Algoritamski nerješivi masovni algoritamski problemi

Ilona, Benko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ilona Benko

ALGORITAMSKI NERJEŠIVI MASOVNI ALGORITAMSKI PROBLEMI

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ilona Benko

Matični broj: 44026/15–R

Studij: Informacijski sustavi

ALGORITAMSKI NERJEŠIVI MASOVNI ALGORITAMSKI PROBLEMI

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Mirko Čubrilo

Varaždin, lipanj 2018.

Ilona Benko

Izjava o izvornosti

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

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Pojam „algoritam“ je suštinski intuitivan. Prilikom opisivanja istog, formalizacija se čini kao korektnija opcija zbog nepreciznosti i problema s interpretacijom govornog jezika. To je učinjeno pomoću Turingovog stoja, koji se ističe kao jedno od pogodnih rješenja imajući u vidu glavna svojstva i zahtjeve koje svaki algoritam mora zadovoljavati. Nadalje, opisan je pojam izračunjivosti po Turingu, pojam univerzalnog Turingovog stroja, te problem zaustavljanja Turingovog stroja. U tom kontekstu, formulirana je metoda dokazivanja algoritamske nerješivosti masovnih algoritamskih problema metodom svodivosti njihove rješivosti na nerješive masovne algoritamske probleme. Metoda je izložena i primijenjena nad nekoliko primjera dokazivanja nerješivosti nerješivih masovnih algoritamskih problema.

Ključne riječi: pojam algoritma, masovni algoritamski problemi, Turingovi strojevi, univerzalni Turingov stroj, algoritamska nerješivost

Sadržaj

1. Uvod

Tematika ovog rada usko je vezana uz teoriju izračunljivosti i teoriju automata. Iako se čini da ova područja strogo spadaju u granu teorijske matematike, iz njih su se izrodila računala kakva danas poznajemo. Korijeni teorije izračunljivosti krenuli su kad su K. Gödel, A. Turing i A. Church otkrili masovne algoritamske probleme koji se ne mogu riješiti pomoću računala, poput problema odlučivanja u domeni logike prvog reda [1]. Ne postoje univerzalni algoritmi koji bi mogli riješiti te probleme. Razvojem raznih teorijskih modela, ovo područje je pomoglo konstruiranju današnjih računala. Teorija izračunljivosti bavi se klasifikacijom problema na rješive i nerješive, te je usko povezana s teorijom složenosti. Teorija automata bavi se definiranjem svojstava i matematičkih modela izračunljivosti. Ona predstavlja primjenu formalnih definicija izračunljivosti i daje koncepte važne za neteorijska područja računarstva. Primjerice, model konačnih automata koristi se kod kompajlera i sklopovlja.

Kada želimo formalno opisati pojam algoritma, imamo nekoliko korektnih opcija, primjerice korištenje formalnih jezika, λ-račun koji je koristio Church ili apstraktne strojeve stanja koje je koristio Turing. Sve navedene opcije međusobno su ekvivalentne. U ovom radu, odlučila sam se za opisivanje algoritma pomoću Turingovih strojeva jer problem zaustavljanja stroja možemo direktno povezati s algoritamskom nerješivosti. Turingov stroj daje detaljan pregled svih ulaza, izlaza, te svih stanja i uvjeta prijelaza koji se nalaze na putu dobivanja rezultata. Ključni postupak kod ovog opisivanja jest apstrahirati algoritam na način da se on jednoznačno opiše pomoću Turingovog stroja. U tom pogledu, potrebno je jasno definiranje skupa simbola stroja. Kada se ovo odradi korektno, bilo koji algoritam se može predstaviti pomoću Turngovog stroja. Ispravnim kodiranjem algoritamskih problema i korištenjem univerzalnog Turingovog stroja može se ispitati njihova rješivost.

U ovom radu navedeni su koncepti korišteni u cilju prezentiranja problematike nerješivih masovnih algoritamskih problema. U drugom i trećem poglavlju objašnjena je povezanost algoritamske rješivosti i zaustavljanja Turingovog stroja, formulirana je metoda dokazivanja algoritamske nerješivosti masovnih problema, te su prethodno navedeni i ovom kontekstu prilagođeni osnovni koncepti i postupci potrebni za njenu realizaciju. U četvrtom poglavlju, metoda dokazivanja algoritamske nerješivosti je pobliže iznesena nad nekoliko masovnih algoritamskih problema. Osobna motivacija leži u entuzijazmu prema ovom području i željom za primjenom navedenih koncepata i teorijskih modela na konkretne probleme.

2. Pojam algoritma

U ovom poglavlju navedene su karakteristike masovnih algoritamskih problema, objašnjena je osnovna problematika opisivanja intuitivnog pojma algoritma i dana je motivacija za nerješive probleme. Definiran je pojam izračunljivosti i dan uvid u izvedbu formalizacije pomoću Turingovog stroja.

2.1. Masovni algoritamski problemi

Za početak je potrebno definirati algoritamski i masovni algoritamski problem. Često se ova dva pojma poistovjećuju, no masovni algoritamski problem je u pravilu poseban tip algoritamskog problema [2]. Algoritamski problem je problem konstruiranja algoritma koji ima određena svojstva, odnosno ima neko specificirano ponašanje. To je zahtjev za davanjem rješenja problema koje zadovoljava određene uvjete. Riješiti problem znači pronaći rješenje, te ukoliko se može utvrditi da takvo rješenje postoji, problem se naziva rješivim. U suprotnom, problem je nerješiv. Postoje brojni nerješivi masovni algoritamski problemi, a prema Ricevom teoremu, bilo koje netrivijalno pitanje o rješivosti Turingovog stroja je nerješiv problem [2].

Masovni problemi se uvijek sastoje od beskonačnog skupa pojedinačnih algoritamskih problema, te su glavno područje primjene teorije algoritama. Prilikom rješavanja masovnog problema, potrebno je odjedanput riješiti sve slučajeve pojedinačnih problema. Problem mora biti općenito zadan i dati rješenje za svaki specifičan slučaj. Količina i vrsta ulaznih podataka, kao i sva ograničenja moraju biti zadani i jasno opisani [3].

I. Primjer algoritamskog problema: Pronaći racionalnu aproksimaciju korijena polinoma $x^2 - x - 1 = 0$ s preciznošću od 10^{-6}.

II. Primjer masovnog problema: Pronaći racionalnu aproksimaciju korijena polinoma $x^2 - x - 1 = 0$ s preciznošću od 10^{-n} za svaki n.

Ideja masovnog problema je za ove potrebe i dalje previše općenita, te je potrebno naći njezin općenitiji ekvivalent, a to je masovni algoritamski problem. Da bi specificirali masovni algoritamski problem, potrebno je navesti [2]:

- skup X: skup algoritamskih problema
- skup Y: skup algoritamskih rješenja
- podskup $E \subset X$: skup ograničenja problema
- podskup $E \subset X \times Y$: skup veza između problema i rješenja

Formalno govoreći, pronaći rješenje masovnog algoritamskog problema jest naći algoritam $X \to Y$ koji transformira svaki od algoritamskih problema $\alpha_i \in E$ u algoritamska rješenja $\beta_j \in Y$, takvih da $\langle \alpha, \beta \rangle \in R$.

III. Primjer masovnog algoritamskog problema:

$$X = E = \mathbb{N}, \qquad Y = \mathbb{Q}$$

$$R = \{\langle n, r \rangle \mid |r - x_0| < 10^{-n}, \qquad gdje\ je\ x_0\ željeni\ korijen\}$$

Izmjenom masovnog problema u korespondirajući masovni algoritamski problem zapravo dobivamo pojam o traženim rješenjima i algoritamskoj rješivosti masovnih algoritamskih problema. Masovni problem se može transformirati u apstraktniji pojedinačni problem. Cilj rješavanja masovnih algoritamskih problema je dati jedno rješenje koje je zapravo algoritam. Postojanje rješenja za neki pojedinačni problem masovnog problema ne mora nužno značiti da i korespondirajući masovni algoritamski problem ima rješenje [3]. Svaki problem u odlučivanju je masovni algoritamski problem i većina masovnih algoritamskih problema se može svesti na probleme u odlučivanju [1].

2.2. Razvoj teorije i potreba za formalizacijom

Pojam algoritma se ne može dovoljno precizno definirati govornim jezikom, iako je intuitivno jasno što bi taj pojam trebao predstavljati. Neformalno govoreći, „algoritam“ je skup jednostavnih instrukcija koje se izvršavaju prilikom rješavanja specifičnog problema. U svakodnevnom životu pojam algoritma često se poistovjećuje s pojmovima „procedura“ i „recept“.

Od davnih vremena, veliki umovi su pokušali dati formalnu definiciju algoritma, a krajem prošlog stoljeća dano je nekoliko već spomenutih formalizacija koje su međusobno ekvivalentne. Do problema dolazi prilikom dokazivanja njihove valjanosti. Dodatan alat je izložen u obliku Church-Turingove teze koja tvrdi da je svaka od tih formalizacija adekvatna za opisivanje intuitivnog pojma algoritma. Kako bi se ovo bolje predočilo, dobro je krenuti od karakterizacije pojma algoritma.

Tvrdnja da je algoritam niz koraka koji za određeni skup ulaznih podataka u konačnom vremenu daju rješenje problema kao određeni skup izlaznih podataka izvodi se iz karakterizacije D.E. Knutha koja je dana idućim svojstvima [4]:

- Konačnost - algoritam mora stati nakon konačnog broja koraka
- Definitnost - svaki korak mora biti precizno definiran i rigorozno specificiran za svaki mogući slučaj izvođenja

- Ulaz - prije početka izvršavanja dano je nula ili više ulaznih podataka koji su uzeti iz specificiranog skupa objekata
- Izlaz - postoji jedan ili više izlaznih podataka koji su u relaciji s ulaznim podatcima
- Efektivnost - sve operacije sadržane u algoritmu moraju biti do te mjere elementarne da ih je čovjek sposoban riješiti u konačnom vremenu koristeći samo olovku i papir

Prije opisani problem je ovom slučaju prisutan kod opisivanja svojstva efektivnosti koje se može jedino rekurzivno opisati [3].

Razvoju teorije nerješivih algoritamskih problema i kretanju u smjeru povezivanja intuitivnog i formalnog pojma algoritama mnogo je doprinio Hilbertov deseti problem čiji naziv glasi "Određivanje rješivosti Diofantske jednadžbe":

"Za danu diofantsku jednadžbu s proizvoljnim brojem nepoznanica i cjelobrojnim koeficijentima (potrebno je) izvesti proces pomoću kojeg se u konačnom broju operacija može odrediti jesu li rješenja jednadžbe u skupu cjelobrojnih brojeva." [5]

Sedamdeset godina kasnije, rješenje ovog problema dao je Y. Matiyasevich oslanjajući se na ranije radove M. Davisa, H. Putnama i J. Robinsona, te je utvrđeno da takav algoritam ne postoji [1]. Hilbertov deseti problem je algoritamski nerješiv.

Za rješavanje ovog problema nerješivosti nije bila dovoljna intuitivna definicija algoritma, već su 1936. godine u radovima Alonza Churcha i Alana Turinga dane ekvivalentne formalizacije pomoću dvije različite metode, λ-računa i strojeva stanja [1]. Iz ovih radova je proizašla Church-Turingova teza ili teza izračunljivosti:

"Problem je algoritamski rješiv ako je rješiv pomoću Turingovog stroja." [6]

Naglasak je na riječi „teza“ iz razloga jer se ne može matematički dokazati njena valjanost. Popularnost ove teze rezultirala je pojavom novih formalizacija pojma algoritma, no sve one su istovjetne formalizaciji pomoću λ-računa i Turingovih strojeva, te se više od toga u povezivanju s intuitivnim pojmom algoritma ne može postići, već korištenje neke od formalizacija ovisi o specifičnom problemu i o tome što se želi postići. U ovom radu koristi se formalizacija pomoću Turingovih strojeva, a navedena teza je ključna u tom postupku jer se svaki algoritam može prikazati pomoću Turingovog stroja i obruno, te (ne)rješivost problema pomoću Turingovog stroja povlači (ne)rješivost masovnog algoritamskog problema.

2.3. Pojam izračunljivosti

Prije detaljnijeg razmatranja same formalizacije, potrebno je još istaknuti granice njenih mogućnosti. Svako računalo, bez obzira na fizička ograničenja resursa, može izračunati svaku teorijski izračuljivu funkciju, te isto tako, ne može izračunati ne izračunljivu funkciju [2]. Izjava da svaki problem koji čovjek može riješiti, može riješiti i računalo, te onaj problem koji čovjek ne može riješiti, ne može riješiti ni računalo ima važnu ulogu. To je samo po sebi jasno i iz Church-Turingove teze koja vrijedi samo za funkcije koje su izračunljive. Ekvivalencija pronalaženju gornje međe neizračunljivih funkcija jest rješavanje problema zaustavljanja koji se pokazao nerješivim za Turingove strojeve, stoga teza iskazuje da funkcija ne može biti efektivno izračunljiva korištenjem bilo koje metode [1]. Iz ovoga slijedi da ako problem zaustavljanja Turingovog stroga za danu funkciju nije rješiv, ta funkcija nije izračunljiva.

Kod efektivno izračunljivih funkcija, korišteni algoritam može izračunati njihove vrijednosti. Ako postoji algoritam za izvršavanje tog zadatka, iz njega se može izvesti stroj koji se tada naziva deterministčkim. Tijekom njegovog rada, točno su specificirani svi budući koraci i stanja u nekom trenutku. Sama izračunljivost po Turingu je definirana na idući način:

"*Pod izračunljivosti Turingovog stroja TS podrazumijeva se konačna sekvenca $\alpha_1, \alpha_2, \ldots, \alpha_p$ trenutnih opisa stanja takva da $\alpha_i \rightarrow \alpha_{i+1}$ (TS) za svaki $1 \leq i < p$ i takva da je α_p terminalna s obzirom na TS. U tom slučaju, $\alpha_p = Rez_{TS}(\alpha_1)$ nazivamo rezultantom od α_1 s obzirom na TS.*" [7]

Nadalje je opisana razlika između izračunljivih, djelomično izračunljivih i neizračunljivih funkcija. Djelomično izračunljive funkcije su one za koje postoji algoritam koji omogućava računanje njihovih vrijednosti u rasponu njihovih domena, no vrijeme izračunavanja ide u beskonačnost ako se kao ulazne vrijednosti uzmu neke vrijednosti koje nisu u domeni. Također, ne može se ni u jednom koraku utvrditi da nema rješenja, odnosno Turingov stroj koji bi se izveo iz tog algoritma bio bi nedeterministički. Kod izračunljivih i neizračunljivih funkcija jasno je postoji li rješenje ili ga nema:

"*Neka je TS Turingov stroj. Tada uz svaki n možemo vezati TS koji je n-narna ili totalna funkcija oblika: $\Psi_{TS}^{(n)} = (x_1, x_2, \ldots x_n)$. Kada za svaku n-torku $(m_1, m_2, \ldots m_n)$ odredimo $\alpha_1 = q_1(m_1, m_2, \ldots m_n)$, možemo razlikovati dva slučaja:*

I. *Postoji izračun, odnosno rezultanta od TS oblika $\alpha_1, \alpha_2, \ldots, \alpha_p$. U tom slučaju*

$$\Psi_{TS}^{(n)} = (m_1, m_2, \ldots m_n) = \langle \alpha_p \rangle = \langle Rez_{TS}(\alpha_1) \rangle.$$

II. *Ne postoji izračun od* $\alpha_1, \alpha_2, \ldots, \alpha_p$*, odnosno, rezultanta* $Rez_{TS}(\alpha_1)$ *nije definirana. U tom slučaju funkcija*

$$\Psi_{TS}^{(n)} = (m_1, m_2, \ldots m_n) \text{ nije definirana.}$$" [7]

Neke od jednostavnih izračunljivih funkcija su primjerice funkcija za računanje sljedbenika nekog broja i funkcije implementacija operacija zbrajanja, množenja i oduzimanja. Kod relativno izračunljivih funkcija, stroj se u nekom trenutku zaustavlja i traži dodatne informacije. U klasi svih rekurzivnih funkcija mogu se izdvojiti i neke njezine važne podklase, poput primitivno rekurzivnih funkcija, koje se proučavaju i zasebno.

Negativni efekt koji se javlja prilikom podijele algoritama u dvije klase, rješive i nerješive jest tendencija da se rješivi problemi smatraju i praktično rješivima i relativno izračunljive funkcije izračunljivima, što bi u budućnosti moglo predstavljati problem [2]. Iako bi sve što je formalno dokazano rješivim trebalo biti i praktično rješivo, postoji problem prostorne i vremenske složenosti.

3. Turingov stroj i metoda dokazivanja

U ovom poglavlju objašnjeni su sastavni elementi i razrađeni principi rada Turingovog stroja, definiran je univerzalni Turingov stroj i objašnjena je općenita logika metode svođenja na kojoj se baziraju svi budući dokazi. Navedeni koncepti potrebni su za razumijevanje i korišteni u formiranju metode dokazivanja algoritamske nerješivosti koja je primijenjena u četvrtom poglavlju.

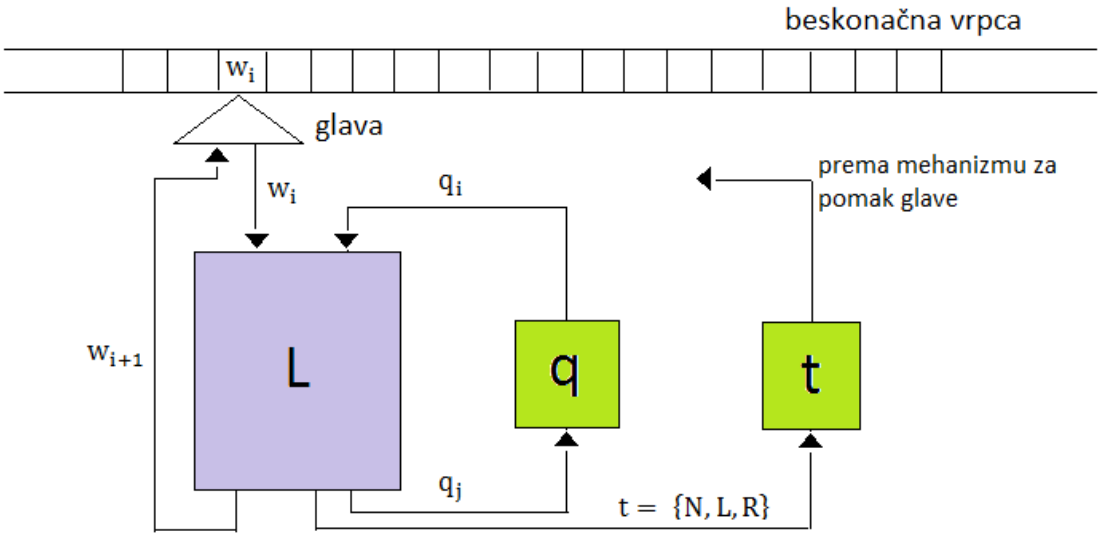
3.1. Model Turingovog stroja

Matematički model Turingovog stroja dan je pomoću n-orke, čiji broj i točan opis određuju željenu varijantu stroja, a u tom se skupu posebno ističe logička funkcija stroja δ koja opisuje dinamiku stroja, odnosno prijelaze iz jednog stanja u novo i izvršavanje elementarnih operacija ili koraka algoritma.

Turingov stroj možemo definirati kao skup (prema [1], [8], [9]):

$$\mathrm{TS} = \langle \mathrm{Q}, \Sigma, \Gamma, \mathrm{b}, \mathrm{T}, \mathrm{q}_0, \mathrm{q}_\mathrm{f}, \delta \rangle$$

- Q je konačan skup svih unutarnjih stanja stroja
- Σ čini vanjska odnosno ulazno-izlazna abeceda stroja bez praznog simbola b
- Γ je unutarnja abeceda stroja koja se nalazi na memorijskoj traci, $\Sigma \cup \mathrm{b} = \Gamma$, sastoji se od simbola $(\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n})$ i praznog simbola b
- b je prazni simbol, $\mathrm{b} \in \Gamma$
- T je skup naredbi za pomak glave za čitanje i pisanje $\mathrm{T} = \{\mathrm{N}, \mathrm{L}, \mathrm{R}\}$
- $\mathrm{q}_0 \subseteq Q$ je skup početnik stanja stroja
- $\mathrm{q}_\mathrm{f} \subseteq Q$ je skup konačnih stanja stroja, pri čemu razlikujemo konačna stanja prihvaćanja q_a i konačna stanja odbijanja q_r
- $\delta: \mathrm{Q} \times \Gamma \rightarrow \mathrm{Q} \times \Gamma \times \mathrm{T}$ je logička funkcija prijelaza stanja gdje $\times$ označava Kartezijev produkt



Slika 1: Shema Turingovog stroja (prema [9])

Ostali fizički elementi koji čine Turingov stroj jesu vanjska memorija koja je realizirana pomoću beskonačne trake koja je neograničena i s lijeva i s desna. Ta traka je podijeljena na polja, s time da se u određenom trenutku na svakom od polja može nalaziti samo jedan simbol iz abecede stroja Γ. Prilikom upisa novog simbola, prethodni se briše, a u samom izračunavanju bitnu ulogu ima prazan simbol b upisom kojeg to polje ostaje prazno.

Poljima na traci se pristupa pomoću glave za čitanje i pisanje koja se na temelju zapisa naredbi za pomak glave T u unutarnjoj memoriji t može pomicati ulijevo, udesno za jedno mjesto ili ostati na istoj poziciji. Glava se uvijek pomiče samo za jedno polje. Druga unutarnja memorija stroja je q, a u nju se bilježi trenutno stanje stroja.

Posljednji sastavni element je primitivna aritmetičko-logička jedinica, odnosno logički blok L u kojem se odvija obrada operacija kao realizacija funkcije $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times T$. Ta funkcija preslikava ulaznu dvojku trenutnog simbola i trenutnog unutarnjeg stanja u izlaznu trojku koja se sastoji od novog simbola koji će se upisati na promatrano polje, te novog stanja i pomaka koji se zapisuju u memorije t i q. Koraci algoritma se bilježe u izlaznim trojkama funkcije prijelaza i svim simbolima u njihovim poljima koji se u tom trenutku nalaze na traci i nazivaju se konfiguracije stroja. U određenom trenutku stroj može imati samo jednu konfiguraciju.

Konfiguracija se može predočiti funkcionalnom shemom ili tablicom stanja Turingovog stroja na način da se u vodećem retku navedu sva unutarnja stanja stroja, a u vodećem stupcu skup simbola vanjske abecede. Osim tog prikaza, može se navesti i pseudokod u taktovima opisan govornim jezikom, unutarnjim stanjima ili funkcijama prijelaza.

Elementarne operacije koje Turingov stroj izvodi mogu se podijeliti u dvije skupine [8]:

I. Operacije zamjene znaka i promjene unutarnjeg stanja stroja u skladu s logičkom funkcijom stroja
II. Pomak glave za čitanje i pisanje

Na početku rada, stroj TS dobiva ulazne podatke $\mathrm{w} = \mathrm{w}_0, \mathrm{w}_1, \dots, \mathrm{w}_\mathrm{n} \in \Sigma$ koji se smještaju na neka uzastopna polja na traci, dok je ostatak trake popunjen praznim simbolima. Glava je pozicionirana na krajnjem lijevom simbolu w_0.

Kada stroj započne s radom, izračun se izvodi u skladu s pravilima koja su opisana funkcijom prijelaza i izvodi se sve dokad stroj ne uđe u konačno stanje q_f, te se u tom trenutku zaustavlja, a na traci je zapisana izlazna supstitucija koja predstavlja rješenje funkcije. U slučaju da se to ne dogodi, stroj nikad ne ulazi u stanje odbijanja ili prihvaćanja, i nikad se ne zaustavlja.

Za Turingov stroj TS kažemo da je primjenjiv za ulaznu informaciju $\mathrm{w} = \mathrm{w}_0, \mathrm{w}_1, \dots, \mathrm{w}_\mathrm{n} \in \Sigma$ ako postoji sekvenca elementarnih operacija $C_1, C_2, \dots, C_k$ za koju vrijedi [1]:

1. C_1 početna konfiguracija stroja TS za ulaz w
2. iz operacije C_i do operacije C_{i+1} se dolazi u jednom koraku algoritma
3. C_k je završna operacija stroja, te je se nakon njenog izvođenja unutarnje stanje stroja jednako q_f

Skup stringova $\mathrm{w} = \mathrm{w}_0, \mathrm{w}_1, \dots, \mathrm{w}_\mathrm{n} \in \Sigma$ za koje je Turingov stroj TS primjenjiv naziva se jezikom TS, nosi oznaku $L(TS)$ i spada u skupinu rekurzivno prebrojivih jezika. Turingov stroj može biti neprimjenjiv za početnu konfiguraciju $\mathrm{w} = \mathrm{w}_0, \mathrm{w}_1, \dots, \mathrm{w}_\mathrm{n} \in \Sigma$ iz dva razloga. Jedan je da ne prihvati ulaz i uđe u stanje odbijanja q_r, a drugi je da ode u petlju i nikad se ne zaustavlja.

3.2. Formalizacija algoritma pomoću Turingovog stroja

Svaki algoritam može se vrlo detaljno i nedvosmisleno opisati definiranim skupom koji određuje Turingov stroj i njegovim raspisivanjem pomoću istog. Turingovi strojevi su vrlo nalik konačnim automatima i predstavljaju precizan model računala opće namjene. Osnovne razlike između konačnih automata i Turingovih strojeva jesu [1]:

- Turingov stroj može obavljati i čitanje i pisanje s memorijske trake
- glava za čitanje i pisanje se može pomicati i ulijevo i udesno
- traka je beskonačne duljine
- posebna konačna stanja prihvaćanja ili odbijanja imaju trenutni efekt

Postoji nekoliko varijanti izvedbe Turingovih strojeva, s time da svaka varijanta ima drugačiju razinu složenosti i pogodna je za različite određene namjene. Razlike u varijantama mogu biti u funkciji prijelaza, izvedbi glave za čitanje i pisanje ili u broju traka koje predstavljaju memoriju. Prostorna složenost je vezna uz odabrani model jer se memorija direktno odnosi na trake, stoga je prije odabira varijante modela potrebno imati detaljan uvid u rad i funkcije algoritma. S vremenskom složenosti je nešto lakše jer se rad stroja odvija u taktovima. Taktovi su vremenski periodi iste duljine za vrijeme kojih se obavlja jedan jednostavni korak algoritma. Zbog toga je za formalizaciju algoritma pomoću Turingovog stroja potrebno raščlaniti problem na elementarne korake koji imaju podjednako trajanje izvođenja.

Na temelju dokaza nezaustavljanja stroja dokazuje se da ne postoji izračun za neizačunljive funkcije jer njihova rezultanta nije definirana. Tada ne postoji traženi jedinstveni algoritam koji kao izlaznu vrijednost vraća rezultat funkcije [7]. U teoriji izračunljivosti i teoriji složenosti, za nerješivi masovni algoritamski problem nije moguće odrediti postoji li takav jedinstveni algoritam koji će uvijek davati točan odgovor tipa da/ne. Takav problem je bilo koje pitanje tipa da/ne s beskonačnim skupom ulaza. Iz tog razloga se uvijek, kada je to moguće, pokušava utvrditi rješivost problema, a ne njegova nerješivost, te se traži algoritam koji za određeni skup ulaza vraća odgovor „da“.

Do toga dolazi kada postoji algoritam koji je konačan, a u suprotnom se ulazi u beskonačnu petlju i algoritam vraća odgovor „ne“. Ta petlja može biti jednostavna, izvoditi se nad uvijek istim skupovima sekvenci, no često je vrlo složena i postoje dinamičke izmjene sekvenci, te se ne mora uvijek utvrditi ista periodičnost [6].

To je najbolje vidljivo prilikom samog rada Turingovog stroja. Inicijalno, traka Turingovog stroja ima na sebi zapisan samo ulazni string, dok su sva preostala polja prazna. Ako stroj treba bilježiti informacije, one se upisuju na traku kao jedan string. Kako bi se te zapisane informacije mogle pročitati, stroj mora pozicionirati glavu za čitanje i pisanje iznad polja na kojem je zapisan string. Stroj tako nastavlja s izračunom sve dokad ne odluči producirati izlaznu informaciju. Izlazi su skupovi konačnih stanja koja se sastoje od podskupova prihvatljivih ili odbijenih stanja koja predstavljaju prihvaćanje ili odbijanje. Kada stroj uđe u stanje prihvaćanja ili odbijanja, algoritam vraća odgovor „da“.

U slučaju da stroj ne uđe u konačno stanje, nastavit će izvođenje izračuna u beskonačnost, stroj se nikada neće zaustaviti i tada je odgovor na početno problemsko pitanje „ne“ [1]. U oba slučaja, radi se o rješivim masovnim algoritamskim problemima. Ako je nemoguće odrediti hoće li stroj ući u konačno stanje prihvaćanja ili odbijanja ili će u

suprotnom ući u beskonačnu petlju, masovni algoritamski problem je nerješiv. Takvi problemi prelaze teoretske granice izračunavanja.

Turing je 1936. prvi puta opisao rad stroja i matematički model na kojem se on temelji [8]. Rad stroja odvija se u jedinicama jednakog trajanja potrebnim za izvođenje jedne elementarne operacije koje se nazivaju taktovi, a prilikom svakog takta se na stroju izvede jedan elementarni korak algoritma. Cjelokupno vrijeme izvođenja algoritma na Turingovom stroju definirana je na idući način:

"*Neka je TS Turingov stroj i neka je w skup ulaznih stringova od TS.*

Tada je t_{TS} vrijeme izvođenja:

$$t_{TS}(w) \coloneqq broj\ taktova\ stroja\ TS\ za\ ulaz\ w$$

Neka je Σ abeceda stroja, $T\colon \mathbb{N}_0 \to \mathbb{N}_0$ funkcija, $A \subseteq \Sigma'$ rješiv jezik, $w = (w_0, w_1, \dots, w_n)$ skup svih stringova sadržanih u Σ i neka je $F\colon \Sigma' \to \Sigma'$ izračunljiva funkcija. Tada:

Turingov stroj TS izračunava funkciju F, odnosno rješava jezik A u vremenu T ako

$$t_{TS}(w) \leq T(|w|)$$ " [9]

Iz gornje definicije možemo zaključiti da je T funkcija veličine ulaza, pa prema tome i vremenska složenost algoritma ovisi isključivo o veličini ulaza. Ako je n neka veličina ulaza, tada za bilo koji string duljine n gornja granica vremenskog izvođenja stroja TS iznosi $T(n)$. Kod opisivanja matematičkog modela Turingovog stroja prostorna složenost je zanemariva iz razloga što stroj u svojoj definiciji ima beskonačnu memoriju predočenu jednom ili više traka beskonačne duljine.

3.3. Univerzalni Turingov stroj

Univerzalni Turingov stroj dobio je takav naziv jer može simulirati bilo koji drugi stroj kada su mu dane njihove standardne definicije, to jest tablice promjena stanja. Radi toga se kaže da je univerzalni Turingov stroj programibilan. Na apstraktnijoj razini, univerzalni Turingov stroj je univerzalna funkcija koja je sama po sebi izračunljiva i koristi se za izračun drugih izračunljivih funkcija. Turing ga je definirao pod nazivom "Univerzalni stroj za izračunavanje“ na idući način:

"Moguće je sastaviti jedan stroj koji se može koristiti za izračunavanje bilo kojeg izračunljivog niza. Ako taj stroj U sadrži traku na početku koje je zapisana standardna definicija stroja M, tada će U izračunavati isti niz kao i M." [6]

Korijeni ideje o pohranjivanju programa proizašli su iz ove definicije. Takav stroj može ujednom čitati formalni opis stroja koji se na njemu simulira, kao i vlastiti ulazni niz. Što se tiče njihovih vremenskih složenosti, ako je $T(n)$ broj koraka nekog algoritma, stroj M ima složenost $O(T(n))$, a ukupna složenost ili složenost stroja U je $O(T(n)\log T(n))$, gdje je n veličina ulaza [10]. Granica prostorne složenosti ostaje ista.

Korištenjem univerzalnog Turingovog stroja U moguće je dokazati rješivost ili nerješivost masovnog algoritamskog problema simuliranog stroja M. Dokazivanje nerješivosti se provodi tako da se pretpostavi da je nerješivi problem rješiv, pa se na temelju svođenja na kontrapoziciju zaključuje da je problem nerješiv. Upravo se to odvija prilikom pokretanja stroja U. Univerzalni Turingov stroj U može kao ulaz prihvatiti bilo koji rekurzivno prebrojiv jezik pri čemu je taj ulaz ujedno i izlaz iz simuliranog stroja M [1].

Rekurzivno prebrojiv jezik je rekurzivno prebrojiv podskup u skupu svih mogućih riječi unutar abecede jezika Turingovog stroja TS [12]. Klasa rekurzivno prebrojivih jezika je vrlo velika i također uključuje neke jezike za koje se ne može mehanički odrediti spadaju li u tu klasu. Ako Turingov stroj M prihvaća neki proizvoljni rekurzivno prebrojiv jezik $L(M)$, tada se ne smije zaustaviti za bilo koji ulaz koji ne pripada jeziku $L(M)$ [11]. Sukladno tome, ako je ulaz w dio jezika $L(M)$, stroj M se mora zaustaviti. Prilikom samog izračunavanja neke funkcije nije moguće reći hoće li stroj M naposlijetku prihvatiti taj ulaz ili će nastaviti s izvođenjem u beskonačnost.

Za izvođenje bilo kakvih formalnih dokaza potrebno je matematički definirati univerzalni Turingov stroj [7]:

Neka je $\varphi(z,x)$ binarna rekurzivna funkcija. Tada postoji Turingov univerzalni stroj U za koji vrijedi:

$$\Psi_U(z,x) = \varphi(z,x)$$

Takav Turingov stroj možemo koristiti za izračunavanje bilo koje djelomično izračunljive jednomjesne funkcije na idući način:

Bilo koji ulazni skup se Gödelovim kodiranjem transformira u ulazni skup prirodnih brojeva, te se tada ti prirodni brojevi nazivaju Gödelovim brojevima. Neka je Z_0 bilo koji Turingov stroj, s pripadajućim Gödelovim brojem z_0. Tada vrijedi:

$$\Psi_U(z_0,x) = \Psi_{Z_0}(x)$$

Ako se broj z_0 nalazi na traci stroja U, te se na slijedećem polju trake nalazi zapisan broj x_0, Turingov stroj U će izračunati vrijednost funkcije $\Psi_{Z_0}(x_0)$.

Ovaj univerzalni stroj U se može koristiti i za izračunavanje n-arnih funkcija, gdje je $n > 1, n \epsilon \mathbb{N}$ na način da se one najprije transformiraju u jednomjesne funkcije. Takav izračun se izvodi na slijedeći način [2]:

Funkcija $f(x_1, x_2, \dots, x_n)$ se zamjenjuje s $g(x) = (1\ Gl\ x, 2\ Gl\ x, \dots, n\ Gl\ x)$, gdje $n\ Gl\ x$ predstavlja n-ti ulazni element transformiran u Gödelov broj. Kako bi izračunali $f(x_1, x_2, \dots, x_n)$ za dane $x_1, x_2, \dots, x_n$, najprije izračunavamo:

$$x = \prod_{k=1}^{n} \mathrm{Pr}(k)^{x_k}$$

gdje oznaka Pr predstavlja prost broj, a zatim izračunavamo vrijednost $g(x)$.

Pretpostavimo da se abeceda stroja U sastoji od elemenata $\{0,1\}$. Bilo koju konačnu abecedu možemo Gödelovim kodiranjem transformirati na $\{0,1\}$, kao i izlazni rezultat, odnosno izračun funkcije stroja M. Ponašanje svakog stroja se može odrediti na temelju njegove funkcije prijelaza, a abeceda stroja, broj stanja i broj traka se mogu isčitati iz tablice stanja. Također je važno da kod svakog stroja postoje dva stanja koja predstavljaju njegovo početno i završno stanje, q_0 i q_f [9].

Kodiranje i sam rad univerzalnog stroja najlakše je predočiti na Turingovom izvornom primjeru iz 1936. Opis je izveden prema [6]. Potrebno je naglasiti da je kodiranje i sama semantika rada stroja ovdje prikazana nešto drugačije nego što se to kasnije odrađivalo u praksi. Nije izvedeno Gödelovo kodiranje, stoga cijela abeceda stroja nije kodirana na $\{0,1\}$, već postoje i dodatni simboli abecede predstavljeni stringovima i za sam prikaz rada stroja kasnije su razvijeni nešto drugačiji opisi i kod stroja U i kod stroja M. Sam način kodiranja je proizvoljan sve dok jednoznačno vrijedi ono što se opisuje.

Tablica 1: Prikaz konfiguracije stroja M prije kodiranja

Trenutna konfigracija	Simbol trake	Izlazna operacija	Pomak glave	Završna konfiguracija
q_1	b	P_0	R	q_2
q_2	b	E	R	q_3
q_3	b	P_1	R	q_4
q_4	b	E	R	q_1

(Izvor: Petzold C, *The Annotated Turing,* 2008.)

U Turingovom primjeru se na ulazu stroja M nalazi skup koji se kodira u šest simbola koji čine abecedu stroja, točnije, $\Gamma = \{\mathrm{A}, \mathrm{C}, \mathrm{D}, \mathrm{R}, \mathrm{L}, \mathrm{N}\}$. Broj konfiguracije stroja $C_1, C_2, \dots, C_k$ predstavljen je simbolom D koji označava separator koji govori da se radi o početku nekog pojma i simbolom A, gdje broj ponavljanja simbola A predstavlja broj unutarnjeg stanja. Tako

je primjerice stanje stroja q_4 u kodiranom obliku zapisano kao DAAAA. Kodiranje na $\{0,1\}$ se provodi na istom principu, samo pomoću dodatnog simbola C. Sam D predstavlja prazan simbol b, 0 je transformirana u DC, a 1 u DCC. Simbol za pomak glave $\{R, L, N\}$ se ne kodiraju.

Tablica 2: Prikaz konfiguracije stroja M nakon kodiranja

Trenutni konfiguracijski kod	Kod simbola trake	Kod izlazne operacije	Kod pomaka glave	Kod završne konfiguracije	Kodirani izlazni skup
DA	D	DC	R	DAA	DADDCRDAA
DAA	D	D	R	DAAA	DAADDRDAAA
DAAA	D	DCC	R	DAAAA	DAAADDCCRDAAAA
DAAAA	D	D	R	DA	DAAAADDRDA

(Izvor: Petzold C, *The Annotated Turing,* 2008.)

Tada kodirani stroj M koji se simulira na stroju U započinje separatorom ; i u čijem su zapisu koraci algoritma razdvojeni separatorom ; izgleda ovako:

; DADDCRDAA; DAADDRDAAA; DAAADDCCRDAAAA; DAAAADDRDA

Ovi simboli su smješteni na zasebnim poljima. Kada se algoritam stroja M izvede na stroju U, dodani su mu posebni simboli i separatori koji označavaju da se radilo o kodu simuliranog stroja, te nakon izračuna niz znakova izgleda ovako:

ee: : ; : : D: : A: : D: : D: : C: : R: : D: : A: : A: : ; : : D: : A: : A: : D: : D: : R: : D: : A: : A: : A: : ; ...

Dvostruko „e“ označava početak, a dvostruki „:“ predstavljaju separatore koda stroja U. Prilikom izvođenja koda koji se mora interpretirati na drugačiji način, poput operacija stroja M dodaju se posebne oznake $u, v, x, y\ i\ z$ kada je fokus na tom određenom nizu znakova, te se brišu nakon što stroj U prijeđe na idući niz znakova koji ima posebnu interpretaciju. Ako pretpostavimo da stroj U trenutno čita drugu konfiguraciju, znak z predstavlja početak koda druge konfiguracije, x predstavlja kraj koda druge konfiguracije. Slika trake stroja U pri tome izgleda ovako:

ee: : ; : : D: : A: : D: : D: : C: : R: : D: : A: : A: : ; z D: : A: : AxD: : D: : R: : D: : A: : A: : A: : ; ...

Za točnu interpretaciju posebnih oznaka, one sve trebaju biti navedene i ispravno kodirane u tablici stanja stroja U.

3.4. Metoda dokazivanja algoritamske nerješivosti

Specifična metoda koja se koristi kod dokazivanja algoritamske nerješivosti algoritamskih problema poznata je pod nazivom „metoda svođenja“ (eng. *reducibility*). Prije izvođenja samih dokaza, potrebno je obrazložiti i približe iskazati samu metodu čiji logički princip je istovjetan načinu izračunavanja rješenja pomoću univerzalnog Turingovog stroja.

Općenito, metoda svođenja je razvijena u području matematike i ima brojne primjene, primjerice kod polinoma, grupa i teorije skupova. Ideja metode je da se neki specifični problem niže razine apstrahira na neki problem više razine, te se nad njim izvrši željeni dokaz. Zatim se na temelju ispravno izvedene apstrakcije rezultat dokaza problema više razine primjenjuje kao rezultat dokaza problema niže razine. Ako bi ovo opisali pomoću Turingovih strojeva, specifični problem niže razine bi se predstavio pomoću Turingovog stroja U, a problem više razine pomoću stroja M.

Tada postupak dokazivanja metodom svođenja izgleda ovako:

1. problem koji se rješava pomoću stroja M je neki određeni problem koji je dokazano nerješiv i to znamo prije korištenja same metode svođenja
2. pretpostavimo da je problem stroja U rješiv
3. izvršimo apstrahiranje problema stroja U na višu razinu problema stroja M tamo gdje je to moguće, odnosno utvrdimo da algoritamska rješivost problema stroja U povlači algoritamsku rješivost problema stroja M, odnosno $U \rightarrow M$
4. na temelju nerješivosti problema stroja M došlo je do zaključka obratom po kontrapoziciji, $\neg M \rightarrow \neg U$, iz čega slijedi da je i problem stroja U nerješiv jer algoritamska nerješivost problema stroja M povlači algoritamsku nerješivost problema stroja U

U slučaju da se treći korak ne može izvesti, nije moguće riješiti problem stroja U. U drugom, trećem i četvrtom koraku jasno nam je što nam je činiti, no prvi korak može predstavljati problem jer se postavlja pitanje kako utvrditi da je problem stroja M nerješiv. S obzirom na to da dokazujemo algoritamsku nerješivost, posebno nas zanimaju slučajevi kada funkcije stroja nisu rekurzivne. Ovdje je još potrebno razmotriti svojstva rekurzivnih i rekurzivno prebrojivih jezika, te definirati komplementarne jezike:

„Neka je J jezik definiran nad alfabetom Σ. Tada se jezik definiran nad alfabetom Σ sastavljen od preostalih simbola koji ne pripadaju J naziva komplementom jezika J i označava se s J'.

Neka su J i J' komplementarni jezici. Tada postoje tri mogućnosti:

- oba jezika, J i J' su rekurzivni
- nijedan od jezika, ni J ni J' nisu rekurzivno prebrojivi
- jedan od jezika je rekurzivno prebrojiv ali nije rekurzivan, dok drugi nije rekurzivno prebrojiv“ [11]

Ako je neki algoritamski problem rješiv, jezik pomoću kojeg se stroj definira je rekurzivan. Temeljni problem koji će poslužiti kao baza u metodi svođenja definira se pomoću jezika:

$$A_{TS} = \{\langle M, w\rangle \mid M\ je\ Turingov\ stroj\ i\ M\ se\ zaustavlja\ za\ ulaz\ w\}$$

Postoji nekoliko načina na koji možemo dokazati njegovu nerješivost. To se također može izvesti korištenjem univerzalnog Turingovog stroja, a ovdje je izvedeno pomoću metode dijagonalizacije, te će se pomoću toga riješiti prvi korak metode svođenja. Nadalje će metodom svođenja izvesti ostatak dokaza nerješivosti problema zaustavljanja Turingovih strojeva.

S obzirom na navedeno, u ovom koraku je cilj metodom dijagonalizacije dokazati da jezik stroja A_{TS} nije rekurzivan, ali je rekurzivno prebrojiv.

Metodu dijagonalizacije je 1873. godine konstruirao matematičar Georg Cantor kada je razmatrao problem mjerenja veličine beskonačnih skupova [1]. Matematičko rješenje do kojeg je došao jest definiranje funkcije f koja vrši preslikavanje sa skupa A na skup B pod uvjetom da se svaki element iz skupa A preslikava u točno jedan element skupa B, te da svaki element skupa B ima svoj izvorni element u skupu A. Takva funkcija $f: A \rightarrow B$ se naziva bijekcija i pomoću nje se izvodi prebrojavanje beskonačnih skupova. Metoda dijagonalizacije je usko vezana uz prikazivanje bijektivnosti.

Kako bi ovo izveli, na početku je potrebno izvršiti Gödelovo kodiranje i formalno definirati stroj M na idući način [11]:

$$\mathrm{M} = \langle\, \mathrm{Q}, \{0,1\}, \{0,1,\mathrm{b}\}, \mathrm{b}, \mathrm{T}, \mathrm{q}_0, \{\mathrm{q}_1\}, \delta\rangle$$

Abeceda stroja Γ sastoji se od $\{0,1,\mathrm{b}\}$, $\mathrm{Q} = \{\,\mathrm{q}_0, \mathrm{q}_1, \ldots \mathrm{q}_\mathrm{n}\}$ je skup svih unutarnjih stanja stroja pri čemu je q_0 početno stanje, a $\mathrm{q}_\mathrm{f} = \{\,\mathrm{q}_1\}$ jedino konačno stanje. Ako je jezik $L \subseteq (0+1)$ prihvatljiv za bilo koji Turingov stroj TS, tada je prihvatljiv i za stroj M s abecedom $\{0,1,\mathrm{b}\}$ [11]. Prema tome, nema potrebe za više od jednim konačnim stanjem jer će se stroj M zaustaviti kada uđe u stanje prihvaćanja. Neka su $\mathrm{X}_1, \mathrm{X}_2$ i X_3 sinonimi simbola $0,1$ i b i neka su $\mathrm{t}_1, \mathrm{t}_2$ i t_3 sinonimi za $\mathrm{T} = \{\mathrm{L}, \mathrm{R}, \mathrm{N}\}$, pomak glave stroja. Tada se generična funkcija tranzicije može prikazati ovako:

$$\delta(\mathrm{q_i}, \mathrm{X_j}) = (\mathrm{q_k}, \mathrm{X_l}, \mathrm{D_m})$$

Ta funkcija kodirana u binarni string prema gore navedenim pravilima izgleda ovako:

$$kod_p : 0^i 10^j 10^k 10^l 10^m \, , p \in [1, r], p \in \mathbb{N}$$

A binarni kod Turingovog stroja M je:

$$111\ kod_1\ 11\ kod_2\ 11 \ldots kod_{r-1}\ 11\ kod_r\ 111$$

Gdje su „111“ oznake početka i kraja koda stroja M, „11“ separatori, a kod_p je binarni string pomoću kojeg je kodiran svaki korak algoritma izveden Turingovim strojem M. Svaki stroj mora biti kodiran na jedinstven način. Stroj M je izveden na ovaj način kako bi univerzalni stroj U na kojem će se on simulirati mogao točno prepoznati separatore i oznake poletka i kraja. Svi stringovi koje A_{TS} čita kodirani su na gore navedeni način. Kako stroj A_{TS} prima sva ulaza $\langle M, w \rangle$, za svaki se najprije učitava M, a iza njega na traci slijedi w nad kojim je također izvedeno Gödelovo kodiranje na $\{0,1\}$.

Sada kada su definirani osnovni parametri, može se početi provoditi metoda dijagonalizacije. Neka se u retcima nalazi skup binarnih kodova M_j, a u stupcima skup riječi s ulaza problema w_i. Neka postoji tablica s i redaka i j stupaca u kojoj je prikazano za $\forall i, \forall j$ je li riječ w_i dio jezika $L(M_j)$. Ako je tvrdnja istinita i riječ w_i jest dio jezika $L(M_j)$, na dijagonali stoji 1, u suprotnom 0.

Tablica 3: Prikaz dijagonalizacije

$i \backslash j$	1	2	3	...
1	0	0	1	...
2	1	1	0	...
3	0	0	1	...
⋮	⋮	⋮	⋮	⋱

Na temelju vrijednosti koje se nalaze na dijagonalama možemo konstruirati jezik L_d kako bi odredili pripadaju li one tom jeziku. Da bi isključili mogućnost da bilo koji Turingov stroj prihvaća L_d, vrijednost $w_i$može biti riječ jezika L_d ako i samo ako je vrijednost polja (i, i) jednaka 0, kako M_i nebi prihvaćao w_i. Pretpostavimo da je neki Turingov stroj TS koji zadrži $M_j$prihvatio jezik L_d. Tada dolazi do kontradikcije jer ako je w_j u L_d, tada je vrijednost (j, j) jednaka 0, što znači da $w_j$nije u $L(M_j)$, što je u kontradikciji s $L_d = L(M_j)$. Ako pretpostavimo da $w_j$nije riječ jezika L_d, tada je vrijednost (j, j) jednaka 1, što povlači da je w_j dio $L(M_j)$, te je

to opet u kontradikciji s $L_d = L(M_j)$. Kako w_j jedino može ili ne može biti dio L_d, pretpostavka $L_d = L(M_j)$ je pogrešna. Prema tome, ne postoji Turingov stroj za koji je L_d prihvatljiv [11].

Ovime je dokazano da jezik L_d nije rekurzivno prebrojiv. To znači da nije prihvatljiv za Turingov stroj M i da se Turingov stroj M neće zaustaviti. Ako na temelju ovih rezultata konstruiramo novi Turingov stroj U koji prepoznaje jezik A_{TS} za koji vrijedi:

$U = Za\ svaki\ \ ulaz\ \langle M, w\rangle, gdje\ je\ M\ Turingov\ stroj, a\ w\ string$:

1. $Pokreni\ M\ za\ ulaz\ w$

2. $Ako\ M\ uđe\ u\ stanje\ prihvaćanja, prihvati, a\ ako\ M\ uđe\ u\ stanje\ odbijanja, odbij$

Možemo zaključiti da stroj U ulazi u petlju za ulaz $\langle M, w\rangle$ u slučaju da M ulazi u petlju za ulaz w. Iz ovoga slijedi da stroj U ne može rješiti problem stroja A_{TS}. Ako bi u nekom slučaju postojao korak algoritma na temelju kojeg se može odrediti da se M ne zaustavlja za w, stroj U bi mogao odbiti takav ulaz i zaustaviti se. Kako to nije slučaj, ovime je dokazano da je problem stroja A_{TS} nerješiv algoritamski problem [1,9].

4. Dokazivanje algoritamske nerješivosti

U ovom poglavlju prikazuje se korištenje metode dokazivanja algoritamske nerješivosti formirane u prethodnom poglavlju nad dokazivanjem nerješivosti nerješivih masovnih algoritamskih problema.

4.1. Problem zaustavljanja Turingovog stroja

U prijašnjem poglavlju je dokazano da je problem stroja A_{TS} nerješiv masovni algoritamski problem. Na njega se nadovezuje problem stroja $HALT_{TS}$ gdje se pitamo hoće li se Turingov stroj zaustaviti za dani ulaz. Cilj dokaza je svesti problem stroja A_{TS} na $HALT_{TS}$ kako bi pomoću njegove nerješivosti dokazali nerješivost problema stroja $HALT_{TS}$. Neka je:

$$HALT_{TS} = \{\langle M, w\rangle \mid M\ je\ Turingov\ stroj\ i\ M\ se\ zaustavlja\ za\ ulaz\ w\}$$

I u ovom slučaju, srž dokaza provodi se svođenjem na kontradikciju. Pretpostavimo da je $HALT_{TS}$ rješiv masovni algoritamski problem. Ako bi problem A_{TS} sveli na $HALT_{TS}$, korištenjem ove pretpostavke mogi bi dokazati da je A_{TS} rješiv.

Za nastavak dokaza potrebno je konstruirati rješavače (eng. *deciders*). Rješavači su vrsta Turingovih strojeva koji nikada ne smiju ući u petlju, odnosno moraju se uvijek zaustaviti na temelju ulaza kojeg prihvaćaju ili odbijaju. Iz tog razloga se još kaže da rješavaju problem ulaza koji im je dan.

„$Neki\ jezik\ je\ rješiv\ ili\ rekurzivan\ ako\ postoji\ Turingov\ stroj\ koji\ ga\ rješava.$“ [1]

Pretpostavimo da postoji Turingov stroj R koji rješava problem za $HALT_{TS}$. Ako imamo takav stroj, možemo ga iskoristiti za konstrukciju stroja S koji rješava A_{TS}. Ako povućemo paralelu sa univerzalnim Turingovim strojevima, u ovom slučaju je univerzalni stroj U predstavljen strojem S, a stroj M je predstavljen strojem R. Sa strojem R možemo testirati zaustavlja li se M za ulaz w:

$R = Za\ svaki\ \ ulaz\ \langle M, w\rangle, gdje\ je\ M\ Turingov\ stroj, a\ w\ string$:

1. $Pokreni\ M\ za\ ulaz\ w$

2. $Ako\ se\ M\ zaustavi, prihvati, a\ ako\ M\ uđe\ u\ petlju, odbij$

Na ovaj način, stroj R će se uvijek zaustaviti. Ako je R ušao u stanje odbijanja, ulaz $\langle M, w\rangle$ nije dio jezika stroja A_{TS}, a ako je R ušao u stanje prihvaćanja, može se napraviti željena simulacija bez neprihvatljive mogućnosti da stroj R uđe u petlju. Pod ovim pretpostavkama, konstruirajmo sada stroj S sa slijedećim postavkama:

$S = Za\ svaki\ \ ulaz\ \langle M, w\rangle, gdje\ je\ M\ kod\ Turingovog\ stroja, a\ w\ string$:

1. $Pokreni\ Turingov\ stroj\ R\ za\ ulaz\ \langle M, w\rangle$

2. $Ako\ R\ odbija, odbij$

3. $Ako\ R\ prihvaća, simuliraj\ M\ za\ ulaz\ w\ sve\ dokad\ se\ ne\ zaustavi$

4. $Ako\ M\ prihvaća, prihvati, a\ ako\ M\ odbija, odbij$

Ovime je izvršen treći korak metode svođenja, stroj R je uspješno apstrahiran na stroj S. Jasno je da ako R rješava problem stroja $HALT_{TS}$, tada S rješava A_{TS}. Rješivost jednog povlači rješivost drugog. U četvrtom koraku svođenja vraćamo se na prijašnje poglavlje gdje je dokazano da je A_{TS}, a kako nerješivost stroja A_{TS} povlači nerješivost stroja $HALT_{TS}$, svođenjem na kontrapoziciju zaključujemo da je i problem stroja $HALT_{TS}$ nerješiv masovni algoritamski problem.

4.2. Problem odlučivanja logike prvog reda

Problem odlučivanja logike prvog reda odnosi se na pitanje rješavanja je li neki logički izraz istinit ili lažan. Univerzalni Turingov stroj u ovom slučaju kao ulazne podatke uzima opis formalnog jezika i željeni logični izraz. Ovaj je problem izvorno postavio David Hilbert 1928. godine pod nazivom „das Entscheidungsproblem“. Desetak godina kasnije Turing i Crurch su neovisno jedan od drugog u svojim radovima pomoću različitih metoda dokazali da je ovaj problem rekurzivno nerješiv [12].

Povezanost logike i teorije izračunljivosti u Turingovim radovima obilježili su korijene razvoja modernih računala. Turing je u svojem radu prikazao da se koraci svakog algoritma mogu opisati logičkim izrazima. Računala su potpuno deterministički strojevi čiji rad je vođen taktom. Ako je u jednom taktu zadana jedna konfiguracija, u idućem je u potpunosti određena, a veza između njih se može logički opisati. Turing je sveo ovaj problem na problem zaustavljanja, što će biti odrađeno u ovom dokazu.

Glavni cilj ovog dokaza je premostiti razlike u samim prirodama problema. Problem zaustavljanja spada u teoriju izračunljivosti, dok je problem u odlučivanju logike prvog reda hibridan problem koji ispituje mogućnost primjene teorije izračunljivosti na logičke probleme, vrši se svođenje s problema izračunljivosti na logiku. Ako se na neki način problem odlučivanja logike prvog reda dovoljno dobro formalizira Turingovim strojevima, preostaje samo primijeniti metodu svođenja. Ovaj dokaz i njegovo izvođenje vrlo su nalik problemu i dokazu kod Cook-ovog teorema, gdje je glavno pitanje može li se zadovoljivost propozicijskog računa potvrditi u polinomnom vremenu [12].

Logika prvog reda zadana je skupom funkcijskih simbola i skupom simbola predikata. Svaki simbol dolazi s određenom potencijom koja je iz skupa prirodnih brojeva, a funkcijski simboli s potencijom 0 predstavljaju konstante. Izrazi se definiraju rekurzivno, tako da su predstavljeni uvjetima. Funkcije se grade od uvjeta, a svaka funkcija je opet izraz. Formule se grade od izraza i simbola predikata. Negacija formule je također formula, konjunkcije, disjunkcije, implikacije i ekvivalencije formula su također formule. Kvantifikatori koji se odnose na određene varijable također predstavljaju formule. Na ove osnove se nadovezuju zakoni logike prvog reda i Gödelov teorem o potpunosti koji govori da je tvrdnja dokaziva prema zakonima, odnosno sukladno izabranoj aksiomatizaciji, ako i samo ako je istinita za sve moguće interpretacije. Na temelju ovoga se izvode dokazi istinitosti tvrdnji. Cilj nam je dokazati da je problem odlučivanja logike prvog reda nerješiv, što znači da ne postoji jedinstveni, univerzalni algoritam koji bi za svaku formulu logike prvog reda bio u stanju utvrditi je li ona istinita u bar jednoj svojoj interpretaciji.

Dokaz je izveden prema [2,7,13]. Za početak je potrebno definirati nešto što će predstavljati univerzalni Turingov stroj iz prijašnjih dokaza pomoću kojeg će se predstaviti problem za koji ćemo pretpostaviti da je rješiv. U tu svrhu, pretpostavimo da imamo neki algoritam A koji će za danu deklaraciju logike prvog reda i neku formulu koja pripada toj logici rješavati problem odlučivanja istinitosti te formule za sve moguće interpretacije.

Neka postoji Turingov stroj koji će za dani ulaz $\langle M, w\rangle$ kreirati logiku prvog reda tako da će deklarirati konstantu λ koja će predstavljati prazni simbol b, unarnu funkciju a za svaki simbol abecede stroja M uz koji se vežu varijable logičkih izraza i binarni predikat f_q za svako stanje q stroja M. Na taj način, $a(w)$ obilježava string aw, a $f_q(x, y)$ označava da stroj M za dani ulaz w može prijeći u konfiguraciju koja ima stanje q, u kojoj se na traci nalaze simboli x i y na način da je x zapisan obrnutim redoslijedom, a y označava početnu poziciju za glavu stroja M.

Pod ovim postavkama, možemo biti sigurni da je izraz $f_{q_0}(\lambda, w)$ istinit, gdje q_0 predstavlja početno stanje stroja, a w je prikaz simbola w sačinjen od simbola konstanta i funkcija dotične logike, te da je početna konfiguracija stroja M ostvariva.

U tom slučaju ako i samo ako stroj M prihvaća ulaz w, vrijedi:

$$\exists x \exists y\colon f_a(x, y),$$ gdje f_a predstavlja konačno stanje prihvaćanja.

Funkcija prijelaza δ se neovisno o akciji, prilikom pomaka udesno predstavlja idućom formulom:

$$\forall x \forall y\colon f_{q_i}(x, ay) \rightarrow f_{q_{i+1}}(bx, y)$$

gdje je q_i neko stanje nakon kojeg stroj prijelazi u stanje q_{i+1}, a je simbol koji se čita s trake, b je simbol koji se zapisuje na traku, → označava logičku implikaciju, a x i y varijable. Sukladno tome i imajući na umu postavke rada stroja, pomak ulijevo izgleda ovako:

$$\forall x \forall y\colon f_{q_i}(cx, ay) \rightarrow f_{q_{i+1}}(x, cby)$$

gdje je c simbol koji je nalazi na lijevom polju. U slučaju da se glava stroja nalazi na krajnjem lijevom polju, pomak ulijevo ima oblik:

$$\forall x \forall y\colon f_{q_i}(\lambda, ay) \rightarrow f_{q_{i+1}}(\lambda, by)$$

U slučaju da je neko polje posjećeno prvi puta, a a je prazan simbol, za svaki od tri navedena slučaja postoje posebne varijante formule gdje se y zamjenjuje s λ.

Neka je K konjunkcija svih gore navedenih izraza sa svim mogućim kombinacijama varijabli. Kako stroj M ima konačno mnogo prijelaza stanja i konačnu abecebu, ukupan broj logičkih implikacija je također konačan, te stoga čini formulu logike prvog reda. Tada cjelokupni rad stroja M za određeni ulaz w možemo prikazati pomoću jedne formule:

$$f_{q_0}(\lambda, w)\ i\ K \rightarrow \exists x \exists y\colon f_a(x, y)$$

Ako i samo ako stroj M prihvaća ulaz w, gore navedeni izračun također dovodi do stanja prihvaćanja. Ovo se ne može odrediti na mehanički način, stoga je potrebno provjeriti je li gore navedeni izraz istinit kako bi mogli rješiti problem zaustavljanja stroja M za dani ulaz w. U ovom slučaju metoda svođenja izgleda ovako:

1. Problem zaustavljanja je dokazano nerješiv i to znamo prije korištenja same metode svođenja.
2. Pretpostavimo da je problem odlučivanja rješiv, odnosno da postoji algoritam A kojim možemo utvrditi istinitost logičkog izraza.
3. Izvršiti apstrahiranje problema odlučivanja na problem zaustavljanja i utvrditi da algoritamska rješivost problema odlučivanja povlači algoritamsku rješivost problema zaustavljanja.
4. Nakon završetka točke 3. vidimo da ne možemo znati prihvaća li stroj M ulaz w, te na temelju nerješivosti zaključujemo da je problem odlučivanja nerješiv.

Kako smo prije dokazali nerješivost problema zaustavljanja, u ovom slučaju je bilo potrebno pretpostaviti da postoji algoritam A i ispravno izvesti apstrahiranje da bi mogli zaključiti nerješivost problema odlučivanja. Na taj način je dokazano da se ne može konstruirati algoritam koji će rješavati baš svaki matematički problem.

5. Zaključak

Nerješivi masovni algoritamski problemi su vrlo brojni. Bilo kakvo općenito problemsko pitanje koje zahtjeva konkretan odgovor tipa da/ne se u slučaju njegove nerješivosti može određenim metodama formalizirati na jedan nerješivi algoritamski problem. Nikad ne možemo pomoću bilo kakvog stroja ili izračuna sa sigurnošću tvrditi o istinitost odgovora na to pitanje.

Vrlo snažan postupak prije samog dokazivanja jest formalizacija pitanja i postupka traženja odgovora pomoću Turingovog stroja. Sama činjenica da završno stanje može biti stanje prihvaćanja ili odbijanja ili ulazak u beskonačnu petlju daje nam za razmišljati o mogućnostima ishoda. Ne samo da postoji opcija kada je odgovor negativan, već postoji nešto kompleksnije, a to je da se odgovor uopće ne može dati, kada stroj ulazi u beskonačnu petlju. U slučaju da želimo to dokazati, koristimo metodu svođenja. Vrlo snažan univerzalni Turingov stroj zapravo predstavlja metodu svođenja formaliziranu u ovoj domeni.

Kako ne možemo dati klasifikaciju rješivih i nerješivih problema, potrebno je za svaki problem pretpostaviti da je rješiv, te zatim pokušati utvrditi i dokazati njegovu nerješivost. Osnovni problem koji služi za bilo kakve daljnje manipulacije u ovoj domeni jest problem zaustavljanja Turingovih strojeva koji služi kao baza dokaza. Za dokazivanje njegove nerješivosti možemo se poslužiti brojnim metodama, no od najveće je važnosti da se to izvede korektno. Nadogradnja dokaza nerješivosti nad dokazom nerješivosti izvodi se rekurzivno, te se tako mogu dokazivati sve složeniji masovni algoritamski problemi. Temelj svega čine teorija automata i teorija izračunljivosti pomoću kojih se definiraju apstraktni izvršitelji, jezici i korektni postupci.

Uvid u nerješive masovne algoritamske probleme nam daje sliku o ograničenosti dubljeg razumijevanja računarstva kao takvog i s obzirom na to, može se činiti nerealno koliko se ono uspjelo daleko razviti do danas. Posljednji dokaz postavlja značajnu teoretsku granicu u području kompjuterizacije matematike i govori da će uvijek postojati neki problemi za koje se ne može izvesti nikakva metoda pronalaska njihovog rješenja na masovnoj osnovi. Nema matematičkih metoda koje bi se mogle primijeniti kako bi nam pružile odgovore čak i na neka elementarna pitanja, već se samo postavljaju dodatne granice iako se sama logika problema naizgled može činiti vrlo jednostavnom. Činjenica da nije moguće dokazati nijednu formalnu definiciju algoritma ostavlja nas u propitkivanju koliko su naše spoznaje zapravo oskudne i hoće li se ikad premostiti granica između ljudskog intuitivnog i računalnog logičnog i hoće li se ikad pomoću jednog moći u potpunosti pojmiti drugo.

Popis literature

[1] Spiser M (1997) *Introduction to the Theory of Computation*. Worcester: PWS Publishing

[2] Uspensky VA, Semenov AL (1993) *Algorithms: Main Ideas and Aplications* (Udžbenik Sveučilišta Lomonosov u Moskvi). Norwell, Massachusetts: Kluwer Academic Publishers

[3] Lovrenčić A (2015) *Uvod u složenost algoritama* (Priručnik Fakulteta organizacije i informatike Sveučilišta u Zagrebu). Varaždin

[4] Knuth DE (1968) *The Art of Computer Programming*. Boston, Massachussets: Addison-Wesley

[5] Yandell B (2002) *The Honors Class*. Natick: AK Peters

[6] Petzold C (2008) *The Annotated Turing*. Hoboken, New Jersey: John Wiley & Sons

[7] Davis M (1958) *Computability and Unsolvability*. New York: McGraw Hill

[8] Ribarić S (2011) *Građa računala - Arhitektura i organizacija računarskih sustava* (Udžbenik sveučilišta u Zagrebu). Zagreb: Algebra

[9] Smid M, Maheshwari A (2017) *Introduction to the Theory of Computation* (Udžbenik sveučilišta Carleton). Ottawa

[10] Arora S, Barak B (2009) *Computational Complexity: A Modern Approach* (Udžbenik sveučilišta Princeton) New Jersey: Princeton University Press

[11] Hopcroft JE, Ullman JD (1979) *Introduction to Automata Theory, Languages and Computation.* Boston, Massachussets: Addison-Wesley

[12] Herken R (1995) The Universal Turing Machine: A Half-Century Survey. New York: Springer-Verlang Wien

[13] Van Glabbeek R, *The Undecidability of First Order Logic* (materijali sveučilišta Stanford), dostupno putem: http://kilby.stanford.edu/~rvg/154/handouts/fol.html

Popis slika

Popis tablica