.

Kod Monte Carlo metode se vrijednosti računaju pomoću slučajnog odabira. Na primjer, ako je potrebno izračunati prosječnu visinu ljudi u nekoj državi, uzimanje vrijednosti svakog čovjeka i stvaranje prosjeka bi bilo naporno i teško izvedivo. Zato se koristi Monte Carlo metoda koja nasumičnim odabirom uzima vrijednosti i iz njih donosi zaključke. Ova se metoda počela

koristiti 40-ih godina 20. stoljeća, a dobila je naziv Monte Carlo jer podsjeća na kockarnicu [47]. Zanimljiv primjer korištenja Monte Carlo metode jesu fraktali. Fraktali su oblici koji se mogu beskonačno mnogo puta ponavljati, tako da se prilikom povećanja ne gubi razlučivost. Mogli bi se ponavljati u beskonačnost, no Monte Carlo metoda određuje ograničenja crtanja. Prilikom približavanja vrhovima fraktala, odabire se sve manje slučajnih točaka formulirajući zadani oblik fraktala. Faktali su od velike važnosti u računalnoj grafici i botanici, a zbog Monte Carlo metode, lako ih je implementirati na računalima [48].

Sljedeća metoda su ekspertni sustavi koji se koriste u umjetnoj inteligenciji. U početku su se koristili za odgovaranje na jednostavna pitanja, a kasnije su poslužili za donošenje odluka. Koriste se u medicini, pri istraživanju srčanih bolesti kod novorođenčadi (1996. godina). Sustav je trebao određivati tretman koji bi se koristio, jer što je dijete mlađe, to postoji veći rizik. Ekspertni sustav se sastojao od tima stručnjaka od kojih je barem jedan pedijatrijski kardiolog, a drugi inženjer informatike. Oni su stvorili rješenje problema kod kojeg je sustav koristio ulazne parametre da bi mogao odrediti kvalitetan tretman i u konačnici spasiti nečiji život [43].

Fuzzy logika je metoda unutar heuristike. Koristi se kada tvrdnjama nije moguće odrediti istinitost, nego se dodaje vrijednost, koja opisuje postotak istinitosti. Primjer primjene je određivanja potrošnje goriva u automobilu. Potrebno je uzimati nekoliko atributa (na primjer: brzina, ubrzavanje i destinacija) u obzir da bi se izračunala optimalna potrošnja. Nad tim atributima se postavljaju vrijednosti u intervalu od 0 do 1 na temelju čega se određuje važnost atributa. Nakon toga, dolazi se do zaključka [49].

4.3.3. Primjene heurističke metode

U računalnoj znanosti, većinom se heurističke metode koriste prilikom poboljšavanja algoritama i smanjenja opterećenja računala. Vjerojatno je da rezultat neće biti najbolji, ali s obzirom na brzinu i sve mogućnosti, taj rezultat će biti među boljima. Na primjer, automobil sadrži algoritam računanja optimalnog puta unutar navigacije. Ako automobil putuje od točke A do točke B, potrebno je između tih točaka pronaći ulice kroz koje mora proći da bi stigao u prihvatljivom vremenskom roku. Navigacija neće računati svaki mogući put od točke A do točke B, nego će uz pomoć faktora (kao što su ceste, ograničenje brzine i promet) izračunati optimalan put. Kada bi navigacija računala svaki mogući put, predugo bi trajalo i vjerojatno izazvalo nezadovoljstvo korisnika. Zato je uz pomoć heurističke metode moguće u kratkom vremenu izračunati jedno od optimalnih rješenja [50].

Unutar rješavanja Booleovog problema bojanja Pitagorinih trojki, koristile su se heurističke metode, kod kojih je bilo potrebno smanjiti broj procesorskih sati. Tijekom korištenja početnih postavki heurističke metode, broj procesorskih sati je iznosio 300,000, što je predstavljalo problem čak i superračunalu. Nakon mnogih istraživanja, saznali su da mogu ručno postaviti novu heuristiku i smanjiti broj kombinacija, što je dovelo do smanjenja na 35,000 procesorskih sati. Zbog novog heurističkog pristupa, superračunalo je bilo u mogućnosti odraditi posao u samo dva dana [38]. Taj problem smo već spomenuli pod nazivom "kombinatorna eksplozija", situacija u kojoj bi jednostavno predugo vremena bilo potrebno da se odradi neki algoritam. Tada je na njemu potrebno napraviti neke preinake da bi se optimizirao [20].

4.4. Simboličke i analitičke metode

Postavlja se pitanje zašto koristiti simboličke, a ne numeričke metode. Kao što je već navedeno, numeričke metode se bave brojevima i njihovim računanjem, dok se simboličke metode koriste za interakciju između varijabli i izraza. Ove metode su u početku bile većinom korištene od strane profesionalnih matematičara i fizičara koji posjeduju moćna računala. No, današnjim razvojem programa i računala svima je omogućeno njihovo korištenje. Leibniz je u prošlosti imao slične ideje, ali zbog slabo razvijene tehnologije, stavio ih je samo na papir. Uz pomoć ovih metoda, sličnih umjetnoj inteligenciji, pojednostavljuje se stvaranje novih dokaza i dokazivanje već postojećih teorema [51]. Za rješavanje problema simboličkim metodama koristi se računalni algebarski sustav (u nastavku CAS), koji se ponekad još naziva simbolički sustav manipulacija. CAS se sastoji od programskih jezika koji imaju mogućnost korištenja simboličkih operacija, a neki od važnijih su:

- LISP

  dizajnirao ga je John McCarthy, a izradio Steve Russell. LISP je programski jezik visoke razine, osmišljen 1958. godine. Ubrzo nakon stvaranja počeo se koristiti za umjetnu inteligenciju, a u računalnu znanost je uveo mnoge pojmove kao što su: stabla podatkovnih struktura, automatsko upravljanje memorijom, objektno-orijentirano programiranje i samoprevođenje [52].

- Maple

  Maple je programski jezik, tvrtke Waterloo Maple Inc., koji pruža procedure za obavljanje simboličkih, numeričkih i grafičkih računanja. Sadržava interaktivno sučelje koje pruža brže stvaranje i lakše snalaženje, kao i mogućnost praćenja objekata stvorenih prilikom programiranja. Bitnu funkcionalnost predstavlja razvijeni sustav traženja i otklanjanja pogrešaka, jer je moguće prolaziti kroz napisani kod i paralelno provjeravati i mijenjati vrijednosti u varijablama [53].

- Mathematica

  Mathematica je programski jezik, tvrtke Wolfram, koji pruža implementirane algoritme za poboljšavanje rada, stvaranje novih funkcionalnosti, meta-algoritama i super funkcija. Osim stvaranja simboličkim metodama, također sadrži podršku za kompleksne brojeve, rješavanje linearnih i nelinearnih jednadžbi, derivacija, integrala i mnogo drugih mogućnosti [54].

- MuPAD

  MuPAD, tvrtke SciFace Software Gmbh & Co., za razliku od Maplea i Mathematice se bazira prvenstveno na rukovanju i radu operacija nad simboličkim izrazima. Sadrži mogućnosti rada s numeričkim metodama u kojima spadaju točni izračuni, aproksimacije i kompleksni brojevi [55].

Svi navedeni programski jezici koriste matematički pseudojezik (MPL, engl. Mathematical Programming Language) koji se sastoji od točno određenih simbola i izraza. Služi osobi za

lakše prelaženje s jednog jezika na drugi. Tijekom matematičkog dokazivanja tvrdnji, potrebno je odmah u početku odabrati dobar alat kojim bi se najbolje opisala i dokazala tvrdnja [56].

Jedan primjer korištenja simboličkih metoda u programu LISP jest računanje korijena iz negativnog broja. Korijen iz negativnog broja ne postoji, ali unosom takvog primjera, moguć je "ispravak", tako da se vrijednost unutar korijena stavi pod apsolutnu vrijednost:

```
ulaz: (sqrt -4)
naredbi sqrt je dan negativni broj. Vratiti (sqrt(abs -4))
izlaz: 2.0
```

Da bi se u simboličkom jeziku mogao koristiti ovakav primjer, potrebno je provjeriti ulazne podatke. Ako je ulaz manji od 0, tada je potrebno staviti broj u apsolutnu vrijednost. Kao što se vidi na navedenom primjeru, računalo zaključuje da se radi o negativnom broju i tada informira korisnika o negativnom broju i njegovom daljnjem korištenju. Nakon toga se računa korijen iz 4 koji iznosi 2. Također se problem javlja tijekom unošenja vrijednosti nula. To se rješava pomoću provjere gdje se ispituje je li vrijednost unutar korijena jednaka nuli. Ako je jednaka nuli, računalo bi vratilo poruku da je nemoguće računanje korijena iz nule [57].

Ako se tvrdnja ne može dokazati pomoću analitičkog pristupa, obično je potrebno korištenje numeričke metode. Kod analitičkog pristupa važno je da se problem dobro poznaje, jer ga inače nije moguće riješiti.

Analitičke metode predstavljaju skup logičkih koraka kojima se izračunava stvarna vrijednost. Uvijek je potrebno dokazati tvrdnje i algoritme na kojima radi računalo [58]. Zato analitičke metode olakšavaju mukotrpan posao koji se danas određuje pomoću računala. Ali uz pomoć analitičkih metoda, otkriveno je da neki problemi imaju analitičko rješenje, iako prije to nije bilo poznato [20].

5. Verifikacija matematički dokaza pomoću računala

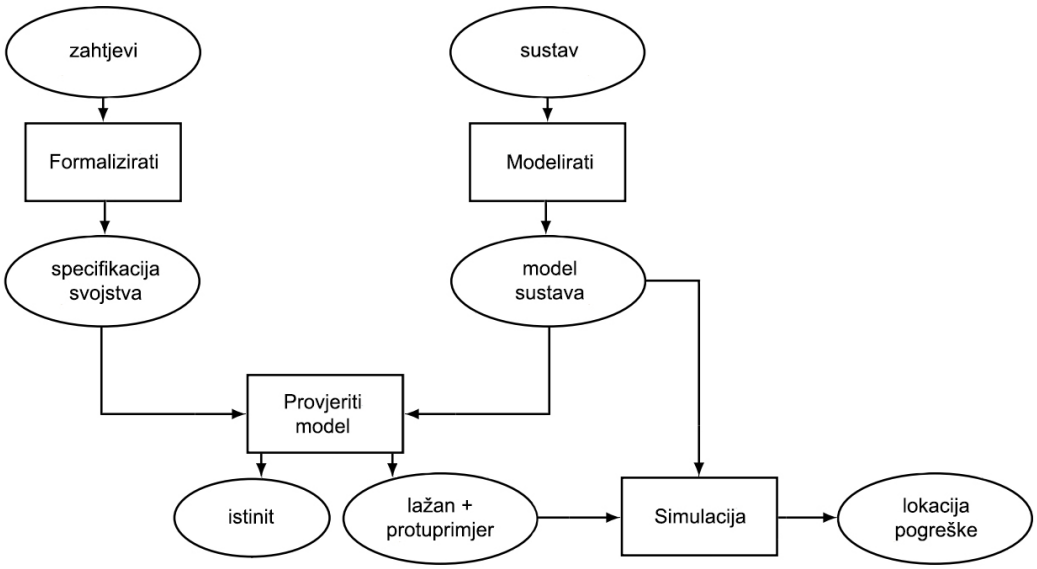
U ovome poglavlju ćemo prikazati zašto je verifikacija u dokazivanju matematičkih, ali i ostalih tvrdnji potrebna. Verifikacija se koristi za pronalaženje i ispravljanje najsitnijih pogrešaka i nedostataka softvera. Pomoću nje želimo utvrditi točnost logičkih izraza i dokaza teorema.

Tako je 1994. godine, tvrtka računalnih procesora Intel, lansirala u prodaju procesor Pentium koji je sadržavao logičku pogrešku. Provedbom pravilne verifikacije, pronašli bi pogrešku i ne bi ostali bez 500 milijuna američkih dolara. Zbog takvih i sličnih slučajeva, potrebno je odmah na početku kvalitetno dizajnirati i implementirati sustav, kako bi se izbjeglo što više pogrešaka prilikom verifikacije. Uz pomoć nje, moguće je identificirati probleme čije bi nerješavanje stvorilo velike troškove za tvrtku [59].

Verifikacija pomoću računala se većinom koristi u formalnim metodama, ali zbog njezine korisnosti, pokazala se kao bitna stavka u bilo kojoj matematičkoj metodi (pogotovo u umjetnoj inteligenciji). U nastavku su navedene verifikacije pomoću računala koje se koriste za provjeravanje algoritama.

5.1. Verifikacija algoritama

Nažalost, u prosjeku se većina vremena troši na prepravljanju algoritama, nego na stvaranje ispravnih i kvalitetnih. Provjeravanje modela je način verificiranja algoritama, a ispituje sve moguće scenarije sustava (engl. brute-force). Naime, ovakav pristup verificiranja problema će sa sigurnošću pronaći sve nedostatke u programu, ali će dugo trajati. Verificiranjem većeg teorem ili dokaz, predstavlja zahtjevan i dugotrajan posao za računalo. Zato se za provjeravanje nekog dokaza koristi sljedeća shema, koja efikasno pronalazi čak i najsitnije pogreške:



Slika 2: Shema provjeravanja modela (Izvor: Baier i Katoen, 2008)

U fazi modeliranja se modelira sustav, dok se u fazi formaliziranja, formaliziraju zahtjevi

tako da bi računalo bilo u mogućnosti koristiti modele za provjeru. Faza provjeravanja modela prethodno navedene modele verificira i analizira. Ako je model istinit, tada se nastavlja dalje i provjerava sljedeći. Inače dolazi do pogreške za koju faza provjeravanje modela generira protuprimjer. U fazi simulacije se koristi model sustava i protuprimjer za pronalaženje lokacije pogreške. Nakon toga se dizajnira novi ili popravlja trenutni model. Tada se ponovno pokreće proces na popravljenom modelu. Ako kojim slučajem, računalo nije u mogućnosti napraviti provjeru modela zbog nedostatka memorije, model se dijeli na manje dijelove te se ponavlja postupak [60].

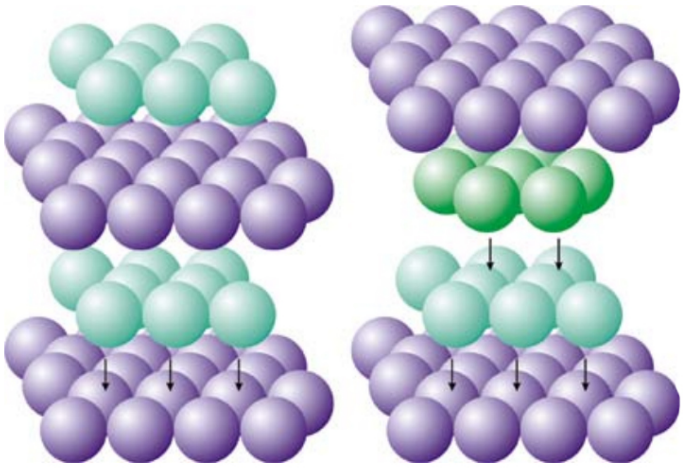
Ove korake u shemi je potrebno kvalitetno programirati zbog mogućnosti da model ne pronađe pogrešku. Važno je napraviti dobru simulaciju koja će sama eliminirati i rješavati pogreške u modelima. Ova faza je veoma važna, pogotovo ako je veći model, zbog uštede na vremenu i računalnom prostoru [60].

IBM je tijekom četiri godine napravio provjeravanje simboličkih modela koristeći formalne verifikacije hardvera. IBM-ov industrijski alat čiji je razvoj fokusiran na nekoliko područja se zove RuleBase. U njega su implementirali jezik zvan "Sugar", kojem kao baza služi logički jezik CTL. Također su implementirati "debugger" koji služi za analiziranje verifikacijskih rezultata. "Debugger" je alat koji pronalazi pogreške i nedostatke unutar programskog koda. Nakon kompajliranja, moguće je provjeravati liniju po liniju napravljenog programskog koda i tako pronalaziti pogreške koje su bitne za rad programa. Na kraju, RuleBase IBM pruža korisniku grafičko sučelje koje ima odličnu interakciju s verifikacijskim procesom gdje olakšava njegovo korištenje [61].

Verifikacija algoritama zahtijeva mnogo vremena i računalne snage. Primjer toga je navedeni teorem četiri boje i Flyspeck projekt koji će se navesti u nastavku. Za oba primjera je bilo potrebno verificirati svako moguće rješenje, pa čak i do tolike razine detaljnosti da se provjerava u kojem se skupu nalaze brojeva. Ovakva verifikacija prilikom računanja pomaže matematičarima da shvate gdje su napravili pogreške i olakšaju njihov ispravak [26].

5.2. Flyspeck projekt

Pretpostavka Flyspeck projekta datira iz 1611. godine kada je J. Kepler pretpostavio da pakiranje sfera (bilo heksagonsko ili kubično gusto pakiranje) ne može imati veću gustoću od $\frac{\pi}{3\sqrt{2}} \approx 74.048\%$ [62]. Prvi sloj heksagonskog gustog pakiranja dobiva se postavljanjem sfera jedne do druge, stvarajući pravilni šesterokut. Tada se drugi sloj sfera stavlja u udubine prvog sloja i nakon toga se treći sloj postavlja točno iznad prvog. Ponavljajući navedene korake, dobiva se heksagonsko gusto pakiranje. Ako želimo kubično gusto pakiranje, tada je potrebno treći sloj sfera postaviti u udubine drugog sloja na koji se opet stavlja prvi sloj sfera [63].



Slika 3: Heksagonsko (lijevo) i kubično (desno) gusto pakiranje (Izvor: Brueckler, 2012)

Najgušće moguće pakiranje sfera naziva se još Keplerova hipoteza. 1975. godine B. Fuller je tvrdio da ima dokaz za navedenu hipotezu, ali je u svome radu samo opisao kubno pakiranje, a ne dokaz njegove istinitosti [62]. Sve do 1998. godine, kada su T. C. Hales i S. Ferguson u radu od oko 300 stranica i 40 000 linija koda dokazali Keplerovu hipotezu. Trebalo je pet godina da se dokaz provjeri, a ocjenjivao ga je tim sudaca, koji zbog kompleksnosti, nije bio u mogućnosti završiti provjeru. Glavni sudac, mađarski matematičar G. F. Tóth, je tvrdio sa sigurnošću od 99% da je dokaz istinit. Ali čak i jedan posto, u matematici može utjecati na odbacivanje dokaza. Hales, nezadovoljan njihovim rezultatima, 2003. godine je započeo projekt pod nazivom "Flyspeck". Cilj projekta je bila izrada formalnog dokaza Keplerove hipoteze. Naziv Flyspeck proizlazi iz akronima FPK (engl. Formal Proof of the Kepler conjecture) gdje Hales navodi da riječ flyspeck znači "istražiti u detalje", što se čini sasvim prikladno za ovakav projekt [64].

Formalna verifikacija je napravljena uz pomoć već spomenutog HOL Light programa koji pruža interaktivno dokazivanje teorema. HOL Light je napravljen na sustavu zvanom LCF koji sadrži biblioteke važne za izradu formalnih analiza. Osim HOL Lighta, za verifikacijske izračune je korišten Isabelle/HOL. Isabelle/HOL podržava oblik računalne refleksije koji omogućava da se izlazni podaci mogu izvesti kao ML, koji se tada mogu integrirani kao pomoćni teoremi [64]. ML je meta jezik, odnosno programski jezik kojeg je stvorio programer R. Milner 70-ih godina prošlog stoljeća [65]. Unutar dokaza je bilo potrebno napraviti sljedeće izračune za koje su korištena računala (programi):

1. Rješenje oko tisuću nelinearnih jednadžbi (HOL Light).
2. Stvaranje grafova koji su provjeravali svaki protuprimjer (Isabelle/HOL). Rezultat grafova je nazvan "TameEnum", a tijekom provjere je trebalo sve provesti i prevesti u HOL Light-u što im je stvorilo većinu problema tijekom projekta.
3. Konačni program se sastojao od mnogo potprograma unutar kojih se koristilo linearno programiranje (HOL Light).

Nakon svih verifikacija, projekt je uspješno završen 2014. godine kada je dokazana Keplerova hipoteza. Taj projekt se može smatrati jednim od najvažnijih primjera u matematičkom dokazivanju pomoću računala [64].

6. Ograničenja u dokazivanju

Zanimljiv primjer ograničenja računala je "borba" između računala zvanog "Deep Blue" i šahovske legende G. Kasparova. Grupa matematičara je osmislila niz pravila i teorema za igranje šaha te ih implementirala u računalo. H. Simon je 1956. godine predvidio da će za deset godina računalo moći pobijediti najboljeg igrača šaha, ali to se ostvarilo tek nakon 41 godine. Prvi susret se dogodio 1996. godine, kada je Kasparov uspio pobijediti IBM-ovo računalo. Može se zaključiti da je moguće pogriješiti u stvaranju pravila. Ipak je čovjek napravio računalo, što znači da i računalo može pogriješiti. Teško je obuhvatiti cijelu domenu i sve moguće poteze koji bi se mogli dogoditi. Ali zato, nakon samo godinu dana, računalo je ipak pobijedilo Kasparova, što dokazuje da je moguće detaljnom analizom pogrešaka stvoriti kvalitetne programe [66].

Programer mora dobro poznavati problem koji opisuje u računalu. Ako ne postoji povezanost između računala i programera, problem se neće u potpunosti riješiti. Mnogo stvari ovise o računalu ali i o programeru. Ako programer loše programira, tada mu neće pomoći niti superračunalo. Isto vrijedi u obrnutom slučaju. Rješavanje matematičkih dokaza pomoću računala, može iscrpiti mnogo snage i zauzeti mnogo prostora na računalu što dovodi do neostvarenog cilja.

Naravno, postoji razlika između računalnog korisnika i programera. Programer stvara programe prilikom čega mora poznavati sve mogućnosti programskog jezika (operacije, tipove podataka, objekte...) kako bi stvorio jezik prilagođeniji korisniku (engl. user-friendly). Na primjer, programski jezik LISP napravljen je kako bi matematičar brzo i jednostavno opisao problem. Zbrajanjem dva broja na običnom kalkulatoru bi izgledalo ovako "2 + 2 =" i nakon toga bi se rješenje prikazalo na ekranu kalkulatora. Kad bi se koristila poljska notacija, zbrajanje dva broja bi izgledalo ovako "2 2 +". LISP za razliku od navedenih notacija, svoje računanje dva broja temelji na interakciji između programa i korisnika bez potrebe da se cijeli napisani program mora kompajlirati [67]. Ovakav način programiranja pomaže matematičarima da brzo i jednostavno opišu dokaz. Važna je interakcija između programera i korisnika jer programer mora stvoriti program na kojim će korisnik moći stvoriti kvalitetan algoritam.

6.1. Ograničenja u računalnom softveru i hardveru

Računalni softver i hardver su tijekom povijesti zbog slabe razvijenosti tehnologije imali ograničenu uporabu. Danas postoje superračunala koja bi mogla riješiti zahtjevan problem u samo nekoliko minuta. Iako su se u prošlosti koristila slaba računala, njihova uporaba je ipak predstavljala brži način računanja nego računanje rukom. Superračunalo, koje je rješavalo Booleov problem bojanja Pitagorinih trojki, generiralo je 200 terabajta dokaza kompresiranih u 68 gigabajtnu verziju. Rješenje je moguće skinuti s interneta, ali samo za rekonstrukciju kompresiranih podataka običnom računalu bi trebalo skoro 30,000 procesorskih sati (skoro 4 godine) [38].

Kod primjene numeričkih metoda je veoma bitan procesor (CPU), jer se numeričke metode temelje na mnogim računanjima koji zahtijevaju brzinu. Za simboličke metode je važno

dobro rasporediti prostor u memoriji. Ne preporučuje se korištenje zajedničkih računala, odnosno računalima koje koristi više osoba odjednom (primjer: server). Rad računala se olakšava tako da se koristi samostalno računalo ili radna stanica s velikim kapacitetom memorije. Osim jakih procesora i memorija velikih kapaciteta, danas su bitne i grafičke kartice računala jer se sve više koriste grafički prikazi koji pomaže korisniku u stvaranju i objašnjenju matematičkih dokaza [68].

Iako se do danas hardver razvio u velikoj mjeri, također se razvija i softver. Završilo je doba gdje je bilo potrebno detaljno poznavanje svakog programa i programskog jezika. Primjer takvog programa je FORTRAN, koji jednostavno funkcionira. Jednostavno znači za programera, a ne za matematičara ili studenta koji ga želi koristiti. Zbog toga se danas sve više stvaraju programi kao Mathematica, koji sadrži grafičke prikaze za olakšavanje učenja i korištenja programa, pa tako i kvalitetnije dokazivanje matematičkih tvrdnji [68].

Na programerima je da naprave programski jezik koji će biti brz i interaktivan s korisnikom, a na korisniku je da efikasno koristi programski jezik i programe. Ako korisnik ne zna kvalitetno stvarati programski kod, tada će usporiti rad računala i postojat će veća šansa da se ne pronađe pogreška. Pogotovo ako se radi o iterativnim sustavima koji se mnogo puta ponavljaju. Na primjer, matematičar se bavi dokazivanjem matematičke tvrdnje kod koje je potrebno provjeriti X podataka. Ako napravi kod koji bi svake iteracije bio pet sekundi sporiji, tada će se procesiranje dokaza na računalu povećati pet puta. Važno je poznavati sve mogućnosti softvera i hardvera, jer se inače stvaraju nepotrebni troškovi.

Osim primjera Booleovog problema bojanja Pitagorinih trojki, ograničenja hardvera je postojalo i u Flyspeck projektu. Tijekom stvaranja algoritma, Hales i njegov student A. Solovyev su izradili formalni dokaz svakog dijela na računalu. Provjeravanje svih dijelova dokaza je trajalo 5,000 procesorskih sati na Microsoft Azure cloudu, što otprilike iznosi sedam mjeseci računalnog procesiranja podataka [26].

6.2. Pogreške u dokazivanju

Tijekom dokazivanja matematičkih tvrdnji javljaju se pogreške. Tako je 1986. godine u dokumentu "Algebraic Cycles and Higher K-Theory" američkog matematičara S. Blocha, pronađena pogreška odmah u početku. Pogreška se nalazila u pomoćnoj tvrdnji, a pronašao ju je matematičar A. Suslin koji je tvrdio da su gotovo sve tvrdnje u navedenom dokumentima bile neutemeljene. Bilo je potrebno nekoliko godina da se ispravi pogreška, a ispravak se sastojao od 30 novih kompleksnih stranica. Nakon toga prošlo je mnogo godina prije nego je dokaz prihvaćen [69].

Prilikom dokazivanja matematičkih tvrdnji je potrebno temeljito planirati problem i pokušati obuhvatiti cijelu domenu. Većina se dokaza provodi verifikacijom na računalu, a da bi se smanjili problemi koji se javljaju u verifikaciji, potrebno je kvalitetno obraditi problem. Većinu problema je moguće riješiti na početku, jer nakon što se stvari zakompliciraju, veoma je teško pronaći pogrešku [69].

7. Zaključak

Matematičko dokazivanje je s napretkom računala dobilo novi način upotrebe. Stvaraju se programi i algoritmi koji sami pronalaze pogreške i dokaze matematičkih tvrdnji. Još uvijek niti jedno računalo nije prošlo Turingov test, ali to bi se moglo dogoditi u bližoj budućnosti. Verifikacija pomoću računala se koristi u poznatim tvrtkama poput Intela, AMD-a i NASA-e da bi pronašli nedostatke u svojim proizvodima. Istu korist imaju i u matematici gdje se uz pomoć njih dokazuje istinitost matematičkih tvrdnji. Ponekad je teško prikazivati matematičke tvrdnje u programskom jeziku, ali da bi ih matematičar mogao dokazati, potrebno je vrlo dobro poznavati problem.

Za izradu dokaza pomoću matematičkih metoda koriste se superračunala, ali svaki dio programa, svaka metoda je sigurno napisana na osobnom računalu koja je tek poslije spojena u jedno i stavljena u superračunalo. Da se svaki dio programa radi na superračunalu bilo bi nepotrebno korištenje energije jer takva računala nisu napravljena za obavljanje jednostavnih operacija. Matematičke metode je moguće koristiti i bez računala, ali ponekad matematički dokazi zahtijevaju mnogo vremena i bez računala ih je nemoguće riješiti. Možda se matematički dokaz sastoji od jednostavnih provjera, ali ako postoji puno iteracija i ponavljajućih računanja, tada je jednostavnije koristiti računalo. Bez računala, Flyspeck projekt i Booleov problem bojanja Pitagorinih trojki vjerojatno nikada ne bi bili dokazani. Računala postoje da obavljaju rutinske poslove, a na matematičarima je da mogu što više vremena provesti na stvaranju novih i kreativnih ideja.

Navedene matematičke metode se također koriste i u umjetnoj inteligenciji, ali skoro svaka metoda koja je spomenuta u ovome radu se preklapa s nekom drugom metodom. U rijetkim slučajevima je moguće riješiti problem samo s jednom metodom, zato je vrlo važno poznavati što više metoda kako bi se njihovim međusobnim kombinacijama mogao efikasnije riješiti matematički problem.

Popis literature

[1] G. Petrović, „Pregled povijesti logike”, *Metodički ogledi*, sv. 20, str. 129–182, ožu 2014. [Na internetu]. Dostupno: https://hrcak.srce.hr/118792. [pristupano: 11.09.2018.]

[2] B. Divjak i T. Hunjak, *Matematika za informatičare*, Varaždin : TIVA, 2004. Str. 10–22.

[3] Microsoft, „Operators (C# Programming Guide)”, srp 2015. [Na internetu]. Dostupno: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/operators. [pristupano: 11.09.2018.]

[4] K. Foth, „Why don't languages include implication as a logical operator?”, sij 2013. [Na internetu]. Dostupno: https://softwareengineering.stackexchange.com/questions/184089/why-dont-languages-include-implication-as-a-logical-operator. [pristupano: 12.09.2018.]

[5] M. Petrač, „Matematički dokaz”, *Osječki matematički list*, sv. 14, str. 157–166, ožu 2015. [Na internetu]. Dostupno: https://hrcak.srce.hr/135204. [pristupano: 12.09.2018.]

[6] M. Bašić, „Matematička indukcija”, *Playmath*, sv. 2, str. 6–12, ožu 2004. [Na internetu]. Dostupno: https://hrcak.srce.hr/2034. [pristupano: 12.09.2018.]

[7] „Definition: Negation Normal Form”, *u Proofwiki*, (bez dat.) [Na internetu]. Dostupno: https://proofwiki.org/wiki/Definition:Negation_Normal_Form. [pristupano: 12.09.2018.]

[8] A. Bonner, „What Was Llull Up To?”, (bez dat.) [Na internetu]. Dostupno: http://www.ramonllull.net/sw_studies/studies_original/compbon.html. [pristupano: 12.09.2018.]

[9] Article, „Gottfried Leibniz”, (bez dat.) [Na internetu]. Dostupno: http://history-computer.com/Dreamers/Leibniz.html. [pristupano: 12.09.2018.]

[10] „Hilbert, Gödel and Turing”, (bez dat.) [Na internetu]. Dostupno: http://www.philocomp.net/home/hilbert.htm. [pristupano: 12.09.2018.]

[11] A. Turing, „On computable numbers, with an application to the entscheidungsproblem”, stu 1936. [Na internetu]. Dostupno: https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf. [pristupano: 12.09.2018.]

[12] Y. Filmus, „Theoretical machines which are more powerful than Turing machines”, tra 2016. [Na internetu]. Dostupno: https://cs.stackexchange.com/questions/55489/theoretical-machines-which-are-more-powerful-than-turing-machines. [pristupano: 12.09.2018.]

[13] „What is the significance of the Entscheidungsproblem ("decision problem")?”, 2014. [Na internetu]. Dostupno: https://www.reddit.com//r/askscience/comments/1q1ds8/what_is_the_significance_of_the/. [pristupano: 12.09.2018.]

[14] M. Rouse, „Turing test”, (bez dat.) [Na internetu]. Dostupno: https://searchenterpriseai.techtarget.com/definition/Turing-test. [pristupano: 12.09.2018.]

[15] N. DuPaul, „The History of Programming Languages Infographic”, tra 2013. [Na internetu]. Dostupno: https://www.veracode.com/blog/2013/04/the-history-of-programming-languages-infographic. [pristupano: 12.09.2018.]

[16] L. De Mol, „Looking for Busy Beavers. A socio-philosophical study of a computer-assisted proof”, stu 2016. [Na internetu]. Dostupno: https://hal.univ-lille3.fr/hal-01396523/document. [pristupano: 12.09.2018.]

[17] A. Neumaier, „Computer-assisted proofs”, (bez dat.) [Na internetu]. Dostupno: https://www.mat.univie.ac.at/~neum/ms/caps.pdf. [pristupano: 12.09.2018.]

[18] M. Malek, „Numerical Analysis”, (bez dat.) [Na internetu]. Dostupno: http://www.mcs.csueastbay.edu/~malek/Class/Approx.pdf. [pristupano: 12.09.2018.]

[19] K. E. Atkinson, „Numerical Analysis”, 2007. [Na internetu]. Dostupno: http://www.scholarpedia.org/article/Numerical_analysis. [pristupano: 12.09.2018.]

[20] J. Dvornik, „Metode rješavanja problema pomoću računala”, *KoG*, sv. 7, str. 19–24, velj 2004. [Na internetu]. Dostupno: https://hrcak.srce.hr/3921. [pristupano: 12.09.2018.]

[21] „What do engineers use Numerical Methods for?”, (bez dat.) [Na internetu]. Dostupno: https://www.reddit.com/r/askscience/comments/302ara/what_do_engineers_use_numerical_methods_for/. [pristupano: 12.09.2018.]

[22] F. Kappetijn i H. Jonkers, „Aircraft Control System Design and Analysis Program Package”, *IFAC Proceedings Volumes*, sv. 12, str. 311–317, ruj 1979. [Na internetu]. Dostupno: https://www.sciencedirect.com/science/article/pii/S1474667017656143. [pristupano: 12.09.2018.]

[23] R. C. Rumpf, „Errors in Computation”, velj 2018. [Na internetu]. Dostupno: http://emlab.utep.edu/ee4386_5301cmee/Topic%201%20--%20Errors%20in%20%20Computation.pdf. [pristupano: 12.09.2018.]

[24] E. Conway, „Mars 1998/1999”, (bez dat.) [Na internetu]. Dostupno: https://www.jpl.nasa.gov/jplhistory/the90/mars-1998-t.php. [pristupano: 12.09.2018.]

[25] Y. J. Hwa, „What is the best programming language for numerical analysis?”, kol 2014. [Na internetu]. Dostupno: https://www.quora.com/What-is-the-best-programming-language-for-numerical-analysis. [pristupano: 12.09.2018.]

[26] J. Avigad, „The Mechanization of Mathematics”, *Notices of the AMS*, sv. 65, str. 681–690, srp 2018. [Na internetu]. Dostupno: https://www.ams.org/journals/notices/201806/rnoti-p681.pdf. [pristupano: 11.09.2018.]

[27] J. Avigad, „Formal methods in mathematics”, svi 2015, [Na internetu]. Dostupno: https://www.andrew.cmu.edu/user/avigad/Talks/australia1.pdf. [pristupano: 12.09.2018.]

[28] D. Galatas, „The Four Color Theorem”, stu 1995, [Na internetu]. Dostupno: http://people.math.gatech.edu/~thomas/FC/fourcolor.html. [pristupano: 12.09.2018.]

[29] W. Bibel, „Early History and Perspectives of Automated Deduction”, (bez dat.) [Na internetu]. Dostupno: http://www.intellektik.de/resources/OsnabrueckBuchfassung.pdf. [pristupano: 12.09.2018.]

[30] J. P. Bridge, „Machine learning and automated theorem proving”, stu 2010. [Na internetu]. Dostupno: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-792.pdf. [pristupano: 12.09.2018.]

[31] A. Avron i M. Sagiv, „Automated Theorem Proving”, (bez dat.) [Na internetu]. Dostupno: http://www.cs.tau.ac.il/~msagiv/courses/ATP/lecture-1.pdf. [pristupano: 12.09.2018.]

[32] A. L. Mann, „A Case Study in Automated Theorem Proving: Otter and EQP”, 2003. [Na internetu]. Dostupno: http://math.colgate.edu/~amann/MA/ma_thesis.pdf. [pristupano: 12.09.2018.]

[33] J. A. Kalman, „Automated Reasoning with Otter”, ruj 2002. [Na internetu]. Dostupno: http://www.rintonpress.com/books/review/myers.html. [pristupano: 12.09.2018.]

[34] J. Urban, „Automated Reasoning for Mizar:Artificial Intelligence through Knowledge Exchange”, (bez dat.) [Na internetu]. Dostupno: https://pdfs.semanticscholar.org/38e8/1fa8491abdac09d57a670e062cf4b4107177.pdf. [pristupano: 12.09.2018.]

[35] J. Harrison, J. Urban i F. Wiedijk, „History of interactive theorem proving”, (bez dat.) [Na internetu]. Dostupno: https://www.cl.cam.ac.uk/~jrh13/papers/joerg.pdf. [pristupano: 12.09.2018.]

[36] L. Jutting, „Checking Landau’s "Grundlagen" in the AUTOMATH system”, 1994. [Na internetu]. Dostupno: https://www.win.tue.nl/automath/archive/pdf/aut046.pdf. [pristupano: 12.09.2018.]

[37] J. Harrison, „The HOL Light theorem prover”, (bez dat.) [Na internetu]. Dostupno: https://www.cl.cam.ac.uk/~jrh13/hol-light/. [pristupano: 12.09.2018.]

[38] M. J. H. Heule, O. Kullmann i V. W. Marek, „Solving and Verifying the boolean Pythagorean Triples problem via Cube-and-Conquer”, svi 2016. [Na internetu]. Dostupno: https://arxiv.org/pdf/1605.00723.pdf. [pristupano: 12.09.2018.]

[39] C. P. Gomes, H. Kautz, A. Sabharwal i B. Selman, „Satisfiability Solvers”, 2008. [Na internetu]. Dostupno: https://www.cs.cornell.edu/gomes/pdf/2008_gomes_knowledge_satisfiability.pdf. [pristupano: 12.09.2018.]

[40] W. Klieber, „Solvers for the Problem of Boolean Satisfiability (SAT)”, kol 2011. [Na internetu]. Dostupno: http://www.cs.cmu.edu/~emc/15414-f11/lecture/lec02_grasp.pdf. [pristupano: 12.09.2018.]

[41] T. A. C. Center, „Cutting-Edge HPC, Visualization, Data Analysis and Data-Intensive Computing”, (bez dat.) [Na internetu]. Dostupno: https://www.tacc.utexas.edu/systems/stampede. [pristupano: 12.09.2018.]

[42] P. Sikavica, T. Hunjak, T. Hernaus i N. B. Ređep, *Poslovno odlučivanje*, Zagreb : Školska knjiga, 2014.

[43] S. Russel i P. Norving, *Artificial Intelligence: A Modern Approach*, New Jersey : Pearson Eduaction, Inc., 2010.

[44] I. Bratko, „Prolog Programming For Artificial Intelligence", 1986. [Na internetu]. Dostupno: https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/PROLOG%20PROGRAMMING%20FOR%20ARTIFICIAL%20INTELLIGENCE%20-%20Ivan%20Bratko.pdf. [pristupano: 12.09.2018.]

[45] M. Triska, „Theorem Proving with Prolog", (bez dat.) [Na internetu]. Dostupno: https://www.metalevel.at/prolog/theoremproving. [pristupano: 12.09.2018.]

[46] E. Tsang, „Combinatorial Explosion", svi 2005. [Na internetu]. Dostupno: https://cswww.essex.ac.uk/CSP/ComputationalFinanceTeaching/CombinatorialExplosion.html. [pristupano: 12.09.2018.]

[47] Scratchapixel, „Mathematical Foundations of Monte Carlo Methods", (bez dat.) [Na internetu]. Dostupno: https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-mathematical-foundations. [pristupano: 12.09.2018.]

[48] K. Brdar, M. Dobrinić i R. Joksić, „Fraktali", lip 2012. [Na internetu]. Dostupno: http://matematika.fkit.hr/novo/izborni/referati/dobrinic_joskic_brdar_fraktali.pdf. [pristupano: 12.09.2018.]

[49] L. Zadeh, „Fuzzy Logic", 1965. [Na internetu]. Dostupno: https://www.rpi.edu/dept/ecse/mps/Fuzzy_Logic.pdf. [pristupano: 12.09.2018.]

[50] „Heuristic Approaches to Problem Solving", velj 2018. [Na internetu]. Dostupno: http://www.101computing.net/heuristic-approaches-to-problem-solving/. [pristupano: 12.09.2018.]

[51] J. Young, „Symbolic Mathematics: A Simplification Program", srp 2012. [Na internetu]. Dostupno: https://github.com/norvig/paip-lisp/blob/master/docs/chapter8.md. [pristupano: 12.09.2018.]

[52] J. McCarthy, „History of Lisp", velj 1979. [Na internetu]. Dostupno: http://www-formal.stanford.edu/jmc/history/lisp/lisp.html. [pristupano: 12.09.2018.]

[53] Maplesoft, „What's New in Maple 2018", 2018. [Na internetu]. Dostupno: https://www.maplesoft.com/products/maple/new_features/. [pristupano: 12.09.2018.]

[54] W. Mathematica, „The world's definitive system for modern technical computing", (bez dat.) [Na internetu]. Dostupno: https://www.wolfram.com/mathematica/. [pristupano: 12.09.2018.]

[55] MatLab, „Symbolic Math Toolbox™ 5 MuPAD® Tutorial", 2008. [Na internetu]. Dostupno: http://www.calvin.edu/~tmk5/research/mupad_tutorial.pdf. [pristupano: 12.09.2018.]

[56] J. S. Cohen, „Computer Algebra And Symbolic Computation”, 2003. [Na internetu]. Dostupno: https://www.ukma.edu.ua/~yubod/teach/compalgebra/%5bJoel_S._Cohen%5d_Computer_algebra_and_symbolic_comp(BookFi.org).pdf. [pristupano: 12.09.2018.]

[57] R. J. Fateman, „Problem Solving Environments And Symbolic Computing”, (bez dat.) [Na internetu]. Dostupno: https://people.eecs.berkeley.edu/~fateman/papers/pse.pdf. [pristupano: 12.09.2018.]

[58] J. Brownlee, „Analytical vs Numerical Solutions in Machine Learning”, tra 2018. [Na internetu]. Dostupno: https://machinelearningmastery.com/analytical-vs-numerical-solutions-in-machine-learning/. [pristupano: 12.09.2018.]

[59] E. M. Clarke i R. P. Kurshan, „Computer-aided verification”, lip 1996. [Na internetu]. Dostupno: http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/Web/People/emc/papers/Papers%20In%20Refereed%20Journals/96_computeraided_verification.pdf. [pristupano: 12.09.2018.]

[60] C. Baier i J.-P. Katoen, „Principles of Model Checking”, 2008. [Na internetu]. Dostupno: http://is.ifmo.ru/books/_principles_of_model_checking.pdf. [pristupano: 12.09.2018.]

[61] I. Beer i dr., „RuleBase: Model Checking at IBM”, (bez dat.) [Na internetu]. Dostupno: https://pdfs.semanticscholar.org/5ddd/29b178e79b47b067baa92f4119f3c698d345.pdf. [pristupano: 12.09.2018.]

[62] E. W. Weisstein, „Kepler Conjecture”, (bez dat.) [Na internetu]. Dostupno: http://mathworld.wolfram.com/KeplerConjecture.html. [pristupano: 12.09.2018.]

[63] F. M. Brueckler, „Matematički božićni poklon s posljedicama”, *Hrvatska Revija*, sv. 3, str. 86–91, 2012. [Na internetu]. Dostupno: http://www.matica.hr/media/uploads/hr/2012/hr-2012-3.pdf. [pristupano: 11.09.2018.]

[64] T. Hales i dr., „A Formal Proof Of The Kepler Conjecture”, sij 2015. [Na internetu]. Dostupno: https://arxiv.org/pdf/1501.02155.pdf. [pristupano: 12.09.2018.]

[65] „ML Programming”, (bez dat.) [Na internetu]. Dostupno: https://www.whoishostingthis.com/resources/ml/. [pristupano: 12.09.2018.]

[66] IBM, „Deep Blue”, (bez dat.) [Na internetu]. Dostupno: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. [pristupano: 12.09.2018.]

[67] J. Young, „Introduction to Lisp”, srp 2012. [Na internetu]. Dostupno: https://github.com/norvig/paip-lisp/blob/master/docs/chapter1.md. [pristupano: 12.09.2018.]

[68] B. R. Donald, D. Kapur i J. L. Mundy, „Symbolic and Numerical Computation for Artificial Intelligence”, 1992. [Na internetu]. Dostupno: https://pdfs.semanticscholar.org/5d82/25cea171a04ac4409704778fcce19d26f1ae.pdf. [pristupano: 12.09.2018.]

[69] V. Voevodsky, „Univalent Foundations”, 2014. [Na internetu]. Dostupno: http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/2014_IAS.pdf. [pristupano: 12.09.2018.]

Popis slika

Popis tablica