Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Metka Selak

Chaitinova konstanta Omega - od definicije do danes

diplomsko delo
na interdisciplinarnem univerzitetnem študiju

Mentor: prof. dr. Borut Robič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Opišite problematiko, ki je privedla do odkritja Chaitinove konstante Omega. Podajte njeno definicijo in opišite njene zanimivejše lastnosti. Predstavite njene posledice v računalništvu in matematiki. Povzemite novejše raziskave v zvezi z Omego in navedite nekaj primerov, ki nakazujejo, kakšna je verjetnost ustavitve pri praktičnih problemih.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Metka Selak sem avtorica diplomskega dela z naslovom:

Chaitinova konstanta Omega - od definicije do danes (angl. *Chaitin's constant Omega - from definition to present*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom prof. dr. Boruta Robiča.
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 3. 6. 2016 Podpis avtorice:

Zahvaljujem se mentorju, prof. dr. Borutu Robiču, za njegovo pomoč, hitre odgovore in vodenje pri pisanju diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
FAS	formal axiomatic sistem	formalni aksiomatski sistem
AIT	algorithmic information theory	algoritemska teorija informacij
TOE	theory of everything	teorija vsega
BF	BrainF*ck	BrainF*ck

Povzetek

Naslov: Chaitinova konstanta Omega - od definicije do danes

V delu je predstavljena problematika, ki je privedla do odkritja Chaitinove konstante Ω, ki je verjetnost ustavitve univerzalnega samoomejenega Turingovega stroja. Podana je njena definicija. Predstavljene so tudi njene zanimivejše lastnosti, med katerimi sta najpomembnejši naključnost in neizračunljivost. Konstanta Ω vnaša naključnost in neizračunljivost v različna področja matematike in računalništva, kar je velika težava za tradicionalen pristop k problemom, ki je še vedno prevladujoč. Predstavljene so tri prevedbe znanih problemov z različnih področij matematike, ki povezujejo bite konstante Ω z rešitvami teh problemov. V nalogi so povzete tudi novejše raziskave v zvezi s konstanto Ω, predvsem na področju računanja točnih začetnih bitov tega v celoti neizračunljivega števila. Na koncu pa je navedenih še nekaj primerov verjetnosti ustavitve pri praktičnih primerih. Predstavitev Chaitinove konstante Ω skuša biti kar se da intuitivna in ob uporabi le najnujnejšega matematičnega aparata še vedno natančna.

Ključne besede: Omega, Chaitinova konstanta, verjetnost ustavitve, problem ustavitve, neizračunljivost, naključnost, Turingov stroj.

Abstract

Title: Chaitin's constant Omega - from definition to present

Problems, that led to the discovery of Chaitin's constant Ω are presented in this work. The constant Ω is the halting probability of the universal self-delimited Turing machine. Its definition is given. Its interesting features are presented, among which randomness and uncomputability are the most important. The constant Ω brings randomness and uncomputability to different areas of mathematics and computer science. This is a great problem for the traditional approach to problem solving. Three transformations of famous problems from different areas of mathematics, which relate bits of Ω to solutions to these problems, are given. Recent researches connected whit the constant Ω, mainly about computing its exact initial bits, are presented. A few examples of the halting probability in practise are listed. The presentation of Chaitin's constant Ω tries to be as intuitive as possible and still exact, while using only the necessary mathematics.

Keywords: Omega, Chaitin's constant, halting probability, halting problem, uncomputability, randomness, Turing machine.

Poglavje 1

Uvod

V diplomskem delu bomo predstavili konstanto Omega, ki jo bomo označevali z grško črko Ω. Leta 1975 jo je odkril Gregory Chaitin in jo zato nekateri imenujejo tudi Chaitinova konstanta. Gre za realno število, ki je hkrati neizračunljivo in naključno. To je bilo zelo pomembno odkritje na področju algoritemske teorije informacij in je zamajalo temelje računalništva in matematike. Podali bomo definicijo konstante in njene pomembnejše lastnosti ter pregledali novejše raziskave na tem področju. Pregledali bomo literaturo s tega področja in naredili povzetek glavnih ugotovitev.

Delo bomo strukturirali na naslednji način. V poglavju 2 bomo najprej predstavili odkritelja konstante, nato pa navedli osnovna zgodovinska dejstva in ideje, ki so pomembno vplivale na odkritje konstante Ω. Po vpeljavi osnovnih pojmov bomo podali njeno definicijo in navedli nekaj njenih pomembnih lastnosti. Izpostavili bomo tudi, zakaj je ravno to realno število tako zanimivo. Za konec tega poglavja pa bomo navedli še dve posledici, ki jih je odkritje prineslo v svet računalništva in matematike. V poglavju 3 bomo predstavili tri probleme z različnih področij matematike in povezavo njihovih rešitev z vrednostjo bitov določene konstante Ω. Dosežke znanstvenikov pri računanju začetnih bitov števila Ω bomo predstavili v poglavju 4. V poglavju 5 bomo predstavili števila imenovana Super Omega, ki so nadgradnja števila Ω. Primere, ki kažejo, kako je z verjetnostjo ustavitve v praksi,

pa bomo podali v poglavju 6. V poglavju 7 bomo za konec podali še nekaj zanimivosti in predloge za nadaljnje branje s tega področja.

Poglavje 2

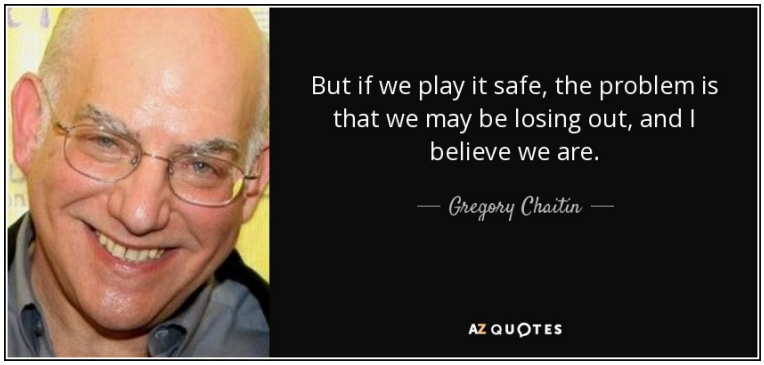
O verjetnosti ustavitve Ω

V tem poglavju bomo predstavili odkritelja[1] konstante Ω in njegovo pot, ki ga je vodila do tega odkritja. Predstavili bomo tudi glavne in najpomembnejše lastnosti konstante Ω in navedli posledice, ki jih je prineslo njeno odkritje. Vsebino bomo večinoma povzeli po knjigi *Meta Maths: The Quest for Omega* [12], kjer je Chaitin natančno predstavil svoje ideje in dokaze. Predvsem v poglavju 6 lahko bralec najde podrobnejše informacije. Druge uporabljene vire bomo posebej citirali.

2.1 Kdo je Gregory Chaitin?

Verjetnost ustavitve Ω nekateri poimenujejo tudi Chaitinova konstanta, saj jo je uvedel Gregory Chaitin. Podatke o njegovem življenju in delu smo povzeli po članku [28]. Gregory John Chaitin je argentinsko-ameriški matematik in računalničar. Rojen je bil leta 1947. Dela na področju algoritemske teorije informacij (AIT) in matematike. Njegova dela so še posebej zaznamovala področje teoretičnega računalništva, natančneje teorije izračunljivosti. Chaitinov najpomembnejši rezultat je ta, da je na svoj način dokazal, da v matematiki obstajajo nerešljivi problemi in da matematika vsebuje naključnost. Prav tako je neodvisno od drugih odkril zahtevnost Kolmogorova.

[1] Matematični platonizem pravi, da vse že obstaja, tudi Ω, le odkriti je treba.



Slika 2.1: Gregory Chaitin in eden njegovih citatov.

Za svoje zasluge je prejel številne nagrade in priznanja, kot so naziv častni doktor znanosti na Univerzi Maine, častni profesor na Univerzi v Buenos Airesu, Leibnizovo medaljo inštituta Wolfram Research in častni naziv doktor filozofije Nacionalne univerze v Cordobi. Bil je raziskovalec pri podjetju IBM, trenutno pa je profesor na Univerzi v Riu de Janeiru. V svojih delih se veliko ukvarja tudi s filozofijo, metamatematiko, metafiziko in metabiologijo. Iz svojih izrekov izlušči filozofske zaključke, s katerimi pa se marsikateri filozof ali logik ne strinja. Dva izmed njegovih slavnih filozofskih zaključkov, ki ju navaja Marcus Chown [17], sta: *"Most of mathematics is true for no particular reason."* in *"Maths is true by accident."* Njegov pogled na svet je razviden tudi iz citata, ki ga prikazuje slika 2.1.

2.2 Pot do konstante Ω

Da bi lahko razumeli sosledje idej, ki je Chaitina pripeljalo do konstante Ω, moramo poznati vsaj nekaj osnov teorije izračunljivosti. V nadaljevanju bomo zato najprej predstavili glavna zgodovinska dejstva povezana s formalizacijo izračunljivosti in povezavo med naključnostjo in izračunljivostjo, ki jo je definiral Chaitin in ki ga je privedla do odkritja konstante Ω.

2.2.1 Formalizacija izračunljivosti

Zgodovinski pregled dogajanja, razvoj idej in vse pomembne dokaze je natančno predstavil Borut Robič v knjigi *The Foundations of Computability Theory*. Izpostavili bomo le nekaj pomembnih mejnikov v zgodovini teorije izračunljivosti, ki smo jih povzeli po omenjeni knjigi, za vse podrobnejše informacije naj bralec pogleda v [25].

Prvi mejnik sega v 1920. leta. Takrat je David Hilbert predstavil svojo idejo, da lahko končen **formalni aksiomatski sistem** (FAS) generira vso matematično resnico. Formalni aksiomatski sistem je natančno določen s svojim simbolnim jezikom, množico aksiomov in množico pravil izpeljevanja. Verjel je, da če bi formalni aksiomatski sistem prejel ustrezno končno množico aksiomov, potem bi lahko povsem mehansko generiral vse izreke. V letih 1920-1928 je Hilbert predstavil svoj program, ki je bil sestavljen iz štirih ciljev, ki bi jih bilo potrebno doseči, da bi uresničil svojo idejo.

Po letu 1930 se je pojavila potreba po formalizaciji pojmov algoritem, računanje in izračunljivost. Znanstveniki Alonzo Church, Kurt Gödel, Jacques Herbrand, Andrey Markov, Alan Turing, Emil Post in drugi so začeli iskati različne računske modele, s katerimi bi te pojme formalizirali. Izkazalo se je, da so vsi njihovi modeli med seboj ekvivalentni, torej, kar lahko izračunamo z uporabo enega izmed njih, lahko tudi z uporabo drugega. Mi se bomo posebej posvetili le Turingovemu stroju, ki ga je uvedel Turing, saj je ta računski model bolj ,,naraven" od drugih. **Turingov stroj** sestoji iz nadzorne enote, ki vsebuje Turingov program, potencialno neskončnega traku in premičnega okna, ki je povezano z nadzorno enoto. Turingova teza o izračunljivosti formalizira algoritem kot Turingov program, računanje kot izvedbo tega programa na Turingovem stroju in izračunljivo funkcijo kot funkcijo, ki jo lahko izračuna Turingov stroj.

Leta 1931 pa je Gödel postavil nov mejnik. Njegova izreka o nepopolnosti sta dokazala, da dva od štirih Hilbertovih ciljev nista dosegljiva. Gödelov prvi izrek govori o tem, da če je formalni aksiomatski sistem neprotisloven in vsebuje vsaj formalno aritmetiko, potem v njegovi teoriji obstajajo trditve o

naravnih številih, ki so resnične, vendar jih ni možno dokazati znotraj tega formalnega aksiomatskega sistema. Drugi izrek pa pove, da se znotraj takega formalnega aksiomatskega sistema ne da dokazati, da je ta neprotisloven. To je bil prvi udarec za Hilbertov program.

Po letu 1940 so začeli odkrivati probleme, za katere so dokazali, da se jih ne da rešiti s Turingovim strojem, torej tudi ne s katerim izmed drugih ekvivalentnih računskih modelov. Turing je odkril enega najbolj poznanih neizračunljivih problemov, to je **problem ustavitve**, ki se glasi: ,,Ali se dani Turingov stroj ustavi pri danem vhodu?" To je bil še drugi udarec za Hilbertov program.

Posledica odkritij Gödla in Turinga je ta, da ne obstaja formalni aksiomatski sistem s končnim številom aksiomov, ki bi lahko generiral vso matematično resnico. Ravno to pa je že eden od zaključkov, ki so tesno povezani s konstanto Ω. Namreč, Chaitin je neobstoj takega formalnega aksiomatskega sistema dokazal na povsem drugačen način, z drugačnim pristopom. Osredotočil se je na pojem naključnosti, ki ga je poskušal formalizirati.

2.2.2 Naključnost in izračunljivost

Prve ideje, ki so Chaitina vodile na poti do konstante Ω, segajo v leto 1686, ko je o njih razmišljal že Gottfried Wilhelm von Leibniz. Chaitin [13] je natančno predstavil Leibnizove ideje, mi bomo navedli le najpomembnejše med njimi, ki so neposredno povezane s konstanto Ω.

Leibniz je obravnaval ključno vprašanje o tem, kako lahko razlikujemo svet, ki ga znanost more razložiti, od tistega, ki ga ne more. Postavil je trditev, da je neka stvar naključna, če je njena predstavitev izjemno zapletena. Naključnost je izenačil z zahtevnostjo. Leibniz je te ideje zapisal, vendar jih nikoli ni obravnaval v globino, zato so stoletja ostale pozabljene. Chaitin pa jih je razvijal naprej. Preden jih podrobneje opišemo, si poglejmo dve drugi ideji, ki govorita o naključnosti.

Najprej poglejmo en primer matematičnih dejstev, ki so naključna. Tak je npr. neskončen niz neodvisnih metov poštenega kovanca. Vse znanje, ki ga

imamo iz preteklosti, nam nikakor ne pomaga pri tem, da bi lahko napovedali naslednji izid. Vsak izid je resničen brez razloga oz. zgolj po naključju. Ti izidi so primer **ireducibilnih** (*slov. nestisljivih*) **matematičnih dejstev**, saj jih nikakor ne moremo izpeljati iz aksiomov, niti jih ne moremo prevesti v enostavnejšo oz. bolj prostorsko učinkovito obliko. Gre torej za matematična dejstva, ki se jih ne da poenostaviti.[2]

Eno izmed definicij naključnosti je ponudil tudi Émile Borel, ki se je ukvarjal z naključnostjo realnih števil, podobno kot tudi Chaitin pri konstanti Ω. Naključno realno število je definiral kot normalno število. Dokazal je namreč, da naključno realno število z verjetnostjo 1 ustreza pogoju normalnosti. Pravimo, da je realno število **normalno**, če se vsaka cifra v zapisu števila pojavi z enako pogostostjo v limiti. Število je **n-normalno**, če za vsak k velja, da ima število v n-tiškem zapisu za vsako od n^k možnih skupin zaporednih k cifer enako pogostost v limiti, to je $\frac{1}{n^k}$. Število je normalno, če je n-normalno za vsak n iz množice naravnih števil. Borel je postavil tudi trditev, da se naključnosti ne da natančno definirati.

Chaitin je predstavil strožjo definicijo naključnosti, saj vanjo ne spadata npr. števili π in e, ker sta izračunljivi in zato reducibilni. To pomeni, da obstaja algoritem za izračun števila π, ki zavzame manj prostora kot pa sama predstavitev števila π (podobno velja tudi za e). Postavil je trditev, da je **naključno** to, kar ni izračunljivo, se pravi je ireducibilno. Zahtevnost podatka je meril z dolžino njegovega dvojiškega zapisa. Na ta način je lahko med seboj primerjal zahtevnost teorije in zahtevnost informacije, ki jo teorija opisuje. Po Chaitinovi definiciji je naključno tako matematično dejstvo, ki ga ne izračuna noben program, ki je prostorsko učinkovitejši od dvojiške predstavitve tega dejstva. Prostorska zahtevnost programa P je dolžina njegovega dvojiškega zapisa. Označimo jo s $|P|$.

Eden glavnih dosežkov Chaitina je ta, da je na neodvisen način dokazal obstoj neizračunljivih problemov. Dokazal je namreč, da znotraj končnega

[2] ,,Ireducibilnost" nima nobene povezave s ,,prevedljivostjo", ki se prav tako pojavlja v računski zahtevnosti.

formalnega aksiomatskega sistema ni mogoče dokazati, da je neki računalniški program ,,eleganten", torej, da krajši program ne more vrniti enakega izhoda oz. rezultata. **Eleganten program** je optimalen stisk svojega izhoda. To pomeni, da je dvojiški zapis takega programa najmanjša možna predstavitev informacije, ki jo nosi rezultat programa. Dokaz poteka na naslednji način. Vzemimo formalni aksiomatski sistem, ki generira vse možne izreke. Pripadajoči generator izrekov P naj bo najmanjši možen, torej tak, ki ima ravno prostorsko zahtevnost generirane teorije, torej ne vsebuje nobene redundance. Moč formalnega aksiomatskega sistema (število bitov informacije, ki jo vsebuje) naj bo definirana kot število bitov informacije, ki jo vsebuje teorija. Uporabimo dokaz s protislovjem. P naj generira vse izreke formalnega aksiomatskega sistema, dokler ne najde dokaza, da $\exists Q : Q\ eleganten \wedge |Q| > |P|$. Če P najde Q, ga požene in vrne njegov izhod kot svojega. Tu pa se pojavi protislovje: P je prekratek, da bi lahko vrnil enak izhod kot Q, saj je Q po predpostavki eleganten. Torej ostane samo ta možnost, da P nikoli ne najde takega Q.

Iz dokaza vidimo, da z uporabo končnega formalnega aksiomatskega sistema ne moremo dokazati, da je program Q eleganten, če je večji od programa P. V danem formalnem aksiomatskem sistemu lahko vedno dokažemo elegantnost le za končno mnogo programov. Če je program daljši, kot je prostorska zahtevnost formalnega aksiomatskega sistema, potem v tem formalnem aksiomatskem sistemu ne moremo dokazati, da je program eleganten. Takojšnja posledica te ugotovitve je, da se algoritmično ne da razpoznati več kot končno mnogo elegantnih programov. Chaitin je dokazal, da iz tega sledi tudi to, da ne obstaja algoritem, ki bi rešil problem ustavitve. To je dokazal s prevedbo. Če bi obstajal algoritem za problem ustavitve, bi ga lahko uporabili za iskanje vseh elegantnih programov. To bi storili tako, da bi generirali vse programe in pri vsakem preverili, ali se ustavi. Ustavljive programe bi potem pognali in primerjali njihove izhode. Za vsak možen izhod bi obdržali samo prvega, ki vrne tak izhod. Če bi programe pregledovali po naraščajoči dolžini, bi na ta način dobili natančno vse elegantne programe.

Ta dva dokaza sta postavila osnovo za definicijo konstante Ω in pokazala, da v matematiki obstaja naključnost in zato ne more obstajati **teorija vsega** (angl. Theory of Everything), to je končna množica aksiomov in pravil, iz katere bi bilo možno izpeljati vsako matematično resnico.

2.3 Definicija konstante Ω

Pri obravnavi ustavitve programov privzemimo, da je vhodni podatek integriran v sam program. To nam dovoljuje izrek o parametrizaciji [25]. Za definicijo konstante Ω je potrebno na problem ustavitve pogledati drugače. Namesto da bi se vprašali, ali se dani program ustavi, raje poglejmo množico vseh možnih programov in naključno izberimo enega. Potem pa se vprašajmo: Kakšna je verjetnost, da se ta naključno izbrani program ustavi? To verjetnost Chaitin poimenuje Ω ali **verjetnost ustavitve**, izrazimo pa jo z realnim številom, ki ima neskončno dolg dvojiški zapis.

Naključno izbran program P sestavimo tako, da vsak bit generiramo z neodvisnim metom poštenega kovanca. Program P mora biti **samoomejen** (*angl. self-delimited*)(tj. noben drug veljaven program, zapisan v dvojiškem zapisu, se ne sme začeti z danim programom). Verjetnost, da na ta način izberemo prav program P, je $\frac{1}{2^{|P|}}$, kjer je $|P|$ dolžina programa P.

Definicija 1 *Ω je verjetnost, da se naključno izbran samoomejen program P ustavi. Dobimo jo tako, da naredimo vsoto verjetnosti naključnega izbora programa P, po vseh ustavljivih programih P.*

$$\Omega = \sum_{P \ se \ ustavi} 2^{-|P|} \tag{2.1}$$

Ključnega pomena pri enačbi (2.1) je to, da so P samoomejeni programi, sicer bi morali konstanto Ω definirati za vsako posamezno velikost programa posebej. Take programe se da enolično dekodirati in zanje velja *Kraftova neenakost*[3], ki zagotovi, da neskončna vsota (2.1) konvergira in je znotraj intervala $[0, 1]$.

[3]Definicija Kraftove neenakosti [30]: Naj bodo znaki abecede $S = \{s_1, s_2, ..., s_n\}$ zako-

2.4 Lastnosti konstante Ω

V tem razdelku si bomo pogledali nekaj lastnosti števila Ω, ki so še posebej zanimive in pomembne za matematiko in računalništvo. Konkretna vrednost števila Ω je odvisna od programskega jezika, ki ga uporabljamo. To sledi iz definicije konstante Ω, saj se verjetnost, da naključno izberemo program P, razlikuje glede na programski jezik, v katerem je napisan P. Vendar pa lastnosti veljajo za vse konstante Ω, ne glede na izbiro jezika.

2.4.1 Število Ω je c.e. realno število

Definicija 2 **N-ti približek števila** Ω, *označen z* Ω_N, *dobimo tako, da poženemo vsak program do velikosti N bitov za N sekund. Vsak k-bitni program ($k \leq N$), ki se ustavi najkasneje v N sekundah, prispeva* $\frac{1}{2^k}$ *k približku* Ω_N. *Zato velja:*

$$\Omega_N = \sum_{|P| \leq N \text{ in } P \text{ se ustavi v največ } N \text{ sekundah}} 2^{-|P|} \tag{2.2}$$

Približne vrednosti monotono rastejo in konvergirajo k Ω. Pravimo, da je Ω **c.e. realno število**[4]. Toda konvergenca je zelo počasna in nikoli ne vemo, kako blizu dejanski vrednosti Ω je približek Ω_N. Raziskovalci si prizadevajo, da bi določili čim več bitov konstante Ω. Več o dosežkih in težavah na tem področju si bomo pogledali v poglavju 4.

2.4.2 Ω je normalno število

Biti števila Ω so enakomerno porazdeljeni, kot bi bili generirani z meti poštenega kovanca. V dvojiškem zapisu se vsaka možna skupina zaporednih cifer 0 in 1 pojavi z enako pogostostjo v limiti.

---

dirani s kodami nad abecedo velikosti r, ki se jih da enolično dekodirati in katerih dolžine so $l_1, l_2, ..., l_n$. Natanko tedaj velja Kraftova neenakost: $\sum_{i=1}^{n} (\frac{1}{r})^{l_i} \leq 1$.

[4]Realno število α je c.e (*angl. computably enumerable, slov. izračunljivo preštevno*), če obstaja izračunljivo, naraščajoče zaporedje racionalnih števil, ki konvergira (ne nujno izračunljivo) k številu α.

2.4.3 Ω je aritmetično število

Število Ω je ekvivalentno po Turingu problemu ustavitve in je zato v razredu Δ_2^0 v aritmetični hierarhiji [25, Theorem 15.3]. Poleg števila Ω obstajajo tudi števila *Super Omega*, ki so v višjih razredih v aritmetični hierarhiji. Več o njih bomo napisali v poglavju 5.

2.4.4 Biti števila Ω so ireducibilna dejstva

Biti števila Ω so ireducibilna matematična dejstva, saj lahko prvih N bitov konstante Ω uporabimo, da rešimo problem ustavitve za vse programe do velikosti N bitov. To pa je ravno N bitov informacije in zato prvih N bitov števila Ω ne vsebuje redundance.

To dokažemo na naslednji način. Računamo približke Ω, dokler ni prvih N bitov pravilnih. Pri tem smo generirali vse ustavljive programe do velikosti N bitov in si zapomnili njihove izhode. Izpišemo nekaj, kar ni vključeno v izhod nobenega izmed njih. Tak izhod ne more biti generiran z uporabo nobenega programa dolžine največ N bitov. Prav zato tudi prvih N bitov števila Ω ne more določiti noben program, dolg največ N bitov. Posledično Ω ustreza Chaitinovemu kriteriju naključnosti oz. ireducibilnosti. Sledi tudi, da lahko končen formalni aksiomatski sistem določi le toliko bitov števila Ω, kot je njegova kompleksnost.

2.4.5 Ω je povezana s celimi števili

Chaitin je pokazal, da je Ω tesno povezana tudi s celimi števili, saj je sestavil eksponentno diofantsko enačbo $L(k, n, x, y, z, \ldots) = R(k, n, x, y, z, \ldots)$, kjer je n parameter, $k, x, y, z, \ldots$ pa so neznanke. Enačba ima neskončno mnogo pozitivnih celoštevilskih rešitev, če je n-ti bit Ω enak 1, in ima samo končno mnogo pozitivnih celoštevilskih rešitev, če je n-ti bit Ω enak 0. Več o tem bomo napisali v poglavju 3.

2.5 Zanimivost števila Ω

Chaitin [12] je dokazal, da je naključno realno število z verjetnostjo 1 ireducibilno. Zakaj je potem ravno Ω tako zanimiva, tako pomembna? Ω je neizračunljivo število, vendar je videti izračunljivo (glejte razdelek 2.4.4). Je ravno čez mejo tega, kar matematika zmore, saj lahko izračunamo vedno boljše približke, vendar nikoli ne vemo, kako blizu dejanske vrednosti so. Ω se pojavlja na raznih področjih matematike, kot so diofantske enačbe (glejte razdelek 2.4.5), algebra, geometrija... Natančneje bomo njeno povezavo s temi področji predstavili v poglavju 3. Ω torej ni samo teoretičen konstrukt, temveč ima velik praktičen pomen. Poleg tega je tesno povezana s Turingovim problemom ustavitve (glejte razdelek 2.4.3). Ta problem je neizračunljiv (natančneje polodločljiv). Tudi verjetnost ustavitve je neizračunljivo število. Vendar prvih N bitov konstante Ω, ki jih lahko izračunamo, da veliko informacije o posameznih problemih ustavitve (glejte razdelek 2.4.4).

2.6 Posledice

Chaitinova odkritja imajo za matematiko in računalništvo pomembne posledice. Za praktično uporabo sta pomembni predvsem dve.

Prva je ta, da je Chaitin pokazal, da je tradicionalni pristop k matematiki v temelju pomanjkljiv. Ireducibilna matematična dejstva ne morejo biti izpeljana iz nobenih načel, preprostejših od dejstev samih. Redukcija na aksiome in sklepanje tu nista možna in zato nimata nobenega smisla. Zaradi ireducibilnih matematičnih dejstev ima matematična resnica ,,neskončno zahtevnost", saj ne obstaja končna množica aksiomov, iz katere bi jo bilo možno vso generirati in zato ne zadostuje noben končen formalni aksiomatski sistem. Treba je neprestano dodajati nove aksiome in nova pravila sklepanja. Za slednje Chaitin predlaga eksperimentalno matematiko, kjer bi nove aksiome pridobivali na podlagi računalniških poskusov. Pokaže tudi, da v resnici noben algoritmični proces ni kreativen, saj je vsa informacija, ki se da izračunati, že implicitno vsebovana v začetnem stanju (tj. v programu in

njegovih vhodnih podatkih). Naključnost je edini vir kreativnosti.

Druga posledica, ki jo Chaitin izpostavi, pa je ta, da je dokazovanje pravilnosti programske kode vnaprej obsojeno na neuspeh. Edini smiseln način ugotavljanja pravilnosti delovanja je eksperimentalen. Tudi razvijanje programske kode mora biti eksperimentalno, vezano na dolgo zaporedje preizkusov in izboljšav. Samo tradicionalna metoda snovanja algoritmov ,,od zgoraj navzdol", ko si celoten program zamislimo prej, kot pričnemo pisati kodo, ne more biti povsem uspešna.

Poglavje 3

Ω na drugih področjih

Konstanta Ω ima posebno velik pomen zato, ker se pojavlja na veliko različnih področjih matematike, ne samo pri algoritemski teoriji informacij, kjer jo je Chaitin definiral. Pojavlja se v problemih geometrije, algebre, teorije števil itn. V nadaljevanju si bomo natančneje pogledali po en problem z vsakega izmed naštetih področij, ki je neposredno povezan z biti števila Ω. Pri vsakem področju bomo predstavili osnovne uporabljene pojme, prevedbo problema ustavitve na problem z danega področja in povezavo bitov konstante Ω z rešitvijo tega problema.

3.1 Ω v teoriji števil

Konstanta Ω se v teoriji števil pojavi pri diofantskih enačbah. Chaitin [12] je predstavil dve različni konstrukciji, ki povezujeta bite števila Ω z diofantskimi enačbami. Prva konstrukcija je njegova, avtorja druge pa sta Toby Ord in Tien D. Kieu. Obe se opirata na definicijo **univerzalne diofantske enačbe** $L(k, n, x, y, z, \ldots) = R(k, n, x, y, z, \ldots)$, ki so jo Yuri Matiyasevich, Martin Davis, Hilary Putnam in Julia Robinson definirali kot rešitev *Hilbertovega 10. problema*[1]. Gre za diofantsko enačbo, ki lahko simulira poljuben izračun

[1]Hilbert je leta 1900 objavil seznam 23 nerešenih matematičnih problemov. Nekateri izmed njih so še vedno nerešeni. Hilbertov 10. problem se glasi: ,, Ali obstaja algori-

(tj. poljubno zaporedje operacij). Tu je k parameter enačbe, $n, x, y, z, \ldots$ pa so neznanke. Parameter k je v bistvu program, $x, y, z, \ldots$ so spremenljivke, n pa je izhod oz. rezultat izračuna. Tisto, kar iščemo, je vrednost neznanke n. Izračun simuliramo na naslednji način: v enačbo $L(k, n) = R(k, n)$ vstavimo vrednost k, to je program, ki ga želimo izvesti. Potem opazujemo vrednosti n, za katere lahko najdemo nabor vrednosti za $x, y, z, \ldots$, pri katerem program k vrne n. Ta enačba je zelo zapletena. Chaitin [12] je v programskem jeziku LISP to enačbo tudi napisal. Enačba je dolga dvesto strani in vsebuje 20.000 neznank. Če izraz k v tem programskem jeziku nima vrednosti, potem tudi enačba nima rešitve. V nasprotnem primeru ima enačba natančno eno rešitev, namreč n, ki je vrednost izraza k. To enačbo lahko uporabimo tudi za simulacijo Turingovega stroja.

Chaitin je leta 1987 uporabil zgornjo univerzalno diofantsko enačbo in predstavil zapleteno eksponentno diofantsko enačbo, katere rešitve določajo bite števila Ω. Vzemimo Turingov stroj $T1$, ki izračuna Ω_k, to je k-ti približek števila Ω, kot je definirano v enačbi (2.2). $T1$ se takoj ustavi, če je n-ti bit Ω_k enak 1, sicer pa se vrti v neskončni zanki. Za dovolj velike vrednosti števila k se vrednost n-tega bita ustali pri pravilni vrednosti, zato za dovolj velike k velja, da se Turingov stroj $T1$ ustavi natanko tedaj, ko je n-ti bit Ω enak 1. Če stroj $T1$ simuliramo z univerzalno diofantsko enačbo, dobimo eksponentno diofantsko enačbo, ki ima natančno eno pozitivno celoštevilsko rešitev, če se $T1$ ustavi, sicer pa nima rešitve. Če torej določimo vrednost n in pustimo, da se k spreminja, potem ima ta ista enačba neskončno mnogo rešitev, če je n-ti bit Ω enak 1, in končno mnogo rešitev, če je n-ti bit Ω enak 0.

Ord in Kieu sta leta 2003 uporabila drugačen pristop. Za izhodišče sta vzela Turingov stroj $T2$, ki se ustavi natanko tedaj, ko je $k > 0$ in $2^n \times \Omega_j > k$ za nek $j = 1, 2, 3, ...$ Stroj $T2$ se torej ustavi natanko tedaj, ko $2^n \times \Omega_j > k > 0$. Če stroj $T2$ simuliramo z univerzalno diofantsko enačbo,

tem, ki odloči, ali je diofantska enačba rešljiva?" Torej ali obstaja algoritem, ki odloči ali neznankam v enačbi lahko določimo celoštevilske vrednosti tako, da zadoščajo enačbi?

dobimo natančno eno celoštevilsko rešitev, če se stroj ustavi, sicer ne dobimo rešitve. Če določimo vrednost n, je enačba $L(k,n) = R(k,n)$ rešljiva natančno za $k = 1, 2, 3, \ldots, \lfloor 2^n \times \Omega \rfloor$. Zato ima $L(n) = R(n)$ natančno $\lfloor 2^n \times \Omega \rfloor$ rešitev, kar je ravno celi del števila, ki ga dobimo, če dvojiški zapis števila Ω premaknemo v levo za n bitov. Skrajno desni bit števila $\lfloor 2^n \times \Omega \rfloor$ je n-ti bit Ω. Če določimo n in spreminjamo k, vrne enačba $L(k,n,x,y,z,\ldots) = R(k,n,x,y,z,\ldots)$ liho število rešitev, če je n-ti bit Ω enak 1, in sodo število rešitev, če je n-ti bit Ω enak 0.

3.2 Ω v algebri

Chaitin [14] je predstavil tudi algebraično karakterizacijo konstante Ω. To je storil tako, da je prevedel problem ,,Ali se Turing-Postov program ustavi?" in posamezne bite konstante Ω v **besedni problem** (*angl. word problem*), iz tega pa dobil enačbo nad besedami s parametrom k. Ta enačba vrne bite v dvojiškem zapisu števila Ω.

Uporabil je **Turing-Postov programski jezik**, ki ga je natančno predstavil v članku [14]. Računski model je Turingov stroj z nekoliko prilagojenim opisovanjem izrazov. Ta programski jezik uporablja abecedo A z α tračnimi znaki, od katerih je eden ,,prazen znak" B (angl. blank), ki označuje prazno polje na traku. Množica ukazov U vsebuje $2\alpha + 3$ ukazov.

$$\begin{aligned} U \;=\; & \{\texttt{move right},\ \texttt{move left},\ \texttt{stop}\} \cup \\ & \cup\, \{\texttt{write i} \mid i \in A\} \cup \\ & \cup\, \{\texttt{go to ukaz k if i} \mid i \in A,\ k \textit{ je zaporedna št. ukaza v programu}\} \end{aligned}$$

Turing-Postov program je zaporedje ukazov iz množice U. Da se program lahko začne izvajati, potrebuje začetni vhod, zapisan na trak Turingovega stroja.

Besedni problem je definiran z abecedo, ki je končna množica znakov, in s končnim številom enačb nad besedami. Besede so nizi znakov nad to abecedo. Besedni problem se nato glasi: ,,Ali lahko dve različni dani besedi nad dano abecedo pretvorimo iz ene v drugo z uporabo danih enačb?"

Za prevedbo problema ustavitve v besedni problem je Chaitin ukaze Turing-Postovega programa prevedel v enačbe nad besedami. Vzel je Turing-Postov program P. Naj bo n število ukazov v P . Uporabil je tračno abecedo, ki vsebuje $\alpha + n + 2$ znakov: poleg znakov, ki so bili v abecedi A, so dodani še znaki $h, q_1, q_2, \ldots q_n, q_{n+1}$. Tu je h poseben znak za ločevanje, $q_1, \cdots, q_{n+1}$ pa so znaki, ki označujejo stanja Turingovega stroja. Dejstvo, da je na traku zapisano neko stanje, se zakodira v določeno besedo nad tračno abecedo. Beseda se konstruira tako, da se prepiše niz znakov, različnih od B, ki je trenutno na traku, na začetek in na konec pa se kot mejnik postavi znak h. Če se bo izvršil i-ti korak programa P, se vstavi znak q_i levo od znaka, ki se je ravnokar prebral. Ukazi določajo enačbe nad tako dobljenimi besedami. Na ta način se izvajanje programa lahko prevede v množico enačb nad besedami. Za boljšo predstavo lahko bralec pogleda primere navedene v [14]. Posebej izpostavimo ukaz `stop`. Ta ukaz se prevede v enačbo $q_n = q_{n+1}$. Če se torej v besednem problemu pojavi znak q_{n+1} pomeni, da se je program ustavil.

Na opisani način lahko prevedemo problem ustavitve v besedni problem. Naj program P začne brati skrajno levi znak niza v. Temu stanju ustreza beseda hq_1vh. Če se P ustavi, se da iz enačb, ki ustrezajo programu P, izpeljati enačbo $hq_1vh = hq_{n+1}h$. To dokažemo tako, da sledimo izvajanju računanja korak za korakom. Po drugi strani pa želimo dokazati tudi to, da če se P nikoli ne ustavi, potem take enačbe ne moremo izpeljati. (V dokazu uporabimo naslednje: vsakič, ko uporabimo enačbo, ki ustreza ukazu, ,,premaknemo" izračun za korak naprej ali pa nazaj, torej naredimo nov korak ali pa zadnji korak razveljavimo.) Dokaz te trditve je Chaitin objavil v članku [14].

Prevedba posameznih bitov konstante Ω v besedni problem poteka preko Turing-Postovega programa. Naj bo Ω_j j-ti približek Ω, kot je definirano v enačbi (2.2). Naj bo a^j niz j črk a in b^k niz k črk b. Vzemimo Turing-Postov program, ki začne brati skrajno levi znak besede $a^j b^k$, in ki se ustavi, če je k-ti bit Ω_j enak 1 ter se ne ustavi, če je ta bit enak 0. Program izračuna k-ti bit Ω_j in se takoj ustavi ali pa se večno vrti v zanki, odvisno od vrednosti

tega bita. Če po zgornji metodi ta program prevedemo v besedni problem, potem velja, da bo enačba $hq_1a^jb^kh = hq_{n+1}h$ izpeljiva natanko tedaj, ko je k-ti bit Ω_j enak 1.

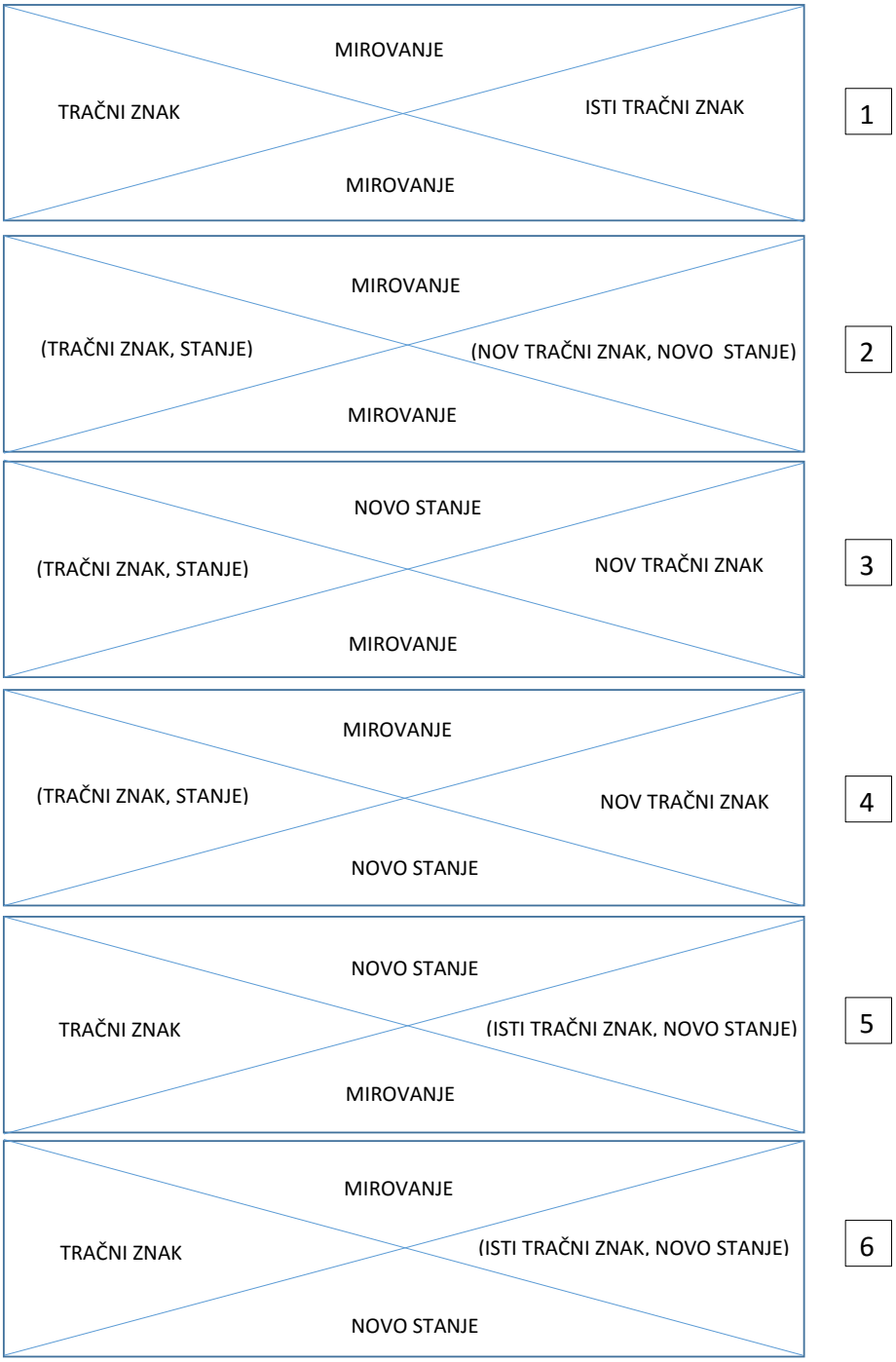
Algebraična karakterizacija bitov števila Ω je torej naslednja. Če določimo vrednost parametra k in spreminjamo j, potem bo množica vseh besed oblike $hq_1a^jb^kh$, ki so enake besedi $hq_{n+1}h$ neskončna, če je k-ti bit Ω enak 1, in končna, če je ta bit enak 0.

3.3 Ω v geometriji

V geometriji se konstanta Ω pojavi pri neskončnem tlakovanju z **Wangovimi ploščicami**. Chaitin [15] je predstavil prevedbo problema ustavitve na problem tlakovanja s parametrom i. Rešitev tega problema določi vrednost i-tega bita v dvojiškem zapisu števila Ω.

Wangovo tlakovanje sestoji iz Wangovih ploščic, ki so enako veliki kvadrati, razdeljeni v 4 trikotnike z diagonalama. Rezine imenujemo *levi, desni, zgornji in spodnji kvadrant.* Vsak kvadrant je določene barve. Podana je končna množica W Wangovih ploščic, kjer ima vsaka ploščica posebno kombinacijo barv svojih kvadrantov. Pravila tlakovanja z Wangovimi ploščicami dovoljujejo, da uporabimo več kopij iste ploščice, ne smemo pa jih vrteti ali zrcaliti. Položene skupaj morajo tvoriti pravilno kvadratno mrežo in sosednje ploščice morajo biti pri ujemajočih se robovih enake barve.

Najprej poglejmo, kako bi z uporabo Wangovega tlakovanja simulirali Turingov stroj z enim dvosmernim neskončnim trakom in eno bralno-pisalno glavo. Vsebino traku Turingovega stroja predstavimo s stolpcem Wangovih ploščic na desni polravnini. Začetno stanje na traku je predstavljeno s skrajno levim stolpcem ploščic. Korak izračuna ustreza premiku v desno na polravnini in je zato tlakovanje **prostorsko-časovni diagram** izračuna. Pri simulaciji privzemimo, da se stanje premika skupaj z bralno-pisalno glavo, torej se premika po prostorsko-časovnem diagramu po sledi premikov bralno-pisalne glave. Da lahko določimo množico W_T Wangovih ploščic, ki ustreza



Slika 3.1: Wangove ploščice za simulacijo Turingovega stroja

Turingovemu stroju T, moramo najprej definirati množico barv. Ta mora biti sposobna predstaviti naslednje: mirovanje, tračne znake, stanja in pare (tračni znak, stanje). Vsaki možni vrednosti naj ustreza posebna barva. Naslednji korak pa je ta, da določimo izbiro barv za vsako ploščico v množici W_T. Glede na dogajanje med izračunom je možnih 6 posebnih vrst barvanj ploščic. Prikazuje jih slika 3.1. Ploščice vrste 1 zagotovijo, da se simulirajo neaktivne celice traku, tj. tiste, ki so oddaljene od bralno-pisalne glave in se v danem koraku z njimi nič ne dogaja (mirovanje). Druge vrste ploščic rabimo za simuliranje aktivnih celic. Najpreprostejši primer je, kadar se bralno-pisalna glava ne premakne, spremenita pa se tračni znak in/ali notranje stanje. To možnost simuliramo s ploščicami vrste 2. Če pa se bralno-pisalna glava premakne, potem aktivna celica postane neaktivna. To simuliramo s ploščicami vrst 3 in 4, odvisno od tega, v katero smer se premakne glava. Zadnja možnost pa je, da neaktivna celica postane aktivna. To možnost simuliramo s ploščicami vrst 5 in 6, spet v odvisnosti od tega, s katere strani se je na dano celico premaknila bralno-pisalna glava.

Pokazali smo, na kakšen način lahko simuliramo Turingov stroj z uporabo Wangovih ploščic. Zdaj lahko konstruiramo predstavitev bitov števila Ω. Najprej konstruiramo končno množico W Wangovih ploščic, ki ustreza naslednjemu Turingovemu stroju. Če ima v začetnem stanju na traku samo i enic in bralno-pisalno glavo na skrajno levi enici, izračuna i-ti bit Ω_k za $k = 1, 2, 3, \ldots$ To je neskončen izračun in posebna stanja s_1 ustrezajo temu, da stroj izpiše 1, torej je i-ti bit Ω_k enak 1. Vse ploščice, ki vsebujejo stanje s_1, so zajete v podmnožico $D \subset W$. Če je i-ti bit Ω enak 1, potem bo tudi i-ti bit Ω_k enak 1, za vse dovolj velike k. V tem primeru bo Turingov stroj obiskal neskončno mnogo stanj s_1 in zato bo v množici D neskončno mnogo ploščic. Obratno velja tudi, da če je i-ti bit Ω enak 0, potem bo tudi i-ti bit Ω_k enak 0, za vse dovolj velike k. Turingov stroj bo obiskal le končno mnogo stanj s_1, v tlakovanju, ki ustreza prostorsko-časovnem diagramu tega izračuna, pa bo v množici D le končno mnogo ploščic.

Podobno, kot smo v razdelku 3.1 videli dve možni konstrukciji, obstaja

tudi v tem primeru konstrukcija, ki uporabi pristop, ki sta ga predstavila Ord in Kieu. V tem primeru se v množici D vedno znajde le končno mnogo ploščic, toda njihovo število je odvisno od vrednosti i-tega bita Ω. Število ploščic v D je sodo, če je i-ti bit enak 0, in liho, če je ta bit enak 1.

Poglavje 4

Računanje bitov konstante Ω

Kot smo že omenili, je Ω pomembna predvsem zato, ker nudi veliko informacije o posameznih problemih ustavitve. Za veliko pomembnih nerešenih matematičnih problemov obstajajo prevedbe na problem ustavitve. Torej bi s poznavanjem rešitev teh problemov lahko rešili veliko odprtih matematičnih vprašanj. Kot smo že videli, je natančna vrednost konstante neizračunljiva, vendar približki, ki jih lahko izračunamo, podajo rešitve problemov ustavitve za programe dolžine največ n bitov, če je prvih n bitov približka točnih. Prav zato se nekateri znanstveniki trudijo, da bi čim natančneje določili vrednost števila Ω, torej, da bi določili čim več začetnih bitov. Cristian S. Calude in Chaitin [4] sta kot primer navedla *Riemannovo hipotezo*[1]. Če bi poznali prvih 7780 bitov števila Ω, bi vedeli, ali je hipoteza pravilna. Charles H. Bennett in Martin Gardner [2] sta zapisala, da že prvih nekaj tisoč bitov, ki jih lahko zapišemo na majhen kos papirja, vsebuje odgovore na več matematičnih vprašanj, kot bi jih lahko zapisali v celotnem vesolju. Kljub temu so Calude, Michael J. Dinneen in Chi-Kou Shu [5] izpostavili, da tudi, če bi znanstvenikom uspelo točno izračunati npr. prvih 10.000 bitov, bi bilo reševanje problemov, ki jih ti biti opisujejo, sicer izračunljivo, vendar stra-

[1]Riemannova hipoteza [31] predvideva, da ima Riemannova zeta funkcija $\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$ ničle na vseh negativnih sodih celih številih in na kompleksnih številih, ki imajo realni del enak $\frac{1}{2}$. Nekateri matematiki so mnenja, da je to najpomembnejši nerešen problem v matematiki.

hotno zahtevno, saj čas, potreben za iskanje ustavljivih programov dolžine manjše od n, raste hitreje kot katerakoli izračunljiva funkcija spremenljivke n.

Računanje približkov števila Ω načeloma ni težko, saj je treba samo generirati vedno več ustavljivih programov. Toda vrednosti bitov, ki jih na ta način pridobimo, niso nujno točne. Če je prvi bit enak 1, potem je gotovo točen; toda če je prvi bit, ki smo ga dobili z računanjem približka, enak 0, potem zaradi možnega prenosa ne vemo, ali je pravilna vrednost 1 ali 0. Enako velja tudi za vse naslednje bite. Samo začetni niz samih enic bi bil zagotovo točen.

V nadaljevanju si bomo pogledali tri različne primere računanja bitov števila Ω. Vrednost te konstante je odvisna od stroja in programskega jezika, ki ga uporabimo pri izračunu, zato ni presenetljivo, da se izračunane vrednosti med seboj razlikujejo.

4.1 Biti konstante Ω pri Solovayevih strojih

Calude, Dinneen in Shu [5] so izpostavili naslednje pomembne lastnosti števila Ω. Vsako c.e. realno število, ki je naključno, je verjetnost ustavitve Ω nekega univerzalnega stroja. Števila Ω so verjetnosti ustavitve **univerzalnih Chaitinovih strojev** (tj. univerzalnih Turingovih strojev, katerih programi so samoomejeni) in vsako število Ω je verjetnost ustavitve neskončne množice takih strojev. Nekateri izmed njih so tudi Solovayevi stroji. **Solovayev stroj** je Turingov stroj, katerega univerzalnost je dokazljiva v Peanovi aritmetiki, hkrati pa noben končni formalni aksiomatski sistem ne more določiti več kot začetni blok enic v dvojiškem zapisu njegove verjetnosti ustavitve Ω. Calude [7] je dokazal, da če je končni formalni aksiomatski sistem aritmetično zdrav, potem je vsako naključno c.e. realno število verjetnost ustavitve nekega Solovayevega stroja. Velja naslednji Solovayev izrek [5]: *V končnem formalnem aksiomatskem sistemu, ki je aritmetično zdrav, je vsako naključno c.e. realno število* $\alpha \in (0, \frac{1}{2})$ *verjetnost ustavitve Solovayevega stroja, ki ne more*

določiti niti enega samega bita števila Ω; to pa ne velja za nobeno tako število $\alpha \in (\frac{1}{2}, 1)$. Toda ista Ω, ki ji pri Solovayevih strojih lahko določimo samo začetni niz enic, je lahko definirana tudi kot verjetnost ustavitve univerzalnega Chaitinovega stroja, ki ni Solovayev stroj. Zato lahko končni formalni aksiomatski sistem, ki ima na voljo neki drug stroj, izračuna več bitov te konstante Ω, vendar vedno samo končno mnogo.

4.2 Biti konstante Ω pri registrskih strojih

Calude, Dinneen in Shu so se izračuna približka vrednosti števila Ω lotili z uporabo registrskih strojev in njihovih programov. **Registrski stroj** je stroj, ki ima končno število registrov z enoličnimi naslovi, v katerih lahko hrani različno velika nenegativna cela števila. Do registrov lahko neposredno dostopa. Je ekvivalenten Turingovemu stroju. **Program registrskega stroja** je končen seznam ukazov, z omejitvijo, da se ukaz `STOP` pojavi samo enkrat in še to kot zadnji ukaz na seznamu. Vhodni podatki so podani v obliki dvojiškega niza in neposredno sledijo ukazu `STOP`. Zaradi teh omejitev so programi registrskih strojev samoomejeni. Natančno predstavitev uporabljenih registrskih strojev in njihovih programov lahko bralec najde v [16]. V nadaljevanju bomo predstavili, kako so izračunali prvih 80 bitov konstante Ω za konkreten registrski stroj. Pozneje so odkrili napako in jo popravili ter izračunali prvih 64 bitov konstante Ω.

4.2.1 80 začetnih bitov števila Ω

Leta 2000 so Calude, Dinnen in Shu [8] natančno izračunali prvih 80 bitov števila Ω za dani registrski stroj in njegov programski jezik. Oba so predstavili v [8]. Uporabili so 7-bitno kodiranje. Da so lahko zmanjšali število programov, ki jih je bilo potrebno simulirati, so namesto splošnih programov registrskih strojev uporabili ,,kanonične programe". **Kanonični program** je program registrskega stroja, v katerem se naslovi pojavljajo v naraščajočem numeričnem redu od 0 naprej, nova imena registrov se poja-

vljajo v naraščajočem leksikografskem redu, v programu pa ni vodilnih ali končnih presledkov. Za vsak registrski program obstaja ekvivalenten kanonični program (tj. ima isto domeno in na njej vrača enake izhode). Uporabili so Java interpreter za registrske stroje, ki ga je definiral Chaitin [16], in z njim testirali ustavitev programov registrskega stroja do velikosti 98 bitov. Programi so se izvajali največ 100 korakov. Če se v tem času niso ustavili, so obveljali za neustavljive. Na ta način so dobili vrednosti prvih 98 bitov števila Ω, ki jih prikazuje enačba (4.1).

$$\begin{aligned}\Omega \;=\;& 0,000000000000000000001000000100000010000001000001 \\ & 000001110010011100010100010100000111010111011101001 \end{aligned} \tag{4.1}$$

Takoj se je pojavilo vprašanje, kako lahko vemo, da je ta rezultat točen, torej, da gre za pravilne bite števila Ω za dani registrski stroj. Potrebno je bilo dokazati, da obstaja neki $N > 0$, za katerega velja, da programi daljši od 98 bitov, ne morejo prispevati toliko, da bi se biti števila Ω pred N-tim bitom spremenili. Calude, Dinneen in Shu [8] so dokazali, da to velja za $N = 81$. Izračun je pokazal, da prispevek programov daljših od 98 bitov lahko povzroči dodatno enico kvečjemu pri 83. bitu izračunanega približka števila Ω. Vendar sta 81. in 82. bit enaka ena, zato bi morebiten prenos pri 83. bitu vplival na vrednosti vse do 81. bita. Zaradi tega so zagotovo točne le vrednosti do vključno 80. bita. Podroben izračun je opisan v [8]. Njihov končni rezultat prikazuje enačba (4.2).

$$\begin{aligned}\Omega \;=\;& 0,000000000000000000001000000100000010000001 \\ & 000001000001110010011100010100010100000 \end{aligned} \tag{4.2}$$

Ustrezna vrednost v desetiškem sistemu je $\Omega = 0.0000004806$. To je bil pomemben rezultat, vendar je Chaitin že istega leta opozoril na napako v njihovem delu. Stroj, ki so ga v članku [8] konstruirali, je sicer univerzalen v Turingovem smislu (tj. lahko simulira katerikoli samoomejeni Turingov stroj),

ni pa univerzalen v smislu algoritemske teorije informacij, ker cena simulacije ni omejena z aditivno konstanto. Zato izračunana verjetnost ustavitve ni število Ω, ampak le c.e. realno število, z nekaterimi lastnostmi, ki so blizu naključnosti. To napako so avtorji odpravili in že dve leti pozneje predstavili novo konstrukcijo, kjer so izračunali prvih 64 bitov števila Ω.

4.2.2 64 začetnih bitov števila Ω

Leta 2002 so Calude, Dinneen in Shu [5] objavili nov članek, v katerem so odpravili omenjeno napako. Še vedno so uporabili 7-bitno kodiranje. Postavili pa so strožje pogoje za kanonične programe. Poleg pogojev, ki so jih zahtevali v [8], so dodali tri dodatne zahteve, in sicer to, da so operandi med seboj ločeni samo z enim presledkom, za naslovi ali operandi ni presledka, ukazi pa so ločeni z enim samim presledkom. Kanonične programe so zaradi optimizacije še stisnili tako, da so izbrisali vse naslove, presledke in dvopičja, več operandov so med seboj ločili z vejico in konstante zamenjali z njihovo ASCII numerično vrednostjo. Na ta način stisnjeni programi so še vedno tudi kanonični programi. Uporabili so isti interpreter kot prvič, generirali stisnjene kanonične programe do velikosti 84 bitov in jih izvajali do največ 100 korakov. Če se v tem času niso ustavili, so veljali za neustavljive. Ker so uporabljali stisnjene kanonične programe, so dobili pri posamezni velikosti bistveno več programov kot v prvem primeru in zato tudi bistveno več ustavljivih programov. Npr. pri prvi konstrukciji so našli pri dolžini 84 bitov 1506 ustavljivih programov, pri drugi konstrukciji pa kar 2559837. Točne rezultate za primerjavo lahko bralec najde v [5] in [8]. Tako so dobili vrednost, ki jo prikazuje enačba (4.3).

$$\begin{aligned}\Omega \;=\; & 0,000000100000010000011000100000110100011111100 \\ & 1011101110100001000001111011011011011101 \end{aligned} \tag{4.3}$$

Podobno kot prvič so morali izračunati N, za katerega velja, da programi daljši od 84 bitov ne morejo prispevati toliko, da bi se biti števila Ω pred

N-tim bitom spremenili. Dokazali so, da to velja za $N = 65$. Izračun je pokazal, da prispevek programov daljših od 84 bitov lahko povzroči kvečjemu dodatno enico pri 68. bitu izračunanega približka števila Ω. Vendar je med 65. in 68. bitom niz enic, zato lahko prenos vpliva na vrednosti vse do 65. bita. Zaradi tega so zagotovo točne le vrednosti do vključno 64. bita. Njihov končni rezultat prikazuje enačba (4.4).

$$\begin{aligned}\Omega \;=\; & 0,000001000000100000110001000011010001111110010111011 \\ & 101000010000 \end{aligned} \tag{4.4}$$

Vrednost, ki so jo dobili, je v desetiškem zapisu približno enaka vrednosti v enačbi (4.5).

$$\Omega \approx 0,0157499939956247687_{[10]} \tag{4.5}$$

Izračunali so torej, da je verjetnost ustavitve naključno konstruiranega programa v jeziku, ki so ga uporabljali, približno 1,575 %. Ta rezultat je pomemben, ker je konstanta Ω danega registrskega stroja enaka konstanti Ω neke množice Turingovih strojev. Pomanjkljivost tega izračuna, ki jo je izpostavil Saxberg [26], je v tem, da so izračunali prvih 64 bitov Ω določenega registrskega stroja. Vemo samo, da obstajajo Turingovi stroji, ki imajo enako verjetnost ustavitve, vendar za zdaj še niso poznani.

4.3 Biti konstante Ω pri Turingovem stroju

Calude in Dinneen [9] sta leta 2007 objavila nadgradnjo dela [5]. Izboljšala in poenostavila sta metodo dela ter ga uporabila na kompaktnem samoomejenem Turingovem stroju, ki sta ga konstruirala. Dokazala sta, da je le-ta univerzalen, če računa v bazi 16 ali v bazi 2. Uspelo jima je natančno izračunati začetne bite dveh konstant Ω za ta stroj, in sicer 43 začetnih bitov števila Ω_{16} v bazi 16 in 40 začetnih bitov števila Ω_2 v bazi 2.

Za osnovo sta tudi tokrat vzela registrski stroj, ki ga je konstruiral že Chaitin [16]. Natančno predstavitev konstrukcije si bralec lahko prebere v

[9]. Na tem mestu bomo podali samo pomembne razlike med to in prejšnjo konstrukcijo. Glavna razlika je ta, da sta tokrat namesto 7-bitnega uporabila 4-bitno kodiranje. Podatke sta podala v dveh različnih formatih: pri bazi 16 kot niz 4-bitnih znakov in pri bazi 2 kot navaden dvojiški niz. Program registrskega stroja je bil sestavljen iz končnega niza ukazov iz množice samo petih ukazov, medtem ko je bilo pri prejšnji konstrukciji na voljo 11 ukazov. Ukaz `STOP` je moral biti še vedno zadnji, neposredno pa so mu sledili vhodni podatki. Podala sta tudi dodatne zahteve za kanonične programe. Obdržala sta vseh 6 zahtev od prej in dodala še tri nove. To so: izbris vseh naslovov, ker se implicitno pojavljajo v naraščajočem vrstnem redu, izbris presledkov in dvopičij in ločitev večkratnih operandov z eno samo vejico. Kljub dodatnim zahtevam je še vedno veljalo, da za vsak program registrskega stroja obstaja enoličen kanonični program, ki mu je ekvivalenten. Jezik njunega registrskega stroja je bil lahko uporabljen na dva načina, v bazi 2 ali v bazi 16. Razlika je bila le v tem, ali je ukaz `BERI` prebral vsak bit posamezno ali pa po 4 bite naenkrat. Dokazala sta, da na ta način poenostavljen jezik registrskega stroja implementira univerzalni Turingov stroj, če dela v bazi 2 ali 16.

Implementirala sta Java simulator za njune programe registrskih strojev, ki kot parameter sprejme niz p, ki je program registrskega stroja, in celoštevilsko konstanto $jtime$, ki je zgornja meja števila skokov, ki se pojavijo v programu. S to konstanto sta omejila čas izvajanja programov. Da sta ugotovila, koliko bitov izračunane konstante Ω je točnih, sta primerjala dve vrednosti, in sicer *spodnjo* in *zgornjo mejo* za vrednost Ω. Spodnjo mejo sta izračunala podobno kot prej, torej sta generirala vedno več in več ustavljivih programov in njihove prispevke prištela k vrednosti spodnje meje za konstanto Ω. Računanje zgornje meje pa je bilo težje. Ideja je bila ta, da sta med generiranjem programov sistematično izločala vse programe, ki so zagotovo neustavljivi. Tako sta pridobivala čedalje boljše zgornje meje. Ko sta med seboj primerjala vrednosti bitov spodnje in zgornje meje, sta videla, koliko bitov približka je zagotovo točnih.

4.3.1 43 začetnih bitov števila Ω v bazi 16

Za računanje v bazi 16 sta Calude in Dinneen generirala programe, katerih velikosti so večkratniki števila 4, vse do velikost 80 bitov. Konstanto *jtime* sta nastavila na vrednost 1000. Vsak program se je torej izvajal vsaj 1000 korakov, večina pa precej več. Vse programe, ki se niso ustavili, sta shranila za poznejšo analizo, ki je bila potrebna za računanje zgornje meje. Začela sta z generiranjem programov dolžine 4 bite, nadaljevala s programi dolžine 8 bitov itd. Pri vsaki dolžini sta izračunala spodnjo in zgornjo mejo za vrednost števila Ω. Na koncu sta med seboj primerjala izračunano spodnjo in zgornjo mejo ter ugotovila, da je vsaj 43 bitov točnih. Njun rezultat prikazuje enačba (4.6).

$$\Omega_{16} = 0,0001000000010000101001110111000100000101110 \tag{4.6}$$

S tem sta izboljšala rezultat iz [5], saj sta tokrat, gledano relativno, izračunala več bitov konstante Ω. Kodiranje je namreč 4-bitno in ne 7-bitno kot prej.

4.3.2 40 začetnih bitov števila Ω v bazi 2

Računanje pri bazi 2 je bilo težje, saj sta morala upoštevati vse dolžine programov in ne le večkratnike števila 4. Programe sta generirala do dolžine 84 bitov. Konstanto *jtime* sta tudi tokrat nastavila na vrednost 1000. Zaradi tega sta morala generirati veliko več programov. Da sta to lahko izvedla, sta uvedla nekaj poenostavitev. Na vsakem koraku sta računala le spodnjo mejo in ne tudi zgornje, slednjo sta izračunala le na koncu. Do dolžine 61 bitov sta generirala vse programe in pri računanju spodnjih mej uporabila prispevke vseh ustavljivih programov. Pri dolžinah od 62 do 84 bitov pa nista več generirala vseh programov, ampak sta postavila dve omejitvi. Nista več razširjala (i) programov, ki bi imeli skupaj s podatki dolžino večjo od 80 bitov in (ii) programov daljših od 30 bitov. Pri računanju zgornje meje sta prilagodila strategijo, uporabljeno pri bazi 16, saj je bilo pri bazi 2 preveč programov, da bi jih lahko generirala. Podrobnosti si lahko bralec pogleda v

[9]. Po končani analizi sta primerjala spodnjo in zgornjo mejo in na ta način pridobila 40 točnih bitov števila Ω. Njun rezultat prikazuje enačba (4.7).

$$\Omega_2 = 0,0001000000010000101001110111000011111010 \tag{4.7}$$

Izračunani vrednosti pričakovano nista enaki, ker je uporabljeno drugačno kodiranje (ki pa vpliva na vrednost števila Ω).

Poglavje 5

Super Omega

Konstanta Ω je dodobra pretresla matematični svet in nekateri matematiki so se le stežka sprijaznili z neizračunljivostjo in naključnostjo, ki ju Ω vnaša v različne veje matematike. Toda Ω še ni vse, kar je predstavil Chaitin. Definiral je tudi števila, ki jih imenuje **Super Omega**. Ta števila je predstavil Marcus Chown [17]. Tudi Super Omege izvirajo iz odkritij Turinga. Turing si je zamislil računalnik, ki je močnejši od katerega koli realnega računalnika, saj zmore odgovoriti na vprašanje, ali se bo določen računalnik ustavil pri izvrševanju danega programa ali ne. Tak računalnik, ki si ga je zamisli Turing, se imenuje **računalnik s prerokom**. Takoj ko je Chaitin uvedel Ω, si je prav tako lahko zamislil tudi preroka, ki bi poznal to število. Toda tudi računalnik s prerokom za Ω bi imel svojo verjetnost ustavitve Ω', ki pa je sam ne bi mogel izračunati. Na enak način kot prej si je možno zamisliti računalnik s prerokom za Ω'. Tudi ta ima svojo neizračunljivo verjetnost ustavitve Ω'', ki jo pozna le neki prerok tretje stopnje, in tako naprej. Chaitin je prepričan, da obstaja neskončno zaporedje števil $\Omega, \Omega', \Omega'', \ldots$, katerih naključnost strogo narašča. Trdi tudi, da obstaja neki prerok neskončno visokega reda (tj. reda, ki je neskončno kardinalno število), ki pozna vse druge $\Omega^{(k)}$, kjer je $k \in \mathbb{N}$.

Sprva Super Omege niso predstavljale velikega odkritja, saj je bilo na prvi pogled videti, da nimajo nobene povezave z realnim svetom in zato v

praksi niso pomembne. Toda Veronica Becher in Sergio Daicz sta pokazala nasprotno. Odkrila sta, (i) da je Ω' enaka verjetnosti, da neskončni izračun vrne samo končno velik izhod in (ii) da je Ω'' enaka verjetnosti, da neskončni izračun, ki vrača neskončno zaporedje, le končno mnogokrat ne bo vrnil izhoda (v neskončnem zaporedju bo manjkalo le končno mnogo števil).

V članku [29] je izpostavljeno, da so Super Omege še bolj naključne od Ω, saj se ne da izračunati niti njihovih začetnih bitov. Super Omege prinašajo v matematiko še večjo naključnost in zato predstavljajo še večjo težavo za tradicionalen pristop. Če bi matematiki kdaj znali izračunati Ω, to ne bi pomenilo velikega napredka, ker bi bilo potem še neskončno Super Omeg $\Omega', \Omega'', \Omega''', \ldots$, ki jih ne bi mogli izračunati.

Poglavje 6

Verjetnost ustavitve v praksi

Že večkrat smo poudarili, da ima problem ustavitve velik praktičen pomen. Ker je neizračunljiv, znanstveniki iščejo alternativne rešitve in poskušajo problem bolje razumeti. V nadaljevanju bomo podali nekaj primerov, kjer so znanstveniki raziskovali problem ustavitve na praktičnih primerih in primerjali njihove zaključke.

6.1 Problem ustavitve in programski jezik BF

Sven Köhler, Chrisitian Schindelhauer in Martin Ziegler [20] so izpostavili, da je pri približnem reševanju problema ustavitve ključnega pomena izbira programskega jezika in kodiranje problema oz. programski sistem, ki ga to kodiranje določa. Veliko programskih sistemov v praksi dovoljuje približno reševanje z asimptotično zanemarljivo relativno napako iz preprostega razloga, ker se delež sintaktično nepravilnih programov z večanjem dolžine približuje vrednosti 1. Takim programskim sistemom pravimo, da **niso gosti**. Proučevanje programskega sistema, ki ni gost, pa ni tako pomembno, saj programi, ki se pojavljajo v praksi, predstavljajo le majhno in zelo razpršeno podmnožico programov v takem sistemu.

Köhler, Schindelhauer in Ziegler [20] so se ukvarjali s problemom ustavitve

na primeru formalnega jezika **BF (BrainF*ck)**, ki je popoln po Turingu[1]. To je zelo preprost programski jezik, ki uporablja abecedo s samo 8 znaki: $\Sigma_{BF} = \{<, >, +, -, ,, ., [,]\}$. Prvih 6 znakov predstavlja ukaze, znaka [in] pa se uporabljata za konstrukcijo zank. BF-program shranjuje podatke na traku podobnem, kot ga ima Turingov stroj. Vsaka celica lahko vsebuje število med 0 in 255 (torej 8 bitov). Vrednost trenutne celice se lahko poveča za 1 z ukazom + in zmanjša z ukazom −. Do drugih celic se lahko dostopa s pomikom levo < ali desno >. Na začetku je vsebina vseh celic traku nastavljena na vrednost 0. Ukaz , vzame 8 bitov vhoda in ga shrani v trenutno celico, medtem ko ukaz . pripne vsebino trenutne celice izhodnemu nizu. Zanke se tvorijo tako, da se da želeno zaporedje ukazov med oklepaja [in].Vsakič, ko se bo zanka izvršila, se preveri, ali trenutna celica vsebuje število različno od 0. Če je tako, se zanka izvrši ponovno, sicer se konča. Sintaksa programa je zelo preprosta, edina zahteva je ta, da morata biti [in] pravilno gnezdena. Da je tak programski sistem popoln po Turingu, je potrebno privzeti še to, da je trak potencialno neskončen[2].

Predstavili so dve različni kodiranji BF-programov. **„Naivno" kodiranje** BF-programa je kodiranje s funkcijo Ψ_p. To je funkcija, ki jo dobimo, če besedo p nad abecedo Σ_{BF} interpretiramo kot kodo nekega BF-programa. Če p ni sintaktično pravilen, velja $\Psi_p :\equiv \uparrow$. Dokazali so, da programski sistem, ki ga dobimo z naivnim kodiranjem BF-programov, ni gost. Zato se vprašanje približnega reševanja problema ustavitve prevede na preverjanje sintaktične pravilnosti in je zato trivialno. Predstavili so tudi **„kompaktno" kodiranje**, tj. kodiranje BF-programov s funkcijo Φ_N. Tu je Φ_N funkcija, ki jo izračuna N-ti sintaktično pravilen BF-program p_N. Dokazali so, da je programski sistem, ki ga določa funkcija Φ_N, gost. Dokazali so tudi, da obstaja univerzalna konstanta $\epsilon > 0$, ki je spodnja meja napake, ki jo naredi katerikoli program, ki poskuša približno rešiti problem ustavitve v tem programskem sistemu.

[1]Formalni jezik je popoln po Turingu, če lahko simulira katerikoli Turingov stroj.

[2]Trak je potencialno neskončen, če se lahko po potrebi podaljšuje.

Tudi Uroš Čibej in sodelavci [18] so se ukvarjali s problemom ustavitve pri programskem jeziku BF. Ugotovili so, da se naključno generirani BF-programi v več vidikih precej razlikujejo od tistih, ki jih je napisal človek. Zato so analizirali več BF-programov, ki jih je napisal človek in izolirali lastnosti, ki odražajo te razlike. To jim je omogočilo razvoj metode za naključno generiranje realističnih BF-programov, ki so po svojih lastnostih bolj sorodni tistim, ki jih napiše človek. To metodo so uporabili pri generiranju BF-programov. Razvili so več metod dokazovanja, da se dani BF-program ne ustavi, in jih uporabili pri eksperimentalnem ocenjevanju verjetnosti ne-ustavitve pri razredu realističnih in razredu povsem naključnih BF-programov. Njihovi rezultati so pokazali, da je empirično problem ustavitve relativno lahek v obeh razredih, saj je velika večina primerkov problema ustavitve odločljiva z uporabo razvitih metod.

6.2 Ustavljivi programi se ustavijo hitro

V nadaljevanju bomo predstavili še nekaj primerov, kjer so raziskovalci na različne načine proučevali problem ustavitve. Ne glede na razlike v pristopih so vsi med svojim delom prišli tudi do ugotovitve, da se v praksi večina programov ustavi hitro ali pa se nikoli ne ustavi.

William B. Langdon in Riccardo Poli [21] sta proučevala problem ustavitve pri von Neumannovi arhitekturi. S praktičnim poskusom sta potrdila teoretične domneve, da je pri *linearnem genetskem programiranju*[3], ki je popoln po Turingu, delež ustavljivih programov zanemarljivo majhen. Poskus sta izvajala na majhnem računalniku, imenovanem T7, ki ima neposredno

[3]Genetsko programiranje [32] je področje umetne inteligence in strojnega učenja. Gre za samodejno pisanje programov po vzoru naravnega izbora. Stroj sam napiše program, ki reši vnaprej zadani problem. Na začetku imamo nekaj naključno napisanih programov. Nato s križanjem, mutacijo in selekcijo dobimo naslednjo generacijo. Če so okoliščine ustrezne, je vsaka naslednja generacija boljša od prednikov. Linearno genetsko programiranje [33] je posebna oblika genetskega programiranja, kjer so programi v populaciji predstavljeni kot zaporedje ukazov strojnega jezika.

dostopen končni spomin. Njegovi programi vsebujejo 7 ukazov. To so `ADD` (seštevanje), `JUMP` (brezpogojni skoki), `BVS` (pogojna vejitev) in štirje različni ukazi kopiranja. Natančno definicijo in predstavitev teh ukazov si bralec lahko pogleda v [21]. Uporabila sta 8-bitno kodiranje. Dolžina naslovov je odvisna od velikosti programa. Naslov mora biti ravno dovolj dolg, da lahko naslovi vsak ukaz v programu. Pri poskusu sta uporabila 96-bitni spomin in 1 dodaten bit za prenos. Uporabila sta programe dolžin od 30 do 16777215 ukazov. Teh programov je preveč, da bi lahko testirala vse, zato sta naključno vzorčila programe posameznih velikosti in na ta način dobila reprezentativen vzorec. Pri vsaki izmed dolžin sta testirala 1000 programov. Vsakega sta izvajala iz naključne začetne točke in z naključnim vhodom, dokler (i) program ni dosegel zadnjega ukaza in se ustavil, (ii) ni bila zaznana neskončna zanka ali (iii) se ni posamezen ukaz izvedel več kot stokrat. Neskončno zanko sta prepoznala tako, da sta sledila izvajanju programa. Če je bila vsebina spomina in zastavice za prenos po izvedenem ukazu identična vsebini, ko se je dani ukaz zadnjič izvedel, potem sta ocenila, da gre za neskončno zanko. Na ta način sta zbirala statistike o tem, koliko ukazov se je izvršilo, koliko programov se je končalo, kakšni tipi in dolžine zank so se pojavljali, kdaj je vstopilo izvajanje programa v prvo zanko... Z opazovanjem teh statistik sta prišla do numeričnih vrednosti, ki veljajo le za računalnik T7, vendar dobljena razmerja veljajo splošno. Pokazala sta, da je v dolgih programih večina zank kratkih in da je verjetnost, da bo program, ki je eno zanko zapustil, ponovno vstopil v novo zanko, skoraj enaka, kot je verjetnost vstopa v zanko na začetku programa, slednje pa je skoraj 1. Na ta način predvidevata, da bo program med izvajanjem tekel iz ene zanke v drugo, dokler ne bo v eni obtičal. Izkazalo se je tudi, da v dolgih programih ne traja dolgo, da se najde kratka zanka, iz katere ni izhoda. Izkazalo se je torej, da delež ustavljivih programov pada proti ničli z naraščajočo dolžino programa. Dolgih programov je eksponentno več kot kratkih, zato gledano absolutno, število ustavljivih programov raste. Toda gledano s stališča verjetnosti je pri von Neumanovi arhitekturi problem ustavitve odločljiv, saj se von Neumanovi programi ne

ustavijo z verjetnostjo 1. Izkazalo se je tudi, da je delež ustavljivih programov dolžine N enak $\frac{1}{\sqrt{N}}$, standardni odklon časa izvajanja programa pa raste kot $\sqrt{N}$. Torej, če bi program izvajali recimo $20 * \sqrt{N}$ ciklov, bi lahko ločili skoraj med vsemi ustavljivimi in neustavljivimi T7-programi.

Tudi Calude in Michael A. Stay [6] sta prišla do podobnih zaključkov, ko sta proučevala verjetnost ustavitve. V svojem delu sta predstavila dva pristopa k računanju verjetnosti, da se N-bitni program ustavi. Chaitin je pri računanju konstante Ω uporabil prvi pristop. Tu je verjetnostni prostor eno-dimenzionalen, tj. prostor vseh možnih programov. V tem primeru se verjetnost, da se N-bitni program P ustavi, računa po enačbi (6.1).

$$Prob_N = \frac{|\{P \in \Sigma^N | P \ se \ ustavi\}|}{2^N} \tag{6.1}$$

Pri samoomejenih Turingovih strojih se $Prob_N$ približuje vrednosti 0, ko gre N proti neskončno, saj postaja vedno bolj verjetno, da je dan N-bitni niz podaljšek krajšega ustavljivega programa. Pri univerzalnem Turingovem stroju, ki ni samoomejen, pa je verjetnost ustavitve za dovolj velik N vedno večja od nič. To velja zato, ker bo od določene točke dalje tak stroj simuliral totalni Turingov stroj (tak, ki se ustavi pri vsakem vhodu) in bo zato vedno prispeval neki pozitiven delež k vrednosti $Prob_N$. V splošnem je 1 najboljša izračunljiva zgornja meja za $Prob_N$, ki jo lahko najdemo. Pri tem pristopu igra pomembno vlogo le to, ali se program ustavi ali ne, čas, v katerem se ustavi, pa ni pomemben.

Calude in Boris Pavlov [11] sta uporabila drugačen pristop. Verjetnostni prostor sta razširila v dve dimenziji, tj. prostor in čas. Opazovala sta verjetnost, da se naključen N-bitni program, ki se ni ustavil v danem času, ustavi do naključnega poznejšega časa. V tem primeru je čas ustavitve bistvenega pomena. Izbrala sta verjetnostno porazdelitev prostora časov ustavitve. Pomembno je, da izbrana porazdelitev zagotavlja, da so dolgi časi izvajanja izjemno redki, zato gledano v limiti niti ni tako pomembno, katero porazdelitev sta izbrala. Calude in Pavlov sta zasnovala kvantno-mehansko napravo, ki je sposobna verjetnostno rešiti problem ustavitve. Njun algoritem je najprej na podlagi dolžine programa in vnaprej določene dovoljene napake 2^{-k}

učinkovito izračunal neko končno časovno mejo T. Kvantna naprava, ki je narejena tako, da dela na naključno izbranem testnem vektorju, je bila potem pognana za T časa. Če je naprava vrnila 0, potem se dani program ustavi, sicer se program ne ustavi z verjetnostjo večjo od $1 - 2^{-k}$. Ta rezultat je pomemben, vendar uporablja nekonvencionalni model kvantnega računanja.

Calude in Stay [6] sta pokazala, da za pridobitev tega rezultata uporaba kvantnih naprav ni nujna. Razvila sta namreč metodo, ki uporabi zgornji algoritem, vendar brez uporabe kvantnega računanja. Vnaprej sta predpostavila neko izračunljivo verjetnostno porazdelitev prostora vseh časov izvajanja. Verjetnostni prostor je potem produkt prostora vseh programov določene dolžine (ali pa vseh možnih programov), kjer so programi enakomerno porazdeljeni, in prostora časa, ki je diskreten in ima vnaprej izbrano verjetnostno porazdelitev. V tem kontekstu sta pokazala, da če je dana konstanta $k > 0$, lahko učinkovito izračunata časovno mejo T tako, da je na produktnem prostoru verjetnost, da se bo N-bitni program pozneje ustavil, če se ni ustavil do časa T, manjša od 2^{-k}. Dokazala sta tudi, da točen čas ustavitve programa algoritmično ni naključen, ker se večina programov bodisi ustavi hitro ali pa se nikoli ne ustavi in da se N-bitni program, ki se ni ustavil do časa 2^{N+c}, kjer je c konstanta, ne more ustaviti ob naključnem poznejšem času.

Vsi predstavljeni primeri kažejo na to, da je v praksi velikokrat možno izračunati dobre približke za rešitve problema ustavitve ali pa vsaj za verjetnost ustavitve.

Poglavje 7

Sklepne ugotovitve

V tem diplomskem delu smo podali definicijo in predstavitev konstante Ω. Pregledali smo, kaj se je na tem področju dogajalo od njene definicije leta 1975 do danes. Povzeli smo glavne dosežke pri računanju bitov konstante in predstavili nekaj področij, na katere vpliva Ω. To delo predstavlja uvod oz. osnovo za nadaljnje raziskovanje v tej smeri. Bralcu, ki ga ta tema zanima, predlagamo v branje še knjige [22, 24, 19, 23, 3]. Na tem področju bi bilo po mojem mnenju zanimivo poiskati še več praktičnih primerov in znanih problemov, ki so povezani s konstanto Ω.

Chaitin velja za velik um in je s svojim delom in s svojimi razmišljanji spodbudil veliko vprašanj. Matematiki, filozofi in drugi znanstveniki so se različno odzivali na njegovo odkritje. Calude je ob Chaitinovem 60. rojstnem dnevu izdal knjigo z naslovom *Randomness & Complexity, from Leibniz to Chaitin* [10], v kateri je zbral nekatere odzive na njegovo delo. Gre za znanstvene in filozofske članke ter za spomine in eseje. Bralec, ki ga tematika zanima, lahko v tej knjigi najde veliko zanimivih pogledov na Chaitinovo odkritje in na posledice, ki jih prinaša.

Osebno me je pri pisanju tega dela najbolj navdušil Chaitinov način razmišljanja in njegov nekonvencionalen pristop k matematiki. Zanimivo je to, da mu je uspelo najti konkreten primer (konstanto Ω), ki je podprl njegove trditve o tem, da je prav naključnost temelj matematike. Presenetljivo

je tudi to, da mu je uspelo naključnost zelo natančno, a vendar enostavno, definirati. Pravi namreč, da je naključno zaporedje tako, ki je ireducibilno, torej je najkrajši opis naključnega zaporedja kar zaporedje samo. Svoj zanimiv pristop k matematiki je po mojem mnenju najlepše opisal z naslednjim izrekom: *"In a way, math isn't the art of answering mathematical questions, it is the art of asking the right questions, the questions that give you insight, the ones that lead you in interesting directions, the ones that connect with lots of other interesting questions - the ones with beautiful answers."*

Zaključimo pa z mislijo Chaitina, ki je bila objavljena v reviji New Scientist leta 2006 v rubriki *Brilliant Minds Forecast the Next 50 Years*. Chaitin je zapisal: *"I hope that by 2056 weird astronomical observations will lead to radical new fundamental physics. I expect people will be tampering with the human genome, which should be fun. In my own field, I hope the current desiccated, formal approach has died out and people are more adventurous and creative."*

Literatura

[1] G. Barmpalias, A. E. M. Lewis. *Chaitin's halting probability and the compression of strings using oracles*. Proceedings of the Royal Society A 467 (2011), str. 2912–2926.

[2] C. H. Bennett, M. Gardner. *The random number omega bids fair to hold the mysteries of the universe*. Scientific American, št. 241 (1979). str. 20-34.

[3] G. S. Boolos, J. P. Burgess, R. C. Jeffrey. *Computability and Logic*. Cambridge University Press, 2002

[4] C. S. Calude, G. J. Chaitin. *What is a Halting Probability*. Notices of the AMS, št. 57, zv. 2 (2010), str. 236-237.

[5] C. S. Calude, M. J. Dinneen, C. K. Shu. *Computing a glimpse of randomness*. Experimental Mathematics, št. 11, zv. 3(2002), str. 361-370.

[6] C. S. Calude, M. A. Stay. *Most programs stop quickly or never halt*. Advances in Applied Mathematics št. 40, zv. 3 (2008), str. 295–308.

[7] C. S. Calude. *Chaitin Omega numbers, Solovay machines and incompletness*. Theoretical Computer Science št. 284(2002), str. 269-277.

[8] C. S. Calude, M. J. Dinneen, C. K. Shu. *Computing 80 initial bits of a Chaitin Omega nuber: Preliminary version*. CDMTCS Research Report, št. 146 (2000).

[9] C. S. Calude, M. J. Dinneen. *Exact Approksimations of Omega Numbers.* International Journal of Bifurcation and Chaos, št. 17, zv. 6 (2007), str. 1937-1954.

[10] C. S. Calude. *Randomness & Complexity, from Leibniz to Chaitin.* World Scientific, 2007.

[11] C. S. Calude, B. Pavlov. *Coins, quantum measurements and Turing's barrier.* Quantum Inf. Process, št. 1, zv. 1-2 (2002), str. 107-127.

[12] G. Chaitin. *Meta Maths: The Quest for Omega.* Atlantic Books, 2006.

[13] G. Chaitin. *Omega and why maths has no TOEs.* Plus magazine (1.12.2005).

[14] G. Chaitin. *An Algebraic Characterization of the Halting Probability.* Fundamenta Informaticae, št. 79 (2007), str. 17-23.

[15] G. Chaitin. *The Halting Probability via Wang Tiles.* Fundamenta Informaticae, št. 86, zv. 4 (2008), str. 429–433.

[16] G. Chaitin. *Algorithmic Information Theory.* Cambridge University press, Cambridge 1987, tretji ponatis 1990.

[17] M. Chown. *The Omega Man*, New scientist (10.3.2001).

[18] U. Čibej, B. Robič, J. Mihelič. *Halting in a simple programming language: random vs. human.* Članek je v pripravi.

[19] R. G. Downey, D. R. Hirschfeldt. *Algorithmic Randomness and Complexity.* Springer, 2010.

[20] S. Köhler, C. Schindelhauer, M Ziegler. *On Approximating Real-World Halting Problems.* In Fundamentals of Computation Theory, M. Liskiewicz and R. Reischuk (Eds.). Lecture Notes in Computer Science, št. 3623. Springer Berlin Heidelberg, str. 454–466.

[21] W. B. Langdon, R. Poli. *The halting probability in von Neumann architectures*. P. Collet (Ed.), et al. EuroGP 2006, LNCS št. 3905, Springer, Heidelberg (2006), str. 225 -237.

[22] M. Li, P. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2008.

[23] P.G. Odifreddi. *Classical Recursion Theory, vol. I*. North-Holland, 1992.

[24] A. Nies. *Computability and Randomness*. Oxford Logic Guides, 2012.

[25] B. Robič. *The Foundations of Computability Theory*. Springer, 2015.

[26] B. Saxberg. *Chaitin's Constant: An Uncomputable Number*. [Online]. Dosegljivo: https://www.math.washington.edu/m̃orrow/336_13papers/brendan.pdf [Dostopano 8.12.2015].

[27] Chaitin's Constant. [Online]. Dosegljivo: http://mathworld.wolfram.com/ChaitinsConstant.html [Dostopano: 03.03.2016]

[28] Gregory Chaitin.[Online]. Dosegljivo:https://en.wikipedia.org/wiki/Gregory_Chaitin [Dostopano 16.12.2015].

[29] Chaitin's constant. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Chaitin's_constant [Dostopano 16.12.2015].

[30] Kraft's inequality. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Kraft's_inequality [Dostopano: 01.02.2016]

[31] Riemann hypothesis. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Riemann_hypothesis [Dostopano: 29.02.2016]

[32] Genetsko programiranje. [Online].
Dosegljivo: https://sl.wikipedia.org/wiki/Genetsko_programiranje
[Dostopano: 12.05.2016]

[33] Linear genetic programming. [Online].
Dosegljivo: https://en.wikipedia.org/wiki/Linear_genetic_programming
[Dostopano: 12.05.2016]

Stvarno kazalo