MATEMATIKA IZVAN MATEMATIKE

Zaboravljena matematička povijest modernog računala

Boris Čulina[1], Dragana Čulina[2] i Marijan Čančarević[3]

Sažetak. U radu je izložena matematička povijest modernog računala o kojoj se izvan matematičke struke malo zna, a koja je vrlo važna za razumijevanje matematičkih i logičkih ideja koje su u osnovi računarstva i nastanka modernog računala (dijagonalna metoda, formalni sustavi, kodiranje, poluizračunljivost i izračunljivost, ekvivalentne formulacije izračunljivosti, univerzalni stroj s uskladištenim programom, neizračunljivost), kao i za razumijevanje važne uloge matematike u računarstvu.

Ključne riječi: dijagonalna metoda; Hilbertov program; formalni sustavi; Gödelovi teoremi, Turingovi strojevi; univerzalni Turingov stroj; problem dijagonalnog zaustavljanja; Church-Turingova teza; Von Neumannov draft

Danas, kad računalska tehnologija napreduje brzinom od koje zastaje dah,
i kad se divimo uistinu izvanrednim dostignućima inženjera,
lako se mogu previdjeti logičari čije su ideje učinile sve to mogućim.

Martin Davis [1]

I. Uvod

Općepoznato je da je moderno računalo, koncepcijski gledano, digitalno univerzalno računalo s uskladištenim programom. Njegov logički opis dao je 1945. godine John von Neumann u radu *First Draft of a Report on the EDVAC* [2]. Kao što i sam naslov rada kaže, u pitanju je bila prva skica za izvještaj o projektu *EDVAC*, projektu izgradnje novog računala pri *Moore School of Electrical Engineering* u SAD. Taj draft je brzo „procurio" izvan projekta i po njemu su napravljena prva moderna računala, počevši s britanskim računalom *Manchester Baby* iz 1948. godine.

Draft nikad nije prerastao u konačni izvještaj. Između ostalog, u draftu nisu citirani izvori niti navedeni doprinosi članova grupe. Tako je taj ključni događaj u povijesti računarstva ostao sporan. Najviše je prijepora bilo oko revolucionarnog koncepta uskladištenog programa. Koliko su tom konceptu pridonijeli glavni inženjeri na projektu John Presper Eckert i John Mauchly, a koliko matematičar John von Neumann koji je kao konzultant uključen u projekt? Najmjerodavniji za odgovor trebao bi biti sam voditelj projekta Herman Goldstine koji je napisao sljedeće: „Prije njegova [von Neumannova] dolaska *Moore School* grupa bila je koncentrirana primarno na tehnološke probleme koji su bili vrlo veliki. Nakon njegova dolaska on je preuzeo vodstvo u logičkim problemima. ... Svakako, i prije von Neumanna ljudi su znali da se

[1]Boris Čulina, Veleučilište Velika Gorica
[2]Dragana Čulina, Visoka škola za informacijske tehnologije, Zagreb
[3]Marijan Čančarević, Visoka škola za informacijske tehnologije, Zagreb

električni krugovi grade za izvršenje aritmetičkih i upravljačkih funkcija, ali su bili primarno koncentrirani na elektroinženjerske aspekte problema. Ti su aspekti bili, naravno, od vitalnog značaja, ali von Neumann je taj koji je prvi dao logički tretman ovom predmetu, kao da je on standardni dio logike ili matematike. ...von Neumann je u svom izvještaju dao logički potpunu analizu strukture *EDVACa*. To je bio njegov prvi veliki doprinos. ... Cijelo njegovo znanje formalne logike pripremilo ga je za to i ono nesumnjivo nosi pečat njegova genija." [3]. Podjela doprinosa koju navodi Goldstine konačno je i očekivana s obzirom na struke sudionika grupe.

Međutim, koje je to znanje iz formalne logike, koje spominje Goldstine, pripremilo von Neumanna za taj revolucionarni korak? Sve do kraja osamdesetih godina prošlog stoljeća o tome se u svijetu računarstva gotovo ništa nije znalo. Uopće se nije spominjao Alan Turing koji je u osnovi sve to napravio već 1936. u svome radu neobičnog naslova *On Computable Numbers with an Application to the Entscheidungsproblem* [4]. Taj rad je von Neumann dobro poznavao jer se i sam dugo bavio područjem kojemu rad pripada. Takva situacija je zaintrigirala Martina Davisa, poznatog eksperta iz područja matematičke logike i teorijskog računarstva. Godine 1987. napisao je članak o Turingovu radu i njegovu utjecaju na von Neumannov draft [5]. Nakon toga sve je više jačala svijest o Turingovoj ključnoj ulozi u nastanku modernog računala. Međutim, kroz Turingov rad u računarstvo su ugrađene ideje i rezultati cijelog jednog smjera u matematici koji se bavio problemima zasnivanja matematike. I to je ta zaboravljena povijest računarstva o kojoj će biti riječi u ovom članku. Ona je puna velikih matematičkih ideja i rezultata, ali i neobičnih i herojskih osoba. To je povijest koju vrijedi (još jednom) ispričati. Između ostalog, ona pokazuje i da računarstvo nije samo inženjerstvo, kako se uobičajeno misli, već su u njemu ravnopravno zastupljeni inženjerstvo i matematika.

Naravno, ova povijest ljudi i ideja ne bi se mogla vjerodostojno izložiti ni u više knjiga, a kamoli u nevelikom članku. Mi smo iz nje izvukli samo neke ključne elemente, a bitnu matematiku neformalno izložili. Za obuhvatnije izlaganje upućujemo čitatelja na knjigu Martina Davisa [1] koja je izdana povodom 100 godina od rođenja Alana Turinga. Za precizno izlaganje matematičkog sadržaja upućujemo čitatelja na predavanja Mladena Vukovića koja se mogu naći na njegovoj web stranici https://www.math. pmf.unizg.hr/hr/publication-croatian-vukovic (na pojedinim mjestima u članku dane su reference na odgovarajuća predavanja). Ako se čitatelju pojedini matematički sadržaj učini pretežak, ništa čudno: pored neuspjeha autora da ga bolje izlože, razlog može biti i u tome da su u pitanju zaista neobične matematičke ideje. Vjerujemo da će čitatelju, i bez udubljivanja u pojedine matematičke dijelove, ovo povijesno putovanje biti zanimljivo i poučno.

II. Cantor dijagonalnom metodom analizira beskonačnost

Našu ćemo priču započeti s Georgom Cantorom (1845. – 1918.) koji je svojim radovima dobrano uzdrmao dotadašnja shvaćanja matematike. Valovi toga potresa, zajedno s pravim matematičkim biserom koji je Cantor otkrio – dijagonalnom metodom, bitno su obilježili matematičku povijest računarstva. „Problemi" su nastali kad je Cantor, baveći se sasvim praktičnim pitanjima konvergencije Fourierovih redova, počeo izučavati dotad „zabranjeno područje" – beskonačne skupove. Ne samo da je pokazao da je skup realnih brojeva bitno više beskonačan nego skup prirodnih brojeva, nego je pokazao da postoji cijela (beskonačna) hijerarhija sve većih beskonačnosti. S matematičkom imaginacijom kakva se rijetko viđa izgradio je jedan cijeli elegantni svijet beskonačnih skupova. Osnovni alat u toj izgradnji bila mu je dijagonalna metoda. O snažnom protivljenju koje je doživio ne samo od teologa i filozofa, već

i od cijele skupine matematičara, i o njegovoj herojskoj borbi za svoje ideje možete pročitati u [1]. O matematici beskonačnih skupova možete pročitati u [6]. Mi ćemo ovdje objasniti osnovnu ideju dijagonalne metode i primijeniti metodu da pokažemo nešto što tada nije bilo u fokusu pažnje – da *postoje neizračunljive funkcije, funkcije za koje ne možemo napisati program koji bi ih računao*. No, prvo ćemo uvesti malo terminologije.

Da funkcija *f* preslikava elemente skupa *A* u elemente skupa *B* označavamo $f : A \to B$. Ako pritom preslikava sve elemente skupa *A* (na svim elementima je definirana), tada kažemo da je totalna. Ako pak poprima za vrijednosti sve elemente iz skupa *B*, tada kažemo da je surjekcija. Ako na različitim ulazima poprima različite vrijednosti ($x_1 \neq x_2 \to f(x_1) \neq f(x_2)$), tada kažemo da je injekcija. Funkciju $f : A \to B$ koja je totalna, surjekcija i injekcija nazivamo bijekcijom između skupova *A* i *B*. Ako postoji bijekcija s nekog skupa *S* u skup (pozitivnih) prirodnih brojeva $\mathbb{N}$, tada kažemo da je skup *S* prebrojivo beskonačan.

Objasnimo sada *dijagonalnu metodu*. Ona nam kaže kako iz zadane liste lista dobiti novu listu (pojam liste ćemo intuitivno razumijevati). Neka imamo prebrojivo beskonačnu listu prebrojivo beskonačnih lista objekata iz nekog skupa *S*: $L_1, L_2 \ldots$. Elemente liste L_n redom ćemo označavati: $L_n(1), L_n(2) \ldots$. Ovu listu lista slikovito možemo predstaviti tablicom u kojoj svaki redak predstavlja jednu listu (Slika 1.)

	1	2	3	. . .
L_1	$L_1(1)$	$L_1(2)$	$L_1(3)$	
L_2	$L_2(1)$	$L_2(2)$	$L_2(3)$	
L_3	$L_3(1)$	$L_3(2)$	$L_3(3)$	
.				.
.				.
.				.

Slika 1.

Dijagonala tablice također nam daje jednu listu: $D(\mathrm{n}) = L_n(n)$ (Slika 2.)

	1	2	3	. . .
L_1	$L_1(1)$	$L_1(2)$	$L_1(3)$	
L_2	$L_2(1)$	$L_2(2)$	$L_2(3)$	
L_3	$L_3(1)$	$L_3(2)$	$L_3(3)$	
.				.
.				.
.				.

Slika 2.

Neka je $t : \mathrm{S} \to \mathrm{S}$ neka totalna funkcija na skupu *S*. Pomoću nje ćemo transformirati dijagonalnu listu u listu $TD(n) = \mathrm{t}(D(\mathrm{n}))$.

Dijagonalni teorem.

1) Ako funkcija *t* svaki objekt dijagonalne liste $D(n)$ preslikava u neki drugi objekt ($TD(n) \neq D(n)$), tada je *TD* nova lista koja je različita od svih lista L_n iz tablice.
2) Ako je *TD* jedna od lista u tablici, tada transformacija nije promijenila član u toj listi koji se nalazi na dijagonali.

Dokaz.

1) Ako je funkcija izmijenila sve članove dijagonalne liste, tada se od prve liste razlikuje na prvom mjestu jer $TD(1) \neq D(1) = L_1(1)$. Iz istog se razloga od druge liste razlikuje na drugome mjestu, itd. Dakle, različita je od svih lista u tablici.
2) Ako je *TD* jedna od lista u tablici, recimo *n*-ta, tad se na njenoj dijagonali nalazi element koji je gledano po dijagonali jednak *D*(*n*), a gledano po listi jednak *TD*(*n*). Dakle, $TD(n) = \mathrm{D}(n)$.

Za ilustraciju upotrebe dijagonalne metode dokazat ćemo da postoje neizračunljive funkcije. Naravno, time preskačemo cijelu ovu povijest koja je, između ostaloga, i precizno

definirala pojam izračunljive funkcije pa danas koristimo već ustaljen pojam: to su funkcije za koje možemo napisati program koji ih računa u nekom programskom jeziku, npr. *Pythonu*.

Prvo ćemo pokazati da skup izračunljivih funkcija $\mathbb{N} \to \mathbb{N}$ možemo smjestiti u prebrojivo beskonačnu listu. Programski jezik *Python* sastavljen je od konačnog skupa simbola koje možemo poredati nekim redoslijedom. Taj redoslijed određuje leksikografski redoslijed među svim stringovima (konačnim nizovima osnovnih simbola) iste duljine. Sve stringove možemo poredati na sljedeći način. Prvo ispišemo u zadanom redoslijedu sve stringove duljine 1. Zatim leksikografski poredamo stringove duljine 2, pa duljine 3, itd. Na taj smo način poredali sve stringove u jednu prebrojivo beskonačnu listu. Sad redom na stringove puštamo *Python* interpreter koji iz te liste izdvaja listu Python programa. Među njima dalje izdvojimo one koji na ulazu i na izlazu imaju jedan prirodan broj. Tako ćemo dobiti prebrojivo beskonačnu listu programa koji računaju funkcije $\mathbb{N} \to \mathbb{N}$ koje nisu nužno totalne (može se dogoditi da na nekim ulazima program nikada ne staje). Time smo ujedno dobili i prebrojivo beskonačnu listu f_n takvih funkcija (neke se mogu i ponavljati). S obzirom da svaku od tih funkcija možemo shvatiti kao listu njenih vrijednosti (ako na nekom ulazu nije definirana, tad joj npr. možemo pridružiti oznaku $\uparrow$), što znači da imamo listu lista prikazanu na Slici 3.

	1	2	3	. . .
f_1	$f_1(1)$	$f_1(2)$	$f_1(3)$	
f_2	$f_2(1)$	$f_2(2)$	$f_2(3)$	
f_3	$f_3(1)$	$f_3(2)$	$f_3(3)$	
.				.
.				.
.				.

Slika 3.

Primijenimo sad dijagonalnu metodu. Dijagonalna lista određuje funkciju $D : \mathbb{N} \to \mathbb{N}$, $D(n) = f_n(n)$. Transformirajmo je u novu funkciju tako da joj svaku vrijednost izmijenimo. Npr. to možemo učiniti na sljedeći način. Ako $D(n)$ nije definirana, stavit ćemo da je $TD(n) = 0$. Ako je pak definirana, stavit ćemo da je $TD(n) = D(n) + 1$. Time je osigurano da je svaki član liste izmijenjen. Po dijagonalnom teoremu, funkcija (lista) *TD* nije nijedna od izračunljivih funkcija (nijedna od lista iz tablice). Dakle, ona je neizračunljiva funkcija.

Uz malo više poznavanja Cantorove matematike beskonačnosti može se lako pokazati da neizračunljivih funkcija ima bitno više (tvore višu beskonačnost) nego izračunljivih funkcija: izračunljivih funkcija ima koliko i prirodnih brojeva, a neizračunljivih koliko i realnih brojeva.

Kad gledamo cijeli postupak kako smo odredili neizračunljivu funkciju *TD*, u njemu mora biti neki neizračunljivi element. Inače bismo imali postupak računanja neizračunljive funkcije *TD*, dakle kontradikciju. Identificirajmo taj element. Listu stringova možemo računski generirati. Puštajući interpreter redom na te stringove možemo računski generirati listu programa. Ispitujući naredbe u tim programima možemo računski generirati listu programa koji računaju funkcije sa $\mathbb{N}$ u $\mathbb{N}$. Puštajući te programe na raznim ulazima možemo računski generirati cijelu tablicu. Jedini je problem što na nekim ulazima programi neće dati izlaz, već će beskonačno raditi. Upravo tu leži odgovor. Kad bismo imali računski postupak (program) kojim bismo za svaki program i ulaz u taj program mogli odrediti staje li program na tom ulazu (tzv. *problem zaustavljanja*), tad bismo cijelu tablicu mogli računski generirati, što bi za posljedicu imalo i da bismo mogli imati program za računanje funkcije *TD*. Budući da je ta funkcija neizračunljiva, to znači da problem zaustavljanja nije izračunljiv. Štoviše, da bismo mogli računati funkciju *TD,* dovoljno bi bilo imati računski postupak koji bi samo na dijagonalnim elementima rješavao problem zaustavljanja. Dakle, ni *problem dijagonalnog zaustavljanja*, tj. problem određenja hoće li *n*-ti program u listi programa stati na ulazu *n*, nije izračunljiv.

No, ovim razmatranjem preskočili smo vremenski redoslijed jer je baš Alan Turing bio prvi čovjek kojemu je palo na um primijeniti dijagonalnu metodu na izračunljive funkcije i izvući gornje zaključke.

III. Hilbert spašava matematiku od paradoksa

Teorija skupova pokazala se vrlo elegantnom i moćnom teorijom u kojoj se lako mogu modelirati i međusobno povezivati razne zamisli, te dokazivati razne tvrdnje o njima. Dapače, pokazala se i nužnom za preciznije određenje osnovnih matematičkih struktura, kao npr. strukture prirodnih brojeva. Tako je teorija skupova u matematici prihvaćena kao osnovna matematička teorija. Međutim, krajem 19. stoljeća počeli su se u njoj otkrivati razni paradoksi. Obično su bili povezani s naprednijim dijelovima teorije, pa se vjerovalo da će, kad se sve malo bolje promisli, paradoksi nestati. Međutim, Bertrand Russell otkrio je 1902. godine vrlo elementaran paradoks, danas ga nazivamo Russellov paradoks, koji je pokazao da su problemi mnogo dublji nego što se mislilo.

Da bismo „otkrili" *Russellov paradoks,* opet ćemo pustiti u pogon dijagonalnu metodu. Sad će naša lista listā sadržavati sve skupove. Cantor je pokazao kako se svi skupovi mogu prebrojiti, samo se umjesto prirodnih brojeva moraju koristiti ordinali. Nama je ovdje jedino važno znati da će liste biti numerirane ordinalima koji su (prirodno) produženje u beskonačnost niza prirodnih brojeva: prirodni brojevi tvore početni dio ordinalnog nizanja. Liste neće biti prebrojivo beskonačne nego bitno veće, ali se može pokazati da sve što je prethodno rečeno o dijagonalnom metodu i sada vrijedi. Dakle, po Cantoru sve skupove možemo ordinalno nanizati u jednu listu: s_1, s_2,... . Za svaki takav skup možemo gledati koji mu je skup element, a koji ne. Tako za svaki skup s_n imamo listu nula i jedinica, gdje je na i-tom mjestu u listi 0 ako skup s_i ne pripada skupu s_n, inače je 1. Tako naša listā lista može izgledati kao na Slici 4.

	s_1	s_2	s_3	. . .
s_1	0	1	0	
s_2	1	1	0	
s_3	1	0	0	
.				.
.				.
.				.

Slika 4.

Npr. iz drugog retka možemo iščitati koji su skupovi elementi skupa s_2 : s_1 mu jest element, s_2 mu jest element, s_3 mu nije element,… Dijagonalna lista također određuje jedan skup d. To je skup kojem s_1 nije element, s_2 mu jest element, s_3 mu nije element,... Zamijenimo li u dijagonalnoj listi jedinice nulama a nule jedinicama, dobit ćemo transformiranu listu koja određuje skup td. To je skup kojem s_1 jest element, s_2 mu nije element, s_3 mu jest element,… Budući da je ova transformacija dijagonalne liste izmijenila sve njezine elemente, po dijagonalnom teoremu ovaj skup nije nijedan od skupova iz tablice. Ali u tablici su svi skupovi. Dobili smo kontradikciju. Još govorimo i o paradoksu jer nije jasno zašto td nije skup.

Često je dijagonalna metoda samo vizualan način razmišljanja kojim naslutimo neki rezultat koji možemo i direktno pokazati. To je i ovdje slučaj. Lako je vidjeti da dijagonalna lista odgovara na pitanje pripada li skup s_n sam sebi: 0 znači da ne pripada, a 1 da pripada. Dakle, dijagonalni skup je skup svih skupova koji sebi pripadaju, $d = \{x \mid x \in x\}$, pa je antidijagonalni skup skup svih skupova koji sebi ne pripadaju, $td = \{x \mid \neg\, x \in x\}$ (znak $\neg$ je znak negacije). Dijagonalni teorem kaže nam da kad bi transformirani skup td bio jedan od skupova iz tablice, onda bi se na dijagonalnom elementu razlikovao na sebi samom – dobili bismo kontradikciju. Sad ćemo tu kontradikciju direktno dobiti. Ovdje dijagonalni element odgovara na pitanje je li $td \in td$. Po uvjetu pripadnosti skupu td, neki skup x mu pripada baš kad sebi ne pripada:

$x \in td \Leftrightarrow \neg\, x \in x$. Primijenimo li to na sam *td*, dobijemo kontradikciju – *td* sebi pripada upravo kad sebi ne pripada:

$$td \in td \Leftrightarrow \neg\, td \in td$$

Otkriće kontradikcija u teoriji skupova koja se afirmirala kao osnovna matematička teorija značilo je da je cijela matematička „zgrada" izgrađena na nesigurnim temeljima. Nastala je prava uzbuna u matematičkom svijetu. Prvih desetljeća 20. stoljeća pokrenuta su tri velika programa spašavanja matematike, odnosno izgradnje čvrstih osnova matematike: logicizam, intuicionizam (konstruktivizam) i formalizam (Hilbertov program). Nijedan od tih programa nije uspio „spasiti" matematiku, ali ju je obogatio nekim od najljepših matematičkih sadržaja (u međuvremenu, matematika je pokazala da joj sasvim dobro ide i bez čvrstih temelja). *Logicizam* (Gottlob Frege, Bertrand Russell, ...) je namjeravao spasiti matematiku tako da je svede na logiku: da pokaže da se matematički pojmovi mogu definirati pomoću logičkih pojmova i da se matematički teoremi mogu dokazati na osnovi logičkih istina. Ne samo da se to pokazalo stranim stvarnom tkivu matematike, već se to jednostavno nije uspjelo napraviti. *Intuicionizam* (Luitzen Egbertus Jan Brouwer, Arend Heyting, ...), kojem bolje pristaje ime konstruktivizam (kad ga oslobodimo od misticizma), inzistirao je na tome da matematika smije koristiti samo konstruktivne metode. On je uvelike utjecao na računarstvo jer je razvio niz konstruktivnih metoda i načina razmišljanja. Međutim, pokazalo se da bi inzistiranje na konstruktivnosti zahtijevalo odbacivanje najljepših i najuspješnijih dijelova matematike, kao npr. diferencijalnog i integralnog računa. Drugim riječima, intuicionizam bi spasio matematiku tako da bi je osakatio. To također nije prošlo. Treći program, *Hilbertov program* (formalizam nije baš sretno odabran naziv za taj program), program je unutar kojega se dalje odvija glavni dio povijesti koju pratimo (mada su i druga dva smjera vrlo značajan dio te povijesti) i zato ćemo njemu posvetiti posebnu pažnju.

Od početka 20. stoljeća pa sve do dolaska nacizma na vlast 30-ih godina, Göttingen u Njemačkoj bio je centar matematičkog svijeta (a i centar svijeta fizike), a njegova glavna zvijezda bio je David Hilbert (1862. – 1943.), čovjek izuzetno širokih nazora, ne samo u matematici, već i u pitanjima ljudskosti. Više o Hilbertu i tom dobu možete pročitati u [1].

Hilbert je zamislio program spašavanja koji bi imao sljedeće glavne crte:

1) Svaku matematičku teoriju treba posve precizno formulirati: točno precizirati jezik, pravila dokazivanja i aksiome teorije. Pri tome skup aksioma mora biti efektivan, a pravila dokazivanja konstruktivna, u smislu da možemo efektivno provjeriti je li nešto aksiom ili ne, odnosno je li neki niz zaključivanja dokaz ili ne.
2) Za tako formuliranu teoriju treba finitnim sredstvima (konstruktivnom manipulacijom sa simbolima) dokazati konzistentnost.

Prema Hilbertu, postizanje ovih dvaju ciljeva osiguralo bi i treći važan cilj, što bi i konstruktiviste zadovoljilo, konstruktivan dokaz konzervativnosti matematičkih teorija: svaka finitno smislena tvrdnja (tvrdnja čiji je sadržaj konstruktivan), koja se može dokazati u toj teoriji, može se dokazati i finitnim sredstvima.

Vidimo da se u programu dosta spominje forma, pa je odatle dobio naziv – formalizam. Ali to ne znači da Hilbert i drugi sudionici nisu matematiku razumijevali kao sadržajnu ljudsku djelatnost.

U formulaciji programa prisutni su pojmovi konstruktivnosti, efektivne provjerljivosti i finitnosti, kojima nije dano precizno značenje, nego se oslanjaju uglavnom na intuiciju. Pojam

efektivne provjerljivosti je tek poslije precizno formuliran u Turingovom radu. Ostali pojmovi – precizan jezik, pravila dokazivanja, konzistentnost i neki drugi koje ćemo spomenuti, tijekom razvoja ovog programa, ali i ostalih dvaju programa, dobili su posve precizno značenje. Precizne formulacije možete naći u [7]. Ovdje ćemo samo „izdaleka" opisati ove pojmove.

S obzirom da čitatelji imaju iskustvo s matematičkim jezikom, precizne jezike koji su razvijeni u matematičkoj logici možemo kratko opisati kao do kraja precizirane varijante matematičkog jezika. Ovdje će biti riječi samo o takozvanim jezicima prvog reda, a mi ćemo ih radi jednostavnosti zvati (logički) jezici. Nadalje, stalno ćemo podrazumijevati interpretirane jezike, jezike koji govore o nečem određenom, prirodnim brojevima, skupovima,... . Svaka tvrdnja takvog jezika je ili istinita ili lažna. Da je u jeziku L tvrdnja φ istinita, označavat ćemo $L \vDash \varphi$.

Najvažnija relacija među tvrdnjama logičkog jezika je relacija *logičke posljedice*. Intuitivno, da iz nekog skupa tvrdnji A logički slijedi tvrdnja φ, znači da samo na osnovi pretpostavke da su sve tvrdnje iz A istinite, neovisno o čemu govore, koristeći značenja jezičnih konstrukcija možemo odrediti istinitost tvrdnje φ. Npr. iz tvrdnje $x > 3$ i $x \in \mathbb{N}$, koristeći jezični smisao veznika „i", možemo logički zaključiti da je $x > 3$ (istinitost početne tvrdnje znači da obje tvrdnje spojene veznikom „i" moraju biti istinite, pa tako mora biti istinita i prva tvrdnja). Nasuprot tome, iz tvrdnje $x > 3$ ili $x \in \mathbb{N}$ ne možemo logički zaključiti da je $x > 3$ (istinitost početne tvrdnje znači da je barem jedna od tvrdnji spojenih veznikom „ili" istinita, ali to ne mora biti prva tvrdnja). Da iz skupa rečenica A logički slijedi rečenica φ, označavamo $A \vDash \varphi$.

Da iz nekog skupa tvrdnji logički slijedi neka tvrdnja obično izvodimo u nizu jednostavnih koraka koji su formalne prirode (ispituje se samo forma rečenica). Tako npr. iz rečenice oblika φ i ψ možemo izvesti rečenicu φ. Takav sustav izvođenja nazivamo *deduktivni sustav*. Niz koraka kojim iz nekog skupa rečenica A pomoću ovog deduktivnog sustava izvedemo rečenicu φ nazivamo *izvodom* u tom sustavu. Deduktivni sustav mora biti efektivan u smislu da mora postojati algoritam koji za svaki niz koraka može odrediti je li to izvod u danom deduktivnom sustavu ili ne. Da je u nekom deduktivnom sustavu iz skupa rečenica A izvediva rečenica φ, označavamo $A \vdash \varphi$.

Naravno, osnovni cilj je da deduktivnim sustavom posve opisujemo logičku posljedicu, tj. da vrijedi svojstvo *potpunosti deduktivnog sustava*:

$$A \vDash \varphi \Leftrightarrow A \vdash \varphi$$

Kurt Gödel je 1929. godine u svom doktorskom radu dokazao potpunost jednog sustava dokazivanja. Prvi takav sustav napravio je Gottlob Frege koji se zbog svojih revolucionarnih radova iz područja logike ponekad naziva i ocem moderne logike. Svi danas upotrebljivi deduktivni sustavi su potpuni. Dok su neki od njih prilagođeni računalu ili pak teorijskim razmatranjima, postoji i jedan sustav, tzv. sustav prirodne dedukcije, koji do kraja precizira način kako mi razmišljamo.

Razmotrimo s računske strane što znači postojanje potpunog deduktivnog sustava za relaciju logičke posljedice. Da bismo ispitali slijedi li logički iz nekog skupa rečenica A rečenica φ, u nekom takvom deduktivnom sustavu izvodimo rečenice iz skupa A (ili damo to računalu da radi). Ako izvedemo rečenicu φ, procedura nam je dala odgovor *Da*. Ako pak φ logički ne slijedi iz A, tada je nikada nećemo u izvođenju dobiti. Dakle, procedura ne može dati odgovor *Ne*! Tako imamo program koji za dani skup rečenica A i rečenicu φ na izlaz daje odgovor *Da* kad je $A \vDash \varphi$, a ne staje kad nije $A \vDash \varphi$. Za relaciju za koju postoji takav program kažemo da je

poluizračunljiva. Naravno, mi bismo htjeli imati program koji daje odgovor *Da* kad je $A \vDash \varphi$ i *Ne* kad nije $A \vDash \varphi$. Za takvu relaciju kažemo da je *izračunljiva*. Hilbert je upravo za relaciju logičke posljedice 1928. godine postavio problem njene izračunljivosti. Upotrijebio je njemačku riječ *Entscheidungsproblem* (problem odluke) koja je postala univerzalno prihvaćena i koju smo već sreli u naslovu Turingovog rada. To je drugi važan sastojak naše priče (prvi je dijagonalna metoda). Vidjeli smo da je Kurt Gödel svojim teoremom potpunosti deduktivnog sustava dokazao da je relacija logičke posljedice poluizračunljiva. Vidjet ćemo da je Alan Turing u svom radu dokazao da ona nije izračunljiva.

Pod *skupom aksioma* smatramo svaki skup tvrdnji nekog logičkog jezika za koji postoji algoritam koji za svaku rečenicu jezika može utvrditi pripada li ona ovom skupu ili ne. Osnovno svojstvo skupa aksioma A je *konzistentnost*, da ne možemo iz njega izvesti neku tvrdnju zajedno s njenom negacijom:

Ne postoji rečenica φ takva da je i $A \vdash \varphi$ i $A \vdash \neg \varphi$.

Može se pokazati da bismo, kad bi sustav bio nekonzistentan, u njemu mogli dokazati sve rečenice. Dakle, takav sustav je neupotrebljiv. Naravno, željeli bismo i da je skup aksioma *korektan*, da se iz njega mogu izvesti samo istinite rečenice jezika:

$$A \vdash \varphi \rightarrow L \vDash \varphi$$

Još bi bolje bilo da je skup aksioma *potpun*, da se iz njega mogu izvesti sve istinite rečenice jezika i samo one (ovaj drugi dio slijedi iz prvog ako je teorija konzistentna):

$$A \vdash \varphi \leftrightarrow L \vDash \varphi$$

Pored toga što je precizirao jezik prvog reda i mnoštvo značajnih pojmova, kao i identificirao značajne probleme, Hilbertov program imao je i početnih uspjeha u njihovu rješavanju. U tom programu je kao Hilbertov asistent važan sudionik bio i von Neumann, što je isto značajno za našu priču. No ono najvažnije se nije uspijevalo napraviti – dokazati konzistentnost teorije prirodnih brojeva, najbazičnije matematičke teorije.

Kad je Hilbert sa 68 godina otišao u mirovinu, još uvijek je bio pun optimizma. Te 1930. godine njegov rodni grad Königsberg dodijelio mu je titulu počasnog građanina. Tom prigodom održao je u Königsbergu predavanje o prirodnim znanostima i logici na kojemu je u svom tipičnom optimističnom stilu tvrdio da nema nerješivih problema. Predavanje je završio poznatim riječima: *Wir müssen wissen – wir werden wissen* (Mi moramo znati – mi ćemo znati). Ironijom sudbine, u Königsbergu je nekoliko dana ranije održan simpozij o osnovama matematike na kojem je mladi austrijski matematičar Kurt Gödel obznanio svoje rezultate koji su srušili Hibertov program.

IV. Gödel kodira matematiku i postavlja granice formalizma

Kurt Gödel (1906. – 1978.) godine je 1931. objavio rad [8] u kojem je sredstvima o kojima dotad nitko nije ni sanjao postigao rezultate koji su mnogima srušili snove.

Mi ćemo ovdje neformalno opisati jednu apstraktnu i semantičku verziju Gödelovih teorema o nepotpunosti aritmetike, jer će nam to omogućiti da se u kratkom prikazu usredotočimo na bitne ideje koje leže u pozadini tih teorema, kao i njihove posljedice za Hilbertov program. Detaljno o Gödelovim teoremima možete saznati u [9], a o njegovoj neobičnoj ličnosti u [1].

Gödelovi teoremi, u tzv. semantičkoj verziji, govore o aksiomima pomoću kojih izvodimo istinite tvrdnje o prirodnim brojevima u jeziku koji sadrži oznaku za nulu (0), za funkciju

sljedbenika (S), za operacije zbrajanja (+) i množenja (·), te za relaciju usporedbe (<) prirodnih brojeva. Ovaj ćemo jezik zvati *jezik aritmetike*.

Gödelov teorem o nepotpunosti aksioma aritmetike (semantička verzija): *Za svaki skup aksioma koji su istinite tvrdnje jezika aritmetike postoji istinita tvrdnja jezika aritmetike koja se iz njih ne može ni dokazati ni oboriti* (*dokazati njena negacija*).

Gödelov teorem o nedokazivosti konzistentnosti aksioma aritmetike (semantička verzija): *Za svaki skup aksioma koji su istinite tvrdnje jezika aritmetike i koji sadrže Peanovu aritmetiku vrijedi da se iz njih ne može ni dokazati ni oboriti tvrdnja koja izražava njihovu konzistentnost.*

Ključno je u ovoj formulaciji, podsjetimo se definicije skupa aksioma, da skup aksioma mora biti izračunljiv. Ovdje nećemo precizirati što je *Peanova aritmetika*, već ćemo samo reći da se ona sastoji od nekih elementarnih istina o prirodnim brojevima i operacijama nad njima iz kojih možemo izvesti svu standardnu aritmetiku prirodnih brojeva. Tako drugi Gödelov teorem zahtijeva da je skup aksioma „dovoljno jak". Napomenimo da teorem vrijedi i za skupove aksioma koji ne sadrže Peanovu aritmetiku nego nešto slabiju, tzv. Robinsonovu aritmetiku. Poslije ćemo navesti tvrdnju koja izražava konzistentnost nekog skupa aksioma. I drugi Gödelov teorem je teorem o nepotpunosti jer također govori o postojanju istinite tvrdnje koja se ne može ni dokazati ni oboriti.

Posljedice za Hilbertov program su devastirajuće. Prvi Gödelov teorem pokazuje da se ne može ispuniti ni prva točka Hilbertova programa: *skup aritmetičkih istina uopće se ne može prikazati kao aksiomatska teorija*! Drugi Gödelov teorem pokazuje da se ne može ispuniti ni druga točka Hilbertova programa. Ma što točno značila finitna sredstva kojima bismo trebali dokazati konzistentnost nekog malo jačeg skupa istinitih aksioma aritmetike *A*, ona se mogu opisati pomoću Peanove aritmetike, pa tako i pomoću *A*. Kad bismo takvim sredstvima mogli dokazati konzistentnost *A,* tada bi se iz *A* mogla izvesti tvrdnja koja izražava konzistentnost od *A*. No, po drugom Gödelovom teoremu to je nemoguće. Dakle, *finitnim sredstvima ne možemo dokazati konzistentnost od A*!

U daljnjem ćemo tekstu ocrtati dvije za našu priču bitne ideje na kojima počiva dokaz prvog Gödelovog teorema, kodiranje i dijagonalizaciju (koju je Gödel preuzeo od Cantora).

Gödela je za dokaz inspirirao *paradoks lašca*, odnosno rečenica koja govori o sebi samoj da je lažna. Kad bi bila istinita, tj. kad bi bilo istina to što kaže, bila bi lažna. Kad bi bila lažna, tj. kad bi bila laž to što govori, bila bi istinita. Dakle, takva je rečenica istinita upravo onda kada je lažna – dobili smo kontradikciju. Gödel se upitao što bi bilo ako ona govori o vlastitoj nedokazivosti, tj. o samoj sebi kaže da je nedokaziva. Podrazumijevat ćemo da se radi o dokazivosti iz nekog istinitog skupa aksioma. Kad bi ta rečenica bila dokaziva, bila bi istinita, pa bi to značilo da nije dokaziva, a to je u kontradikciji s početnom pretpostavkom. Dakle, ona nije dokaziva. To ujedno znači da je istina što govori. Kad bi njena negacija bila dokaziva, onda bi njena negacija bila istinita. To ne može biti jer smo već utvrdili da je sama ta rečenica istinita, pa je njena negacija lažna. Dakle, ni njena negacija nije dokaziva. Tako je rečenica koja govori o sebi da nije dokaziva istinita rečenica koja nije ni dokaziva ni oboriva. Još "samo" preostaje naći takvu rečenicu. Dvije ključne stvari treba riješiti:

1) Kako rečenica aritmetičkog jezika koja govori o brojevima može govoriti o dokazivosti?
2) Kako može govoriti o sebi samoj?

Odgovor na prvo pitanje Gödel je našao u *kodiranju*. Kodirao je jezik aritmetike L tako da, preko kodova, rečenice aritmetike govore o rečenicama aritmetike, dokazima, skupu aksioma, dokazivosti neke rečenice iz nekog skupa aksioma, itd. Sve to uspio je napraviti zahvaljujući tome da su svi pojmovi vezani za formalnu stranu (jezik, aksiomi, dokazi) izračunljivi, pa u ovom kodiranju prelaze u izračunljive operacije s brojevima koje se mogu izraziti u jeziku aritmetike. Istaknut ćemo samo ono što nam je ovdje najvažnije. Budući da u jeziku aritmetike postoji ime za svaki prirodan broj, preko kodiranja imamo i ime za svaku rečenicu. Neka je za rečenicu φ jezika aritmetike njeno ime u jeziku aritmetike $\lceil\varphi\rceil$. Gödel je za svaki skup aksioma A našao predikat jezika $Prov_A$ koji preko kodiranja izražava dokazivost iz aksioma A. Drugim riječima, tvrdnja aritmetičkog jezika $Prov_A(\lceil\varphi\rceil)$ istinita je upravo onda kad je φ dokaziva iz A :

$$A \vdash \varphi \Leftrightarrow L \vDash Prov_A(\lceil\varphi\rceil)$$

Sada možemo opisati i rečenicu koja izražava konzistentnost skupa aksioma A. Takva rečenica kaže da iz A nije izvediva kontradikcija, pa to može biti npr. rečenica $\neg Prov_A(\lceil 0=1 \text{ i } 0\neq 1\rceil)$.

Danas, kad je u računalima sve kodirano u nule i jedinice, od bankovnih računa pa do slika najmilijih, kodiranje i ne izgleda kao neka posebna ideja. Ali u to doba ono je bilo zaista nešto novo. A tek kodirati matematiku: to je bilo revolucionarno. To je treći ključni sastojak naše priče (osim dijagonalizacije i *Entscheidungsproblema*)

Odgovor na drugo pitanje, nalaženje rečenice koja o sebi govori da je dokaziva, Gödel je našao pomoću dijagonalne metode. Pogledajmo sve jednomjesne predikate jezika aritmetike, npr. „x je paran broj", „x je prost broj",... . Sve njih možemo npr. leksikografski numerirati i tako dobiti listu svih jednomjesnih predikata jezika aritmetike: $P_1(x)$, $P_2(x)$,... . Svaki taj predikat možemo preko kodiranja primijeniti na druge predikate. Na primjer, tvrdnja $P_1(\lceil P_3(x)\rceil)$ kaže da predikat $P_3(x)$ ima svojstvo P_1. Tako dobivamo listu listā:

	$P_1(x)$	$P_2(x)$	$P_3(x)$	
$P_1(x)$	$P_1(\lceil P_1(x)\rceil)$	$P_1(\lceil P_2(x)\rceil)$	$P_1(\lceil P_3(x)\rceil)$	.....
$P_2(x)$	$P_2(\lceil P_1(x)\rceil)$	$P_2(\lceil P_2(x)\rceil)$	$P_2(\lceil P_3(x)\rceil)$	.....
$P_3(x)$	$P_3(\lceil P_1(x)\rceil)$	$P_3(\lceil P_2(x)\rceil)$	$P_3(\lceil P_3(x)\rceil)$	.....
..	.....	.....	.....	.....

Po dijagonali imamo predikate koji su primijenjeni na sebe same:

$$P_1(\lceil P_1(x)\rceil), P_2(\lceil P_2(x)\rceil),\ldots$$

Sad ćemo transformirati dijagonalu tako da ćemo na imena ovih rečenica u jeziku L primijeniti predikat $\neg Prov_A(x)$:

$$\neg Prov_A(\lceil P_1(\lceil P_1(x)\rceil)\rceil), \neg Prov_A(\lceil P_2(\lceil P_2(x)\rceil)\rceil),\ldots$$

Ove rečenice redom govore: nije dokaziva rečenica $\lceil P_1(\lceil P_1(x)\rceil)\rceil$, nije dokaziva rečenica $\lceil P_2(\lceil P_2(x)\rceil)\rceil$,...

Operaciju koja prebacuje predikat $P(x)$ u rečenicu $P(\lceil P(x)\rceil)$, nastalu primjenom predikata

na sebe sama, nazivamo dijagonalizacijom. Gödel je pokazao da, preko kodiranja, i nju možemo izraziti u jeziku aritmetike funkcijskim simbolom koji ćemo nazvati *d*. Tako prethodnu transformiranu listu možemo prevesti u listu:

$$\neg Prov_A\left(d\left(\ulcorner P_1(x)\urcorner\right)\right), \neg Prov_A\left(d\left(\ulcorner P_2(x)\urcorner\right)\right), \ldots$$

Ove rečenice redom govore: nije dokaziva dijagonalizacija rečenice $\ulcorner P_1(x)\urcorner$, nije dokaziva dijagonalizacija rečenice $\ulcorner P_2(x)\urcorner$,... .

Ta je lista dobivena primjenom predikata $\neg Prov_A\left(d(x)\right)$ na ostale predikate. Dakle, taj predikat nalazi se u tablici. Neka je to predikat iz tablice indeksa δ : $P_\delta(x)$. Po dijagonalnoj lemi, član ove liste koji se nalazi na dijagonali nije se izmijenio ovom transformacijom. Ali šta to znači? Kad idemo po toj listi, svaka rečenica u listi govori da nije dokaziva rečenica na dijagonali koja je u istom stupcu: $P_\delta\left(\ulcorner P_1(x)\urcorner\right)$ govori da nije dokaziva rečenica $\left\ulcorner P_1\left(\ulcorner P_1(x)\urcorner\right)\right\urcorner$, $P_\delta\left(\ulcorner P_2(x)\urcorner\right)$ govori da nije dokaziva rečenica $\left\ulcorner P_2\left(\ulcorner P_2(x)\urcorner\right)\right\urcorner$,... . U tablici je strelicom naznačeno ovo referiranje:

	$P_1(x)$	$P_2(x)$	$P_3(x)$	
$P_1(x)$	$P_1(\ulcorner P_1(x)\urcorner)$	$P_1(\ulcorner P_2(x)\urcorner)$	$P_1(\ulcorner P_3(x)\urcorner)$	.....
$P_2(x)$	$P_2(\ulcorner P_1(x)\urcorner)$	$P_2(\ulcorner P_2(x)\urcorner)$	$P_2(\ulcorner P_3(x)\urcorner)$	.....
$P_3(x)$	$P_3(\ulcorner P_1(x)\urcorner)$	$P_3(\ulcorner P_2(x)\urcorner)$	$P_3(\ulcorner P_3(x)\urcorner)$	.....
..	.....	.....	.....	.....
$P_\delta(x)$	$P_\delta(\ulcorner P_1(x)\urcorner)$	$P_\delta(\ulcorner P_2(x)\urcorner)$	$P_\delta(\ulcorner P_3(x)\urcorner)$	$P_\delta(\ulcorner P_\delta(x)\urcorner)$

Vidimo da rečenica na presjeku ovog retka s dijagonalom referira na samu sebe. Dakle, ona o sebi govori da nije dokaziva. To je rečenica $\neg Prov_A\left(d\left(\left\ulcorner \neg Prov_A\left(d(x)\right)\right\urcorner\right)\right)$, tzv. Gödelova rečenica.

Nas je zanimalo da nađemo rečenicu koja o sebi kaže da nije dokaziva. No, na isti ovaj način mogli bismo za bilo koji predikat jezika *P* naći rečenicu koja o sebi govori da za nju vrijedi *P*. Samo bismo u prethodnom razmatranju zamijenili $\neg Prov_A$ sa *P*.

Da malo približimo ovaj neobičan način dijagonalnog samoreferiranja, pokazat ćemo kako ga možemo dobiti u prirodnom jeziku. Npr. dijagonalizacija (skraćeno dijag.) rečenice *Kurt čita x* je rečenica *Kurt čita „Kurt čita x”*. Sad ćemo „popravljati” referiranje dok ne dobijemo rečenicu koja referira na sebe, tj. rečenicu koja kaže da Kurt čita baš tu rečenicu. Radi preglednosti, strelicom ćemo označavati što Kurt čita i u zagrade napisati je li to samoreferiranje:

Kurt čita „Kurt čita *x*” → Kurt čita *x* (NE)

Kurt čita dijag. od „Kurt čita *x*” → Kurt čita „Kurt čita *x*” (NE)

Kurt čita dijag. od „Kurt čita dijag. od *x*” → Kurt čita dijag. od „Kurt čita dijag. od *x*” (DA!)

Originalni Gödelovi teoremi puno su dalekosežniji od semantičke verzije koju smo mi naveli. Oni uopće ne spominju semantički pojam istine, već umjesto uvjeta da aksiomi budu istiniti koriste slabiji uvjet – da su aksiomi konzistentni. O toj općenitijoj verziji i dokazu možete saznati u [9]. Ovdje ćemo samo navesti kakve su posljedice ovih teorema za naše znanje.

Sve naše imalo ambicioznije teorije (za koje obično ne znamo jesu li istinite) sadrže Peanovu aritmetiku (uvijek nešto brojimo), bilo da su matematičke (teorija brojeva, Euklidova geometrija, teorija skupova,...), bilo fizikalne (klasična mehanika, kvantna mehanika,...), ili

pak iz nekog drugog područja. Tako su na njih primjenjivi Gödelovi teoremi: one su nužno nepotpune i ne možemo dokazati njihovu konzistentnost sredstvima u čiju smo ispravnost sigurni. Stalno zahtijevaju kreativan akt upotpunjena i spremnost suočavanja s njihovom nekonzistentnošću. S obzirom da računalima možemo predati samo ovaj formalizirani dio razmišljanja koji je nepotpun i nesiguran, Gödelovi teoremi ujedno postavljaju i granice na računalno simuliranje ljudskog razmišljanja.

V. Turing rješava *Entscheidungsproblem* i „usput" daje matematički opis univerzalnog računala

Alan Turing (1912. – 1954.) u Cambridgeu je u Velikoj Britaniji u proljeće 1935. godine pohađao predavanja Maxa Newmana o osnovama matematike koja su završila dokazom prvog Gödelovog teorema nepotpunosti. Tu je upoznao tri glavna sastojka za svoj rad: *Entscheidungsproblem*, kodiranje i dijagonalnu metodu. Bio je uvjeren da ne postoji algoritam kojim bi se rutinski ispitalo slijedi li logički jedna rečenica iz skupa drugih rečenica. Počeo je razmišljati kako bi to dokazao. To razmišljanje završilo je jednim od ključnih radova matematike i računarstva 20. stoljeća, radom *On Computable Numbers with an Application to the Entscheidungsproblem* [4]. Gledano s današnjeg stajališta, naslov rada nesretno je odabran jer ništa u naslovu ne upućuje na, za budućnost, ključne rezultate: u radu je precizno definiran pojam algoritma i izračunljivosti, napisan je prvi programerski priručnik s bibliotekom pomoćnih potprograma, precizno je opisano univerzalno računalo s uskladištenim programom (napisan prvi interpreter), precizno su opisani prvi algoritamski nerješivi problemi (problem dijagonalnog zaustavljanja je najpoznatiji), i na kraju je riješen Entscheidungsproblem (relacija logičke posljedice nije izračunljiva). Što su to izračunljivi brojevi – za ove rezultate nije ni bitno, pa ih nećemo ni spominjati. U knjizi [1] ne samo da možete pročitati o Turingovom životu, već ta knjiga sadrži i krasnu analizu njegova rada. Mi ćemo ovdje prikazati samo najbitnije elemente.

Dokazati da ne postoji algoritam bitno je drugačiji tip problema od problema nalaženja algoritma kojim će se riješiti neki problem. Da bi se našao algoritam, baš i ne moramo točno znati što je algoritam. Dovoljno je da se na konkretnom primjeru ljudi slože da je to algoritam. Tako svi znamo za algoritam zbrajanja brojeva, za Euklidov algoritam, za algoritam sortiranja podataka, itd., a da možda i ne znamo precizno reći što je algoritam. Ali za dokazati da nema algoritma moramo točno precizirati taj pojam. Tako je Turing prvo morao precizirati pojam algoritma. Vidjeli smo da se i u Hilbertovom programu i u Gödelovom radu stalno spominju algoritmi i izračunljivost. Tako je i za preciziranje formalizma matematičke logike bilo važno da se dade precizna definicija algoritma.

Turing je krenuo od analize rada čovjeka koji ima zadatak npr. zbrojiti dva broja, ili pak izvršiti neku drugu rutinsku simboličku proceduru (u to doba ti su se ljudi zvali *computers*). Bile su mu važne dvije stvari u analizi: da rastavi njegov rad na dijelove za koje je posve jasno da su rutinski i da pri tome ništa ne ispusti. Prvi uvjet osigurava da će zaista dobiti nedvojben opis algoritama, a drugi da će taj opis biti potpun, da neki algoritmi neće biti izostavljeni. Njegova analiza bila je toliko precizna da je čak i čovjek koji se može smatrati oličenjem preciznosti, Kurt Gödel, izjavio: „Turing je izvan svake sumnje dao korektnu definiciju mehaničke izračunljivosti". Tom analizom Turing je pokazao da se sve rutinske procedure mogu rastaviti na korake koji su sasvim trivijalni. Svaki korak sastoji se od čitanja simbola na nekom mjestu (prazno mjesto predstavljeno je posebnim simbolom *blank*), upisivanja simbola na to mjesto (ostavljanje istog simbola možemo shvatiti kao upisivanje tog simbola, a brisanje kao upisi-

vanje znaka *blank*) i eventualnog prelaska na drugo mjesto. Što se stvarno u danom koraku učini, ovisi o tome koji je simbol pročitan i kakvo je, po Turingovim riječima, trenutno „stanje uma" (trenutna namjera) izvršitelja postupka. Dakle, ovisno o učitanom simbolu i stanju uma, izvršitelj će na dano mjesto upisati simbol, eventualno se pomaknuti na novo mjesto i prijeći u novo stanje uma. Ilustrirajmo na jednom primjeru jedini sastojak koji nije konkretne prirode – ideju stanja uma. Npr. ako se nalazimo na početku nekog na papiru napisanog nepraznog niza simbola koji ne sadrži simbol praznine i želimo izbrisati zadnji član niza, ideja rješenja je da, krenuvši od početka niza, dođemo do prve praznine nakon niza (jer tako možemo identificirati gdje je kraj niza), vratimo se jedno mjesto ulijevo i tu izbrišemo znak. Dakle, na početku nam je namjera pronaći mjesto koje ćemo izbrisati. S tom namjerom se pomičemo za jedno mjesto udesno. Svaki simbol na koji naiđemo, a koji nije simbol praznine, ponovo ćemo upisati (ostaviti ga netaknutog) i pomaknuti se za jedno mjesto udesno. Tako ćemo napredovati po nizu korak po korak, uvijek s istom namjerom, dok ne naletimo na prazninu i vratimo se jedno mjesto unazad. Tad ćemo promijeniti namjeru. Nova namjera je da ćemo izbrisati znak na mjestu na kojem se nalazimo. I time smo izvršili zadatak. Jasno je da sve konkretne akcije (čitanje simbola, upisivanje simbola, pomicanje na susjedno mjesto) može napraviti i stroj. No, što je sa stanjima uma (namjerama)? Turing je zaključio da izvršenje rutinskih procedura zahtijeva konačan skup stanja uma, pa ih tako možemo strojno simulirati s konačnim brojem unutarnjih stanja stroja. Dakle, svaki takav korak može izvršiti i stroj. Zaključak je da izvršitelja bilo koje procedure možemo zamijeniti odgovarajućim strojem.

Tako je Turing na sljedeći način zamislio računske strojeve koje danas nazivamo *Turingovi strojevi*, i koji obavljaju rutinske poslove umjesto ljudi. Svaki takav stroj

1) ima konačan skup unutarnjih stanja;
2) ima traku koja je sastavljena od ćelija i koja se proteže po volji daleko u oba smjera;
3) ima glavu koja se u svakom koraku nalazi nad jednom ćelijom iz koje može pročitati simbol i u koju može upisati simbol, te se može pomaknuti za jedno mjesto ulijevo ili udesno.

Pojedini Turingov stroj (dizajniran za rješavanje određenog zadatka) je zadan (određen)

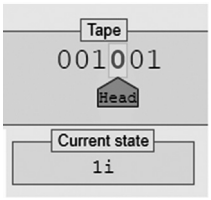
1) konačnim skupom Q simbola unutarnjih stanja među kojima je jedno istaknuto kao početno stanje (ovi se simboli mogu sastojati od više znakova);
2) konačnim skupom S simbola na traci koje stroj može pročitati sa trake ili upisati na traku (među njima je uvijek simbol praznine *blank*);
3) konačnim skupom petorki koje ćemo niže opisati.

Ono što određuje Turingov stroj upravo je njegov skup petorki. Skup petorki možemo shvatiti kao program (*Turingov program*) koji taj stroj realizira (nije bitan redoslijed petorki), a svaku petorku kao naredbu, mada one samo opisuju što stroj radi. Svaka petorka sastavljena je od pet informacija $\left(q_i, R, W, A, q_f\right)$ sa značenjem: „ako si u stanju q_i pročitao na traci simbol R, tada upiši simbol W, učini akciju A i prijeđi u stanje q_f. Pri tome su moguće tri akcije: „otiđi udesno za jedno mjesto" (oznaka „r"), „otiđi ulijevo za jedno mjesto" (oznaka „l") ili „ostani na istom mjestu" (oznaka „*"). Kakva je točno sintaksa naredbe (petorke), nije bitno. Mi ćemo je zapisivati stringom $q_i\, R : W\, A\, q_f$ (između navedenih simbola su praznine). Znak „:" ubačen je samo da se razlikuju parametri početka koraka (početno stanje i učitani simbol) od onog što je učinjeno u koraku (koji je simbol upisan, koja akcija učinjena i u koje je stanje stroj prešao). Skup petorki mora imati jedno važno svojstvo: u skupu ne smiju biti dvije petorke s ista prva dva simbola. Ovime je osiguran jednoznačni odgovor stroja na učitani simbol i

stanje u kojem se nalazi.

Skup petorki determinira rad stroja na danom ulazu na traci i početnom položaju glave. Petorka, kojoj su početni parametri početno stanje i učitani simbol, s ostala tri simbola određuje što će stroj učiniti. Tako će stroj dospjeti u neko eventualno novo stanje u kojem čita novi simbol. Sada petorka koja počinje s tim stanjem i učitanim simbolom određuje što će stroj učiniti itd. Kad se stroj nađe u nekom stanju i pročita simbol za koji ne postoji petorka koja započinje s tim stanjem i učitanim simbolom, tada stroj prestaje s radom.

Stanje računa u nekom koraku posve je opisano stanjem trake (zapisom koji je trenutno na traci), trenutnim unutarnjim stanjem stroja i položajem glave na traci. Stanje možemo slikovito prikazati kao na slici desno.



Lijevo i desno od zapisanog stringa su praznine. Ova je slika preuzeta s web stranice http://morphett.info/turing/turing.html#SyntaxInfo na kojoj se nalazi simulator Turingova stroja. Tu možete napisati program koji treba izvršavati Turingov stroj i pratiti njegovo izvršenje za dani input, početno unutarnje stanje i početni položaj glave. Koriste se iste oznake za petorke kao i u ovom članku, osim što se ne piše znak „:".

Na primjer, Turingov stroj koji treba izbrisati zadnji znak u stringu sastavljenom od slova *A* i *B*, s tim da mu je početni položaj glave na prvom znaku stringa a početno stanje *p*, određen je sljedećim skupom petorki (redoslijed petorki nije bitan, "_" je *blank* simbol):

p A : A r p
p B : B r p
p _ : _ l b
b A : _ * b
b B : _ * b

Prve tri naredbe kažu stroju da napreduje udesno dok ne naiđe na prazninu, kad se treba vratiti jedno mjesto ulijevo (namjera *p*). Preostale mu dvije kažu da upiše prazninu (namjera *b*).

Da bi pokazao da je njegova analiza ispravna, da ovi primitivni strojevi (koji naizgled ništa značajno ne mogu izračunati) uistinu mogu izračunati sve što se može izračunati, Turing je u radu isprogramirao niz sve težih algoritama. Pri tome je razvio ideju potprograma, shvativši njihovu važnost, te napravio cijelu malu biblioteku pomoćnih programa. Tako Turinga možemo shvatiti i kao prvog programera u modernom smislu te riječi.

Sad je mogao precizno definirati da je izračunljiva funkcija ona funkcija za koju postoji Turingov stroj koji je računa – za dane argumente izračuna vrijednost funkcije (ako funkcija na nekom ulazu nije definirana, tada stroj nikad ne staje). Također, neka relacija je izračunljiva ako za nju postoji Turingov stroj koji može za sve objekte ispitati jesu li u toj relaciji ili nisu. Relacija je pak poluizračunljiva ako postoji Turingov stroj koji za sve ulaze može utvrditi da jesu u relaciji baš onda kad jesu, a nikad ne staje kad nisu. Možemo definirati i što je to algoritam – svaki Turingov program (konačni skup petorki). Ili možda manje "službeno" – svaka uputa koja se može prevesti u Turingov program.

A onda je Turing došao na ideju koja je promijenila budućnost. Ma koji Turingov program napisali, znamo ga primijeniti na bilo koji ulaz na traci. *Znači da je i primjena programa na ulaz rutinska stvar – radimo je po određenom algoritmu. Dakle, ako je Turingova analiza*

dobra, i za taj algoritam mora postojati Turingov program (Turingov stroj) koji ga realizira! I Turing ga je u radu zaista i napisao. Napisao je program (opisao stroj) koji na ulaz uzima bilo koji program i ulaz u taj program, a na izlaz daje izlaz tog programa na danom ulazu. Ako taj njegov rezultat gledamo softverski, to je prvi napisani *interpreter*. Ako ga gledamo hardverski, to je moderno računalo: *univerzalni stroj s uskladištenim programom*. Ovdje treba napomenuti da je univerzalnost također bitna komponenta Turingova rada. Turing je opisao sve moguće algoritme, a ne samo one koji rade s brojevima. Njegovi strojevi koriste bilo kakve simbole, a ne samo znamenke. Sve do EDVAC projekta strojevi su bili građeni uglavnom za rješavanje numeričkih problema i nitko nije o njima razmišljao kao o univerzalnim strojevima, osim jednog čovjeka, matematičara Charles Babbagea (1791. – 1871.). On je, 100 godina prije, zamislio analitički stroj kao univerzalan stroj. Samo program ne bi bio uskladišten, već bi se izvana unosio na bušenim karticama. No, tad je još bilo prerano za takav stroj. Kad je pisao svoj rad, Turing nije znao za Babbageove ideje. Sam *Entscheidungsproblem* prirodno je vodio univerzalnosti pojma algoritma jer je relacija logičke posljedice relacija između bilo kojih tvrdnji koje mogu govoriti o bilo čemu, ne samo o brojevima.

U pisanju univerzalnog programa Turing je bitno koristio Gödelovu ideju kodiranja. Univerzalni program prima na ulaz ostale programe u kodiranom obliku. Kodovi koje je Turing koristio nisu bili brojevi (nizovi znamenki), već određeni nizovi simbola, ali sve je vrlo lako prevodivo u brojeve.

Turing je neizračunljivost problema dijagonalnog zaustavljanja riješio na način na koji smo ilustrirali Cantorovu dijagonalnu metodu. Ovdje ćemo ga također riješiti dijagonalnom metodom, ali direktnije. Gledat ćemo sve Turingove strojeve. Idući preko svih prirodnih brojeva, možemo ispitati (možemo čak napisati Turingov program koji će to učiniti) je li pojedini broj kôd Turingova stroja ili ne. Tako imamo prvi kôd, drugi kôd,… Na taj način možemo prebrojiti i programe. Imamo prvi program P_1, drugi program P_2,... Sad ćemo, radi jednostavnosti, uzeti da je kôd prvog programa broj 1, drugog broj 2, itd. Za svaki od ovih programa gledat ćemo na kojim se prirodnim brojevima zaustavlja, a na kojima ne. Ako se zaustavlja, stavit ćemo znak ↓, a ako se ne zaustavlja, znak ↑. Tako imamo listu programa gdje svaki program određuje listu „gore-dolje" strelica (Slika 5.)

	1	2	3	. . .
P_1	↓	↑	↓	
P_2	↑	↑	↓	
P_3	↑	↓	↓	
.				.
.				.
.				.

Slika 5.

Kad bi postojao Program *H* koji bi za svaki program mogao utvrditi zaustavlja li se on na svom kôdu, njegova bi lista izlaza odgovarala dijagonalnoj listi: samo bi umjesto znaka ↓ sad bio odgovor *True* (zaustavlja se), a umjesto znaka ↑ bio bi odgovor *False* (ne zaustavlja se). Taj bismo program (dijagonalnu listu) mogli transformirati tako da ga ukomponiramo u program *TD* koji ga nadopunjuje na sljedeći način: ako program *H* na danom ulazu izbaci odgovor *True*, program ga nadopunjava beskonačnom petljom, a ako izbaci odgovor *False*, program se zaustavlja. Tako je lista programa *TD* upravo suprotna dijagonalnoj listi. Po dijagonalnom teoremu, taj program nije u listi svih programa. Dobili smo kontradikciju. Dakle, ne postoji program koji rješava problem dijagonalnog zaustavljanja.

Već smo napomenuli da dijagonalna metoda može dati ideju kako nešto jednostavnije dokazati. Tako i ovdje možemo napraviti direktan dokaz. Kad bi postojao program *H*, mogli bismo napraviti program *TD* kojem je niže dan pseudokod:

```
Require: x
1: if H(x) then
2:     go to 1
3: end if
```

Stavimo li na ulaz ovog programa njegov vlastiti kôd, ako *H* odgovori da on staje na svom kôdu, on neće stati na svom kôdu, a ako odgovori da neće stati na svom kôdu, on će stati na svom kôdu. Dobili smo kontradikciju. Dakle, program koji rješava pitanje dijagonalnog zaustavljanja ne postoji.

Na kraju rada Turing je riješio *Entscheidungsproblem* tako da je rad Turingova stroja na danom ulazu opisao u odgovarajućem matematičkom jeziku. Pronašao je u tom jeziku rečenicu *A* koja kaže da Turingov stroj koda *n* na ulazu ima broj *n*, i pronašao je rečenicu φ koja kaže da će se generirani račun u jednom koraku zaustaviti. Pokazao je da iz *A* logički slijedi φ upravo onda kad će stroj stati na vlastitom kôdu. Tako, kad bismo imali Turingov program za ispitivanje logičke posljedice, imali bismo i program za rješavanje problema dijagonalnog zaustavljanja. Dakle, ne postoji program koji rješava *Entscheidungsproblem*.

Turingovi rezultati počivaju na pretpostavci da se Turingovim strojevima mogu opisati svi algoritmi. Ma koliko njegova analiza bila precizna, ostaje mogućnost da postoje algoritmi koji se ne mogu opisati Turingovim strojevima. Postojanje takvih algoritama značilo bi da je Turingova definicija izračunljivosti nepotpuna, da njegov univerzalni stroj uopće nije univerzalan, da problem zaustavljanja i *Entscheidungsproblem* možda ipak jesu algoritamski rješivi. Kad je rad bio pri samom kraju, u Cambridge je stigao novi broj *American Journal of Mathematics* u kojem je bio članak *An Unsolvable Problem of Elementary Number Theory* koji je napisao Amerikanac Alonzo Church. U tom su članku definirana dva pojma izračunljivosti, Churchov-Kleeneov λ *račun* i Gödel-Herbrandov *pojam opće rekurzivnosti*. Church je u članku pokazao da su ta dva pojma ekvivalentna i da je, u smislu tih dvaju ekvivalentnih pojmova izračunljivosti, problem spomenut u tom članku algoritamski nerješiv. Church u tom radu ne spominje *Entscheidungsproblem*, ali je jasno da negativno rješenje tog problema slijedi iz Churchovih rezultata, što je Church te iste godine u jednom drugom članku i pokazao. Turingov odgovor je bio da je napisao dodatak svom članku u kojem spominje Churchov rad i skicira dokaz ekvivalentnosti njegovog pojma izračunljivosti s pojmovima izračunljivosti iz Churchova članka.

Tako su 1936. godine „na svjetlo dana" izašla tri ekvivalentna pojma izračunljivosti od kojih je za Turingov bilo najuvjerljivije da odgovara intuitivnom pojmu izračunljivosti. Zanimljivo je da te tri formulacije odgovaraju danas klasičnim programskim paradigmama. Turingova formulacija odgovara imperativnom stilu programiranja u kojem se program sastoji od niza naredbi, i taj stil programiranja poslije je postao dominantan u računarstvu. λ račun je pak najjednostavniji funkcijski jezik koji je postao osnova za sve funkcijske jezike, odnosno za funkcijski stil programiranja: program je opis funkcije koja se iz tog opisa može računati na svakom ulazu. Treća formulacija, Gödel-Herbrandova, pojam opće rekurzivnosti (mada, koliko je poznato autorima, nije imao utjecaja na razvoj računarstva), u osnovi je formulacija koja odgovara logičkom stilu programiranja: program je opis problema, a ulaz u program je upit na koji se iz opisa logičkim izvođenjem dobije odgovor. Tako su se te godine pojavila sva tri klasična modela izračunljivosti, odnosno programerskim rječnikom rečeno, tri klasične programske paradigme. Štoviše, iste je godine Amerikanac Emil Post objavio članak u kojem je opisao pojam izračunljivosti koji se samo u nebitnim detaljima razlikuje od Turingova opisa.

Iste je godine i Stephen Kleene objavio članak u kojem je dan direktan matematički opis klase izračunljivih funkcija. Dodajmo još tome da je Emil Post imao rezultate nepotpunosti prije Gödela i rezultate o postojanju algoritamski nerješivih problema prije Turinga i Churcha, samo ih nikad nije dotjerao do razine objave. On je i tvorac tzv. *Postovih sustava* koji daju još jednu ekvivalentnu formulaciju izračunljivosti. Današnje formalne gramatike u računarstvu samo su specijalan slučaj Postovih sustava.

Razne formulacije pojma izračunljivosti pokazale su se međusobno ekvivalentne. To je dobra osnova da budemo jako sigurni da tako definirana izračunljivost u potpunosti obuhvaća taj pojam. To je sadržaj poznate *Church-Turingove teze*:

Izračunljivost se u intuitivnom smislu podudara s izračunljivošću u smislu ekvivalentnih formulacija pojma izračunljivosti.

Ovu tezu ne možemo dokazati jer bi to zahtijevalo da preciziramo pojam intuitivne izračunljivosti. Tako bismo dobili novu formulaciju pojma izračunljivosti, pa bi samo dokazali ekvivalentnost te formulacije s postojećima. Međutim, tezu možemo oboriti tako da pronađemo funkciju koja je u intuitivnom smislu izračunljiva, a nije izračunljiva u smislu ekvivalentnih formulacija. No takva funkcija još nije pronađena i vjeruje se da i neće biti pronađena. Za svaki novi programski jezik koji pretendira na univerzalnost primjene (npr. *Python*) zahtijeva se da bude Turing potpun: to znači da daje novu ekvivalentnu formulaciju pojma izračunljivosti. Ne želimo da bude slabiji od toga jer bi tada bio neadekvatan, a po Church-Turingovoj tezi to je maksimalno što možemo postići sa svakim programskim jezikom.

VI. John von Neumann daje matematički opis modernog računala

Dolaskom nacizma na vlast mnogi poznati znanstvenici napuštaju Njemačku, Austriju, Italiju i susjedne zemlje. To je doba najvećeg odljeva ljudi slobodnog duha iz kontinentalne Europe. Jedna grupa matematičara smjestila se u Princetonu, između ostalih Kurt Gödel i John von Neumann (1903. – 1957.). Gödel se tamo povezao s grupom koja se bavila izračunljivošću (Church, Kleene). I Turing se od 1936. do 1938. pridružio toj grupi. Jednu od preporuka za stipendiju dao mu je upravo von Neumann. Štoviše, kad je Turing završio svoj boravak u Princetonu, von Neumann mu je ponudio da bude njegov asistent, ali Turing se ipak odlučio vratiti u Englesku. Zabilježena su mnoga svjedočanstva ljudi koji su tada surađivali s von Neumannom koja tvrde da je on bio dobro upoznat s Turingovim radom i da je imao visoko mišljenje o Turingu. Čak je znao suradnike savjetovati da pročitaju Turingov rad, a i spominjao ga je na raznim predavanjima. S obzirom da se jedan dio života i sam bavio zasnivanjem matematike, nedvojbeno se može utvrditi da je von Neumann u potpunosti poznavao Turingov rad i razumio njegovu važnost. I upravo je to ono ključno matematičko naslijeđe koje je von Neumann donio u *EDVAC* projekt i na osnovi postojeće tehnologije postavio koncepcijski model modernog računala – von Neumannovu arhitekturu.

I tako smo se na kraju ovog „ekspres" putovanja kroz povijest vratili na početak naše priče. Ova matematička povijest računarstva, prije svega Turingov utjecaj, dugo je ostala sakrivena uslijed niza okolnosti. Pri tome čestitost –on Neumanna uopće ne dolazi u pitanje. On je i poslije spominjao važnost Turingova rada u krugu svojih suradnika i prijatelja, kao i na predavanjima. Jednostavo, to je bilo turbulentno poslijeratno doba, pogotovo u računarstvu koje je doživjelo pravu revoluciju i nije bilo vremena za „dijeljenje ordena". I sam von Neumann

bio je vrlo dinamična osoba uključena u razne projekte i razna područja matematike. Prošlo je čak nekoliko godina kad je prvi put doznao da je njegov draft postao naširoko poznat. Turingovom zaboravu zasigurno je pridonijela i von Neumannova prerana smrt 1957. godine.

Bez obzira što se nadamo da je sada jasnije koje je matematičko blago von Neumann sa sobom donio u *EDVAC*, njegov udio u stvaranju modernog računala i dalje je ogroman. Navedimo na kraju što je svojim matematičkim naslijeđem i matematičkim pristupom von Neumann pridonio u famoznom draftu koji je u originalnom tisku sadržavao stotinjak stranica:

1) Prije svega to je von Neumannova koncepcijska (logička) organizacija računala koja je dobro poznata pa je ovdje nećemo opisivati. Ona je sinteza Turingove teorijske analize i tadašnjih tehnoloških mogućnosti.
2) Von Neumannov draft je, da se poslužimo riječima Hermana Goldstinea, *„vrhunska analiza i sinteza cjelokupnog razmišljanja koje se odvijalo u EDVAC-u od kraja 1944. do proljeća 1945. godine"* i jedan je od najvažnijih dokumenata iz povijesti računarstva.
3) Von Neumann je uveo slikovitu notaciju, djelomično preuzetu iz rada MacCullocha i Pittsa *A logical calculus of the ideas immanent in nervous activity* iz 1943. godine (jedinog rada koji spominje u draftu), koja se i danas koristi u logičkom opisu elektroničkih dijelova računala.
4) Predložio je kompletan skup naredbi za *EDVAC*. U sljedećem je radu pomoću njih sastavio program za sortiranje niza podataka. Izborom problema želio je ilustrirati univerzalnost stroja jer su prethodnici *EDVAC-a* bili rađeni za rješavanje numeričkih problema.

Literatura:

1. M. Davis, *The Universal Computer*, CRC Press, 2012
2. J. von Neumann, „*First draft of a report on the EDVAC*" Moore School of Electrical Engineering, University of Pennsylvania, 1945
3. H. H Goldstine, *The Computer from Pascal to von Neumann*, Princeton University Press, 1972
4. A. Turing, „*On computable numbers with an application to the entscheidungsproblem*", Proceedings of the London Mathematical Society, 48:230-267, 1936
5. M. Davis, „*Mathematical logic and the origin of modern computers*". In Phillips, E. R., editor, Studies in the History of Mathematics, pages 137-165. MathematicalAssociation of America, 1987
6. M. Vuković, „*Teorija skupova*", predavanja, PMF – Matematički odsjek, 2015
7. M. Vuković, „*Matematička logika i izračunljivost*", predavanja, PMF – Matematički odsjek, 2016
8. K. Gödel, „*Über formal unentscheidbare sätze der principia mathematica und verwandter systeme, i*", Monatshefte für Mathematik und Physik, 38:173-198, 1931
9. M. Vuković, „*Primijenjena logika*", predavanja, PMF – Matematički odsjek, 2011.