SVEUČILIŠTE U ZAGREBU

FILOZOFSKI FAKULTET

ODSJEK ZA FILOZOFIJU

AKADEMSKA GODINA 2012. / 2013.

Teorija izračunljivosti i neodlučivost logike prvog reda

DIPLOMSKI RAD

MENTOR: Prof. dr. sc. Davor Lauc

STUDENT: Željko Brcković

ZAGREB 2013.

Sadržaj

I. Uvod 4

II. Izračunljivost i rekurzija 5

III. Efektivni postupak 8

IV. Turingov stroj 9

1. Definicija 9

2. Dijagonalna funkcija, problem zaustavljanja i neizračunljivost 14

3. Proširenja Turingovog stroja 18

a) Proširenje alfabeta 18

b) Uvođenje većeg broja završnih stanja 19

c) Uvođenje dodatnih traka 20

4. Redukcije Turingovog stroja 22

a) Uklanjanje upute S 23

b) Redukcija trake 23

V. Registarski stroj 24

VI. Rekurzivne funkcije 27

VII. Ekvivalencija različitih formalizacija izračunljivosti 30

VIII. Formalni jezici 34

1. Osnovne definicije i terminologija 34

2. Formalni jezik i gramatika 34

IX. Ekvivalencija Turingovog stroja i formalne gramatike 37

X. Logika prvog reda 41

1. Uvod 41

2. Jezik logike prvog reda 43

a) Sintaksa 43

b) Semantika 46

XI. Problem odlučivosti logike prvog reda 47

Literatura 55

I. Uvod[1]

Teorija izračunljivosti je područje matematičke logike koje se počelo razvijati početkom tridesetih godina dvadesetog stoljeća. Polazišna točka teorije izračunljivosti je formaliziranje pojma efektivnog postupka. Pod efektivnim postupkom (algoritmom)[2] intuitivno shvaćamo postupak kojim polazimo od nekih inicijalno zadanih objekata (ulaznih podataka) i nakon konačnog broja koraka dolazimo do rezultata, a pritom provodimo unaprijed eksplicitno zadana pravila (upute). Ovu intuitivnu definiciju u teoriji izračunljivosti zamjenjujemo formalnom, a to otvara put konkretnijem razmatranju dosega efektivnog rješavanja problema. Prihvaćanjem formalne definicije efektivnog postupka brzo dolazimo do prvog važnog rezultata teorije izračunljivosti – otkrića „nerješivih" problema.

Već na samom početku razvoja ove discipline pojavilo se više različitih formalizacija efektivnog postupka[3]. Mi ćemo se u prvom redu baviti Turingovom formulacijom izračunljivosti putem apstraktnih strojeva, ali ćemo uz to razmotriti i pojam registarskog stroja i rekurzivne funkcije. Svaka od navedenih formulacija predstavlja drugačiji pristup problemu efektivne izračunljivosti i one su nastale u različitim povijesnim kontekstima. Bez obzira na te razlike, razmatranjem odnosa između ovih formulacija dolazimo do zaključka da su one međusobno ekvivalentne.

Nakon toga privremeno prelazimo na drugu temu – formalne jezike. Koncept formalnog jezika je u matematici i logici poznat dugo, ali se teorija formalnih jezika počela ozbiljnije razvijati tek pedesetih godina dvadesetog stoljeća u sklopu lingvističkih istraživanja Noama Chomskog. On je uz pomoć formalnih jezika pokušao dati precizan opis strukture prirodnog jezika. Od Chomskijevih radova potječe i koncept formalne gramatike koji ćemo detaljnije istražiti. Formalna gramatika je aparat kojim se specificira formalni jezik. Rangiranjem formalnih gramatika po složenosti Chomsky je stvorio hijerarhiju prema kojoj su formalne gramatike

[1] Sadržaj poglavlja o izračunljivosti i formalnim jezicima je već prikazan u mojem drugom diplomskom radu: Brcković Ž.; Rekurzivno prebrojivi jezici i Turingov stroj – diplomski rad; Filozofski fakultet, Zagreb, 2013. Temeljna ideja iza ovog rada je primjena tih sadržaja na problem odlučivosti za logiku prvog reda.

[2] Termine "algoritam" I "efektivni postupak" koristim kao sinonime.

[3] Na primjer: Gödel-Herbrandov pojam generalno rekurzivne funkcije, Churchov pojam λ-definabilne funkcije, Turingov pojam izračunljive funkcije.

podijeljene u četiri tipa. Iako svaki od ovih tipova ima veliku važnost u praksi, mi ćemo se koncentrirati na tip najsloženijih gramatika u hijerarhiji – gramatike bez ograničenja. Jezici koji su opisani takvim gramatikama čine skup rekurzivno prebrojivih jezika.

Sada povezujemo rezultate teorije izračunljivosti i teorije formalnih jezika prikazom formalne gramatike kao moguće formulacije izračunljivosti. Zatim izvodimo dokaz ekvivalencije između gramatika i Turingovih strojeva.

Nakon što smo obradili osnovne definicije i ustanovili bitna svojstva različitih formulacija izračunljivosti, primjenjujemo te rezultate na jedan konkretan primjer od velikog povijesnog značaja – problem odlučivosti za logiku prvog reda. Radi se o problemu pronalaska efektivnog postupka kojim bi se moglo za bilo koju formulu logike prvog reda ustanoviti da li je valjana (logički istinita). Problem je 1928. godine predstavio David Hilbert i okarakterizirao ga temeljnim problemom matematičke logike. Dokazano je da takav postupak nije obuhvaćen niti jednom od navedenih formalnih definicija izračunljivosti, a to sugerira da nije efektivno rješiv ni u intuitivnom smislu. U ovom radu je prikazana struktura dokaza neodlučivosti uzimanjem u obzir formalne gramatike kao modela izračunljivosti. Preciznije rečeno, problem odlučivanja o pripadnosti niza nekom formalnom jeziku definiranom gramatikom je sveden na problem odlučivanja o valjanosti implikacije. Na temelju nerješivosti prvog problema zaključujemo da je nerješiv i drugi.

II. Izračunljivost i rekurzija

Teorija izračunljivosti se često naziva i teorijom rekurzije. Međutim, uzimajući u obzir značenja termina „izračunljivost“ i „rekurzija“ teško je vidjeti gdje je poveznica. Ona se ne može razumjeti bez shvaćanja povijesnog konteksta u kojem je teorija izračunljivosti, odnosno teorija rekurzije nastala.[4]

Intuitivni pojam **izračunljivosti** u smislu postojanja efektivnog postupka za rješavanje određenog problema poznat je još od davnina. Već kod Euklida (330. pr. Kr.) nalazimo poznati

[4] Soare, R. I.; The History and Concept of Computability // *Handbook of Computability Theory* / ur. Eward R. Griffor; Elsevier Science B. V., 1999; Str. 3 - 36

primjer algoritma za pronalaženje najvećeg zajedničkog djelitelja. Daljnjim razvojem teorijske matematike i otkrivanjem raznih efektivnih postupaka za rješavanje matematičkih problema rastao je i interes za sam princip računanja. 1642. godine Blaise Pascal je osmislio računalni stroj za zbrajanje koji se može smatrati prvim digitalnim računalom. Gottfried Wilhelm Leibniz je u 17. stoljeću razmatrao koncept univerzalnog računa kojim bi se paradigma algoritamskog rješavanja problema mogla primijeniti i na mnogo šire područje od matematike. 1834. godine Charles Babbage je razvio ideju „analitičkog stroja“ - programibilnog stroja koji može provoditi širok spektar računskih operacija.

Pojam **rekurzije** s druge strane naizgled nema velike veze s pojmom izračunljivosti. Riječ „rekurzija“ dolazi od latinske riječi „recursio“ što znači „vratiti se“. Kada govorimo o rekurziji u matematici, mislimo na princip definiranja nekog postupka (funkcije) pozivajući se na taj isti postupak. Takve definicije još nazivamo i induktivnim definicijama. One na prvi pogled djeluju nevaljano jer krše princip ne-cirkularnosti ali to ne predstavlja nikakav problem. Rekurzivna definicija nekog postupka se uvijek sastoji od dva glavna djela. Prvi dio definicije govori kako postupak primijeniti na neki skup najjednostavnijih instanci problema i on nije cirkularan. Drugi dio definicije govori kako postupak primijeniti na bilo koju složeniju instancu problema pritom pretpostavljajući da znamo kako postupak primijeniti na jednostavnije instance. Ilustrirat ćemo to na primjeru rekurzivne definicije funkcije *faktorijel*. Intuitivno ovu funkciju možemo razumjeti na sljedeći način: $f(x) = x! = 1 * 2 * 3 \dots * x$. Evo rekurzivne definicije:

$$f(1) = 1$$
$$f(x+1) = f(x) * (x+1)$$

U prvom redu je prikazano kako doći do rezultata kada funkciju primijenimo na najjednostavniju instancu problema (broj 1). U drugom redu je prikazano kako funkciju primijeniti na bilo koji veći broj $x + 1$ ako već znamo kako funkciju primijeniti na manji broj x. Postupak računanja vrijednosti funkcije za broj 3 izgleda ovako:

$$f(3) = f(2) * 3 = (f(1) * 2) * 3 = (1 * 2) * 3 = 6$$

Primjere rekurzivnih definicija osnovnih aritmetičkih funkcija možemo naći već u radovima Dedekinda s kraja 19. stoljeća. Iz istog vremena potječe i poznata aksiomatizacija aritmetike Giuseppea Peana gdje je eksplicitno prikazan princip rekurzivnog definiranja funkcija.

Pojmovi rekurzije i izračunljivosti su dovedeni u vezu tek tridesetih godina 20. stoljeća. Razvoj u matematici koji je do toga doveo je bio iniciran radom Davida Hilberta s početka 20. stoljeća. Hilbert je predložio da se svi matematički dokazi reduciraju na puke mehaničke postupke manipuliranja sa znakovima lišenih bilo kakvog intuitivnog značenja. Svrha ovog projekta bila je omogućiti dokaz konzistentnosti aritmetike. Naime, svođenjem matematičkog dokaza na konačne precizno definirane mehaničke postupke onemogućila bi se pojava kontradikcije u aritmetici. Hilbertov program je time potaknuo interese matematičara za detaljnije istraživanje pojma mehaničkog postupka i pronalazak njegove preciznije formulacije.

Jednu takvu formulaciju je djelomično razvio Kurt Gödel u sklopu svojeg „dokaza o nepotpunosti" iz 1931. godine[5], a kasnije je doradio prema sugestijama Herbranda. Gödel i Herbrand su ponudili rekurzivnu definiciju svih efektivno izračunljivih funkcija u aritmetici. Takve funkcije je Gödel nazvao „generalno rekurzivnim funkcijama". U to vrijeme pojavile su se i druge formulacije izračunljivih funkcija koje su se također bazirale na principu rekurzije.[6] 1930. Alonzo Church je razmatrao skup λ-definabilnih funkcija i predložio je da se s njim poistovjeti skup svih funkcija koje su izračunljive u intuitivnom smislu. Taj prijedlog je danas poznat pod nazivom „Churchova teza". 1936. godine Alan Turing je ponudio novu formulaciju efektivnog postupka koja predstavlja potpuno drugačiju paradigmu u teoriji izračunljivosti. On je razmatrao hipotetske strojeve (agente) koji manipuliraju znakovima prema unaprijed precizno zadanim pravilima. Turing je smatrao da će se takvim strojevima moći izračunati sve funkcije koje su izračunljive u intuitivnom smislu (Turingova teza). Iako je Turingova formulacija izračunljivosti potpuno drugačija od onih prethodnih, dokazano je da se one ipak podudaraju. Drugim riječima, funkcije koje su definirane kao generalno rekurzivne

[5] Gödel, K.; On Formally Undecidable Propositions of Principia Mathematica and Related Systems; Dover Publications Inc., 1992.

[6] λ-definabilne funkcije (Church), μ-rekurzivne funkcije (Kleene)

(Gödel/Herbrand), λ-definabilne (Church) ili μ-rekurzivne funkcije (Kleene) su iste one koje su izračunljive na Turingovom stroju.

Na taj način je stvorena veza između pojmova rekurzije i izračunljivosti koja se održala do danas u sinonimima „teorija rekurzije“ i „teorija izračunljivosti“. Iako su oba termina još uvijek u upotrebi, u nastavku ćemo koristiti samo termin „teorija izračunljivosti“.

III. Efektivni postupak

Prije nego razmotrimo različite formalizacije pojma efektivnog postupka, prvo moramo konkretizirati što točno pod time mislimo. Ranije smo rekli da pod **efektivnim postupkom** intuitivno razumijemo postupak provođenja eksplicitno zadanih pravila. Postupak polazi od zadane instance određenog problema i nakon konačnog broja koraka dovodi do njegovog rješenja. Problem koji se može riješiti takvim postupkom ćemo zvati **efektivno rješivim problemom**. Ovdje nismo precizno specificirali što razumijemo pod „problemom“. Problema s kojima se susrećemo u matematici i logici ima najrazličitijih vrsta. Bez obzira na to, ako je neki problem efektivno rješiv, možemo očekivati da će se različite instance toga problema i njihova rješenja moći precizno izraziti u određenoj notaciji. Kada to prihvatimo, svaki problem se svodi na neku funkciju koja preslikava iz skupa ulaznih podataka (prezentacija različitih instanci problema u određenoj notaciji) u skup izlaznih podataka (prezentacije rješenja toga problema za svaku njegovu instancu). Notacije u kojima će biti izraženi ulazni i izlazni podaci mogu biti različite, ali se sve mogu svesti na jednu: Izraze u bilo kojoj notaciji uvijek ćemo moći kodirati prirodnim brojevima. Zgodnu primjenu takvog kodiranja nalazimo u već spomenutom Gödelovom dokazu nepotpunosti iz 1931. godine.[7] Gödelova ideja je bila da se prvo svakom znaku u notaciji pridruži jedinstven prirodni broj. To će biti kodni broj toga znaka. Zatim se svakom izrazu u notaciji također pridružuje jedinstven kodni broj. To se čini na sljedeći način: Neka su znakovi od kojih se sastoji proizvoljni izraz duljine n kodirani brojevima $x_1, x_2 \dots x_n$. Uzmimo sada prvih n prostih brojeva: $p_1, p_2, \dots p_n$. Kodni broj izraza će biti ${p_1}^{x_1} * {p_2}^{x_2} * \dots {p_n}^{x_n}$. Činjenica da svaki prirodni broj ima jedinstvenu rastavu na proste faktore garantira

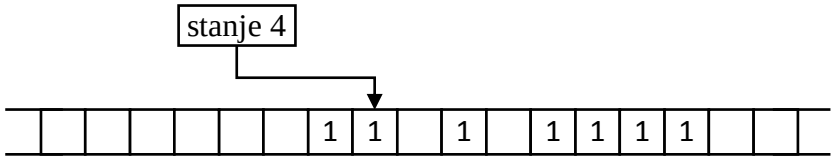
[7] Gödel, K.; On Formally Undecidable Propositions of Principia Mathematica and Related Systems; Dover Publications Inc., 1992.

nam da će svaki izraz dobiti jedinstven kodni broj. Idejom takvoga kodiranja je uvedeno jedno bitno pojednostavljenje - svaki problem formuliran u bilo kojoj notaciji je sada svodiv na funkciju nad prirodnim brojevima, a pojam efektivnog postupka u širem smislu je sveden na pojam računanja vrijednosti takvih funkcija. Za funkciju (nad prirodnim brojevima) kažemo da je **efektivno izračunljiva** (ili jednostavno izračunljiva) ako postoji efektivni postupak koji nas vodi od argumenta funkcije do njezine vrijednosti. Ako precizno odredimo granicu između izračunljivih i neizračunljivih funkcija nad prirodnim brojevima, time ćemo indirektno odrediti i granicu između efektivno rješivih i nerješivih problema u širem smislu. To nam dozvoljava da se u nastavku posvetimo samo funkcijama nad prirodnim brojevima i zanemarimo pojam problema u širem smislu.

IV. Turingov stroj

1. Definicija

Turingov stroj je apstraktni matematički model računala. On funkcionira na temelju konačnog skupa vrlo jednostavnih pravila, ali njegove mogućnosti se uzimaju kao univerzalni model bilo kakvog efektivnog postupka. Počet ćemo s neformalnom definicijom: Turingov stroj se sastoji od tri glavna djela: trake, glave za čitanje/pisanje i jedinice za kontrolu stanja. Traka je beskonačna u oba smjera i podijeljena u polja. Svako polje na traci može biti prazno ili sadržavati znak 1. Ponekad ćemo reći da prazna polja sadrže „prazan“ znak 0. Na traci uvijek može biti samo konačan broj jedinica iz čega slijedi da će uvijek biti beskonačno mnogo praznih polja.



Za Turingov stroj kažemo da se u svakom trenutku nalazi u jednom od konačno mnogo stanja. Glava Turingovog stroja se u svakom trenutku nalazi nad jednim od polja na traci. Kažemo da Turingov stroj (ili njegova glava) „pokazuje“ na to polje. Ovisno o sadržaju toga polja i stanju

u kojem se Turingov stroj nalazi, pokrenut će se niz od tri radnje. Prva radnja će biti jedna od sljedećih:

1. Upisivanje jedinice u pokazano polje (ako je u tom polju već jedinica onda se ne čini ništa)
2. Brisanje jedinice u pokazanom polju (ako je polje već prazno onda se ne čini ništa)

Druga radnja će biti jedna od sljedećih:

1. Pomicanje glave za jedno polje udesno
2. Pomicanje glave za jedno polje ulijevo
3. Ostajanje na mjestu

Zadnja radnja će biti prijelaz u novo stanje ili ostanak u istom. Koji niz od tri radnje će se dogoditi, to je jasno specificirano pravilima. Svako pravilo govori koji niz radnji se treba provesti ovisno o trenutnom stanju i znaku u pokazanom polju.

Na primjer, jedno od pravila može glasiti: „Ako se nalaziš u stanju 4 i na polju je jedinica, briši jedinicu, pomakni glavu ulijevo i ostani u istom stanju“. Na taj način je u svakom trenutku radnja Turingovog stroja u potpunosti determinirana pravilima. Zato se kaže da je Turingov stroj u ovoj formulaciji **deterministički**. Kako Turingov stroj ne bi morao raditi u beskonačnost, uvijek će postojati jedno **završno** stanje za koje nema specificiranih uputa. Kada ga dosegne, Turingov stroj će prestati s radom.

Ovo neformalno objašnjenje ćemo sada konkretizirati formalnom definicijom.[8] Turingov stroj M je par $M = (Q, \delta)$ gdje je:

> Q konačan skup stanja koja će inače biti označena nizom znaka: $q_0, q_1, \ldots$ Prvo stanje u nizu (q_0) će uvijek biti početno, a zadnje stanje u nizu završno.

[8] Ova formulacija Turingovog stroja je pojednostavljena verzija formulacija preuzetih iz:
Hopcroft J. E., Motwani R., Ullman J.; Introduction to Automata Theory, Languages, and Computation (Second Edition); Addison-Wesley, 2001.; str. 319
Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.; str. 23

δ funkcija prijelaza čiji su argumenti stanje q i znak na traci X, a vrijednost je trojka (Y, D, p) gdje je:

Y znak koji se piše u polje (ili se briše sadržaj polja ukoliko je Y znak 0)

D smjer u kojem se pomiče glava ili uputa da glava ostane na istom mjestu. Za to ćemo koristiti znakove: L (pomak ulijevo), R (pomak udesno), S (ostajanje na mjestu)

p sljedeće stanje iz skupa Q

Sada možemo zamisliti kako bi Turingov stroj mogao računati funkciju nad prirodnim brojevima: Prvo bi upisali na traku argument funkcije (broj bi mogli zapisati u unarnoj notaciji jer imamo na raspolaganju samo jedan znak[9]), postavili bi glavu na krajnji lijevi znak i pokrenuli bi stroj[10]. Ako Turingov stroj ikada dosegne završno stanje, a na traci ostane zapis jednoga broja i pritom glava pokazuje na krajnju lijevu jedinicu njegovog zapisa, taj broj ćemo smatrati vrijednošću funkcije. Ako se to ne dogodi, smatrat ćemo da funkcija koju stroj računa nije definirana za taj argument. To može biti u tri slučaja slučaja:

1. Turingov stroj se nikad ne zaustavlja.
2. Turingov stroj se zaustavlja u stanju koje nije završno.
3. Turingov stroj se zaustavlja u završnom stanju ali nije slučaj da se na traci nalazi zapis jednog broja i da glava pokazuje na krajnju lijevu jedinicu u nizu.

Uzet ćemo za primjer Turingov stroj koji računa funkciju *sljedbenik* definiranu s $s(x) = x + 1$. Prije nego specificiramo stroj razmotrit ćemo princip po kojem bi on trebao raditi. Inicijalno će se na traci nalaziti niz od x jedinica koji predstavlja unarni zapis argumenta x. Turingov stroj će pokazivati na prvu jedinicu s lijeve strane. Glava stroja će se sada pomaknuti za jedno polje ulijevo i na to mjesto će zapisati jedinicu. Nakon toga će stroj prijeći u završno stanje. Jasno je da će na kraju traka sadržavati niz s jednom jedinicom više nego na početku. Stroj će

[9] U unarnoj notaciji broj n se piše kao niz od n jedinica. Ako želimo domenu proširiti nulom, onda možemo nulu označiti jednom jedinicom i općenito svaki broj n s $n + 1$ jedinica.

[10] Važno je unaprijed dogovoriti početnu poziciju glave Turingovog stroja jer će o njoj ovisiti vrijednost funkcije.

na kraju opet pokazivati na krajnju lijevu jedinicu niza, tako da je konačan niz prihvatljiv kao vrijednost funkcije. To je konvencija koje ćemo se držati u nastavku - svaki Turingov stroj prilikom računanja funkcije inicijalno mora pokazivati na krajnju lijevu jedinicu početnog niza, a završiti pokazivanjem na krajnju lijevu jedinicu rezultirajućeg niza. Ukoliko ovaj drugi uvjet nije zadovoljen, smatrat ćemo da funkcija koju Turingov stroj računa nije definirana za taj argument.[11] Za svaku funkciju koja je na taj način izračunljiva na nekom Turingovom stroju reći ćemo da je **Turing-izračunljiva**.

Formalno stroj koji računa funkciju *sljedbenik* je predstavljen uređenim parom: (Q, δ) gdje je:

$$Q = \{q_0, q_1, q_2\}$$
$$\delta = \{ ((q_0, 1), (1, L, q_1)), ((q_1, 0), (1, S, q_2)) \}$$

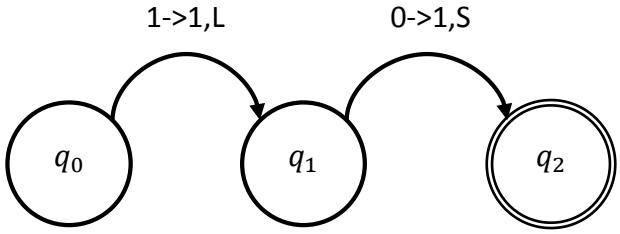
Funkcija prijelaza se može preglednije prikazati tablicom:

	0	1
q_0		$1, L: q_1$
q_1	$1, S: q_2$	

U lijevom stupcu su pobrojana sva stanja osim završnog. U gornjem redu su znakovi trake. Kad god vrijedi $\delta((q, X)) = (Y, D, p)$, polje u redu označenom sa stanjem q i stupcu označenom sa znakom X popunjeno je trojkom (Y, D, p). Red označen sa završnim stanjem je izostavljen jer funkcija prijelaza nikada neće biti definirana za završno stanje. Čak i za stanja koja nisu završna funkcija prijelaza može ostati nedefinirana (kao što je u našem slučaju za parove $(q_0, 0)$ i $(q_1, 1)$. Ako Turingov stroj ikada dođe u situaciju da se nalazi u stanju koje nije završno, a nema definiranu sljedeću radnju, prestat će s radom. U takvim situacijama ćemo reći da funkcija koju taj Turingov stroj računa nije definirana za argument.

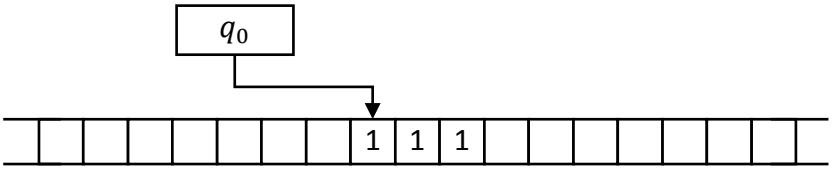
[11] Uvođenje ove konvencije omogućuje jednostavno „komponiranje“ strojeva - tamo gdje je završio jedan stroj može nastaviti drugi. Drugim riječima, ako su nam zadana dva stroja koji računaju neke funkcije, možemo lako dobiti stroj koji računa kompoziciju tih funkcija.

Drugi popularni prikaz Turingovog stroja je pomoću grafa:



Svaki čvor u grafu predstavlja jedno od stanja. Završno stanje je označeno dvostrukom kružnicom. Strelice prikazuju prijelaze između stanja, a zapisi nad strelicama trenutno pokazan znak na traci, te upute za pisanje novog znaka i pomicanje glave.

Pogledajmo sada kako se navedeni Turingov stroj ponaša kada mu zadamo kao argument broj 3. Inicijalno se na traci nalazi niz od 3 jedinice (unarni zapis broja 3). Turingov stroj se nalazi u stanju q_0 i pokazuje na krajnju lijevu jedinicu niza:



Potpuni opis nekoga trenutka u radu Turignovog stroja nazivamo **konfiguracijom**. Rad Turingovog stroja je zapravo niz prijelaza iz jedne konfiguracije u drugu.[12] Na slici je prikazana inicijalna konfiguracija. Konfiguracije ćemo u nastavku pisati o ovakvom obliku: $1_0 11$. Rad Turingovog stroja koji računa $s(3)$ je sada prikazan nizom konfiguracija:

$$1_0 11 \text{ -> } 0_1 111 \text{ -> } 1_2 111$$

Mogli bi pokušati konstruirati i Turingove strojeve za računanje nekih složenijih funkcija uključujući i one s većim brojem argumenata, na primjer: zbrajanje, oduzimanje, množenje, dijeljenje... Jedna metoda računanja funkcija s n argumenata na Turingovom stroju je da se na početku argumenti zadaju kao n nizova jedinica odvojenih razmakom, a glava stroja se

[12] Nova konfiguracija može biti jednaka prethodnoj ako postoji pravilo $\delta((q, X)) = (Y, S, p)$ gdje je $X = Y$ i $q = p$. Ako Turingov stroj ikada dosegne konfiguraciju gdje se primjenjuje takvo pravilo, neće se nikada zaustaviti jer će sve buduće konfiguracije biti jednake.

postavi na krajnju lijevu jedinicu prvog niza. Time je svakom Turingovom stroju pridružen niz funkcija koje on računa: jedna unarna, jedna binarna i općenito za svaki $n \geq 1$ po jedna n-arna funkcija. Druga metoda računanja n-arne funkcije na Turingovom stroju može se provesti na indirektan način svođenjem svih n-arnih funkcija na unarne. To se može provesti kodiranjem n-torki prirodnim brojevima. Za to bi mogli koristiti opet ranije prikazan Gödelov princip kodiranja. Svaka n-torka $a_1, a_2, \dots a_n$ bi se kodirala brojem ${p_1}^{a_1} * {p_2}^{a_2} * \dots {p_n}^{a_n}$ gdje je $p_1, p_{2,} \dots p_n$ prvih n prostih brojeva. Naravno, morali bismo prije toga pokazati da je odgovarajuća funkcija dekodiranja Turing-izračunljiva.

Svakome tko se prvi put susreće s ovom temom može se činiti nevjerojatnim da bi tako jednostavan stroj mogao računati bilo koju znatno složeniju funkciju, a kamoli svaku efektivno izračunljivu funkciju u intuitivnom smislu. Taj stav se mijenja tek nakon susreta s konkretnim primjerima kompliciranijih strojeva[13]. Za sada prestajemo s daljnjim specificiranjem konkretnih Turingovih strojeva i posvećujemo se problemu ograničenja pojma Turing-izračunljivosti.

2. Dijagonalna funkcija, problem zaustavljanja i neizračunljivost

Postavlja se pitanje: Da li je svaka funkcija nad prirodnim brojevima Turing izračunljiva? Može se relativno jednostavno pokazati da to nije slučaj. Da bi izveli taj dokaz, morat ćemo prvo pronaći sustav kodiranja Turingovih strojeva, odnosno treba nam metoda kojom će se svakom Turingovom stroju pridružiti jedinstven prirodni broj. Ranije smo vidjeli da se svaki Turingov stroj može u potpunosti specificirati tablicom. Nadalje, svaka tablica Turingovog stroja može se napisati u jednom redu kao niz petorki: $(q_0, 0, Y_0, D_0, p_0)$, $(q_0, 1, Y_1, D_1, p_1)$, $(q_1, 0, Y_2, D_2, p_2)$... Ovdje pretpostavljamo da je svaki red u tablici popunjen. Naša definicija Turingovog stroja to ne zahtjeva i već smo vidjeli primjer Turingovog stroja za funkciju *sljedbenik* koji je bio specificiran nepotpunom tablicom. Međutim, dodavanjem uvjeta da tablica uvijek mora biti potpuna nećemo promijeniti skup funkcija koje Turingovi strojevi mogu izračunati. Svaka funkcija koju računa Turingov stroj s nepotpunom tablicom može se

[13] Primjeri se mogu naći u:
Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.

izračunati i Turingovim strojem s potpunom tablicom. Taj stroj dobivamo tako da na svako prazno mjesto u tablici prvog stroja stavimo „beskonačnu petlju“ u obliku pravila $\delta\big((q,X)\big) = (Y,S,p)$ gdje je $X = Y$ i $q = p$. Dobiven stroj će „upasti“ u beskonačnu petlju svaki put kad prvi stroj zapne u stanju koje nije završno. Rekli smo da funkciju smatramo jednako nedefiniranom za argument bilo da se Turingov stroj nikada ne zaustavi ili da se zaustavi u stanju koje nije završno.

Tablicu Turingovog stroja koji računa funkciju *sljedbenik* bi mogli popuniti ovako:

	0	1
q_0	$0, S: q_0$	$1, L: q_1$
q_1	$1, S: q_2$	$1, S: q_2$

S obzirom da smo išli od vrha tablice prema dolje, prva dva znaka u petorkama su predvidiva pa ih možemo izostaviti. Ostaje nam niz trojki: (Y_0, D_0, p_0), (Y_1, D_1, p_1), (Y_2, D_2, p_2)... Ako izostavimo zagrade i zareze dobivamo niz: $Y_0\ D_0\ p_0\ Y_1\ D_1\ p_1\ Y_2\ D_2\ p_2$... Sada svakom znaku pridružujemo broj na sljedeći način:

1. Svaki q_n dobiva broj $n + 1$ (kako bi se izbjegla nula)
2. Znakovi 0 i 1 dobivaju brojeve 1 i 2 (kako bi se izbjegla nula)
3. Znakovi L, R, i S dobivaju redom brojeve 1,2 i 3

Time od niza znakova dobivamo niz kodnih brojeva: x_1, x_2, x_3 ... Različiti znakovi u ovom slučaju nisu dobili jedinstvene brojeve, ali to neće uzrokovati nikakve poteškoće jer su mjesta različitih tipova znakova u nizu fiksirana. Na primjer, ako se broj 1 pojavljuje na nekom mjestu $1 + k * 3$ ($k \geq 1$) u nizu, znamo da taj broj predstavlja znak 0 koji se upisuje na traku, a ne stanje q_0 ili pravilo L. Na kraju primjenjujemo Gödelovu metodu kodiranja i Turingov stroj dobiva jedinstven kodni broj: ${p_1}^{x_1} * {p_2}^{x_2} * {p_3}^{x_3}$ ... Ako poredamo kodne brojeve Turingovih strojeva po veličini, indirektno smo poredali i unarne funkcije koje oni računaju: f_1, f_2, f_3 ...

Neka je **dijagonalna funkcija** d definirana na sljedeći način:

$$d(x) = \begin{cases} 2 \text{ ako je } f_x(x) = 1 \\ 1 \text{ inače} \end{cases}$$

Funkcija d ne može biti Turing-izračunljiva. Kad bi postojao Turingov stroj koji računa d, onda bi funkcija d morala biti u nizu $f_1, f_2, f_3 \ldots$ Pretpostavimo da je $d = f_k$ za neki k. Prema definiciji to bi značilo da je $f_k(k) = 2$ ako i samo ako je $f_k(k) = 1$, što je kontradiktorno.

Možda na prvi pogled nije jasno zašto se funkcija d naziva „dijagonalnom". Pogledajmo prikaz vrijednost funkcija $f_1, f_2, f_3 \ldots$ u obliku tablice:

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

Funkcija d je definirana obzirom na dijagonalu u tablici. Za bilo koji argument x, $d(x)$ će se razlikovati od vrijednosti na dijagonali u stupcu x. Sada vidimo zašto se d ne može nalaziti u tablici. Kad bi se d nalazila redu k, njezina vrijednost bi se po definiciji morala razlikovati od vrijednosti na dijagonali u stupcu k.[14]

Sad kad imamo metodu kodiranja Turingovih strojeva, možemo zamisliti **univerzalni Turingov stroj**. To je stroj koji za argumente x_1, x_2 računa vrijednost koju bi izračunao Turingov stroj s kodnim brojem x_1 za argument x_2. Naravno, nisu svi prirodni brojevi kodovi Turingovih strojeva. Slučajeve kada x_1 nije kodni broj nijednog stroja možemo zanemariti. S pojmom univerzalnog Turingovog stroja dolazi ideja da nije potrebno za svaki problem konstruirati poseban stroj za njegovo rješavanje. Dovoljan nam je jedan univerzalni stroj koji se može programirati za rješavanje različitih problema. U slučaju univerzalnog Turingovog stroja, prvi argument x_1 predstavlja program.

[14] Dijagonalnu metodu koristio je Georg Cantor u svojem poznatom dokazu iz 1891. godine o postojanju neprebrojivih skupova.

Sada ćemo ramotriti još jedan poznati primjer nerješivog problema - problem zaustavljanja (*Halting problem*). Problem je pronaći efektivni postupak kojim će se moći za bilo koji Turingov stroj i prirodni broj ustanoviti da li će se Turingov stroj zaustaviti kada mu se taj broj zada kao ulazni argument. Prvo moramo navedeni problem svesti na funkciju nad prirodnim brojevima.

To bi će biti binarna funkcija h definirana na sljedeći način:

$$h(x_1, x_2) = \begin{cases} 1 \text{ ako se Turingov stroj s kodnim brojem } x_1 \text{ zaustavlja za argument } x_2 \\ 2 \text{ inače} \end{cases}$$

Pretpostavimo da postoji Turingov stroj H koji računa funkciju $h(x_1, x_2)$. Sada bi mogli lako dobiti stroj H' koji računa unarnu funkciju h' definiranu sa: $h'(x) = h(x, x)$. Taj stroj bi prvo napravio kopiju ulaznog niza tako da od x dobijemo (x, x), a zatim bi vratio glavu na početak i nastavio iste operacije koje obavlja i stroj H. Stroj H' zapravo provjerava za broj x da li se Turingov stroj s kodnim brojem x zaustavlja ako je pokrenut nad brojem x. Ovisno o odgovoru na traku se ispisuje 1 ili 2 .

Sada dolazimo do stroja H'' kojega dobivamo iz H' tako da pravila za ispisivanje broja 2 zamijenimo pravilima za ispisivanje broja 1, a pravila za ispisivanje broja 1 zamijenimo beskonačnom petljom. H'' sada računa funkciju $h''(x)$ definiranu sa:

$$h''(x) = \begin{cases} 1 \text{ ako je } h'(x) \neq 1 \\ \text{nedefinirano inače} \end{cases}$$

H'' provjerava za broj x da li se Turingov stroj s kodnim brojem x zaustavlja kad je pokrenut nad brojem x. Ako je odgovor "da" (što znači da je $h'(x) = 1$) onda se H'' nikada neće zaustaviti jer h'' nije definiran. Ako je odgovor „ne" (što znači da je $h'(x) \neq 1$), onda H'' ispisuje broj 1 i zaustavlja se. Što će se dogoditi ako H'' pokrenemo nad vlastitim kodnim brojem? Ispada da se H'' za vlastiti kodni broj zaustavlja ako i samo ako se ne zaustavlja. Dakle, H'' ne postoji, a to znači da ne postoji niti H. Funkcija h nije Turing-izračunljiva.

Vidjeli smo dvije funkcije koje nisu Turing-izračunljive. Zapravo većina funkcija nad prirodnim brojevima nije Turing-izračunljiva jer je skup funkcija neprebrojiv, dok je skup Turingovih

strojeva prebrojiv.[15] Ako Turingov stroj prihvatimo kao univerzalni model efektivnog postupka (kao što sugerira Turingova teza), zaključujemo da postoje problemi koji nisu efektivno rješivi. Turingovu tezu nazivamo samo „tezom" (a ne teoremom) jer ne postoji rigorozni dokaz za nju. Takav dokaz nije moguć jer se Turingova teza referira na intuitivni pojam efektivnog postupka koji nije precizno definiran. Bez obzira na to, iskustvo ide u prilog ovoj tezi obzirom na to da nije otkriven niti jedan primjer postupka koji u teoriji ne bi bio svodiv na radnju Turingovog stroja, a koji bi se mogao smatrati efektivnim u intuitivnom smislu. Nadalje, konvencionalna digitalna računala su ekvivalentna s Turingovim strojevima s ograničenom memorijom (trakom), a tijekom ubrzanog razvoja tehnologije proteklih desetljeća se pokazao ogroman doseg njihovih mogućnosti.

3. Proširenja Turingovog stroja

Turingov stroj, kako smo ga definirali na početku ovog poglavlja je krajnje jednostavan. To čini specificiranje strojeva za rješavanje složenijih funkcija izrazito napornom zadaćom. Čak i najjednostavnije specifikacije Turingovih strojeva za računanje temeljnih aritmetičkih funkcija kao što su oduzimanje i množenje, mogu djelovati komplicirano. U ovom djelu ćemo razmotriti neka od mogućih proširenja Turingovog stroja koja znatno olakšavaju tu zadaću. Ako je Turingova teza istinita, niti jedno proširenje Turingovog stroja koje i dalje održava princip efektivnog postupka u intuitivnom smislu ne bi moglo proširiti skup Turing-izračunljivih funkcija. Mi ćemo ipak za svako proširenje sugerirati kako bi se moglo bez pozivanja na Turingovu tezu dokazati da je nova, proširena varijanta Turingovog stroja zapravo ekvivalentna prvotnoj.

a) Proširenje alfabeta

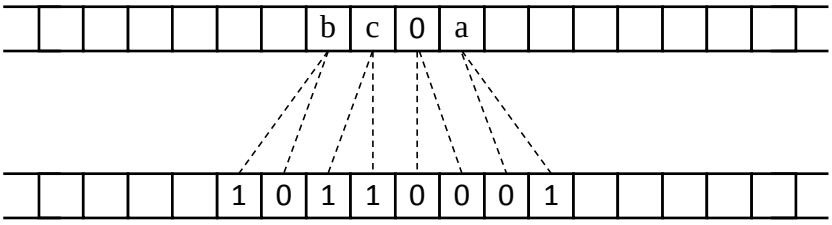
Prvo razmatramo mogućnost proširenja skupa znakova koje Turingov stroj koristi (**alfabet**). Umjesto da imamo samo binarni alfabet s jedinicama i nulama (prazninama) mogli bi koristiti neki opsežniji alfabet. Turingov stroj s binarnim alfabetom uvijek može simulirati radnju stroja

[15] Skup Turing-izračunljivih funkcija se može postaviti u korelaciju jedan na jedan s prirodnim brojevima, što smo već pokazali. Skup svih funkcija nad prirodnim brojevima se ne može jer ćemo za bilo koji poredak funkcija moći definirati dijagonalnu funkciju koja nije u njemu.

s većim alfabetom. Da bi to pokazali, prvo je potrebno kodirati sve znakove proširenog alfabeta. S obzirom da svaki prošireni alfabet mora imati jedan znak koji predstavlja prazninu, njega ćemo kodirati nulom. Svi ostali znakovi dobit će redom prirodne brojeve počevši od 1. Sada se svaki zapis na traci u proširenom alfabetu može prevesti zapisom u binarnom alfabetu s jedinicom i nulom na način da se svaki znak zapiše kao binarna reprezentacija njegovog kodnog broja. Na primjer, neka je proširen alfabet skup $\{0, a, b, c\}$ gdje znak 0 predstavlja prazninu. Znakovima pridružujemo redom brojeve:

$$0 \to 0 \qquad a \to 1 \qquad b \to 2 \qquad c \to 3$$

Izraz $bc0a$ sada možemo predstaviti na sljedeći način:



Kad bi prošireni alfabet imao više znakova, morali bi rezervirati i više polja za binarni kod svakog znaka. Ako je broj znakova u alfabetu k, bit će potrebno $\lceil \log_2 k \rceil$ polja za kod svakog znaka. Od Turingovog stroja s proširenim alfabetom uvijek možemo dobiti Turingov stroj s binarnim alfabetom koji izvršava istu funkciju ali umjesto različitih znakova prvog stroja upisuje njihove kodove u binarnom zapisu.

b) Uvođenje većeg broja završnih stanja

Često se koristi definicija Turingovog stroja s većim brojem završnih stanja. Do sada smo Turingove strojeve promatrali samo kao strojeve za računanje funcija nad prirodnim brojevima pa nismo imali potrebu za više od jednog završnog stanja. Bez obzira na to, svaki stroj s većim brojem završnih stanja se može svesti na ekvivalentan stroj sa samo jednim završnim stanjem: odaberemo jedno od završnih stanja q_f i onda svako drugo završno stanje dopunimo pravilom prijelaza u q_f.

c) Uvođenje dodatnih traka

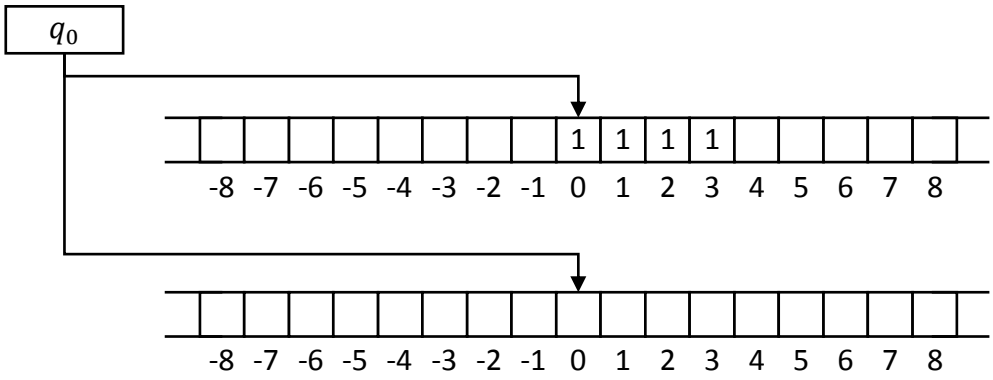
Sada razmatramo Turingov stroj koji pralelno radi na više traka. Svaka traka ima zasebnu glavu koja obavlja radnje neovisno o ostalima. Inicijalno, na prvoj traci se nalazi ulazni podatak i glava prve trake je pozicionirana na prvi znak ulaznog niza. Sve ostale trake su inicijalno prazne (inicijalna pozicija ostalih glava nije važna jer su sva polja na praznim trakama jednaka). Na kraju Turingov stroj završava s rezultatom ispisanim na prvoj traci. Ostale trake služe kao pomoćna memorija. U svakom koraku radnja svake glave je determinirana stanjem Turingovog stroja i znakom na koji ona pokazuje. Ako Turingov stroj ima k traka, funkcija prijelaza δ će imati oblik:

$$\delta\big(q, (X_1, X_2, \dots X_k)\big) = \big((Y_1, D_1), (Y_2, D_2), \dots (Y_k, D_k), p\big)$$

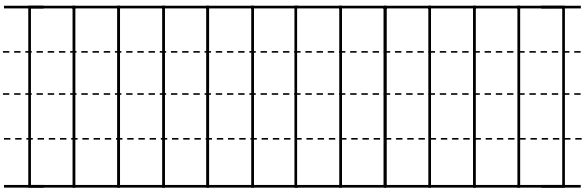
gdje je $X_1, X_2, \dots X_k$ niz znakova na koje redom pokazuju svaka od k glava, $Y_1, Y_2, \dots Y_k$ niz znakova koje redom upisuju svaka od k glava i p je novo stanje.

Ovo proširenje također ne utječe na skup funkcija koje će biti izračunljive. Nećemo se baviti rigoroznim dokazivanjem ove tvrdnje. Samo ćemo sugerirati kako bi se mogla dokazati ekvivalentnost između Turingovog stroja s dvije trake i binarnim alfabetom, te Turingovog stroja s jednom trakom. Iz ovog primjera (uzimajući u obzir prethodne dokaze) biti će jasno na koji način bi se ovaj dokaz mogao provesti za sve slučajeve.

Neka je T_2 Turingov stroj s dvije trake i binarnim alfabetom $\{1,0\}$. Prvo ćemo indeksirati polja obje trake kako bi se kasnije mogli na njih referirati. Polja na koja glave pokazuju u inicijalnom stanju će dobiti indeks 0. Sva polja na lijevoj strani će dobiti negativne indekse, a polja na desnoj strani pozitivne:



Sada tražimo ekvivalentan Turingov stroj T s jednom trakom koji će simulirati T_2. Zamislimo da je traka stroja T podijeljena po dužini na 4 segmenta:



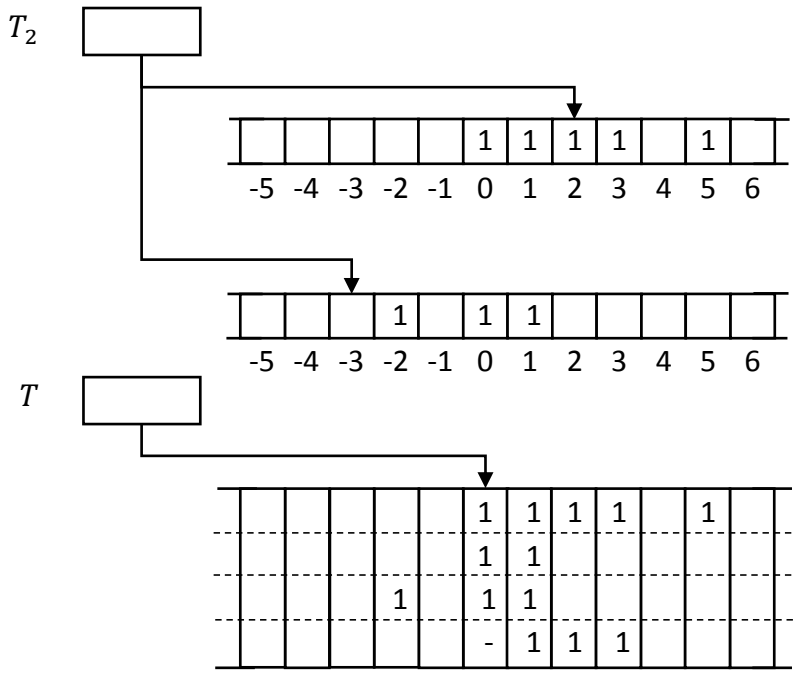
Time je svako polje na traci podijeljeno na 4 podpolja. Po definiciji Turingov stroj svako polje tretira kao cjelinu i ne može zapisivati zasebne znakove u svako od tih podpolja. Taj problem možemo riješiti tako da za uzmemo za stroj T prošireni alfabet od 16 znakova:

$$\begin{Bmatrix} 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \\ 0'\ 0'\ 0'\ 0'\ 1'\ 1'\ 1'\ 1'\ 0'\ 0'\ 0'\ 0'\ 1'\ 1'\ 1'\ 1 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \end{Bmatrix}$$

Alfabet je zapravo kartezijev produkt $\{1,0\}^4$, odnosno skup svih četvorki (X_1, X_2, X_3, X_4) gdje je $X_1, X_2, X_3, X_4 \in \{1,0\}$. Sada upisivanje jedinice u primjerice treće podpolje odozgora možemo ostvariti upisivanjem četvorke $(X_1, X_2, 1, X_4)$ gdje su X_1, X_2, X_4 odabrani tako da ne modificiraju ostala podpolja.

Drugi i četvrti segment trake možemo koristiti kao kopije dviju traka stroja T_2, a prvi i treći segment će služiti za „pamćenje“ pozicije svake od glava stroja T_2. Pozicija glave se može prikazati unarnom reprezentacijom indeksa polja na koje ona pokazuje. S obzirom da indeksi obuhvaćaju nulu i negativne brojeve možemo pojednostaviti pisanje indeksa tako da stroju T

dodamo znak „-“ na način da alfabet stroja nadopunimo četvorkama $(-, X_2, -, X_4)$ za svaki $X_2, X_4 \in \{1,0\}$. Evo konkretnog primjera kako bi se trake stroja T_2 mogle u određenoj konfiguraciji prikazati na stroju T:



Naravno, stroj T će imati znatno veći broj stanja od stroja T_2 i njihove konfiguracije nisu u korelaciji jedan na jedan. Kada stroj T_2 napravi jedan korak, stroj T treba simulirati promjenu njegovih traka na drugom i četvrtom segmentu svoje trake, a zatim mora ažurirati nove pozicije glava upisivanjem indeksa na prvi i treći segment svoje trake. To je relativno složena operacija i stroj T bi prilikom njenog izvršavanja promijenio velik broj konfiguracija. Nas u ovom trenutku ne zanima rigorozna specifikacija stroja T. Ovaj primjer samo pokazuje ideju po kojoj bi se dokaz ekvivalentnosti strojeva T_2 i T mogao provesti. Potpuno analogno tome mogli bi simulirati i Turingov stroj s bilo kojim brojem traka na stroju sa samo jednom trakom.

4. Redukcije Turingovog stroja

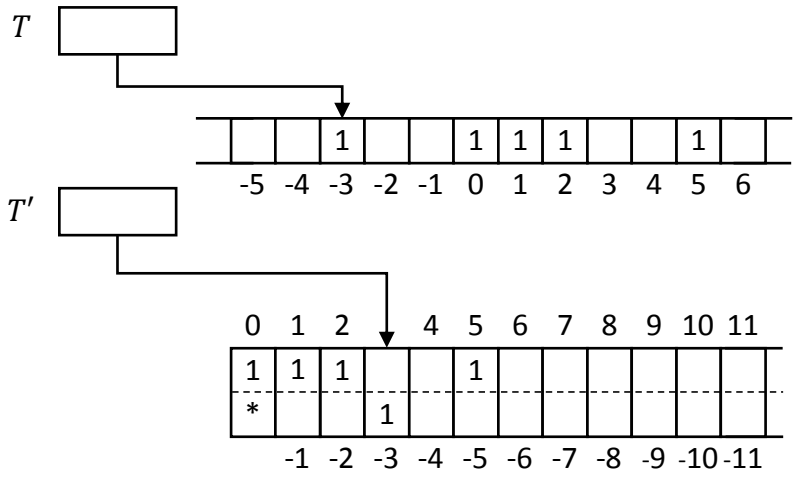
Turingov stroj, kako smo ga definirali na početku, je izrazito jednostavan. No, ipak je moguće provesti neke daljnje redukcije koje neće bitno utjecati na njegove mogućnosti.

a) Uklanjanje upute S

Jedna vrlo jednostavna redukcija koju možemo provesti je uklanjanje upute S po kojoj glava ostaje na mjestu. Ako to učinimo, Turingov stroj će morati u svakom koraku pomaknuti glavu ulijevo ili udesno. To ne predstavlja nikakav problem. Ako želimo da se glava nakon upisivanja znaka nalazi na istom mjestu, možemo je pomaknuti ulijevo i zatim u drugom koraku vratiti pomakom udesno bez promjene na traci. Preciznije, svako pravilo oblika $\delta(q, X) = (Y, S, p)$ mogli bi zamijeniti pravilima $\delta(q, X) = (Y, L, q')$, $\delta(q', 1) = (1, R, p)$, $\delta(q', 0) = (0, R, p)$ gdje je q' neko novo stanje.

b) Redukcija trake

Sada razmatramo Turingov stroj s polovičnom trakom - stroj čija je traka beskonačna samo u jednom smjeru. Ovo se na prvi pogled može učiniti značajnijom redukcijom koja bi mogla ograničiti mogućnosti rezultirajućeg stroja. Pokazat ćemo da to ipak nije slučaj. Turingov stroj s polovičnom trakom je ekvivalentan originalnom Turingovom stroju. Neka je T običan Turingov stroj kako je zadan na početku poglavlja. Sada tražimo Turingov stroj T' s polovičnom trakom koji će simulirati rad stroja T. Prvo ćemo polja na traci stroja T indeksirati na isti način kao što smo to već činili ranije, a zatim ćemo traku podijeliti na dva djela: lijevi koji čine polja s negativnim indeksima i desni. Sada polovičnu traku stroja T' dijelimo vodoravno na dva segmenta: gornji segment će predstavljati desnu polovicu trake stroja T, a donji lijevu. Polja lijeve polovice trake stroja T će biti prikazana u donjem segmentu obrnutim redoslijedom:



Znak * je uveden kao granica između lijeve i desne polovice trake. Alfabet stroja T je sada skup:

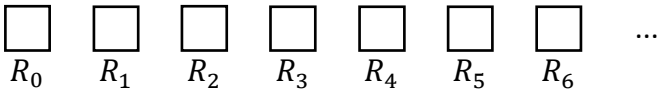
$$\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0' & 0' & 1' & 1' & *' & * \end{Bmatrix}$$

Način na koji T' simulira T je relativno jednostavan. Pravila stroja T' možemo podijeliti na dva skupa. Jedan skup pravila se koristi kada T' simulira rad na desnoj polovici trake stroja T, a drugi kada simulira rad na lijevoj polovici. Svaki put kad T upisuje znak X na desnoj polovici trake, T' će upisati par (X, Y) gdje je Y isti znak koji se nalazi na donjem podpolju. Svaki put kad T pomakne glavu na desnoj polovici trake, to će analogno pratiti i T'. Stvar se mijenja kada glava stroja T dođe do polja s indeksom 0. Ako sada T učini pomak ulijevo, T' će učiniti pomak udesno i prebacit će se na drugi skup pravila koji ga instruira da radi „obrnuto“. Znak * je ovdje postaljen kako bi T' „znao“ kada je dosegnuo granicu. Sada kad god T upisuje znak X na lijevoj polovici trake, T' će upisati par (Y, X) gdje je Y isti znak koji se nalazi na gornjem podpolju. Svaki put kad T pomakne glavu na lijevoj polovici trake, T' će napraviti pomak u suprotnom smjeru.

Može se postaviti pitanje: „Zašto bi uopće uvodili redukcije Turingovog stroja ako će to otežati specifikaciju strojeva za rješavanje konkretnih problema?“. Obično jednostavnost formulacije izračunljivosti koju smo prihvatili znatno olakšava izvođenje dokaza neizračunljivosti za određene probleme. Ako pokažemo da se neka funkcija ne da izračunati na reduciranom Turingovom stroju, znamo da neće biti izračunljiva niti na proširenim verzijama Turingovog stroja.

V. Registarski stroj

U ovom poglavlju bavimo se drugim modelom računalnog stroja - **registarskim strojem**. Memoriju registarskog stroja čini beskonačan niz polja (registara). U svakom polju može biti upisan bilo koji prirodni broj. Prazno polje ćemo interpretirati kao da je u njemu upisana nula. Svakom registru je pridružena „adresa“, odnosno redni broj počevši od nule.



Za razliku od Turingovog stroja koji svojim memorijskim lokacijama (poljima na traci) pristupa sekvencijalno, registarski stroj može u svakom trenutku izravno pristupiti bilo kojem registru kako bi promijenio njegov sadržaj. Takav model puno bliže odgovara konvencionalnim računalima s memorijom nasumičnog pristupa (Random Access Memory). Rad registarskog stroja je određen konačnim nizom pravila. Svako pravilo ima svoj redni broj. Postoje različite ekvivalentne formulacije registarskog stroja. Mi uzimamo definiciju registarskog stroja koji radi na temelju sljedećih pravila:[16]

1. Uvećaj broj u registru R_n za 1 i prijeđi na pravilo k.
2. Ako je registar R_n prazan prijeđi na pravilo k, inače umanji broj u njemu za 1 i prijeđi na pravilo l.
3. Završi s radom

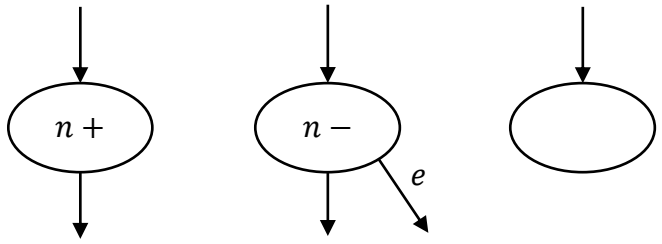
Registarski stroj počinje s prvim pravilom. U nizu pravila uvijek će se nalaziti jedno završno pravilo. Po dogovoru završno pravilo će biti na kraju niza. Ako registarski stroj ikada dosegne završno pravilo, prekinut će s radom.

Sada možemo zamisliti proces računanja funkcije nad prirodnim brojevima na registarskom stroju. Inicijalno će se u prvih n registara nalaziti argumenti n-arne funkcije. Ako registarski stroj ikad prestane s radom, u registru R_{n+1} će ostati vrijednost funkcije. Ako se registarski stroj nikada ne zaustavi, smatrat ćemo funkciju nedefiniranom za zadane argumente. Evo jednostavnog primjera pravila registarskog stroja koji računa funkciju *sljedbenik* definiranu s $s(x) = x + 1$

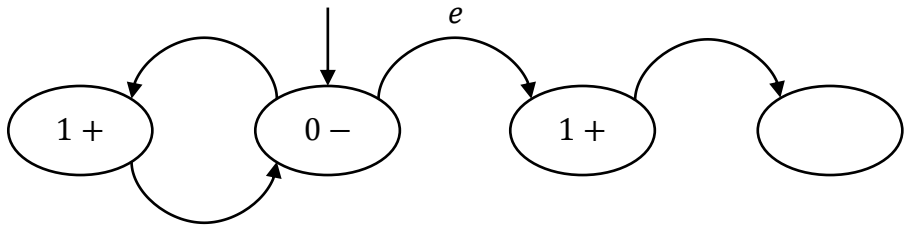
1. Ako je R_0 prazan prijeđi na pravilo 3, inače umanji broj u njemu za 1 i prijeđi na pravilo 2.
2. Uvećaj broj u R_1 za 1 i prijeđi na pravilo 1.
3. Uvećaj broj u R_1 za 1 i prijeđi na pravilo 4.
4. Završi s radom.

[16] Ova formulacija registarskog stroja preuzeta je iz:
Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.; str. 45

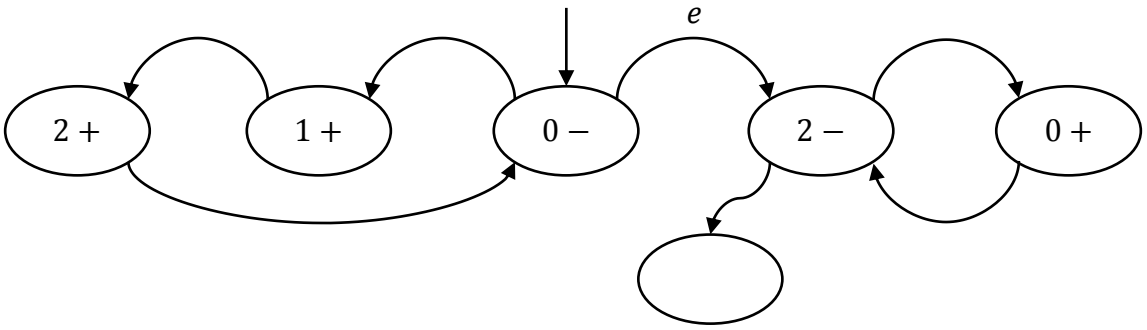
Prva dva pravila naizmjence umanjuju broj u R_0 i uvećavaju broj u R_1. Kad je sadržaj registra R_0 dosegnuo nulu, u R_1 će se nalaziti argument koji je inicijalno bio u R_0. Sada se trećim pravilom uvećava taj broj za jedan i završava se s radom. Na kraju nam u R_1 ostaje vrijednost funkcije. Pravila registarskog stroja se mogu preglednije prikazati grafom. Svako od tri pravila se označava odgovarajućim čvorom na sljedeći način:



Drugo pravilo ima dva izlaza. Izlaz označen s *e* se koristi kada je registar R_n prazan. Evo grafa registarskog stroja za računanje funkcije *sljedbenik*:



Jedan od nedostataka ovoga stroja je to što je na kraju prebrisan registar R_0 u kojemu se nalazio argument. Taj problem možemo općenito lako izbjeći na način da prije računanja funkcije sve argumente prekopiramo u nove registre i onda radimo s tim kopijama. Proces kopiranja broja iz R_0 u R_1 se može ostvariti na ovaj način:



Ovaj stroj „prazni“ R_0 i „puni“ R_1 i R_2. Kad je R_0 prazan, u R_1 i R_2 će se nalaziti dvije kopije broja koji se inicijalno nalazio u R_0. Sada se sadržaj registra R_2 prazni u registar R_0. Taj stroj zapravo računa funkciju identiteta koja kao vrijednost vraća vlastiti argument.

Ne bi bilo preteško konstruirati i registarske strojeve za računanje nekih malo kompleksnijih funkcija kao što su zbrajanje i množenje.[17] Zadatak pronalaženja stroja za računanje određene funkcije se sada čini malo lakšim nego što je to bio slučaj kod Turingovih strojeva. Iako registarski strojevi djeluju „moćnije“ od Turingovih, pokazat će se da to nije slučaj. Svaka funkcija koja je izračunljiva na registarskom stroju je i Turing-izračunljiva.

VI. Rekurzivne funkcije

U ovom poglavlju razmatramo bitno drugačiji pristup problemu formalizacije pojma intuitivno izračunljive funkcije. Dosadašnji pristup se sastojao od definiranja apstraktnog stroja za računanje, a zatim poistovjećivanja skupa funkcija izračunljivih na tom stroju sa skupom intuitivno izračunljivih funkcija. Strojevi koje smo definirali bili su krajnje jednostavni i mogli su obavljati vrlo ograničen skup bazičnih operacija. Ako njihove radnje prihvatimo kao opći model računanja, možemo reći da se svaka izračunljiva funkcija u konačnici svodi na nekoliko krajnje jednostavnih operacija. Sada primjenjujemo nešto direktniji pristup u formalnom definiranju izračunljivih funkcija. Prvo odabiremo neke jednostavne funkcije koje su trivijalno izračunljive, a zatim dajemo pravila pomoću kojih od izračunljivih funkcija možemo dobiti nove izračunljive funkcije.[18] Ideja je da skup funkcija koji smo na taj način specificirali obuhvati sve intuitivno izračunljive funkcije. **Osnovne funkcije** od kojih polazimo su definirane na sljedeći način:

[17] Ovdje prestajemo s navođenjem konkretnih primjera registarskih strojeva. Više zanimljivih primjera može se naći u:
Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.; str. 45 - 62

[18] Postupak je na neki način sličan zasnivanju formalnog aksiomatskog sustava gdje počinjemo specifikacijom skupa polaznih tvrdnji (aksioma) čija je istinitost obično trivijalna, a zatim uvodimo pravila izvođenja kojima od aksioma napredujemo prema ostalim tvrdnjama sustava (teoremima).

$z(x) = 0$

$s(x) = x'$ gdje je x' sljedbenik broja x [19]

Za svake $n \geq 1$ i $1 \leq m \leq n$ imamo n-arnu funkciju p_m^n definiranu sa:

$p_m^n(x_1, x_2, \dots x_n) = x_m$

z je konstantna funkcija koja za svaki argument daje vrijednost 0. s je funkcija *sljedbenik* koju smo već ranije definirali. Ona uvijek daje broj koji je za 1 veći od argumenta. p_m^n je funkcija *projekcije* koja prima n argumenata, a vrijednost joj je m-ti argument.

Sada uvodimo pravila kojima ćemo od osnovnih funkcija moći sastavljati nove. Prvo pravilo je **kompozicija**:

Od m-arne funkcije f i n-arnih funkcija $g_1, g_2, \dots g_m$ dobivamo n-arnu funkciju h definiranu s:

$$h(x_1, x_2, \dots x_n) = f\big(g_1(x_1, x_2, \dots x_n), g_2(x_1, x_2, \dots x_n), \dots g_m(x_1, x_2, \dots x_n)\big)$$

Na primjer, od funkcija s i z možemo kompozicijom dobiti konstantnu funkciju čija je vrijednost uvijek 1:

$$c_1(x) = s\big(z(x)\big)$$

Uzastopnom primjenom kompozicije mogli bi za svaki n dobiti konstantnu funkciju $c_n(x)$.

Sljedeće pravilo je **primitivna rekurzija**:

Od n-arne funkcije f i $(n+2)$-arne funkcije g dobivamo $(n+1)$-arnu funkciju h definiranu s:

$$h(x_1, x_2, \dots x_n, 0) = f(x_1, x_2, \dots x_n)$$
$$h(x_1, x_2, \dots x_n, y') = g\big(x_1, x_2, \dots x_n, y, h(x_1, x_2, \dots x_n, y)\big)$$

[19] Ovdje namjerno ne koristim formulu $s(x) = x + 1$ iako bi ona jednako mogla poslužiti kao intuitivna definicija funkcije s. Razlog tome je što želim izbjeći nesporazum koji bi mogao nastati kad bi čitatelj formulu $s(x) = x + 1$ shvatio kao formalnu definiciju funkcije s pomoću funkcije zbrajanja, dok je glavna ideja u tome da se složenije funkcije kao što je zbrajanje definiraju na temelju osnovnih funkcija.

Ovo je opći format rekurzivnog definiranja funkcija. Prvi red predstavlja bazu rekurzivne definicije. Ovdje definiramo vrijednost funkcije h kad je zadnji argument 0 i za to koristimo neku poznatu funkciju f. U drugom redu definiramo vrijednost funkcije h za slučaj kad je na mjestu zadnjeg argumenta neki veći broj y'. Kod definicije koristimo neku poznatu funkciju g i pozivamo se na vrijednost funkcije h kad je na mjestu zadnjeg argumenta prethodnik broja y', odnosno: y.

Uvođenjem ovoga pravila znatno je proširen skup funkcija koje su nam sada dostupne. Sve te funkcije zajedno nazivamo **primitivno rekurzivnima**.

Uzet ćemo kao primjer rekurzivnu definiciju zbrajanja sum. Obično se rekurzivna definicija zbrajanja piše u sljedećem obliku:

$$x + 0 = x \qquad\qquad x + y' = (x + y)'$$

Međutim, mi ovu definiciju moramo prilagoditi formatu koji je prethodno zadan. Prvi dio definicije biti će:

$$sum(x, 0) = p_1^1(x)$$

Za drugi dio $h(x, y') = g(x, y, (h(x, y)))$ potrebno je pronaći odgovarajuću funkciju g. Ta funkcija treba uzeti zadnji argument i uvećati ga za 1. g definiramo kompozicijom na sljedeći način:

$$g(x_1, x_2, x_3) = s\big(p_3^3(x_1, x_2, x_3)\big)$$

Konačna definicija funkcije sum je:

$$sum(x, 0) = p_1^1(x)$$
$$sum(x, y') = s\left(p_3^3\big(x, y, sum(x, y)\big)\right)$$

Pravila kompozicije i primitivne rekurzije dovoljna su nam za definiranje vrlo širokog spektra funkcija. Međutim, ona nam nisu dovoljna za definiranje svih intuitivno izračunljivih funkcija. U poglavljima o Turingovim strojevima uzimali smo u obzir i parcijalne funkcije koje nisu definirane za sve moguće argumente. Njih također smatramo izračunljivima ako postoji

efektivni postupak za dobivanje vrijednosti u svim onim slučajevima kad je funkcija definirana za argument. U onim slučajevima kad funkcija nije definirana dozvoljavamo da postupak nikada ne završava. Pravilima kompozicije i primitivne rekurzije uvijek od totalnih funkcija (funkcija koje su definirane za sve moguće argumente) dobivamo nove totalne funkcije. S obzirom da su sve osnovne funkcije totalne možemo biti sigurni da nikada nećemo dobiti parcijalnu funkciju. Čak i ako zanemarimo parcijalne funkcije, dosadašnja pravila nam nisu dovoljna da dobijemo svaku efektivno izračunljivu totalnu funkciju.[20] Da bi upotpunili skup dostupnih funkcija uvodimo pravilo **minimizacije**:

Od $(n+1)$-arne funkcije f dobivamo n-arnu funkciju h definiranu s:

$$h(x_1, x_2, \dots x_n) = \begin{cases} y \text{ ako je } f(x_1, x_2, \dots x_n, y) = 0, a \text{ za svaki } t \leq y \\ f(x_1, x_2, \dots x_n, t) \text{ je definirano i različito od } 0 \\ \\ \text{nedefinirano inače} \end{cases}$$

Drugim riječima, funkcija $h(x_1, x_2, \dots x_n)$ daje najmanji y za koji je $f(x_1, x_2, \dots x_n, y) = 0$ ukoliko je za svaki $t \leq y$ $f(x_1, x_2, \dots x_n, t)$ definirano. Ako nema takvog broja y, $h(x_1, x_2, \dots x_n)$ nije definirano.

Uvođenjem ovoga pravila završili smo formalnu specifikaciju izračunljivih funkcija. Funkcije koje smo time definirane nazivamo **rekurzivnim funkcijama**. Pretpostavka da je svaka intuitivo izračunljiva funkcija rekurzivna naziva se **Churchovom tezom**.

VII. Ekvivalencija različitih formalizacija izračunljivosti

Sada se posvećujemo odnosima između triju različitih formalizacija izračunljivosti: Turingov stroj, registarski stroj i rekurzivne funkcije.

Neka je R proizvoljno zadan registarski stroj. Stroj R će prilikom rada pristupati registrima koji su eksplicitno navedeni u pravilima. S obzirom da je skup pravila konačan, znamo da će stroj

[20] Poznati primjer totalne izračunljive funkcije koja nije primitivno rekurzivna je *Ackermannova funkcija*: Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.; str. 84

R prilikom rada koristiti samo neki konačan broj registara k. Sada možemo zamisliti Turingov stroj T s k traka koji pohranjuje sadržaj svakoga registra na zasebnu traku kao niz jedinica. Svaku promjenu koju R učini u registru R_n Turingov stroj T može simulirati promjenom broja jedinica na n-toj traci. Graf za T mogao bi se dobiti direktno iz grafa za R tako da se svaki čvor nadopuni odgovarajućim nizom naredbi kojima se uvećava ili umanjuje broj jedinica na određenoj traci. Ovo je neformalni opis strukture dokaza kojim se može pokazati da je svaka funkcija izračunljiva na registarskom stroju ujedno i Turing-izračunljiva.

Razmotrimo sada odnos između rekurzivnih funkcija i registarskog stroja. Svaka od osnovnih funkcija može se vrlo jednostavno izračunati na registarskom stroju. Vidjeli smo već primjer računanja funkcije *sljedbenik:* s. Funkciju z računa najjednostavniji registarski stroj koji ne vrši nikakvu izmjenu u registrima. Za računanje funkcije p_m^n registarski stroj jednostavno treba prekopirati broj iz registra R_m u registar R_{n+1}.

Računanje vrijednosti funkcije h definirane kompozicijom

$$h(x_1, x_2, \dots x_n) = f\big(g_1(x_1, x_2, \dots x_n), g_2(x_1, x_2, \dots x_n), \dots g_m(x_1, x_2, \dots x_n)\big)$$

za argumente $x_1, x_2, \dots x_n$ će početi uzastopnim računanjem vrijednosti $g_1(x_1, x_2, \dots x_n)$, $g_2(x_1, x_2, \dots x_n), \dots g_m(x_1, x_2, \dots x_n)$ i njihovim kopiranjem u posebne registre. Zatim će se te vrijednosti prekopirati u prvih m registara i služit će kao argumenti za funkciju f.

Računanje vrijednosti funkcije h definirane primitivnom rekurzijom

$$h(x_1, x_2, \dots x_n, 0) = f(x_1, x_2, \dots x_n)$$
$$h(x_1, x_2, \dots x_n, y') = g\big(x_1, x_2, \dots x_n, y, h(x_1, x_2, \dots x_n, y)\big)$$

za argumente $x_1, x_2, \dots x_n, y$ će početi pražnjenjem sadržaja registra R_{n+1} (koji sadrži argument y) u neki slobodan registar R_k. Zatim će se izračunati vrijednost $f(x_1, x_2, \dots x_n)$ i registar R_{n+1} s tom vrijednošću će se isprazniti u R_{n+2}. Sada će registri $R_0, R_1, \dots R_{n+2}$ sadržavati redom vrijednosti $x_1, x_2, \dots x_n, 0, h(x_1, x_2, \dots x_n, 0)$ i može se primijeniti funkcija g. Nakon toga se njezina vrijednost prebacuje u R_{n+2}, broj u R_{n+1} se uvećava za 1 i opet se primjenjuje funkcija g. Nakon svakog računanja funkcije g umanjuje se broj u R_k koji služi kao

brojač. Kad se R_k isprazni izvršili smo dovoljan broj koraka i došli smo do vrijednosti $h(x_1, x_2, \dots x_n, y)$.

Računanje vrijednosti funkcije h definirane minimizacijom

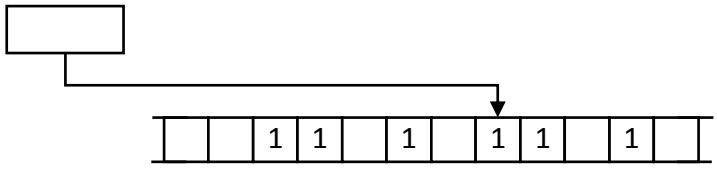
$$h(x_1, x_2, \dots x_n) = \begin{cases} y \text{ ako je } f(x_1, x_2, \dots x_n, y) = 0, a \text{ za svaki } t \leq y \\ f(x_1, x_2, \dots x_n, t) \text{ je definirano i različito od } 0 \\ \\ \text{nedefinirano inače} \end{cases}$$

za argumente $x_1, x_2, \dots x_n$ započet će računanjem funkcije f (R_{n+1} je inicijalno prazan i služi kao zadnji argument funkcije f). Nakon toga se provjerava da li je u registru R_{n+2} (koji sada sadrži vrijednost funkcije f) nula. Ako nije, broj u R_{n+1} se uvećava za 1 i opet se primjenjuje funkcija f. Postupak se ponavlja sve dok se prilikom provjere u R_{n+2} ne nađe nula. Ako se to dogodi, znači da smo pronašli najmanji y koji zadovoljava uvjet. Time je pokazano da je svaka rekurzivna funkcija izračunljiva na registarskom stroju, a time i na Turingovom stroju.

Sada je potrebno još dokazati i da je svaka Turing-izračunljiva funkcija rekurzivna. Ovaj dokaz je relativno složen i da bi u potpunosti opisali njegovu strukturu bilo bi potrebno značajno produžiti poglavlje o rekurzivnim funkcijama kako bi se sve pomoćne funkcije u dokazu mogle opravdano koristiti. S obzirom da je u ovom radu u prvi plan stavljen Turingov stroj kao formulacija izračunljivosti, zadovoljit ćemo se samo vrlo grubim opisom dokaza.[21] Prethodno smo pokazali da se tablica svakog Turingovog stroja može kodirati prirodnim brojem. Da bi iz takvog kodnog broja natrag dobili potrebne informacije o pravilima prijelaza, trebali bismo primijeniti neke poznate aritmetičke funkcije za koje se može pokazati da su rekurzivne. Osim kodiranja tablice potrebno je kodirati i konfiguracije stroja. To znači da treba kodirati informaciju o stanju, poziciji glave i sadržaju na traci za svaki trenutak u računanju. Stanja se lako mogu kodirati njihovim rednim brojevima. Praktična metoda za kodiranje sadržaja na traci je da se traka podijeli na dva djela – lijevi i desni, gdje desni dio počinje od polja na koje

[21] Potpuni dokaz čiju strukturu opisujem nalazi se u:
Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007.; str. 88-93.

pokazuje glava. Sada svaka polovica trake sada sadrži jedan broj u binarnom zapisu (broj na desnoj polovici moramo čitati u suprotnom smjeru). Na primjer, na traci



lijevi i desni dio su predstavljeni brojevima 11010 i 1011 (26 i 11 u decimalnom zapisu). Cijela konfiguracija je trojka koju čine ta dva broja zajedno sa brojem stanja. Informacija o poziciji glave je ovdje implicitno sadržana jer znamo da glava pokazuje na prvo polje desnog djela trake. Trojka koja predstavlja konfiguraciju može se sada već korištenom Gödelovom metodom kodirati kao jedan broj. Da bi iz kodnog broja konfiguracije mogli dobiti svaki član trojke, opet su nam potrebne neke jednostavne aritmetičke funkcije za koje se može pokazati da su rekurzivne. Sad kad imamo rekurzivnim funkcijama opisane sustave kodiranja tablica strojeva i njihovih konfiguracija, mogu se tražiti rekurzivne funkcije kojima će se opisati rad stroja. Cilj je doći do rekurzivne funkcije $F(m, x)$ koja za kodni broj Turingovog stroja m i za ulazni argument x daje vrijednost funkcije koju računa taj stroj (ukoliko ona postoji, inače je vrijednost od F nedefinirana). Tada se za bilo koji Turingov stroj s kodnim brojem m pripadajuća funkcija f koju on računa može definirati kompozicijom iz rekurzivnih funkcija:

$$f(x) = F\big(c_m(x), p_1^1(x)\big)$$

gdje je c_m konstantna funkcija čija je vrijednost m. Time je pokazano da je svaka Turing-izračunljiva funkcija ujedno i rekurzivna.

Na temelju ovih dokaza pokazana je ekvivalentnost između tri različite formulacije izračunljivosti. Time je potvrđena i ekvivalentnost između Turingove i Churchove teze koje se zbog toga obično obuhvaćaju jednim imenom: „Church-Turingova teza“.

U nastavku prelazimo na drugu temu koja je usko povezana s dosadašnjom: teorija formalnih jezika. Razmotrit ćemo formalni aparat za specifikaciju jezika – formalnu gramatiku, a zatim ćemo pokazati da se ona može uzeti kao još jedna formulacija izračunljivosti koja je ekvivalentna dosadašnjima.

VIII. Formalni jezici

1. Osnovne definicije i terminologija

Alfabet je bilo koji neprazan konačan skup znakova. Neki primjeri alfabeta su:

$$\{a, b, c, d, e\}, \qquad \{0,1\}, \qquad \{0,1,2,3,4,5,6,7,8,9\}$$

Ako znakove iz alfabeta (ne nužno sve) poredamo u konačan niz (moguće s ponavljanjem istih znakova), dobivamo ***niz znakova*** nad tim alfabetom. Nizove znakova možemo povezivati u nove nizove. Tu operaciju nazivamo **nastavljanje** ili **konkatenacija**. Neki nizovi znakova nad alfabetom $\{a, b, c, d, e\}$ su: $b, da, bdab, cccc$.

Duljina niza znakova α je broj znakova od kojih se on sastoji i označavamo ga s $|\alpha|$. Na primjer: $|b| = 1$, $|bdab| = 4$. Važno je upamtiti da nizovima znakova smatramo samo nizove konačne duljine. Niz znakova može biti i prazan. S obzirom da takav niz ne sadrži niti jedan znak, označavat ćemo ga posebnim znakom: ε. Duljina praznog niza je 0 ($|\varepsilon|$=0). ε je jedini niz koji se može izgraditi nad bilo kojim alfabetom.

Niz β je ***podniz*** niza α ako je $\alpha = \alpha_1\gamma\alpha_2$ i $\beta = \gamma$. Na primjer, niz da je podniz niza $bdab$. Svaki niz je podniz samoga sebe. ε je podniz svakog niza.

Ako se u nekom nizu ponavlja samo jedan znak, taj niz možemo pisati skraćeno kao $\boldsymbol{x}^d$ gdje je $\boldsymbol{x}$ znak iz niza, a d je duljina niza. Na primjer, niz $cccc$ možemo označiti kao: c^4.

Nad istim alfabetom moguće je izgraditi prebrojivo beskonačno mnogo nizova. Ako s $\mathcal{A}$ označimo neki alfabet, skup svih mogućih nizova nad tim alfabetom označavat ćemo s $\mathcal{A}^*$.[22]

2. Formalni jezik i gramatika

Ako je $\mathcal{A}$ alfabet, ***formalni jezik*** $\mathcal{L}$ nad tim alfabetom je bilo koji podskup od $\mathcal{A}^*$. Neka je alfabet $\mathcal{A}$ definiran s $\mathcal{A} = \{a, b, c\}$. Neki primjeri formalnih jezika nad $\mathcal{A}$ su:

[22] Znak $*$ se općenito koristi kao unarni operator nad skupovima znakova ali i skupovima nizova. Ako je S skup nizova, S^* će biti skup svih nizova koje možemo sastaviti konkatenacijom nizova iz S. S^* će uvijek biti beskonačan osim u slučajevima kad je $S = \emptyset$ ili $S = \{\varepsilon\}$.

$$\{a, b, c\}, \{a, b, c, aa, bb, cc, ab, ba, bc, cb, ac, ca\}, \{a, aa, aaa, aaaa, \dots\}$$

Elemente formalnog jezika nazivamo ***rečenicama***.

S obzirom da je formalni jezik podskup beskonačnog skupa $\mathcal{A}^*$, on sam može biti beskonačan (što je često slučaj kod formalnih jezika s kojima se susrećemo u praksi). Ako je formalni jezik beskonačan, nemoguće ga je definirati popisom svih njegovih rečenica. U tom slučaju moramo na neki drugi način jednoznačno odrediti njegove rečenice. Postoji nekoliko različitih načina na koje se formalni jezik može specificirati. Najpoznatiji su gramatike i automati. U nastavku se posvećujemo gramatikama.

Gramatiku definiramo kao uređenu četvorku $(\mathcal{N}, \mathcal{T}, \mathcal{S}, \mathcal{P})$ gdje je:

$\mathcal{N}$ konačan skup ***neterminalnih znakova (neterminala)***. Neterminali nisu dio formalnog jezika specificiranog gramatikom već samo služe za njegov opis.

$\mathcal{T}$ konačan skup ***terminalnih znakova (terminala)***. To je alfabet jezika kojega gramatika specificira. $\mathcal{N}$ i $\mathcal{T}$ moraju biti disjunktni ($\mathcal{N} \cap \mathcal{T} = \emptyset$).

$\mathcal{P}$ konačan skup uređenih parova nizova (***produkcija***) definiran sa:

$$\{(\alpha, \beta): \alpha = \alpha_1 \boldsymbol{A} \alpha_2;\ \alpha_1, \alpha_2, \beta \in (\mathcal{N} \cup \mathcal{T})^*; \boldsymbol{A} \in \mathcal{N}\}$$ [23]

$\mathcal{S}$ poseban neterminal iz $\mathcal{N}$ kojega nazivamo ***početnim znakom***.

Za neterminale ćemo uvijek koristiti velika slova, a za terminale mala slova i brojke. Početni znak će uvijek biti S. Produkcije ćemo pisati u ovom obliku: $\alpha \rightarrow \beta$ (čitamo kao: „α producira β"). Evo jednog primjera gramatike:

$$\mathcal{N} = \{S, A\} \quad \mathcal{T} = \{0,1\}$$

$$\mathcal{P} = \{(S, 1A), (A, 1A), (A, 0A), (A, \varepsilon)\} \qquad \mathcal{S} = S$$

[23] Drugim riječima, $\mathcal{P}$ je skup uređenih parova (α, β) gdje je α niz znakova koji sadrži najmanje jedan neterminal i bilo koji broj terminala, a β je niz znakova koji sadrži bilo koji broj neterminala i terminala (moguće je i $\beta = \varepsilon$)

Možemo je prikazati na pregledniji način ovako:

$S \to 1A \qquad A \to 0A$

$A \to 1A \qquad A \to \varepsilon$

Ako više produkcija u skupu $\mathcal{P}$ ima jednake nizove na lijevoj strani kao što je to slučaj u prethodnom primjeru, gramatiku možemo skraćeno napisati ovako:

$S \to 1A \qquad A \to 0A \mid 1A \mid \varepsilon$

Nizove $1A$, $0A$ i ε ovdje nazivamo alternativama za A.

Gramatika specificira formalni jezik na način da generira njegove rečenice. Postupak dobivanja pojedinih rečenica iz gramatike nazivamo ***izvođenjem rečenica***. Prije nego počnemo sa objašnjenjem toga postupka, definirat ćemo nekoliko važnih pojmova:

U gramatici $(\mathcal{N}, \mathcal{T}, \mathcal{S}, \mathcal{P})$:

Niz α ***izravno izvodi*** niz β ($\alpha \Longrightarrow \beta$) ako je $\alpha = \alpha_1\gamma\alpha_2, \beta = \beta_1\delta\beta_2$ i $\gamma \to \delta$ je produkcija u $\mathcal{P}$.

Niz α_0 ***izvodi*** niz α_n ($\alpha_0 \overset{n}{\Longrightarrow} \alpha_n$) ako postoje nizovi $\alpha_1, \alpha_2, \ldots \alpha_{n-1}$ takvi da $\alpha_i \Longrightarrow \alpha_{i+1}$ za svaki $0 \leq i \leq n$-1. U nekim situacijama nećemo navoditi broj n nego ćemo pisati $\alpha \overset{*}{\Longrightarrow} \beta$ što znači da postoji broj n takav da $\alpha \overset{n}{\Longrightarrow} \beta$. U tom slučaju $\alpha_0 \Longrightarrow \alpha_1, \alpha_1 \Longrightarrow \alpha_2, \ldots \alpha_{n-1} \Longrightarrow \alpha_n$ zovemo **niz izvođenja** i skraćeno ga pišemo kao: $\alpha_0 \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \cdots \Longrightarrow \alpha_{n-1} \Longrightarrow \alpha_n$.

Niz α je **rečenična forma ako je** $\alpha = S$ ili $S \overset{*}{\Longrightarrow} \alpha$.

Rečenična forma α je rečenica jezika opisanog gramatikom ako je $\alpha \in \mathcal{T}^*$. Jezik generiran gramatikom $\mathcal{G}$ može se označiti s $\mathcal{L}(\mathcal{G})$.

Postupak izvođenja rečenice provodimo na sljedeći način: Polazimo od početnog znaka S i iz njega izravno izvodimo neku rečeničnu formu, odnosno pronalazimo takav niz α_1 da $S \Longrightarrow$

α_1.[24] Ako α_1 nije rečenica, iz α_1 izvodimo sljedeću rečeničnu formu, odnosno pronalazimo takav niz α_2 da $\alpha_1 \Longrightarrow \alpha_2$.[25] Postupak nastavljamo tako dugo dok ne dođemo do rečenice α_n. Na kraju ga možemo u potpunosti prikazati nizom izvođenja: $S \Longrightarrow \alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \cdots \Longrightarrow \alpha_{n-1} \Longrightarrow \alpha_n$.

Evo nekih izvođenja u gramatici iz prethodnog primjera:

$$S \Longrightarrow 1A \Longrightarrow 1 \qquad S \Longrightarrow 1A \Longrightarrow 11A \Longrightarrow 11$$

$$S \Longrightarrow 1A \Longrightarrow 10A \Longrightarrow 10 \qquad S \Longrightarrow 1A \Longrightarrow 10A \Longrightarrow 100A \Longrightarrow 100$$

Ova gramatika opisuje jezik prirodnih brojeva u binarnom zapisu.

Mogli bismo konstruirati znatno složenije gramatike koje opisuju neke zanimljivije jezike ali ovdje ćemo stati s daljnjim navođenjem primjera i posvetit ćemo se odnosu između gramatike i prethodnih formulacija izračunljivosti.[26]

IX. Ekvivalencija Turingovog stroja i formalne gramatike

Turingov stroj smo do sada koristili kao sredstvo za specificiranje funkcija. S druge strane, gramatiku koristimo kao sredstvo za specificiranje jezika. Da bismo uspostavili vezu između Turingovih strojeva i gramatika, moramo uspostaviti vezu između funkcija i jezika.

Funkcija f može se definirati skup uređenih parova (x, y) takvih da $f(x) = y$, a skup uređenih parova se može poistovjetiti s jezikom kojega čine nizovi $1^x 01^y$.

S druge strane, svaki jezik $\mathcal{L}$ se može interpretirati kao funkcija koja je definirana samo nad elementima i $\mathcal{L}$ (funkcija čija domena je $\mathcal{L}$). Vrijednosti funkcije ovdje nisu relevantne.

[24] α_1 je bilo koja alternativa za S.

[25] Iz α_1 izvodimo sljedeću rečeničnu formu na način da pronađemo neku produkciju $\gamma \to \delta$ gdje je γ podniz od α_1. Zatim na nekom mjestu njegovog pojavljivanja u α_1 podniz γ zamijenimo nizom δ.

[26] Neki zanimljiviji primjeri gramatika prikazani su u:
Brcković Ž.; Izvođenje gramatika bez ograničenja - završni rad; Filozofski fakultet, Zagreb, 2009.

Dakle, problem računanja funkcije može se svesti na problem prepoznavanja određenog jezika. Obrnuto, problem prepoznavanja jezika svodiv je na problem računanja određene funkcije.

Ako imamo Turingov stroj T koji računa funkciju f, možemo pronaći stroj T' takav da se T' zaustavlja ako i samo ako mu je inicijalno zadan niz 1^x01^y za koji vrijedi $f(x) = y$. Za stroj T' kažemo da je **prepoznavač** jezika, a jezik koji on prepoznaje možemo značiti s $\mathcal{L}(T')$. T' bi se mogao definirati kao stroj s dvije trake. Inicijalno se na prvoj traci zadaje niz 1^x01^y. T' kopira 1^x na drugu traku i na njoj simulira rad stroja T. Ako se T zaustavlja, uzima se rezultirajući niz drugoj na traci i uspoređuje se s 1^y. Ako su nizovi jednaki, znači da je $f(x) = y$ i niz 1^x01^y se prihvaća. Inače se pokreće beskonačna petlja.

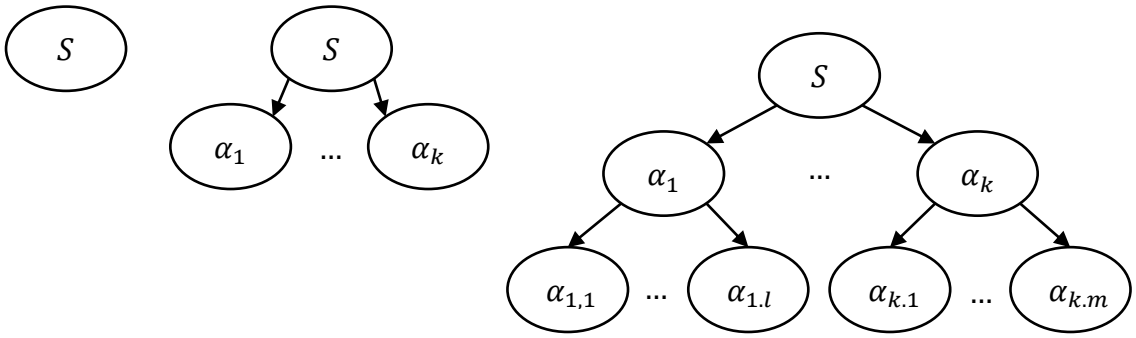
S druge strane, ako imamo stroj T koji prepoznaje jezik $\mathcal{L}$, možemo lako dobiti stroj T' koji računa funkciju s domenom $\mathcal{L}$. Do T' dolazimo tako da stroju T dodamo pravila za ispisivanje neke vrijednosti i postavljanje glave na prvi znak. To će biti vrijednost funkcije za zadani niz.

Sad kad je pokazana veza između računanja funkcije i prepoznavanja jezika možemo se posvetiti samo Turingovom stroju kao prepoznavaču i pokazati ekvivalenciju s formalnom gramatikom.

Neka je zadana proizvoljna gramatika $\mathcal{G}$ koja specificira jezik $\mathcal{L}(\mathcal{G})$. Tražimo Turingov stroj T koji prepoznaje jezik $\mathcal{L}(\mathcal{G})$. T će biti Turingov stroj s više traka. Na prvoj traci će se inicijalno nalaziti ulazni niz α. Stroj prvo provjerava da li je α niz terminala. Ako nije, pokreće se beskonačna petlja jer α ne može biti rečenica. U suprotnom T nastavlja s radom. Na drugu traku se ispisuje prva rečenična forma izvođenja u gramatici $\mathcal{G}$, a to je po dogovoru uvijek početni znak S. Sada se provjerava da li je trenutna rečenična forma (početni znak S) jednaka nizu α.[27] Ako nije, na traku se ispisuje svih k rečeničnih formi $\alpha_1, \alpha_2, \ldots \alpha_k$ koje se mogu dobiti izravnim izvođenjem iz trenutne rečenične forme S:[28]

[27] Naravno, na početku ovaj uvjet neće biti ispunjen jer je α niz terminala, a S je neterminal.
[28] Njihov broj će uvijek biti konačan.

α_1 sada postaje nova trenutna rečenična forma i ponavlja se isti postupak - provjerava se da li je α_1 jednak nizu α (Za pamćenje pozicije na kojoj je trenutna rečenična forma može se koristiti treća traka). Ako α_1 nije jednaka zadanom nizu α, na kraj niza $S, \alpha_1, \alpha_2, \ldots \alpha_k$ se dopisuje svih l rečeničnih formi $\alpha_{1.1}, \alpha_{1.2}, \ldots \alpha_{1.l}$ koje se mogu dobiti izravnim izvođenjem iz α_1. Sada na traci imamo $S, \alpha_1, \alpha_2, \ldots \alpha_k, \alpha_{1.1}, \alpha_{1.2}, \ldots \alpha_{1.l}$ i α_2 postaje nova trenutna rečenična forma. Ovaj postupak se može razumjeti kao konstrukcija stabla. Počinje se od korjena S. Zatim se uvode svi susjedni čvorovi korijena, zatim svi susjedni čvorovi ovih čvorova itd...



Postupak može završiti na dva načina:

1. T nema više rečeničnih formi na raspolaganju (to znači da je cijelo stablo konstruirano, odnosno T je došao do kraja sadržaja na drugoj traci). U tom slučaju pokreće se beskonačna petlja.
2. Trenutna rečenična forma je jednaka zadanom nizu α. U tom slučaju T ulazi u završno stanje što znači da je prihvaćen niz α.

Dakle, T se prepoznaje niz α ako i samo ako α pripada jeziku $\mathcal{L}(\mathcal{G})$. Time je pokazano da za svaku gramatiku $\mathcal{G}$ postoji stroj T takav da je $\mathcal{L}(\mathcal{G}) = \mathcal{L}(T)$.

Sada prelazimo na drugi dio dokaza. Neka je zadan Turingov stroj (prepoznavač) T s jednom trakom. Tražimo gramatiku $\mathcal{G}$ takvu da je $\mathcal{L}(\mathcal{G}) = \mathcal{L}(T)$. Ideja je da gramatika na početku

izgenerira rečeničnu formu $XQ_0\alpha X$ gdje je α bilo koji niz terminala.[29] Za to mogu poslužiti produkcije:

$S \to ZQ_0AZ$ $\quad\quad$ $A \to \varepsilon \mid a_1A \mid a_2A \ldots \mid a_nA$ za svaki terminal a_i

Niz $Q_0\alpha$ se može shvatiti kao opis inicijalne konfiguracije stroja T. Neterminal Q_0 predstavlja početno stanje, a α predstavlja ulazni niz na traci. Činjenica da se Q_0 nalazi neposredno ispred α znači da inicijalno glava stroja pokazuje na prvi znak niza α. Neterminali Z s jedne i druge strane služe za proizvoljno dodavanje nula, odnosno produživanje trake po potrebi. Za to će služiti produkcije:

$Z \to 0Z \mid \varepsilon$

Sada za svako pravilo stroja T uvodimo odgovarajuće produkcije:

Za svako pravilo $\delta\big((q_k, a_m)\big) = (a_n, D, q_l)$ uvodimo produkciju:

$Q_ka_m \to a_nQ_l$

Za svako pravilo $\delta\big((q_k, a_m)\big) = (a_n, L, q_l)$ uvodimo niz produkcija:

$a_iQ_ka_m \to Q_la_ia_n$ za svaki terminal a_i

Za svako pravilo $\delta\big((q_k, a_m)\big) = (a_n, S, q_l)$ uvodimo produkciju:

$Q_ka_m \to Q_la_n$

Na kraju za završno stanje Q_f uvodimo produkciju:

$Q_f \to \varepsilon$

Rečenične forme gramatike potpuno će analogno pratiti konfiguracije Turingovog stroja. Gramatika će generirati niz α ako i samo ako T dolazi do završnog stanja za argument α,

[29] Q_0 treba razumjeti kao jedan znak (neterminal).

odnosno ako T prepoznaje α. Time je pokazano da za svaki Turingov stroj T postoji gramatika $\mathcal{G}$ takva da je $\mathcal{L}(\mathcal{G}) = \mathcal{L}(T)$.

Kad smo definirali proces prepoznavanja jezika na Turingovom stroju, mogli smo postaviti uvjet da se stroj nužno mora za svaki zadani niz zaustaviti, a na traku mora ispisati potvrdan ili negativan odgovor ovisno o tome da li je niz prihvaćen ili ne. Takav proces bi sigurno bio poboljšanje jer bi Turingov stroj za svaki zadani niz nakon konačnog broja koraka dao odgovor na pitanje: Da li taj niz pripada jeziku? Međutim, uvođenjem ovoga uvjeta bi ograničili skup jezika koji se na taj način mogu specificirati. Jezici koje prepoznaju takvi strojevi zovu se **rekurzivnima**. Općenito, jezik je rekurzivan ako postoji efektivna metoda koja za svaki zadani niz nakon konačnog broja koraka dovodi do odgovora na pitanje: Da li taj niz pripada jeziku? S druge strane, jezike prepoznate Turingovim strojevima za koje nije nužno da se uvijek zaustave zovemo **rekurzivno prebrojivima**. Rekurzivno prebrojivi jezici su oni za koje postoji nepotpuna metoda koja dovodi do potvrdnog odgovora za one nizove koji pripadaju jeziku, ali ne nužno i za nizove koji ne pripadaju jeziku. Za takve nizove metoda može trajati u beskonačnost. Ekvivalencijom između gramatika i Turingovih strojeva pokazano je da gramatike također opisuju rekurzivno prebrojive jezike. To znači da ne postoji efektivna metoda kojom bi uvijek mogli za zadanu gramatiku $\mathcal{G}$ i niz α utvrditi da li je α rečenica jezika $\mathcal{L}(\mathcal{G})$.

X. Logika prvog reda

1. Uvod

Logika se tradicionalno bavi proučavanjem određene vrste svojstava sudova i odnosa između sudova. Bitna karakteristika tih svojstava i odnosa je to što su oni određeni isključivo formom sudova, a ne sadržajem. **(Logičku) formu** suda ugrubo možemo opisati kao strukturu koja se otkriva nakon što apstrahiramo od sadržaja suda. Uzmimo za primjer sljedeći sud:

„Svi psi su sisavci.“

Pojmove „psi“ i „sisavci“ možemo u ovom sudu izdvojiti kao nositelje sadržaja. Prema njima znamo „o čemu se govori“ u sudu. Ako apstrahiramo od njihovog značenja i zamijenimo ih znakovima F i G, dobivamo:

„Svi F su G“

To što smo dobili nazivamo **formulom**. Ona predstavlja logičku formu prethodnog suda. Slova F i G imaju ulogu varijabli. Na njih možemo gledati kao na prazna mjesta u formuli. Ako ih popunimo odgovarajućim izrazima, dobit ćemo novi sud.

Izrazi koji su ostali u formuli („svi“, „su“) nemaju više veze sa sadržajem suda. Takvi izrazi su prisutni u svakom diskursu neovisno o temi i području kojem diskurs pripada. Drugim riječima, izraze kao što su „svi“, „su“, „i“, „ili“, „ako“ smo uvijek prisiljeni koristiti jer su oni sredstva pomoću kojih konstruiramo sudove bilo koje vrste. S druge strane, izraze „psi“ i „sisavci“ ćemo vjerojatno koristiti samo u onim situacijama kad želimo govoriti o psima i sisavcima.

Odnos između sudova koji je oduvijek bio od posebnog interesa za logiku je odnos **logičke posljedice**. To je odnos u kojem se konkluzija zaključka treba nalaziti prema premisama da bi zaključak smatrali valjanim. Kao što smo ranije rekli, taj odnos (kao i drugi logički odnosi) je determiniran isključivo logičkom formom sudova koji čine zaključak (premisa i konkluzije). Uzmimo za primjer sljedeći zaključak:

Svi psi su sisavci.
Svi sisavci su životinje.

Svi psi su životinje.

Apstrahiranjem od sadržaja premisa i konkluzije dobivamo **logičku formu zaključka**:

Svi F su G.
Svi G su H.

Svi F su H.

Kad stavimo konkretne izraze na mjesta F, G i H, konkluzija će sigurno biti istinita ako su istinite sve premise. Formu zaključka s takvim svojstvom nazivamo **valjanom formom**. Zaključak s takvom formom je **valjan zaključak**.

Temeljni zadatak logike je sistematiziranje valjanih formi zaključaka.[30] S obzirom na raznolikost formi zaključaka koje se mogu pojaviti u svakodnevnom diskursu, potrebno je prvo učiniti selekciju. Dio logike kojim ćemo se u nastavku baviti naziva se **logika prvog reda**.

Korištenje prirodnog jezika za reprezentaciju logičkih formi stvara određene poteškoće u bavljenju logikom.[31] Ta poteškoća se rješava uvođenjem formalnog jezika za predstavljanje logičkih formi.

2. Jezik logike prvog reda

a) Sintaksa

Alfabet formalnog jezika logike sastojat će se od sljedećih simbola:[32]

a) Beskonačan skup predikatnih varijabli s indeksima (gornji indeks označava „arnost" predikatne varijable):

$P^1 \;\; P^2 \;\; P^3 \;\; \ldots$

$P_1^1 \;\; P_1^2 \;\; P_1^3 \;\; \ldots$

$P_2^1 \;\; P_2^2 \;\; P_2^3 \;\; \ldots$

$\ldots$

Ugrubo, predikatne varijable predstavljaju prazna mjesta u formuli za izraze koji označavaju svojstva i relacije u najširem smislu (predikate). Na primjer, u sudu „svi ljudi su smrtni" možemo identificirati dva svojstva: „biti čovjekom" (čovječnost) i svojstvo smrtnosti. Takva svojstva se predstavljaju unarnim predikatima. U sudu „svatko voli nekoga" možemo identificirati binarnu relaciju koja postoji između svake dvije osobe koje se vole. Takva relacija se predstavlja

[30] Logiku zapravo zanima odnos logičke posljedice između sudova bez obzira na to da li ti sudovi čine neki stvaran ili mogući zaključak, dokaz ili argument.

[31] Više o ovoj temi u:
Strawson, P. F.; Introduction to Logical Theory; Methuen & Co LTD 1964.

[32] U poglavlju o formalnim jezicima alfabet je definiran kao konačan skup znakova. Ovdje kao alfabet uzimamo prebrojivo beskonačan skup simbola. Međutim, simboli kao što je F_1^2 mogu se gledati kao složeni izrazi sastavljeni od konačnog skupa jednostavnih znakova. Da smo uveli linearnu notaciju za indeksiranje i umjesto F_1^2 pisali $F_{(2,1)}$, mogli bi nad konačnim alfabetom $\{F, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, (,)\}$ definirati pravila za sastavljanje predikatnih varijabli.

binarnim predikatom. Općenito, predikatna varijabla $\boldsymbol{P_n^m}$ *se treba shvatiti kao prazno mjesto za m-arni predikat.*

b) Beskonačan skup individualnih varijabli s indeksima: $x\ y\ z\ x_1\ y_1\ z_1$ …

Individualne varijable predstavljaju prazna mjesta u formuli za vlastita imena ili zamjenice. Vlastito ime je izraz koji označava točno jedan određeni entitet. Na primjer, u sudu „Sokrat je filozof“ riječ „Sokrat“ stoji kao vlastito ime za određenu osobu. Riječ „filozof“ s druge strane ima ulogu predikata i pripada prethodnoj kategoriji izraza (predikatima).

c) (Istinosno funkcionalni) veznici: ¬ ∧ ∨ → ↔

U tablici su uz standardne nazive prikazani i izrazi u prirodnom jeziku čija funkcija približno odgovara ovim veznicima:

… ¬ …	**Negacija**: **„**Nije slučaj da ....“
… ∧ …	**Konjunkcija**: **„**... i ...“
… ∨ …	**(Uključujuća) disjunkcija**: „... ili ...“
… → ⋯	**Implikacija**: „Ako ... onda ...“
… ↔ ⋯	**Ekvivalencija**: „... ako i samo ako ...“

d) Kvantifikatori: ∀ ∃

Uz kvantifikator se uvijek pojavljuje individualna varijabla:

$\forall \boldsymbol{x}$ ...	**Univerzalni kvantifikator**: „Za svaku stvar x vrijedi ...“
$\exists \boldsymbol{x}$ ...	**Egzistencijalni kvantifikator**: „Postoji stvar x za koju vrijedi...“

e) Znak za jednakost: $=$

f) Pomoćni znakovi (zagrade i zarez): ( ) ,

Jezik logike predikata čine formule. Formula je niz znakova definiran sljedećim pravilima:

- Ako je $\boldsymbol{P_n^m}$ predikatna varijabla, a $\boldsymbol{x_1, x_2, \dots x_m}$ su individualne varijable (ne nužno različite), tada je $\boldsymbol{P_n^m(x_1, x_2, \dots x_m)}$ (atomarna) formula.
- Ako su $\boldsymbol{x}$ i $\boldsymbol{y}$ individualne varijable, tada je $\boldsymbol{x = y}$ (atomarna) formula.
- Ako je $\boldsymbol{A}$ formula i $\boldsymbol{x}$ individualna varijabla, tada su $\forall \boldsymbol{x}\, \boldsymbol{A}$ i $\exists \boldsymbol{x}\, \boldsymbol{A}$ formule.
- Ako je $\boldsymbol{A}$ formula, tada je $\neg \boldsymbol{A}$ formula.
- Ako su $\boldsymbol{A}$ i $\boldsymbol{B}$ formule, tada su $(\boldsymbol{A} \wedge \boldsymbol{B})$, $(\boldsymbol{A} \vee \boldsymbol{B})$, $(\boldsymbol{A} \rightarrow \boldsymbol{B})$ i $(\boldsymbol{A} \leftrightarrow \boldsymbol{B})$ formule.
- Ništa drugo nije formula.

Prije nego prijeđemo na semantiku potrebno je definirati nekoliko važnih sintaktičkih pojmova:

- Pojavljivanje individualne varijable $\boldsymbol{x}$ u formuli $\boldsymbol{A}$ nazivamo **slobodnim** ako to nije pojavljivanje u podformuli oblika $\forall \boldsymbol{x}\, \boldsymbol{B}$ ili $\exists \boldsymbol{x}\, \boldsymbol{B}$. Inače pojavljivanje varijable $\boldsymbol{x}$ nazivamo **vezanim**.
- Neka je $\boldsymbol{A}$ formula u kojoj nijedno slobodno pojavljivanje varijable $\boldsymbol{x}$ nije u podformuli oblika $\forall \boldsymbol{y}\, \boldsymbol{B}$ ili $\exists \boldsymbol{y}\, \boldsymbol{B}$. S $\boldsymbol{A}\left(\boldsymbol{x}/\boldsymbol{y}\right)$ ćemo označavati formulu koja je dobivena iz formule $\boldsymbol{A}$ zamjenom varijable $\boldsymbol{x}$ na svakom mjestu njezinog slobodnog pojavljivanja s varijablom $\boldsymbol{y}$.
- Formule oblika $\neg \boldsymbol{x = y}$ ćemo skraćeno pisati kao $\boldsymbol{x \neq y}$.

b) Semantika

Interpretacija jezika logike prvog reda je odabir nekog skupa D funkcije v s sljedećim svojstvima (Skup D tada nazivamo **domenom kvantifikacije**):

- Svakoj individualnoj varijabli $\boldsymbol{x}$ funkcija v pridružuje jedan element iz D.

  $v(\boldsymbol{x}) = a$ gdje je $a \in D$.

- Svakoj predikatnoj varijabli $\boldsymbol{P_n^m}$ funkcija v pridružuje podskup kartezijevog produkta D^m.

  $v(\boldsymbol{P_n^m}) = A$ gdje je $A \subseteq D^m$.

- Svakoj atomarnoj formuli $\boldsymbol{P_n^m(x_1, x_2, \dots x_m)}$ funkcija v pridružuje instinosnu vrijednost „istina“ ($\top$) ili „neistina“ ($\bot$) prema sljedećem pravilu:

  $v\big(\boldsymbol{P_n^m(x_1, x_2, \dots x_m)}\big) = \top$ ako je $\big(v(\boldsymbol{x_1}), v(\boldsymbol{x_2}), \dots v(\boldsymbol{x_m})\big) \in v(\boldsymbol{P_n^m})$.

  Inače je $v\big(\boldsymbol{P_n^m(x_1, x_2, \dots x_m)}\big) = \bot$.

- Složenim formulama funkcija v pridružuje instinosne vrijednosti prema sljedećim pravilima:

  $v(\neg \boldsymbol{A}) = \top$ ako je $v(\boldsymbol{A}) = \bot$. Inače je $v(\neg \boldsymbol{A}) = \bot$.

  $v\big((\boldsymbol{A} \wedge \boldsymbol{B})\big) = \top$ ako je $v(\boldsymbol{A}) = \top$ i $v(\boldsymbol{B}) = \top$. Inače je $v\big((\boldsymbol{A} \wedge \boldsymbol{B})\big) = \bot$.

  $v\big((\boldsymbol{A} \vee \boldsymbol{B})\big) = \bot$ ako je $v(\boldsymbol{A}) = \bot$ i $v(\boldsymbol{B}) = \bot$. Inače je $v\big((\boldsymbol{A} \vee \boldsymbol{B})\big) = \top$.

  $v\big((\boldsymbol{A} \to \boldsymbol{B})\big) = \bot$ ako je $v(\boldsymbol{A}) = \top$ i $v(\boldsymbol{B}) = \bot$. Inače je $v\big((\boldsymbol{A} \to \boldsymbol{B})\big) = \top$.

  $v\big((\boldsymbol{A} \leftrightarrow \boldsymbol{B})\big) = \top$ ako je $v(\boldsymbol{A}) = v(\boldsymbol{B})$. Inače je $v\big((\boldsymbol{A} \leftrightarrow \boldsymbol{B})\big) = \bot$.

- Svakoj formuli $\forall \boldsymbol{x A}$ i $\exists \boldsymbol{x A}$ funkcija v pridružuje instinosnu vrijednost prema sljedećim pravilima:

  $\forall \boldsymbol{x}\, \boldsymbol{A} = \top$ ako za svaki $a \in D$ vrijedi $v\left(\boldsymbol{A}\left(\boldsymbol{x}/\boldsymbol{y}\right)\right) = \top$ gdje je $\boldsymbol{y}$ individualna varijabla takva da $v(\boldsymbol{y}) = a$. Inače je $\forall \boldsymbol{x}\, \boldsymbol{A} = \bot$.

  $\exists \boldsymbol{x}\, \boldsymbol{A} = \top$ ako postoji $a \in D$ za koji vrijedi $v\left(\boldsymbol{A}\left(\boldsymbol{x}/\boldsymbol{y}\right)\right) = \top$ gdje je $\boldsymbol{y}$ individualna varijabla takva da $v(\boldsymbol{y}) = a$. Inače je $\exists \boldsymbol{x}\, \boldsymbol{A} = \bot$.[33]

[33] Ako za neki $a \in D$ ne postoji individualna varijabla $\boldsymbol{y}$ takva da je $v(\boldsymbol{y}) = a$, proširujemo jezik uvođenjem nove individualne varijable $\boldsymbol{z}$ i funkciju v proširujemo pravilom: $v(\boldsymbol{z}) = a$.

- Svakoj formuli $\boldsymbol{x} = \boldsymbol{y}$ funkcija v pridružuje istinostnu vrijednost prema sljedećem pravilu:

  $v(\boldsymbol{x} = \boldsymbol{y}) = \top$ ako je $v(\boldsymbol{x}) = v(\boldsymbol{y})$. Inače je $v(\boldsymbol{x} = \boldsymbol{y}) = \bot$.

Odabirom interpretacije svaka formula dobiva po jednu istinosnu vrijednost.

Uvodimo nekoliko važnih semantičkih pojmova:

- Formula $\boldsymbol{A}$ je **konzistentna** ako i samo ako je istinita pod nekom interpretacijom. Inače je **inkonzistentna**.
- Formula $\boldsymbol{A}$ je **valjana** ili **logički istinita** ako i samo ako je istinita u svakoj interpretaciji.
- Skup formula Γ je **konzistentan** ako postoji interpretacija koja čini istinitima sve formule u Γ. Inače je Γ **inkonzistentan**.
- Formula $\boldsymbol{A}$ je **logička posljedica** skupa formula Γ ako i samo ako ne postoji interpretacija koja čini sve formule u Γ istinitima, a $\boldsymbol{A}$ neistinitom. U tom slučaju ćemo još reći da Γ **implicira** $\boldsymbol{A}$.

Kao što je rečeno, jezik logike prvog reda čine formule. Formule nemaju istinosnu vrijednost. One samo služe za prikazivanje logičke forme sudova. Da bi dobili „pravi“ jezik u kojem ćemo moći izraziti sudove, dovoljno je da skup predikatnih varijabli zamijenimo skupom predikata, skup individualnih varijabli dopunimo skupom vlastitih imena (individualnih konstanti), te odaberemo standardnu interpretaciju za uvedene simbole. Simbole koje smo na taj način uveli nazivamo **nelogičkim simbolima**. Ostali simboli su **logički**.

XI. Problem odlučivosti logike prvog reda

Problem odlučivosti logike prvog reda sastoji se u pronalasku efektivnog postupka kojim bi se moglo za proizvoljnu formulu ustanoviti da li je valjana. Problem je opće poznat pod nazivom „Entscheidungsproblem“ i prezentirao ga je David Hilbert 1928. godine. Danas znamo da je to još jedan od nerješivih problema, to jest, postupak rješavanja takvog problema nije moguće definirati na temelju standardnih modela izračunljivosti kao što su Turingovi strojevi ili rekurzivne funkcije. Dokaze za tu tvrdnju su neovisno jedan o drugom izveli Alonzo Church (1935-1936.) i Alan Turing (1936.). U ovom radu je prikazana struktura drugačijeg dokaza u

kojem se uzima formalna gramatika kao model izračunljivosti. Ranije smo pokazali da je problem odlučivanja o pripadnosti niza jeziku opisanom formalnom gramatikom nerješiv. Sada ćemo taj problem svesti na problem odlučivosti za logiku prvog reda. Drugim riječima, pokazat ćemo kako za proizvoljnu gramatiku $\mathcal{G}$ i niz terminala α konstruirati formulu u jeziku logike prvog reda koja je valjana ako i samo ako je $\alpha \in \mathcal{L}(\mathcal{G})$. Kad bi problem odlučivanja o valjanosti formule bio rješiv, dobili bismo indirektno i efektivni postupak odlučivanja o pripadnosti proizvoljnog niza terminala α proizvoljnom jeziku $\mathcal{L}(\mathcal{G})$. Pošto znamo da je drugi problem nerješiv, zaključujemo da prvi problem također mora biti nerješiv.[34]

Prvo definiramo jezik kojim ćemo moći govoriti o zadanoj gramatici $\mathcal{G}$. Zatim pronalazimo skup rečenica Γ koje u standardnoj interpretaciji opisuju sva bitna svojstva zadane gramatike $\mathcal{G}$ i niza terminala α. Zatim pronalazimo rečenicu R u tom jeziku koja pod standardnom interpretacijom tvrdi $\alpha \in \mathcal{L}(\mathcal{G})$. Na kraju pokazujemo da skup formula Γ implicira rečenicu R ako i samo ako je $\alpha \in \mathcal{L}(\mathcal{G})$. Kad smo to učinili, problem utvrđivanja istinitosti rečenice $\alpha \in \mathcal{L}(\mathcal{G})$ je sveden na problem provjeravanja da li skup rečenica Γ implicira R. Ovaj drugi problem je ekvivalentan provjeravanju da li je implikacija $(A_1 \wedge A_2 \wedge \ldots A_n) \rightarrow R$ (gdje je $\{A_1, A_2, \ldots A_n\} = \Gamma$) valjana.

Prije nego započnemo sa specifikacijom jezika, uvest ćemo sustav kodiranja znakova i nizova znakova gramatike $\mathcal{G}$ nenegativnim cijelim brojevima. To će nam omogućiti da indirektno govorimo o svojstvima znakova i nizova znakova referirajući se na njihove kodne brojeve.[35]

Prvo svakom znaku iz skupa $\mathcal{N} \cup \mathcal{T}$ zadane gramatike $\mathcal{G}$ pridružujemo jedinstven nenegativan cijeli broj na sljedeći način: Odaberemo neki uređaj na skupu $\mathcal{N} \cup \mathcal{T} = \{a_1, a_2, a_3 \ldots\}$ i zatim svakom znaku $a_i \in \mathcal{N} \cup \mathcal{T}$ pridružimo kodni broj $c_c[a_i] = i$. Broj $c_c[a_i]$ ćemo zvati kodnim brojem znaka a_i, a c_c ćemo zvati funkcijom kodiranja zakova. Svaki niz $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$ sada možemo predstaviti nizom kodnih brojeva znakova koji se u njemu pojavljuju.

[34] Glavna ideja ovog dokaza preuzeta je iz: Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press, 2007., str. 126

[35] Na taj način možemo domenu kvantifikacije u standardnoj interpretaciji jezika svesti na skup nenegativnih cijelih brojeva.

Primjer:

Neka su znakovi niza $aHbbd$ kodirani na sljedeći način:

$$c_c[a] = 5 \qquad c_c[H] = 2 \qquad c_c[b] = 7 \qquad c_c[d] = 6$$

Niz $aHbbd$ će sada biti predstavljen nizom kodnih brojeva:

$$5,2,7,7,6$$

Na svako mjesto u nizu ćemo se referirati njegovim rednim brojem počevši od nule. Na primjer, u prethodnom nizu se broj 5 pojavljuje na mjestu s rednim brojem 0. Sada svakom nizu $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$ pridružujemo jedinstven kodni broj $c_s[\alpha]$ prema Gödelovom sustavu kodiranja uz dodatno pravilo za prazan niz: $c_s[\varepsilon] = 1$.[36, 37]

U standardnoj interpretaciji našeg jezika domena kvantifikacije će biti skup nenegativnih cijelih brojeva. Kao individualne varijable ćemo uz $x, y, z, x_1, \ldots$ koristiti i $k, l, m, n, \alpha, \beta, \gamma, \delta, \alpha_1, \ldots$ Iako sve individualne varijable imaju istu domenu, koristit ćemo različite vrste varijabli da bi istaknuli kako interpretiramo brojeve na koje one upućuju u različitim kontekstima. Varijable $x, y, z \ldots$ ćemo koristiti kad se želimo referirati na kodne brojeve znakova iz skupa $\mathcal{N} \cup \mathcal{T}$. Varijable $\alpha, \beta, \gamma, \delta, \alpha_1, \ldots$ ćemo koristiti kad se želimo referirati na kodne brojeve nizova znakova iz $(\mathcal{N} \cup \mathcal{T})^*$. Varijable k, l, m, n ćemo koristiti u svim ostalim slučajevima kad brojeve na koje se referiramo ne interpretiramo kao kodove.

Sada dajemo popis jednog djela primitivnih nelogičkih simbola u našem jeziku zajedno s objašnjenjima njihovih značenja u standardnoj interpretaciji:

0	*Vlastito ime (individualna konstanta) koje označava broj* 0.
$S(m, n)$	*Predikat s značenjem: „Broj n je sljedbenik broja m“*
$+(k, l, m)$	*Predikat s značenjem: „Broj m je zbroj brojeva k i l“*
$<(m, n)$	*Predikat s značenjem: „Broj m je manji od broja n“*

[36] Više o Gödelovom sustavu kodiranja na str. 8.

[37] Valja napomenuti da nije svaki nenegativan cijeli broj postao kodni broj znaka ili niza, ali to ne predstavlja problem.

$P(\alpha, x, n)$ *Predikat s značenjem: „Na mjestu n u nizu s kodnim brojem α se nalazi znak s kodnim brojem x“*

$D(\alpha, \beta)$ *Predikat s značenjem: „Postoji produkcija u gramatici $\mathcal{G}$ gdje je niz s kodnim brojem α na lijevoj, a niz s kodnim brojem β na desnoj strani te produkcije.“*

$G(\alpha)$ *Predikat s značenjem: „Niz s kodnim brojem α je rečenična forma gramatike $\mathcal{G}$“*[38]

U nastavku ćemo radi jednostavnosti umjesto „niz s kodnim brojem α“ samo govoriti „niz α“ i umjesto „znak s kodnim brojem x“ ćemo govoriti „znak x“. To ne bi trebalo dovesti do nejasnoća ako stalno imamo na umu da domena kvantifikacije nije skup znakova i nizova znakova, nego je skup nenegativnih cijelih brojeva koje u određenim kontekstima interpretiramo kao kodove znakova i nizova znakova.

Definicije:

$$\boldsymbol{A}(\boldsymbol{n}') \overset{def}{\longleftrightarrow} \exists \boldsymbol{k}\,\big(S(\boldsymbol{n}, \boldsymbol{k}) \wedge \boldsymbol{A}(\boldsymbol{k})\big)^{39}$$

$$\boldsymbol{A}(\boldsymbol{k} + \boldsymbol{l}) \overset{def}{\longleftrightarrow} \exists \boldsymbol{m}\big(+(\boldsymbol{k}, \boldsymbol{l}, \boldsymbol{m}) \wedge \boldsymbol{A}(\boldsymbol{m})\big)$$

$$-(\boldsymbol{k}, \boldsymbol{l}, \boldsymbol{m}) \overset{def}{\longleftrightarrow} +\,(\boldsymbol{l}, \boldsymbol{m}, \boldsymbol{k})$$

$$\boldsymbol{A}(\boldsymbol{k} - \boldsymbol{l}) \overset{def}{\longleftrightarrow} \exists \boldsymbol{m}\big(-(\boldsymbol{k}, \boldsymbol{l}, \boldsymbol{m}) \wedge \boldsymbol{A}(\boldsymbol{m})\big)$$

$$(\boldsymbol{m} < \boldsymbol{n}) \overset{def}{\longleftrightarrow} <(\boldsymbol{m}, \boldsymbol{n}) \qquad (\boldsymbol{m} \leq \boldsymbol{n}) \overset{def}{\longleftrightarrow} (\boldsymbol{m} < \boldsymbol{n}) \vee (\boldsymbol{m} = \boldsymbol{n})$$

$$(\boldsymbol{m} > \boldsymbol{n}) \overset{def}{\longleftrightarrow} (\boldsymbol{n} < \boldsymbol{m}) \qquad (\boldsymbol{m} \geq \boldsymbol{n}) \overset{def}{\longleftrightarrow} (\boldsymbol{n} \leq \boldsymbol{m})$$

Za svaki broj definiramo njemu pripadajuću individualnu konstantu na sljedeći način:

$$\boldsymbol{A}(1) \overset{def}{\longleftrightarrow} \boldsymbol{A}(0') \qquad \boldsymbol{A}(2) \overset{def}{\longleftrightarrow} \boldsymbol{A}(1') \qquad \boldsymbol{A}(3) \overset{def}{\longleftrightarrow} \boldsymbol{A}(2') \qquad \ldots$$

[38] To zapravo znači da postoji niz izvođenja koji vodi od početnog znaka gramatike $\mathcal{G}$ do tog niza.

[39] $\boldsymbol{A}(\boldsymbol{n}')$ označava formulu u kojoj se $\boldsymbol{n}'$ pojavljuje na mjestu koje bi inače zauzimala individualna varijabla ili individualna konstanta ($\boldsymbol{n}'$ se pojavljuje kao term), a $\boldsymbol{A}(\boldsymbol{k})$ je formula dobivena iz $\boldsymbol{A}(\boldsymbol{n}')$ zamjenom simbola $\boldsymbol{n}'$ na svakom mjestu njegovog pojavljivanja s nekom novom varijablom $\boldsymbol{k}$ koje nema u $\boldsymbol{A}$. Isto vrijedi i za ostale definicijske ekvivalencije takvog tipa.

Sada navodimo jedan dio rečenica iz skupa Γ. To su rečenice koje opisuju temeljna svojstva brojeva:

1. $\forall x\ (0 \neq x')$ — *0 nije sljedbenik nijednog broja.*
2. $\forall x\ \forall y\ (x' = y' \leftrightarrow x = y)$ — *Sljedbenik i prethodnik svakog broja su jedinstveni.*
3. $\forall x\ \neg(x < 0)$ — *0 je najmanji broj.*
4. $\forall x\ \forall y\ (x' = y \rightarrow x < y)$ — *Broj je uvijek manji od svojeg sljedbenika.*
5. $\forall x\ \forall y\ \forall z\ \big((x < y \wedge y < z) \rightarrow x < z\big)$ — *Relacija „manje" je tranzitivna.*
6. $\forall x\ \forall y\ (x < y \rightarrow x \neq y)$ — *Nijedan broj nije manji od samoga sebe.*
7. $\forall x\ (x + 0) = x$ — *Rekurzivna definicija zbrajanja.*
8. $\forall x\ \forall y\ (x + y') = (x + y)'$

Definicije:

$$\boldsymbol{A}\big(\alpha_{[\boldsymbol{n}]}\big) \overset{def}{\longleftrightarrow} \exists \boldsymbol{x}\ (P(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{n}) \wedge \boldsymbol{A}(\boldsymbol{x}))$$

$L(\alpha, l)$ — *Predikat s značenjem: „Duljina niza α je l (niz α ima l znakova)"*

$$L(\boldsymbol{\alpha}, \boldsymbol{l}) \overset{def}{\longleftrightarrow} \forall \boldsymbol{k}\ \big(\boldsymbol{k} < \boldsymbol{l} \leftrightarrow \exists \boldsymbol{x}\ P(\boldsymbol{\alpha}, \boldsymbol{x}, \boldsymbol{k})\big)$$

$$\boldsymbol{A}(|\boldsymbol{\alpha}|) \overset{def}{\longleftrightarrow} \exists \boldsymbol{l}\ \big(L(\boldsymbol{\alpha}, \boldsymbol{l}) \wedge \boldsymbol{A}(\boldsymbol{l})\big)$$

U skup Γ dodajemo rečenicu:

9. $\forall \alpha\ \exists l\ (|\alpha| = l)$

Svaki niz ima (konačnu) duljinu.

Definicije:

$C(\alpha, \beta, \gamma)$ — *Predikat s značenjem: „Niz γ je rezultat konkatenacije nizova α i β"*

$$C(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \overset{def}{\longleftrightarrow}$$
$$\forall \boldsymbol{k}\ \big(\boldsymbol{k} < |\boldsymbol{\alpha}| \rightarrow \boldsymbol{\gamma}_{[\boldsymbol{k}]} = \boldsymbol{\alpha}_{[\boldsymbol{k}]}\big)\ \wedge$$
$$\forall \boldsymbol{k}\big(\ (\boldsymbol{k} \geq |\boldsymbol{\alpha}| \wedge \boldsymbol{k} < |\boldsymbol{\alpha}| + |\boldsymbol{\beta}|) \rightarrow \boldsymbol{\gamma}_{[\boldsymbol{k}]} = \boldsymbol{\beta}_{[\boldsymbol{k}-|\boldsymbol{\alpha}|]}\big) \wedge (|\boldsymbol{\gamma}| = |\boldsymbol{\alpha}| + |\boldsymbol{\beta}|)$$

$$\boldsymbol{A(\alpha_1\alpha_2)} \overset{def}{\longleftrightarrow} \exists\boldsymbol{\beta}\ \big(C(\boldsymbol{\alpha_1, \alpha_2, \beta}) \wedge \boldsymbol{A(\beta)}\big)$$

$$\boldsymbol{A(\alpha_1, \alpha_2, \alpha_3)} \overset{def}{\longleftrightarrow} \exists\boldsymbol{\beta}\ \exists\boldsymbol{\gamma}\ \big(C(\boldsymbol{\alpha_1, \alpha_2, \beta}) \wedge C(\boldsymbol{\beta, \alpha_3, \gamma}) \wedge \boldsymbol{A(\gamma)}\big)$$

U skup Γ dodajemo sljedeće dvije rečenice:

10. $\forall\alpha\ \forall\beta\ \exists\gamma\ (\gamma = \alpha\beta)$

    Postoji rezultat konkatenacije za bilo koja dva niza.

11. $\forall\beta\ \forall n\ \forall m\ \exists\alpha_1\ \exists\alpha_2\ \exists\alpha_3$

    $$\big((n + m \leq |\beta|) \rightarrow (|\alpha_1| = n \wedge |\alpha_2| = m \wedge \beta = \alpha_1\alpha_2\alpha_3)\big)$$

    Za proizvoljni niz β postoji svaki njegov podniz. Drugim riječima, koji god segment zadanog niza β da odaberemo, postoji podniz α_2 koji odgovara tom segmentu.

Definicije:

$T(\alpha, \beta_1, \beta_2, n, \gamma)$ *Predikat s značenjem: „Niz γ je rezultat zamjene niza β_1 nizom β_2 na mjestu n u nizu α“*[40]

$$\boldsymbol{T(\alpha, \beta_1, \beta_2, n, \gamma)} \overset{def}{\longleftrightarrow} \exists\boldsymbol{\alpha_1}\ \exists\boldsymbol{\alpha_2}\ (\boldsymbol{\alpha = \alpha_1\beta_1\alpha_2} \wedge \boldsymbol{\gamma = \alpha_1\beta_2\alpha_2} \wedge |\boldsymbol{\alpha_1}| = \boldsymbol{n})$$

$$\boldsymbol{A\left(\alpha_{\beta_2}^{\beta_1}(n)\right)} \overset{def}{\longleftrightarrow} \exists\boldsymbol{\gamma}\ \big(T(\boldsymbol{\alpha, \beta_1, \beta_2, n, \gamma}) \wedge \boldsymbol{A(\gamma)}\big)$$

$\alpha \Rightarrow \gamma$ *Predikat s značenjem: „Niz α izravno izvodi niz β u gramatici $\mathcal{G}$“*

$$\boldsymbol{\alpha \Rightarrow \gamma} \overset{def}{\longleftrightarrow} \exists\boldsymbol{\beta_1}\ \exists\boldsymbol{\beta_2}\ \exists\boldsymbol{n}\ \left(D(\boldsymbol{\beta_1, \beta_2}) \wedge \boldsymbol{\gamma = \alpha_{\beta_2}^{\beta_1}(n)}\right)$$

Za svaku produkciju u $\mathcal{G}$ dodajemo po jednu rečenicu u skup Γ na sljedeći način:

Neka je $a_0, a_1, \ldots a_n \rightarrow b_0, b_1, \ldots b_m$ produkcija u $\mathcal{G}$. U skup Γ dodajemo rečenicu:

[40] n-ti znak niza α je prvi znak niza β_1.

12. $\exists\alpha\ \exists\beta$

$(\alpha_{[0]} = c_c[a_0] \wedge \alpha_{[1]} = c_c[a_1] \wedge \ldots \wedge \alpha_{[n]} = c_c[a_n] \wedge\ |\alpha| = n' \wedge$

$\beta_{[0]} = c_c[\beta_0] \wedge \beta_{[1]} = c_c[\beta_1] \wedge \ldots \wedge \beta_{[m]} = c_c[\beta_m] \wedge\ |\beta| = m' \wedge D(\alpha, \beta))$[41]

Postoje nizovi α i β takvi da je $\alpha \rightarrow \beta$ produkcija u $\mathcal{G}$.

Za početni znak S gramatike $\mathcal{G}$ u skup Γ dodajemo rečenicu:

13. $\exists\alpha\ \left(\alpha_{[0]} = c_c[S] \wedge |\alpha| = 1 \wedge G(\alpha)\right)$

Niz S je rečenična forma gramatike $\mathcal{G}$.

Na kraju u skup Γ dodajemo rečenicu:

14. $\forall\alpha\ \forall\beta\ \left((G(\alpha) \wedge \alpha \Rightarrow \beta) \rightarrow G(\beta)\right)$

I*zravnim izvođenjem iz rečenične forme u $\mathcal{G}$ dobivamo novu rečeničnu formu u $\mathcal{G}$.*

Završili smo sa specifikacijom skupa Γ.

Za svaki niz $\alpha = a_0, a_1 \ldots a_n$ ćemo koristiti metajezični izraz $\mathbb{G}(\alpha)$[42] kojim ćemo se referirati na rečenicu:

$\exists\alpha\ \left(\alpha_{[0]} = c_c[a_0] \wedge \alpha_{[1]} = c_c[a_1] \wedge \ldots \wedge \alpha_{[n]} = c_c[a_n] \wedge\ |\alpha| = n' \wedge G(\alpha)\right)$

Rečenica $\mathbb{G}(\alpha)$ pod standardnom interpretacijom znači: „niz α je rečenična forma gramatike $\mathcal{G}$“. Ako je α niz terminala, rečenica $\alpha \in \mathcal{L}(\mathcal{G})$ je istinita ako i samo ako Γ implicira $\mathbb{G}(\alpha)$.[43] Problem odlučivanja o istinitosti tvrdnje $\alpha \in \mathcal{L}(\mathcal{G})$ je time sveden na provjeravanje da li skup

[41] Kad se izraz oblika $c[a_0]$ pojavljuje u kontekstu rečenice objektnog jezika, treba ga shvatiti kao individualnu konstantu za kodni broj $c[a_0]$ koja je uvedena definicijom kako je prikazano na str. 50. Isto vrijedi i za ostale slične slučajeve.

[42] Važno je razlikovati izraze $\mathbb{G}(\alpha)$ i $G(\alpha)$. $G(\alpha)$ je izraz u objektnom jeziku koji pod standardnom interpretacijom znači: „Niz s kodnim brojem α je rečenična forma gramatike“. $\mathbb{G}(\alpha)$ je izraz u metajeziku i označava rečenicu u objektnom jeziku koja pod standardnom interpretacijom znači: „Niz α je rečenična forma gramatike“.

[43] Prikazana je temeljna struktura dokaza. Za potpuni dokaz potrebno je još dokazati ovu ekvivalenciju. Dokaz implikacije s desna nalijevo je relativno jednostavan. Pod standardnom interpretacijom su sve rečenice u Γ istinite, pa ako vrijedi implikacija, zaključujemo da je istinita i rečenica $\mathbb{G}(\alpha)$ koja zapravo znači $\alpha \in \mathcal{L}(\mathcal{G})$. Za implikaciju s lijeva nadesno je dokaz znatno složeniji. Ako je $\alpha \in \mathcal{L}(\mathcal{G})$ istinito, postoji niz izvođenja $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_n$ u gramatici $\mathcal{G}$ gdje je α_0 niz kojega čini samo početni znak gramatike $\mathcal{G}$, a $\alpha_n = \alpha$. Treba za svaki $0 \leq i \leq n$ matematičkom indukcijom dokazati da Γ implicira $\mathbb{G}(\alpha_i)$.

rečenica Γ implicira $\mathbb{G}(\alpha)$, odnosno da li je formula $(A_1 \wedge A_2 \wedge \ldots A_n) \rightarrow \mathbb{G}(\alpha)$ (gdje je $\{A_1, A_2, \ldots A_n\} = \Gamma$) valjana. Obzirom da je prvi problem nerješiv, zaključujemo da je drugi problem također nerješiv.

Literatura

- Boolos G. S., Burgess J. P., Jeffrey R. C.; Computability and Logic (Fifth Edition); Cambridge University Press 2007
- Brcković Ž.; Izvođenje gramatika bez ograničenja - završni rad; Filozofski fakultet, Zagreb, 2009.
- Brcković Ž.; Rekurzivno prebrojivi jezici i Turingov stroj – diplomski rad; Filozofski fakultet, Zagreb, 2013.
- Church A.; Introduction to Mathematical Logic; Princeton University Press 1956.
- Dovedan Han Z.; Formalni jezici i prevodioci – regularni izrazi, gramatike, automati; Element 2012.
- Gödel, K.; On Formally Undecidable Propositions of Principia Mathematica and Related Systems; Dover Publications Inc. 1992.
- Handbook of Computability Theory (ur. Eward R. Griffor); Elsevier Science B. V. 1999.
- Hopcroft J. E., Motwani R., Ullman J.; Introduction to Automata Theory, Languages, and Computation (Second Edition); Addison-Wesley 2001.
- Kleene, S. C.; Introduction to Metamathematics; Wolters-Noordhoff Publishing and North-Holland Publishing Company 1971.
- Kleene, S. C.; Mathematical Logic; Dover Publications Inc. 2002.
- Strawson, P. F.; Introduction to Logical Theory; Methuen & Co LTD 1964.