

Uvod u teorijsko računarstvo

Milan Banković
Katedra za računarstvo i informatiku
Matematički fakultet
Univerzitet u Beogradu

29. mart 2026.

Sadržaj

1	Uvod	5
1.1	Oblasti teorijskog računarstva	5
1.1.1	Teorija izračunljivosti	5
1.1.2	Teorija složenosti izračunavanja	6
1.1.3	Algoritmi i strukture podataka	6
1.1.4	Teorija formalnih jezika i automata	6
1.1.5	Semantika programskih jezika	7
1.1.6	Teorija tipova	7
1.1.7	Kvantno računarstvo	7
1.1.8	Matematička logika u računarstvu	7
1.1.9	Ostale oblasti teorijskog računarstva	8
1.2	Teorijsko računarstvo u okviru računarstva	8
1.3	Značaj teorijskog računarstva	8
1.4	Struktura teksta	9
2	Matematička logika	11
2.1	Iskazna logika	11
2.1.1	Sintaksa iskazne logike	11
2.1.2	Semantika iskazne logike	14
2.1.3	Normalne forme u iskaznoj logici	17
2.1.4	Problem SAT	18
2.2	Logika prvog reda	20
2.2.1	Sintaksa logike prvog reda	20
2.2.2	Semantika logike prvog reda	24
2.2.3	Normalne forme u logici prvog reda	28
3	Izračunljivost	29
3.1	Intuitivni pojam algoritma	29
3.2	Formalni pojam algoritma	31
3.2.1	μ -rekurzivne funkcije	31
3.2.2	URM programi	43
3.2.3	Čerčova teza	46
3.3	Kodiranje	47
3.3.1	Kodiranje uređenih parova i torki	48
3.3.2	Kodiranje konačnih nizova brojeva	49
3.3.3	Kodiranje hijerarhijskih struktura	51
3.4	Enumeracija izračunljivih funkcija	51
3.4.1	Kodiranje URM programa	51

3.4.2	Enumeracija URM programa	53
3.4.3	Univerzalni programi i funkcije	53
3.4.4	$s - m - n$ teorema	55
4	Problemi odlučivanja	59
4.1	Odlučivost	59
4.2	Rekurzivni skupovi	64
4.3	Poluodlučivost	65
4.4	Polurekurzivni skupovi	68
4.5	Odlučivost u logici	70
5	Izračunljivost i formalna matematika	75
5.1	Deduktivni sistemi	75
5.1.1	Prirodna dedukcija	77
5.2	Logičke teorije	87
5.2.1	Peanova aritmetika	90
5.3	Gedelove teoreme	94
5.3.1	Izrazivost izračunljivih funkcija na jeziku aritmetike . . .	94
5.3.2	Kodiranje termova, formula i dokaza	97
6	Složenost izračunavanja	101
6.1	Tjuringova mašina	102
6.2	Definicija složenosti	111
6.3	Složenost problema odlučivanja	114
6.4	Klasa P	115

Glava 1

Uvod

Teorijsko računarstvo (engl. *theoretical computer science* (TCS)) je oblast računarstva koja proučava matematičke osnove izračunavanja. Kao takvo, teorijsko računarstvo je takođe i oblast matematike, tj. nalazi se u *preseku* matematike i računarstva.

Teorijsko računarstvo nam daje odgovore na sledeća pitanja:

- na koji način se izračunavanje može formalno matematički opisati (tj. kakvi sve *modeli izračunavanja* postoje)?
- koje su granice formalnog izračunavanja (tj. šta je moguće *efektivno* izračunati)?
- koja je cena takvog izračunavanja (tj. koja je *složenost*)?
- na koji način možemo formalno matematički *rezonovati* o procesima izračunavanja (tj. proučavati svojstva izračunavanja i dokazivati korektnost izračunavanja)?

1.1 Oblasti teorijskog računarstva

Savremeno teorijsko računarstvo uključuje veliki broj oblasti. Pritom, granice između različitih oblasti nisu uvek sasvim jasne, jer postoji dosta preklapanja. Takođe, mnoge oblasti računarstva koje imaju jaku matematičku podlogu se ponekad svrstavaju u teorijsko računarstvo, iako to istorijski gledano nije slučaj. Najzad, neke oblasti koje se obično svrstavaju u teorijsko računarstvo u sebi imaju sadržaje koji imaju mnoge praktične aspekte, pa se zato te oblasti ponekad svrstavaju i u druge discipline računarsva koje nisu deo teorijskog računarstva. Otuda spisak koji sledi nije konačan ni definitivan, ali jeste sastavljan iz oblasti koje su najznačajnije i za koje ne postoji nikakva dilema da spadaju u teorijsko računarstvo u najvećoj mogućoj meri.

1.1.1 Teorija izračunljivosti

Teorija izračunljivosti (engl. *computability theory*) se bavi matematičkim modelima izračunavanja i daje nam odgovor na pitanje šta je efektivno izračunljivo. Istorijski, ovo je prva grana teorijskog računarstva koja je nastala,

pre svega za potrebe formalizacije matematike i automatizacije formalnog rezonovanja. U okviru ove teorije, razvijeni su različiti formalni modeli izračunavanja (tj. opisivanja algoritama) za koje je pokazano da su međusobno ekvivalentni, ali su nam dali različite poglede na proces računanja i omogućili bolje razumevanje tog procesa. Ovo je omogućilo dalji praktičan razvoj računarstva, kroz, na primer, dizajn različitih vrsta programskih jezika. Sa druge strane, teorija izračunljivosti je poznata po brojnim *negativnim* rezultatima, koji se tiču granica formalne izračunljivosti. Zanimljivo je da su te granice uspostavljene pre nego što je napravljen prvi elektronski računar (tokom 30tih godina 20. veka).

1.1.2 Teorija složenosti izračunavanja

Teorija složenosti izračunavanja (engl. *computational complexity theory*) proučava cenu izračunavanja u smislu resursa koji su za neko izračunavanje potrebni. U te resurse pre svega ubrajamo potrebno vreme (izraženo u broju računskih koraka), ali i memorijski prostor. Teorija složenosti razmatra osnovne *klase složenosti*, tj. razvrstava probleme prema tome da li su npr. rešivi u *polinomskom* ili *eksponencijalnom* vremenu ili prostoru. Mnogi odnosi među ovim klasama su još uvek otvoreni, tako da je ova oblast poznata po mnogim još uvek nerazrešenim pitanjima (jedno od najčruvenijih je „da li je $P = NP$?”).

1.1.3 Algoritmi i strukture podataka

Oblast *algoritmi i strukture podataka* (engl. *algorithms and data structures*) se bavi dizajnom efikasnih postupaka izračunavanja. To podrazumeva metode dizajna i konstrukcije efikasnih algoritama, kao i struktura podataka nad kojima ti algoritmi rade. Takođe, ova oblast proučava i metode za praktičnu analizu algoritama, u smislu utvrđivanja njihove složenosti. Za razliku od teorije složenosti izračunavanja koja se bavi „grubom” analizom (npr. da li se neki problem može rešiti u polinomskom vremenu ili ne), analiza algoritama podrazumeva preciznu analizu složenosti konkretnog algoritma (npr. da li je algoritam složenosti $O(n^2)$ ili $O(n \log n)$?). Za takvu analizu se proučavaju razne matematičke metode poput *rekurentnih relacija* i *funkcija generatrisa*.

1.1.4 Teorija formalnih jezika i automata

Teorija formalnih jezika i automata (engl. *formal languages and automata theory*) je vrlo bliska teoriji izračunljivosti, a proučava klase formalnih jezika, poput *regularnih*, *kontekstno slobodnih* i *kontekstno zavisnih jezika*. U praksi, u ove formalne jezike spadaju razni jezici koji se koriste u računarstvu, poput programskih jezika, jezika za obeležavanje, konfiguracionih jezika, jezika za računarsko i matematičko modelovanje, ali i jezika koji se koriste u, na primer, matematičkoj logici. Takođe, ova oblast razmatra formalne modele algoritama za *prepoznavanje* različitih klasa jezika (tj. koji omogućavaju da se utvrdi da li proizvoljna niska pripada jeziku ili ne). Ovakvi modeli se nazivaju *automati* (npr. *konačni automati*, *potisni automati*, *linearno-ograničeni automati*). Automati su jednostavniji modeli izračunavanja u odnosu na opšte modele koji se izučavaju u okviru teorije izračunljivosti. U praksi, automati se koriste prilikom dizajna i konstrukcije *programskih prevodilaca*.

1.1.5 Semantika programskih jezika

Oblast *semantika programskih jezika* (engl. *programming languages semantics*) je oblast koja je u preseku teorijskog računarstva i *teorije programskih jezika* (koja, pored teorijskih, ima i svoje praktične aspekte). Semantika programskih jezika omogućava da formalno logički opisujemo *značenje* naših programa. Ovo je u praksi veoma značajno za konstrukciju programskih prevodilaca. Takođe, formalni opis semantike nam omogućava i da rezonujemo o našim programima i dokazujemo njihovu ispravnost.

1.1.6 Teorija tipova

Teorija tipova (engl. *type theory*) je još jedna oblast koja ima svoje teorijske i praktične aspekte, pa se kao takva nalazi u preseku teorijskog računarstva i drugih oblasti, poput pomenute teorije programskih jezika. U teorijskom smislu, teorija tipova uspostavlja korespondenciju između matematičke logike i računarskih programa („teorema = tip”, „dokaz = program”). U praktičnom smislu, teorija tipova je značajna zato što izučava različite modele tipizacije podataka u programskim jezicima, što ima svoju praktičnu primenu prilikom dizajna novih i unapređenja postojećih programskih jezika.

1.1.7 Kvantno računarstvo

Kvantno računarstvo (engl. *quantum computing*) je jedna od novijih oblasti teorijskog računarstva koja proučava modele izračunavanja zasnovane na kvantnoj mehanici. Kao i u klasičnom računarstvu, matematički modeli predstavljaju prethodnicu realnim mašinama koje ta izračunavanja mogu da obavljaju. Tako su različiti algoritmi kvantnog računarstva poznati već decenijama, a tek u novije vreme nastaju realni kvantni računari koji neke od tih algoritama mogu i izvršavati.

1.1.8 Matematička logika u računarstvu

Matematička logika u računarstvu (engl. *logic in computer science*) je široka oblast koja pokriva različite veze između matematičke logike i računarstva. Kao takva, ona je deo teorijskog računarstva, jer izučava matematičke osnove izračunavanja i veze matematike i računarstva. Ipak, matematička logika u računarstvu nije izolovana oblast teorijskog računarstva, već predstavlja „lepak” koji povezuje više oblasti teorijskog računarstva u jednu celinu. Na primer, teorija izračunljivosti je iznikla iz matematičke logike i predstavlja neraskidivu vezu između matematike i računarstva. Teorija složenosti se takođe prožima sa matematičkom logikom (na primer, prvi problem za koji je dokazano da je *NP-kompletna* je problem zadovoljivosti formule iskazne logike, *problem SAT*). Semantika programskih jezika je zasnovana na metodama matematičke logike za formalizovanje semantike programskih jezika i rezonovanje o njima. Teorija tipova direktno uspostavlja vezu između matematičke logike i računarskih programa.

Sa druge strane, računarske metode se široko koriste u automatizaciji procesa logičkog rezonovanja. Oblasti poput *automatskog rezonovanja* i *interaktivnog dokazivanja teorema* koriste algoritme za automatsko generisanje i proveru

ispravnosti formalnih matematičkih dokaza, kao i za automatsko rešavanje problema zadatih na jeziku matematičke logike. Može se reći da na ovaj način računarstvo vraća svoj dug matematičkoj logici, za njen nemerljiv doprinos razvoju računarstva.

1.1.9 Ostale oblasti teorijskog računarstva

Često se u literaturi kao deo teorijskog računarstva spominju i mnoge druge oblasti, poput *kriptografije*, *paralelno i distribuirano izračunavanje*, *teorija informacija*, *simboličko izračunavanje*, *računarska geometrija* i sl. Mnoge od ovih oblasti zaista imaju značajan presek sa teorijskim računarstvom. Ipak, ne može se reći da apsolutno pripadaju teorijskom računarstvu, s obzirom da u značajnoj meri pripadaju i drugim oblastima matematike i računarstva.

1.2 Teorijsko računarstvo u okviru računarstva

Računarstvo (engl. *computer science* (CS)) danas predstavlja samostalnu naučnu disciplinu. Ona je istorijski iznikla iz teorijskog računarstva, pa samim tim i iz matematike. Međutim, danas postoji živa akademska rasprava da li je savremeno računarstvo više matematika ili je više inženjerstvo. Ono što je sigurno je da računarstvo danas predstavlja hibridnu disciplinu koja uključuje svoje matematičke, inženjerske, pa čak i društvene aspekte.

U klasičnoj definiciji, računarstvo je nauka koja se bavi teorijskim i praktičnim aspektima izračunavanja, obrade podataka i procesiranja informacija. Teorijsko računarstvo dominantno pokriva ove „teorijske” aspekte. Pored teorijskog računarstva, računarstvo uključuje mnoge praktične discipline, poput arhitekture i organizacije računara, operativnih sistema i računarskih mreža, dizajna programskih jezika i programskih prevodilaca, softverskog inženjerstva, baza podataka, informacionih sistema, veštačke inteligencije i sl. Iako ove oblasti ne spadaju u teorijsko računarstvo, većina njih imaju dodirnih tačaka sa teorijskim računarstvom ili vuku svoje korene iz nekih od oblasti teorijskog računarstva.

1.3 Značaj teorijskog računarstva

Teorijsko računarstvo je od fundamentalnog značaja za razvoj računarstva i formalnog matematičkog rezonovanja. Ovaj značaj se ogleda u sledećem:

- iz teorijskog računarstva je izniklo čitavo moderno računarstvo. Otu- da glavni rezultati teorijskog računarstva prožimaju i sve druge oblasti računarstva i utiču na njihov razvoj. Mnoge ideje nastale u teorijskom računarstvu se kasnije preuzimaju u drugim oblastima računarstva, gde dobijaju svoju praktičnu realizaciju (npr. *računari sa uskladištenim programom* su praktična realizacije *univerzalnih programa*).
- teorijsko računarstvo uspostavlja granice formalne ali i praktične izračunljivosti. Dakle, ono nam govori šta je moguće rešiti algoritamski, šta je teorijski moguće, ali je u praksi veoma teško sa postojećim resursima, a šta nikada neće moći da bude rešeno na taj algoritamski način.

- teorijsko računarstvo proučava modele koji omogućavaju razvoj metoda za formalnu verifikaciju softvera i hardvera. Metode koje se koriste u statičkoj analizi softvera poput *provere modela*, *apstraktne interpretacije* i *simboličkog izvršavanja* se direktno oslanjaju na matematičku logiku, teoriju automata i semantiku programskih jezika.
- teorijsko računarstvo omogućava i usmerava razvoj drugih postojećih, kao i novih oblasti računarstva, kroz razvoj programskih jezika, novih softverskih tehnologija, novih metoda i tehnika koje se primenjuju raznim oblastima računarstva, i sl.

1.4 Struktura teksta

U nastavku ovog teksta biće izložene osnove nekih od oblasti teorijskog računarstva. Akcenat će biti na teoriji izračunljivosti, teoriji složenosti, kao i na vezama matematičke logike i teorijskog računarstva. Sa druge strane, neke od oblasti će namerno biti izostavljene, s obzirom da se u većem obimu proučavaju u okviru drugih obaveznih predmeta na Matematičkom fakultetu. U te oblasti spadaju teorija jezika i automata, kao i algoritmi i strukture podataka. Najzad, neke oblasti će prosto biti izostavljene zbog ograničenog obima teksta.

Glava 2

Matematička logika

Matematička logika predstavlja glavnu sponu između matematike i teorijskog računarstva. Zbog toga u ovoj glavi dajemo pregled osnovnih logičkih pojmova koji će nam biti potrebni u nastavku teksta. Iako će mnogi od ovih pojmova čitaocu biti poznati, ovde ih iznosimo zarad potpunosti izlaganja. Fokus će biti na iskaznoj logici i logici prvog reda.

2.1 Iskazna logika

Iskazna logika predstavlja najjednostavniji logički okvir u kome se može vršiti formalno rasuđivanje. Iskazne formule se grade nad iskazima, koji predstavljaju elementarne činjenice koje mogu biti tačne ili netačne. Iskazna logika je značajna jer predstavlja osnovu za druge, bogatije logike. Takođe, mnogi praktični problemi se mogu izraziti i rešavati u okviru iskazne logike, te je ona stoga korisna i sama po sebi.

2.1.1 Sintaksa iskazne logike

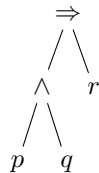
Definicija 2.1. Neka je dat prebrojiv skup *iskaznih slova* (atoma) P . *Iskazna formula* se dobija konačnom primenom sledećih pravila:

- $\top$ i $\perp$ su iskazne formule (*logičke konstante*)
- $p \in P$ je iskazna formula (*atom*)
- ako je F iskazna formula, tada je i $\neg F$ iskazna formula
- ako su F i G iskazne formule, onda su i $F \wedge G$, $F \vee G$, $F \Rightarrow G$ i $F \Leftrightarrow G$ iskazne formule

Skup svih iskazni formula nad P označavaćemo sa $\mathcal{F}_P$.

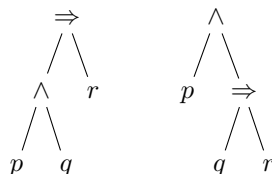
Napomena 2.1. Treba napraviti razliku između *konkretne sintakse* i *apstraktne sintakse*. U apstraktnoj sintaksi, formula se posmatra kao hijerarhijska struktura, izgrađena primenom logičkih veznika na jednostavnije formule. Formulu predstavljamo *stablom apstraktne sintakse* (engl. *abstract syntax tree* (AST)) u čijim se unutrašnjim čvorovima nalaze iskazni veznici ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$), a u listovima se nalaze logičke konstante ili atomi. Iz strukture ovog stabla, jasno je koji

se iskazni veznik primenjuje na koje potformule, tj. u potpunosti je definisana struktura formule. Na primer, ako imamo sledeće stablo:

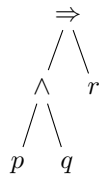


jasno je da je u pitanju *implikacija* ($\Rightarrow$) primenjena na dve potformule, od kojih je prva *konjunkcija* ($\wedge$) dve atomičke formule p i q , a druga je atomička formula r . Apstraktna sintaksna stabla su zgodna kao strukture podataka za predstavljanje formula u računarskim programima.

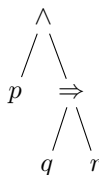
Sa druge strane, apstraktna sintaksa nije pogodna za zapisivanje formula u okviru teksta. U tu svrhu koristimo konkretnu sintaksu. U konkretnoj sintaksi, formula se predstavlja niskom simbola, poput $p \wedge q \Rightarrow r$. Problem sa konkretnom sintaksom je što se ona može tumačiti *višeznačno*: ista konkretna sintaksa može odgovarati različitim apstraktnim sintaksama, tj. može predstavljati različite formule. Na primer, formula $p \wedge q \Rightarrow r$ u konkretnoj sintaksi može odgovarati dvema različitim formulama u apstraktnoj sintaksi:



Kako bi konkretna sintaksa jednoznačno određivala apstraktnu sintaksu formule, potrebno je definisati *prioritete* i *asocijativnosti* operatora. Prema uobičajenoj konvenciji, redosled prioriteta, počev od najvišeg, je sledeći: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, pri čemu su svi binarni veznici desno asocijativni. Prioritet i asocijativnost se mogu promeniti *zagradama*. Imajući to u vidu, formula $p \wedge q \Rightarrow r$ je sintaksno identična formuli $(p \wedge q) \Rightarrow r$, tj. odgovara apstraktnoj sintaksi:



Sa druge strane, konkretna sintaksa $p \wedge (q \Rightarrow r)$ bi odgovarala apstraktnoj sintaksi:



Primer 2.1. U nastavku navodimo primere iskaznih formula u konkretnoj sintaksi, uz komentare vezane za prioritete i asocijativnosti iskaznih veznika:

- $p \vee q \wedge r$ je u sintaksnom smislu isto što i $p \vee (q \wedge r)$, s obzirom da $\wedge$ ima viši prioritet od $\vee$
- $(p \vee q) \wedge r$: ovde smo zagradama promenili prioritet, tj. sada se najpre primenjuje $\vee$, pa zatim $\wedge$
- $p \wedge q \wedge r$: ova formula predstavlja isto što i $p \wedge (q \wedge r)$, s obzirom da $\wedge$ ima desnu asocijativnost
- $(p \wedge q) \wedge r$: ovim smo promenili asocijativnost operatora, tako da se najpre primenjuje levi $\wedge$ veznik
- $p \Leftrightarrow q \Rightarrow r$ je isto što i $p \Leftrightarrow (q \Rightarrow r)$, zbog prioriteta operatora
- $p \Rightarrow q \Rightarrow r$ je isto što i $p \Rightarrow (q \Rightarrow r)$, zbog desne asocijativnosti operatora $\Rightarrow$.

Napomena 2.2. Konkretna sintaksa iskaznih formula predstavlja *jezik* nad azbukom $P \cup \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \top, \perp, (,)\}$, tj. skup niski nad ovom azbukom koje su formirane u skladu sa sintaksnim pravilima datim definicijom 2.1, uz dodatak da ako niska F predstavlja formulu, tada i (F) takođe predstavlja formulu.

Definicija 2.2. Složenost formule $c(F)$ se definiše na sledeći način:

- $c(\top) = c(\perp) = c(p) = 0$ (gde je $p \in P$)
- $c(\neg F) = c(F) + 1$
- $c(F \wedge G) = c(F \vee G) = c(F \Rightarrow G) = c(F \Leftrightarrow G) = c(F) + c(G) + 1$

Napomena 2.3. Iz gornje definicije sledi da je složenost formule jednaka broju iskaznih veznika koje formula sadrži.

Primer 2.2. Složenost formule $(p \vee q) \wedge \neg r$ je 3, dok je složenost formule $\neg(\top \Rightarrow q)$ jednaka 2.

Definicija 2.3. Zamena formule G formulom H u formuli F (u oznaci $F[G \rightarrow H]$) se definiše na sledeći način:

- $F[G \rightarrow H] = H$, ako je $F = G$
- $\top[G \rightarrow H] = \top$, $\perp[G \rightarrow H] = \perp$
- $p[G \rightarrow H] = p$, ako je $p \in P$ i $G \neq p$
- $(\neg F)[G \rightarrow H] = \neg F[G \rightarrow H]$
- $(F_1 \wedge F_2)[G \rightarrow H] = F_1[G \rightarrow H] \wedge F_2[G \rightarrow H]$
- $(F_1 \vee F_2)[G \rightarrow H] = F_1[G \rightarrow H] \vee F_2[G \rightarrow H]$
- $(F_1 \Rightarrow F_2)[G \rightarrow H] = F_1[G \rightarrow H] \Rightarrow F_2[G \rightarrow H]$
- $(F_1 \Leftrightarrow F_2)[G \rightarrow H] = F_1[G \rightarrow H] \Leftrightarrow F_2[G \rightarrow H]$

Napomena 2.4. Zamena $F[G \rightarrow H]$ se, prema prethodnoj definiciji, vrši tako što se sva pojavljivanja potformule G u formuli F zamene formulom H . Zamena je čisto sintakсна operacija – njome se vrši transformacija jedne formule u drugu.

Primer 2.3. Zamenom formule $p \wedge q$ formulom $q \wedge p$ u formuli $p \wedge q \Rightarrow r$ dobijamo formulu $q \wedge p \Rightarrow r$, dok zamenom formule $p \wedge q$ formulom q u formuli $(p \wedge q) \vee (r \Rightarrow (p \wedge q))$ dobijamo $q \vee (r \Rightarrow q)$.

2.1.2 Semantika iskazne logike

Definicija 2.4. *Iskazna valuacija* je funkcija $v : P \rightarrow \{0, 1\}$, tj. funkcija koja svakom iskaznom slovu pridružuje vrednost 0 (netačno) ili 1 (tačno).

Napomena 2.5. S obzirom da je P prebrojiv skup, valuacije možemo razumeti i kao beskonačne nizove nula i jedinica. Otuda je broj mogućih valuacija nad P neprebrojivo beskonačan.

Definicija 2.5. *Interpretacija* I_v određena valuacijom v je funkcija $I_v : \mathcal{F}_P \rightarrow \{0, 1\}$ koja svakoj formuli pridružuje vrednost 0 (netačno) ili 1 (tačno) na sledeći način:

- $I_v(\top) = 1$
- $I_v(\perp) = 0$
- $I_v(p) = v(p)$ (za $p \in P$)
- $I_v(\neg F) = 1$ akko $I_v(F) = 0$
- $I_v(F \wedge G) = 1$ akko je $I_v(F) = 1$ i $I_v(G) = 1$
- $I_v(F \vee G) = 1$ akko je $I_v(F) = 1$ ili $I_v(G) = 1$
- $I_v(F \Rightarrow G) = 1$ akko je $I_v(F) = 0$ ili $I_v(G) = 1$
- $I_v(F \Leftrightarrow G) = 1$ akko je $I_v(F) = I_v(G)$

Primer 2.4. Neka je data valuacija v takva da je $v(p) = 1$, $v(q) = 0$ i $v(r) = 1$. Tada je formula $p \wedge q \Rightarrow r$ logički tačna u valuaciji v (tj. $I_v(p \wedge q \Rightarrow r) = 1$), dok je formula $p \Rightarrow q \wedge r$ logički netačna u v (tj. $I_v(p \Rightarrow q \wedge r) = 0$).

Definicija 2.6. Ako je $I_v(F) = 1$, tada kažemo da je valuacija v *model* formule F . Formula je *zadovoljiva* ako ima bar jedan model. U suprotnom, formula je *nezadovoljiva*.

Napomena 2.6. Zadovoljivost iskazne formule se teorijski može ispitati veoma jednostavno. Naime, iako je skup iskaznih slova P beskonačan (pa samim tim je takav i skup valuacija nad P), u svakoj formuli se pojavljuje samo konačno mnogo iskaznih slova iz P . Ako se u formuli F pojavljuje n različitih iskaznih slova, tada mi imamo 2^n suštinski različitih valuacija koje treba razmotriti (sve valuacije koje se poklapaju na tih n promenljivih su suštinski iste iz ugla formule F). Kako je broj valuacija koje treba ispitati konačan, možemo odrediti vrednost formule u svakoj od njih i videti da li bar jedna od njih zadovoljava formulu. Ovak postupak je poznat i kao *metod istinitosnih tablica*.

Primer 2.5. Neka je data formula $p \wedge q \Rightarrow r$. Njoj odgovara sledeća istinitosna tablica:

p	q	r	$p \wedge q$	$p \wedge q \Rightarrow r$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Iz ove tablice je jasno da je formula zadovoljiva.

Napomena 2.7. Prethodni postupak, iako jednostavan, nije naročito efikasan, s obzirom da broj valuacija koje treba ispitati raste eksponencijalno sa brojem promenljivih n . Zbog toga se često razmatraju „pametniji” postupci utvrđivanja zadovoljivosti iskaznih formula koji u praksi rade efikasnije.

Definicija 2.7. Formula F je *tautologija* (*valjana*, *validna*) ako je tačna u svakoj valuaciji. Formula je F *poreciva* ako nije tautologija.

Teorema 2.1. Formula F je tautologija, akko je $\neg F$ nezadovoljiva. Formula F je poreciva akko je $\neg F$ zadovoljiva.

Dokaz. Očigledno sledi iz definicije ovih pojmova. □

Primer 2.6. Formula iz primera 2.5 nije tautologija, s obzirom da postoji valuacija u kojoj nije tačna (tj. formula je poreciva).

Primer 2.7. Formula $(p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$ je tautologija. Zaista, njena tablica istinitosti izgleda ovako:

p	q	$\neg p$	$\neg q$	$p \Rightarrow q$	$\neg q \Rightarrow \neg p$	$(p \Rightarrow q) \Rightarrow (\neg q \Rightarrow \neg p)$
0	0	1	1	1	1	1
0	1	1	0	1	1	1
1	0	0	1	0	0	1
1	1	0	0	1	1	1

Dakle, vidimo da je tačna u svim valuacijama različitim po p i q .

Definicija 2.8. Skup formula Δ je *zadovoljiv* ako postoji valuacija v takva da istovremeno zadovoljava sve formule skupa Δ .

Primer 2.8. Skup $\Delta = \{p \wedge \neg q, q \Rightarrow p\}$ je zadovoljiv, jer valuacija u kojoj je $v(p) = 1$ i $v(q) = 0$ zadovoljava obe formule skupa Δ . Sa druge strane, skup $\Gamma = \{p \wedge q, q \Rightarrow \neg p\}$ je nezadovoljiv, jer da bi prva formula bila zadovoljena, mora da važi $v(p) = v(q) = 1$, što drugu formulu čini netačnom. Dakle, ne postoji valuacija u kojoj su obe formule tačne.

Definicija 2.9. Formula F je logička posledica skupa formula Δ (u oznaci $\Delta \models F$) ako svaka valuacija v koja zadovoljava skup formula Δ ujedno zadovoljava i formulu F . Specijalno, F je logička posledica formule G (u oznaci $G \models F$) ako svaka valuacija koja zadovoljava G zadovoljava i F .

Primer 2.9. Formula $p \vee q$ je logička posledica formule $p \wedge q$. Zaista, valuacija v zadovoljava $p \wedge q$ akko je $v(p) = v(q) = 1$. U svim takvim valuacijama tačna je i formula $p \vee q$. Otuda važi $p \wedge q \models p \vee q$.

Stav 2.1. Važi $\Delta \models A$ akko je skup $\Delta \cup \{\neg A\}$ nezadovoljiv.

Dokaz. Neka je v proizvoljna valuacija koja zadovoljava skup Δ . Kako je $\Delta \models A$, to znači da je i A tačna u valuaciji v , pa je $\neg A$ netačna u v . Otuda skup $\Delta \cup \{\neg A\}$ nije zadovoljen ni jednom valuacijom. Obratno, ako je skup $\Delta \cup \{\neg A\}$ nezadovoljiv i ako je v proizvoljna valuacija koja zadovoljava skup Δ , tada ta valuacija ne može zadovoljavati $\neg A$. Otuda ona zadovoljava A , pa je $\Delta \models A$. $\square$

Definicija 2.10. Formule F i G su *logički ekvivalentne* (u oznaci $F \equiv G$) ako je $F \models G$ i $G \models F$.

Napomena 2.8. Prema prethodnoj definiciji, formule F i G su logički ekvivalentne ako je za svaku valuaciju v ispunjeno $I_v(F) = I_v(G)$, tj. ako se formule isto interpretiraju u svakoj valuaciji.

Teorema 2.2. Relacija $\equiv$ je relacija ekvivalencije (refleksivna, simetrična i tranzitivna) na skupu svih formula $\mathcal{F}_P$.

Dokaz. Očigledno. $\square$

Napomena 2.9. Formule koje pripadaju istoj klasi ekvivalencije po relaciji $\equiv$ su *semantički jednake*, tj. imaju isto značenje, iako se sintaksno mogu razlikovati.

Primer 2.10. U nastavku navodimo neke dobro poznate logičke ekvivalencije koje se nazivaju i *logički zakoni*:

- $\neg\neg F \equiv F$ (zakon dvojne negacije)
- $F \wedge G \equiv G \wedge F$, $F \vee G \equiv G \vee F$ (zakoni komutacije)
- $(F \wedge G) \wedge H \equiv F \wedge (G \wedge H)$, $(F \vee G) \vee H \equiv F \vee (G \vee H)$ (zakoni asocijacije)
- $F \wedge F \equiv F$, $F \vee F \equiv F$ (zakoni idempotencije)
- $F \wedge \neg F \equiv \perp$, $F \vee \neg F \equiv \top$ (zakon kontradikcije i zakon tautologije)
- $F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$, $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$ (zakoni distribucije)
- $F \wedge (F \vee G) \equiv F$, $F \vee (F \wedge G) \equiv F$ (zakoni apsorpcije)
- $F \Rightarrow G \equiv \neg F \vee G$ (zakon eliminacije implikacije)
- $F \Leftrightarrow G \equiv (F \Rightarrow G) \wedge (G \Rightarrow F) \equiv (\neg F \vee G) \wedge (\neg G \vee F)$ (zakon eliminacije ekvivalencije)
- $(F \Rightarrow G) \equiv (\neg G \Rightarrow \neg F)$ (zakon kontrapozicije)
- $\neg(F \wedge G) \equiv \neg F \vee \neg G$, $\neg(F \vee G) \equiv \neg F \wedge \neg G$ (De Morganovi zakoni)
- $F \wedge \perp \equiv \perp$, $F \wedge \top \equiv F$ (i slični zakoni za ostale veznike)

Ispravnost ovih logičkih ekvivalencija ostavljamo čitaocu za vežbu.

Sledeću važnu teoremu navodimo bez dokaza. Ona nam omogućava da izvodimo zaključke o zadovoljivosti beskonačnih skupova na osnovu zadovoljivosti njihovih konačnih potskupova.

Teorema 2.3 (Kompaktnost iskazne logike). *Skup Δ iskaznih formula je zadovoljiv ako je zadovoljiv svaki njegov konačan potskup.*

Ekvivalentno, ako je Δ nezadovoljiv, tada postoji neki njegov konačan potskup koji je nezadovoljiv.

Posledica 2.1. $\Delta \models F$, ako postoji neki konačan potskup $\Delta' \subseteq \Delta$ takav da $\Delta' \models F$.

Dokaz. Važi $\Delta \models F$ ako je $\Delta \cup \{\neg F\}$ nezadovoljiv, a to važi ako postoji konačan potskup od $\Delta \cup \{\neg F\}$ koji je nezadovoljiv. Bez ograničenja opštosti, možemo pretpostaviti da taj potskup sadrži $\neg F$ (uvek ga možemo dodati i potskup će ostati nezadovoljiv), tj. da je taj potskup oblika $\Delta' \cup \{\neg F\}$, gde je $\Delta' \subseteq \Delta$. Ovaj skup je nezadovoljiv ako je $\Delta' \models F$. $\square$

Sledeća teorema daje semantičko opravdanje sintaksne operacije zamene.

Teorema 2.4 (Teorema o zameni). *Ako je $G \equiv H$, tada je $F[G \rightarrow H] \equiv F$.*

Dokaz. Indukcijom po strukturi formule. $\square$

Napomena 2.10. Prema prethodnoj teoremi, ako u formuli F zamenjujemo neku potformulu njoj ekvivalentnom potformulom, rezultat će biti formula koja je ekvivalentna sa polaznom formulom F . Ovakve zamene se nazivaju *ekvivalentne transformacije*.

Primer 2.11. Kako važi $p \wedge q \equiv q \wedge p$ (zakon komutacije za konjunkciju), na osnovu teoreme o zameni važi $p \wedge q \Rightarrow r \equiv q \wedge p \Rightarrow r$.

2.1.3 Normalne forme u iskaznoj logici

Definicija 2.11. *Literal* je ili iskazno slovo ili negacija iskaznog slova iz P . *Elementarna disjunkcija* (ili *klauza*) je ili literal ili disjunkcija literala. *Elementarna konjunkcija* je ili literal ili konjunkcija literala. Za formulu kažemo da je u *konjunktivnoj normalnoj formi* (KNF) ako je konjunkcija klauza. Za formulu kažemo da je u *disjunktivnoj normalnoj formi* (DNF) ako je disjunkcija elementarnih konjunkcija.

Teorema 2.5. *Za svaku formulu F postoji njoj ekvivalentna formula u KNF-u (DNF-u) F' koja joj je logički ekvivalentna.*

Dokaz. Teorema sledi iz efektivnog postupka svodenja na KNF (DNF) putem ekvivalentnih transformacija. Pritom, koriste se zakoni eliminacije implikacije i ekvivalencije, De Morganovi zakoni, zakon dvojne negacije, kao i zakoni distribucije. $\square$

Napomena 2.11. Formulu F' iz prethodne teoreme nazivamo KNF-om (DNF-om) formule F .

Primer 2.12. Formula $(p \wedge q) \Rightarrow (r \wedge s)$ se može transformisati ekvivalentnim transformacijama na sledeći način:

$$\begin{aligned} (p \wedge q) \Rightarrow (r \wedge s) &\equiv \\ \neg(p \wedge q) \vee (r \wedge s) &\equiv \\ (\neg p \vee \neg q) \vee (r \wedge s) &\equiv \end{aligned}$$

Ova formula je u DNF-u, jer je disjunkcija elementarnih konjunkcija $\neg p$, $\neg q$ i $r \wedge s$. Ako želimo KNF, dalje možemo primeniti distributivni zakon $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$:

$$\begin{aligned} (\neg p \vee \neg q) \vee (r \wedge s) &\equiv \\ (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee s) &\equiv \end{aligned}$$

Sada imamo KNF koji se sastoji iz dve klauze: $\neg p \vee \neg q \vee r$ i $\neg p \vee \neg q \vee s$.

Napomena 2.12. Primetimo da KNF (DNF) neke formule u opštem slučaju nije jedinstven. U praksi smo često zainteresovani za KNF (DNF) minimalne složenosti. Postupci pronalaženja takvog KNF-a (DNF-a) date formule su poznati kao *postupci minimizacije*. Među poznatijim takvim postupcima su *metod Karnoovih mapa* i *metod Kvín-Meklaskog*.

2.1.4 Problem SAT

Definicija 2.12 (Problem SAT). Problem ispitivanja zadovoljivosti iskazne formule u KNF-u naziva se *problem SAT*.

Napomena 2.13. Naziv SAT potiče od engleske reči *Satisfiability*.

Napomena 2.14. Ponekad se u literaturi problemom SAT naziva problem ispitivanja zadovoljivosti proizvoljne iskazne formule. U tom slučaju se problem ispitivanja zadovoljivosti KNF formule eksplicitno označava kao *CNF-SAT* (engl. *conjunctive normal form satisfiability*).

Napomena 2.15. Da bi neka valuacija zadovoljila iskaznu formulu u KNF-u, ona mora istovremeno zadovoljiti sve njene klauze, tj. potrebno je u toj valuaciji bude tačan bar po jedan literal iz svake klauze. Ovo problem SAT čini kombinatornim problemom, koji je, iako veoma jednostavno definisan, često vrlo težak za rešavanje. O ovome ćemo više govoriti u poglavlju 6.

Primer 2.13. Razmotrimo zadovoljivost KNF formule $(p_1 \vee \neg p_3) \wedge (p_2 \vee p_3 \vee \neg p_1) \wedge (\neg p_1 \vee \neg p_3 \vee p_2) \wedge (\neg p_1 \vee \neg p_2) \wedge (p_1 \vee p_3)$. Pretpostavimo da postoji valuacija v koja zadovoljava ovu formulu. Ispitaćemo slučajeve po vrednosti $v(p_3)$. Pretpostavimo da je $v(p_3) = 1$. Tada je $I_v(\neg p_3) = 0$, pa kako bi klauza $p_1 \vee \neg p_3$ bila zadovoljena, mora da važi $v(p_1) = 1$. Dalje, kako je $I_v(\neg p_1) = 0$, da bi klauza $\neg p_1 \vee \neg p_2$ bila zadovoljena, mora da važi $I_v(\neg p_2) = 1$, tj. $v(p_2) = 0$. Međutim, sada je $I_v(\neg p_1 \vee \neg p_3 \vee p_2) = 0$. Dakle, ne može da važi $v(p_3) = 1$. Pretpostavimo sada da važi $v(p_3) = 0$. Sada, zbog klauze $p_1 \vee p_3$ mora da važi $v(p_1) = 1$. Otuda je ponovo $v(p_2) = 0$ (zbog klauze $\neg p_1 \vee \neg p_2$), pa će važiti $I_v(p_2 \vee p_3 \vee \neg p_1) = 0$. Dakle, ni za $v(p_3) = 1$ ni za $v(p_3) = 0$ formula ne može biti zadovoljena takvom valuacijom. Otuda formula nije zadovoljiva.

Napomena 2.16. Jedan od najpoznatijih postupaka za ispitivanje zadovoljivosti KNF formule (tj. za rešavanje problema SAT) je *DPLL procedura* (engl. *Davis-Putnam-Logemann-Loveland*). Zasnovan je na pretrazi sa elementima rezonovanja, od kojih je najznačajniji *jedinična propagacija* (ako su u klauzi svi

literali osim jednog netačni, tada taj preostali literal mora biti tačan). Postupak demonstriran u prethodnom primeru predstavlja jednostavan primer primene DPLL procedure.

Problem SAT je izuzetno primenljiv u praksi, jer se mnogi problemi mogu svesti na instance ovog problema. O tome govore sledeći primeri.

Primer 2.14 (Bojenje grafa). Pretpostavimo da je dat neusmeren graf $G = (V, E)$, sa skupom čvorova $V = \{1, \dots, n\}$ i skupom grana $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. Potrebno je svaki čvor grafa obojiti jednom od tri boje *crvena*, *zelena* ili *plava*, tako da nikoja dva susedna čvora u i v (tj. čvora za koje postoji grana $\{u, v\} \in E$) nisu obojena istom bojom.

Ovaj problem se može jednostavno svesti na problem SAT. Neka iskazno slovo c_i znači „čvor i je obojen crvenom bojom”. Slično neka p_i i z_i imaju isto značenje za plavu i zelenu boju, respektivno. Svaki čvor mora biti obojen tačno jednom bojom, što možemo opisati sledećim klauzama:

$$(c_i \vee p_i \vee z_i) \wedge (\neg c_i \vee \neg p_i) \wedge (\neg c_i \vee \neg z_i) \wedge (\neg p_i \vee \neg z_i)$$

Ovakve klauze formiramo za svako $i \in \{1, \dots, n\}$. Dalje, za svaka dva susedna čvora i i j imaćemo sledeće klauze:

$$(\neg c_i \vee \neg c_j) \wedge (\neg p_i \vee \neg p_j) \wedge (\neg z_i \vee \neg z_j)$$

Ove klauze zabranjuju da čvorovi i i j budu obojeni istom bojom. Konjunkcija svih navedenih klauza daje KNF formulu koja će biti zadovoljiva akko je moguće obojiti graf na opisani način. Pritom, svaka valuacija koja zadovoljava ovu formulu će odgovarati jednom ispravnom bojenju grafa.

Primer 2.15. Tri đaka, Miloš, Zoran i Petar uče strane jezike: engleski i nemački. Ako se zna da bar dvojica govore engleski, i bar dvojica govore nemački, tada postoji učenik koji govori oba jezika. Dokazati.

Opisani problem možemo predstaviti iskaznom formulom. Neka su m_e i m_n redom iskazna slova sa značenjem „Miloš govori engleski” i „Miloš govori nemački”. Sa sličnim značenjem, uvodimo i iskazna slova z_e i z_n (za Zorana) i p_e i p_n (za Petra). Prethodno tvrđenje se može predstaviti kao sledeća iskazna formula:

$$\begin{aligned} & ((m_e \wedge z_e) \vee (m_e \wedge p_e) \vee (z_e \wedge p_e)) \quad \wedge \\ & ((m_n \wedge z_n) \vee (m_n \wedge p_n) \vee (z_n \wedge p_n)) \quad \Rightarrow \\ & ((m_e \wedge m_n) \vee (z_e \wedge z_n) \vee (p_e \wedge p_n)) \end{aligned}$$

Potrebno je dokazati da je ova formula tautologija. Da bismo to uradili, možemo dokazati da je negacija ove formule nezadovoljiva. Ova formula je oblika $A \wedge B \Rightarrow C$, pa je njena negacija oblika $A \wedge B \wedge \neg C$, tj. imamo formulu:

$$\begin{aligned} & ((m_e \wedge z_e) \vee (m_e \wedge p_e) \vee (z_e \wedge p_e)) \quad \wedge \\ & ((m_n \wedge z_n) \vee (m_n \wedge p_n) \vee (z_n \wedge p_n)) \quad \wedge \\ & \neg ((m_e \wedge m_n) \vee (z_e \wedge z_n) \vee (p_e \wedge p_n)) \end{aligned}$$

Svođenjem na logički ekvivalentnu KNF formu, dobijamo:

$$\begin{aligned} & (m_e \vee z_e) \wedge (m_e \vee p_e) \wedge (z_e \vee p_e) \quad \wedge \\ & (m_n \vee z_n) \wedge (m_n \vee p_n) \wedge (z_n \vee p_n) \quad \wedge \\ & (\neg m_e \vee \neg m_n) \wedge (\neg z_e \vee \neg z_n) \wedge (\neg p_e \vee \neg p_n) \end{aligned}$$

Prilikom svođenja na KNF, primenjivali smo i zakon idempotencije $A \vee A \equiv A$ i apsorpcije $(A \vee B) \wedge A \equiv A$ radi uprošćavanja. Za dobijenu KNF formulu se može utvrditi da je nezadovoljiva (proveriti za vežbu!), odakle je polazna formula tautologija.

Napomena 2.17. Prethodni primer je demonstrirao postupak *dokazivanja tautologičnosti pobijanjem* (engl. *refutation*). Ovaj postupak se zasniva na teoremi 2.1, tj. činjenici da je formula F tautologija akko je njena negacija $\neg F$ nezadovoljiva. Time se problem tautologičnosti svodi na problem zadovoljivosti, a zatim svođenjem na KNF na problem SAT.

Softverski alati za rešavanje problema SAT su poznati pod nazivom *SAT rešavači* (engl. *SAT solvers*). Moderni SAT rešavači su zasnovani na algoritmi-ma koji su mnogo efikasniji od metode istinitosnih tablica (poput CDCL algoritma). SAT rešavači zasnovani na CDCL algoritmu su u stanju da za relativno kratko vreme ispituju zadovoljivost KNF formula koje sadrže na hiljade iskaznih slova, kao i desetine hiljada klauza. Zahvaljujući tome, mnogi teški kombinatorni problemi su u prethodnih par decenija rešeni upravo svođenjem na SAT.

2.2 Logika prvog reda

Logika prvog reda proširuje iskaznu logiku time što omogućava precizniji opis atomičnih iskaza, uz pomoć funkcijskih i predikatskih simbola koji u datom kontekstu mogu imati specifično značenje. Na primer, umesto da elementarne iskaze predstavljamo iskaznim atomima $p, q, \dots$ (za koje samo možemo da zadržimo da li su tačni ili netačni), možemo imati atome poput $a < b$ ili $(a + b) \mid c$ ili $parno(a)$, čije značenje zavisi od značenja pridruženog simbolima koji se u njima pojavljuju. Takođe, logika prvog reda uvodi pojam *promenljivih* koje se mogu *kvantifikovati*, čime se omogućava izražavanje svojstava poput „za svako x važi...” ili „postoji x za koje važi...”.

2.2.1 Sintaksa logike prvog reda

Definicija 2.13. *Signatura* (ili *jezik*) prvog reda je uređena trojka $\mathcal{L} = (\Sigma, \Pi, ar)$ pri čemu je Σ neprazan skup *funkcijskih simbola*, Π je neprazan skup *predikatskih (relacijskih) simbola* ($\Pi \cap \Sigma = \emptyset$), a $ar : \Sigma \cup \Pi \rightarrow \mathbb{N}$ je funkcija koja svakom simbolu iz $\Sigma \cup \Pi$ pridružuje *arnost*.

Definicija 2.14. Neka je data signatura $\mathcal{L} = (\Sigma, \Pi, ar)$ i neprazan skup *promenljivih* V . *Term* nad $\mathcal{L}$ se dobija konačnom primenom sledećih pravila:

- promenljiva $x \in V$ je term
- ako je $a \in \Sigma$ i $ar(a) = 0$, tada je a term
- ako je $f \in \Sigma$ i $ar(f) = n > 0$, a $t_1, \dots, t_n$ su termovi, tada je i $f(t_1, \dots, t_n)$ term.

Atom nad $\mathcal{L}$ je ili $p \in \Pi$ takvo da je $ar(p) = 0$, ili je oblika $p(t_1, \dots, t_n)$, pri čemu je $p \in \Pi$ i $ar(p) = n > 0$, a $t_1, \dots, t_n$ su termovi nad $\mathcal{L}$. *Formula* nad $\mathcal{L}$ se dobija konačnom primenom sledećih pravila:

- $\top$ i $\perp$ su formule

- atom nad $\mathcal{L}$ je formula
- ako je F formula, tada je i $\neg F$ formula
- ako su F i G formule, tada su i $F \wedge G$, $F \vee G$, $F \Rightarrow G$ i $F \Leftrightarrow G$ formule
- ako je F formula, a x promenljiva iz V , tada su $\exists x.F$ i $\forall x.F$ formule.

Napomena 2.18. Iz prethodne definicije vidimo koji je smisao *arnosti*: arnost simbola određuje na koliko se podtermova on može primeniti. Specijalan slučaj su funkcijski simboli arnosti 0 – oni se ne primenjuju na podtermove i sami za sebe predstavljaju termove (dakle, ne pišemo $a()$, već samo a). Takve simbole nazivamo *simbolima konstanti*. Slično, predikatski simboli arnosti 0 se ne primenjuju na termove, već sami za sebe predstavljaju atome, pa samim tim i formule. Ovakvi atomi odgovaraju iskaznim slovima u iskaznoj logici. Samim tim, svaka iskazna formula je i ujedno i formula prvog reda, u skladu sa ovom definicijom. Otuda se logika prvog reda može smatrati uopštenjem iskazne logike, gde sada dozvoljavamo i složeniju strukturu atoma.

Primer 2.16. Neka je data signatura $\mathcal{L} = (\Sigma, \Pi, ar)$, gde je $\Sigma = \{a, f, g\}$, pri čemu je $ar(a) = 0$, $ar(f) = 1$ i $ar(g) = 2$. Neka su x i y promenljive iz V . Tada su a , x , $f(a)$, $g(a, f(x))$, $g(g(x, y), f(f(a)))$ termovi nad $\mathcal{L}$. Sa druge strane, $f(x, y)$ nije ispravan term nad $\mathcal{L}$ jer f nije arnosti dva. Dalje, ako je $\Pi = \{p, q\}$, gde je $ar(p) = 1$, a $ar(q) = 2$, tada su $p(f(a))$ i $q(f(a), g(x, y))$ atomi nad $\mathcal{L}$, dok $p(x, y)$ to nije, s obzirom da je arnost simbola p jednaka 1.

Napomena 2.19. Kao i u iskaznoj logici, i ovde je potrebno definisati prioritete i asocijativnosti veznika, kako bi konkretna sintaksa bila jednoznačna. Za iskazne veznike zadržaćemo iste prioritete i asocijativnosti kao i u iskaznoj logici. Što se tiče kvantifikatora ($\exists, \forall$), smatraćemo da oni imaju najniži mogući prioritet. Otuda, formula $\exists x.p(x) \wedge q(x, a)$ će biti sintaksno identična formuli $\exists x.(p(x) \wedge q(x, a))$. Sa druge strane, ako želimo da se kvantifikator odnosi samo na $p(x)$, moramo ga staviti u zagrade: $(\exists x.p(x)) \wedge q(x, a)$.

Napomena 2.20. Ponekad se i funkcijski i predikatski simboli mogu zapisivati u *operatorskoj* notaciji. Na primer, ako imamo funkcijski simbol $+$ arnosti 2, obično umesto $+(a, b)$ pišemo $a + b$ (tj. zapisujemo ga kao infiksni operator). Slično, ako imamo funkcijski simbol $?$ arnosti 1, tada umesto $?(a)$ često pišemo $?a$ ili $a?$ (tj. zapisujemo ga kao prefiksni ili sufiksni operator). U slučaju da imamo takve simbole u signaturi, moramo za njih takođe definisati prioritete i asocijativnosti, kako bi konkretna sintaksa takvih termova i atoma bila jednoznačna. Na primer, ako imamo binarne funkcijske operatore $+$ i $\cdot$, tada moramo znati koji su im prioriteti, ako želimo da konkretna sintaksa terma $a + b \cdot c$ bude jednoznačna.

Definicija 2.15. *Slobodna i vezana pojavljivanja promenljive u formuli definišemo na sledeći način:*

- ako je A atom, tada su sva pojavljivanja promenljivih u A slobodna
- sva slobodna (vezana) pojavljivanja promenljivih u formuli F su slobodna (vezana) i u $\neg F$
- sva slobodna (vezana) pojavljivanja promenljivih u formulama F i G su slobodna (vezana) i u formulama $F \wedge G$, $F \vee G$, $F \Rightarrow G$ i $F \Leftrightarrow G$

- sva slobodna (vezana) pojavljivanja promenljivih u formuli F su slobodna (vezana) i u formulama $\exists x.F$ i $\forall x.F$, izuzev slobodnih pojavljivanja promenljive x u formuli F koja su sada vezana kvantifikatorom u formuli $\exists x.F$ (odnosno $\forall x.F$).

Napomena 2.21. Iz gornje definicije sledi da su sve promenljive slobodne, osim onih koje su vezane kvantifikatorom. Takođe, svaki kvantifikator vezuje samo slobodna pojavljivanja kvantifikovane promenljive u potformuli. Na primer, ako imamo formulu $\forall x.p(x) \Rightarrow (\exists x.q(x))$, spoljni kvantifikator $\forall x$ vezuje samo pojavljivanje promenljive x u $p(x)$, s obzirom da je pojavljivanje u $q(x)$ već bilo vezano i u potformuli unutrašnjim kvantifikatorom $\exists x$.

Primer 2.17. U formuli $\forall x.p(x, y) \Rightarrow (\exists y.q(y))$ pojavljivanje promenljive x je vezano kvantifikatorom $\forall x$. Sa druge strane, prvo pojavljivanje promenljive y (u $p(x, y)$) je slobodno, dok je drugo pojavljivanje (u $q(y)$) vezano kvantifikatorom $\exists y$.

Primer 2.18. U formuli $(\forall x.p(x)) \wedge (\forall x.q(x))$ imamo dva pojavljivanja promenljive x i oba su vezana, ali različitim kvantifikatorima.

Definicija 2.16. Formula je *zatvorena* (ili *rečenica*) ako ne sadrži slobodna pojavljivanja promenljivih.

Primer 2.19. U formuli $\forall x.\forall y.p(x, y) \Rightarrow (\exists z.q(x, z) \wedge q(z, y))$, sve promenljive su vezane odgovarajućim kvantifikatorima. Dakle, ova formula je rečenica.

Definicija 2.17. Za formulu (ili term) kažemo da je bazna, ako ne sadrži promenljive (ni slobodne ni vezane).

Napomena 2.22. Svaka bazna formula je rečenica, ali obrnuto ne mora da važi.

Primer 2.20. Term $f(a)$ je bazni, dok $f(x)$ to nije. Slično, formula $p(f(a)) \Rightarrow q(a) \vee q(b)$ je bazna, dok $\forall x.\forall y.p(f(x)) \Rightarrow q(x) \vee q(y)$ to nije (ako je rečenica).

Napomena 2.23. Umesto $\forall x_1.\forall x_2.\dots\forall x_n.F$ ćemo kraće pisati $\forall x_1x_2\dots x_n.F$. Slično za egzistencijalni kvantifikator.

Definicija 2.18. *Zamena* formule G formulom H u formuli F (u oznaci $F[G \rightarrow H]$) se definiše na sledeći način:

- $F[G \rightarrow H] = H$, ako je $F = G$
- $\top[G \rightarrow H] = \top$, $\perp[G \rightarrow H] = \perp$
- $p[G \rightarrow H] = p$, ako je $p \in P$ i $G \neq p$
- $(\neg F)[G \rightarrow H] = \neg F[G \rightarrow H]$
- $(F_1 \wedge F_2)[G \rightarrow H] = F_1[G \rightarrow H] \wedge F_2[G \rightarrow H]$
- $(F_1 \vee F_2)[G \rightarrow H] = F_1[G \rightarrow H] \vee F_2[G \rightarrow H]$
- $(F_1 \Rightarrow F_2)[G \rightarrow H] = F_1[G \rightarrow H] \Rightarrow F_2[G \rightarrow H]$
- $(F_1 \Leftrightarrow F_2)[G \rightarrow H] = F_1[G \rightarrow H] \Leftrightarrow F_2[G \rightarrow H]$
- $(\forall x.F)[G \rightarrow H] = \forall x.F[G \rightarrow H]$

- $(\exists x.F)[G \rightarrow H] = \exists x.F[G \rightarrow H]$.

Napomena 2.24. Prethodna definicija je gotovo identična definiciji zamene u iskaznoj logici (definicija 2.3) uz dodatak slučaja koji pokriva kvantifikatore. Dakle, i ovde se zamena vrši tako što se sva pojavljivanja potformule G u formuli F zamenjuju formulom H .

Pored zamene *formule formulom*, u logici prvog reda definišemo i zamenu *promenljive termom*.

Definicija 2.19. Zamena promenljive x termom t u termu s (formuli F), u oznaci $s[x \rightarrow t]$ ($F[x \rightarrow t]$) definiše se na sledeći način:

- $x[x \rightarrow t] = t$
- $y[x \rightarrow t] = y$ ($y \in V$ i $y \neq x$)
- $a[x \rightarrow t] = a$, gde je a simbol konstante
- $f(t_1, \dots, t_n)[x \rightarrow t] = f(t_1[x \rightarrow t], \dots, t_n[x \rightarrow t])$, gde je f funkcijski simbol arnosti n
- $\top[x \rightarrow t] = \top$
- $\perp[x \rightarrow t] = \perp$
- $p[x \rightarrow t] = p$, gde je p iskazni atom
- $p(t_1, \dots, t_n)[x \rightarrow t] = p(t_1[x \rightarrow t], \dots, t_n[x \rightarrow t])$, gde je p predikatski simbol arnosti n
- $(\neg F)[x \rightarrow t] = \neg F[x \rightarrow t]$
- $(F \wedge G)[x \rightarrow t] = F[x \rightarrow t] \wedge G[x \rightarrow t]$
- $(F \vee G)[x \rightarrow t] = F[x \rightarrow t] \vee G[x \rightarrow t]$
- $(F \Rightarrow G)[x \rightarrow t] = F[x \rightarrow t] \Rightarrow G[x \rightarrow t]$
- $(F \Leftrightarrow G)[x \rightarrow t] = F[x \rightarrow t] \Leftrightarrow G[x \rightarrow t]$
- $(Qx.F)[x \rightarrow t] = Qx.F$
- $(Qy.F)[x \rightarrow t] = Qy.F[x \rightarrow t]$, gde je $y \neq x$ i t ne sadrži promenljivu y
- $(Qy.F)[x \rightarrow t] = Qy'.F[y \rightarrow y'][x \rightarrow t]$, gde je $y \neq x$, y je sadržana u t , a y' nije sadržana ni u $Qy.F$ ni u t

gde je $Q \in \{\forall, \exists\}$.

Napomena 2.25. Iz prethodne definicije, zaključujemo da se zamena promenljive x termom t vrši tako što se sva *slobodna* pojavljivanja promenljive x zamene termom t . Pritom, važno je da ni jedna od promenljivih koje sadrži t ne postanu vezane prilikom zamene. Da se to ne bi dogodilo, može biti neophodno izvršiti *preimenovanje vezane promenljive*, što je pokriveno poslednjim slučajem u gornjoj definiciji (y se preimenuje u novu promenljivu y' , a zatim se izvrši zamena x sa t).

Pod preimenovanjem vezane promenljive (koje nazivamo i *alfa konverzija*) u formuli $Qy.F$ ($Q \in \{\forall, \exists\}$) podrazumevamo sledeću transformaciju:

$$Qy.F \mapsto Qz.F[y \rightarrow z]$$

gde je z promenljiva koja se ne pojavljuje u $Qy.F$. Za formule F i G kažemo da su *alfa ekvivalentne* (u oznaci $F \equiv_\alpha G$) ako se jedna može dobiti od druge samo primenom alfa konverzija. Na primer, formule $\forall x.p(x)$ i $\forall y.p(y)$ su alfa ekvivalentne. Slično, formule $\forall x.p(x) \Rightarrow (\exists y.q(x, y))$ i $\forall u.p(u) \Rightarrow (\exists v.q(u, v))$ su alfa ekvivalentne.

Primer 2.21. Pretpostavimo da želimo da izvršimo zamenu $[x \rightarrow f(y)]$ u formuli $\forall y.p(y) \Rightarrow q(x, y) \vee (\exists x.h(x, y))$. Zameniće se samo prvo pojavljivanje promenljive x , jer je drugo pojavljivanje (u $h(x, y)$) vezano egzistencijalnim kvantifikatorom. Dodatno, prilikom zamene prvog pojavljivanja promenljive x sa $f(y)$, promenljiva y u $f(y)$ bi postala vezana spoljnim univerzalnim kvantifikatorom, tj. dobili bismo $\forall y.p(y) \Rightarrow q(f(y), y) \vee (\exists x.h(x, y))$. Prema gornjoj definiciji, ovo nije ispravno, već je potrebno prvo preimenovati vezanu promenljivu y nekim novim imenom koje se do tada nije pojavljivalo u formuli (npr. sa z). Dakle, najpre imamo alfa konverziju polazne formule u $\forall z.p(z) \Rightarrow q(x, z) \vee (\exists x.h(x, z))$. Sada možemo bezbedno zameniti slobodno pojavljivanje promenljive x sa $f(y)$, pa imamo: $\forall z.p(z) \Rightarrow q(f(y), z) \vee (\exists x.h(x, z))$.

2.2.2 Semantika logike prvog reda

Definicija 2.20 ($\mathcal{L}$ -struktura). *Struktura* nad datom signaturom $\mathcal{L}$ ($\mathcal{L}$ -struktura) $\mathcal{D} = (D, \cdot^{\mathcal{D}})$ se sastoji iz nepraznog skupa D (koji nazivamo *domen* ili *univerzum* strukture $\mathcal{D}$) i preslikavanja $\cdot^{\mathcal{D}}$ koje:

- svakom simbolu konstante a signature $\mathcal{L}$ pridružuje fiksirani element $a^{\mathcal{D}} \in D$
- svakom funkcijskom simbolu f arnosti $n > 0$ signature $\mathcal{L}$ pridružuje funkciju $f^{\mathcal{D}} : D^n \rightarrow D$
- svakom iskaznom atomu p signature $\mathcal{L}$ pridružuje vrednost $p^{\mathcal{D}}$ iz skupa $\{0, 1\}$ (*netačno* ili *tačno*)
- svakom predikatskom simbolu p arnosti $n > 0$ signature $\mathcal{L}$ pridružuje relaciju $p^{\mathcal{D}}$ dužine n nad D , tj. $p^{\mathcal{D}} \subseteq D^n$.

Napomena 2.26. $\mathcal{L}$ -struktura zapravo određuje interpretaciju jezika $\mathcal{L}$, tako što fiksira skup i funkcije (operacije) i relacije nad tim skupom kojima će se interpretirati funkcijski i predikatski simboli jezika. Na primer, ako u imamo jezik $\mathcal{L}$ gde imamo konstante 0 i 1, binarne funkcijske simble $+$ i $\cdot$ i binarni predikatski simbol $\leq$, jedna $\mathcal{L}$ -struktura bi mogla da bude $(\mathbb{N}, 0, 1, +, \cdot, \leq)$, gde su sa 0, 1, $+$, $\cdot$, $\leq$ redom označeni prirodni brojevi nula i jedan, operacije sabiranja i množenja prirodnih brojeva, kao i relacija „manje ili jednako” nad skupom prirodnih brojeva.

Definicija 2.21 (Valuacija prvog reda). *Valuacija prvog reda* nad $\mathcal{L}$ -strukturuom $\mathcal{D} = (D, \cdot^{\mathcal{D}})$ je funkcija $v : V \rightarrow D$ koja svakoj promenljivoj iz V pridružuje jedan element domena D .

Napomena 2.27. $\mathcal{L}$ -struktura nam interpretira simbole signature $\mathcal{L}$. Međutim, u našim formulama se pojavljuju i promenljive koje nisu deo signature. Valuacija nam služi da odredi značenje promenljivih koje se u formuli javljaju.

Napomena 2.28. Treba jasno razlikovati valuaciju prvog reda od iskazne valuacije. Iskazna valuacija interpretira iskazne atome kao tačne ili netačne. U tom smislu, uopštenje iskazne valuacije je zapravo $\mathcal{L}$ -struktura, koja takođe, između ostalog, interpretira iskazne atome u jeziku $\mathcal{L}$ (ako postoje). Sa druge strane, valuacija prvog reda interpretira promenljive prvog reda kao elemente domena D . Nesrećna okolnost je to što se za ova dva suštinski različita pojma u literaturi obično koristi isti naziv, što može da dovede do zabune.

Definicija 2.22 (Interpretacija). Neka je data $\mathcal{L}$ -struktura $\mathcal{D}$ i valuacija prvog reda v . *Interpretacija* (ili *vrednost*) terma t u strukturi $\mathcal{D}$ i valuaciji v (u oznaci $\mathcal{D}_v(t)$) se definiše na sledeći način:

- $\mathcal{D}_v(x) = v(x)$, za $x \in V$
- $\mathcal{D}_v(a) = a^{\mathcal{D}}$, ako je a simbol konstante iz $\mathcal{L}$
- $\mathcal{D}_v(f(t_1, \dots, t_n)) = f^{\mathcal{D}}(\mathcal{D}_v(t_1), \dots, \mathcal{D}_v(t_n))$, ako je f funkcijski simbol arnosti $n > 0$ iz $\mathcal{L}$.

Interpretacija (ili *vrednost*) formule F u strukturi $\mathcal{D}$ i valuaciji v (u oznaci $\mathcal{D}_v(F)$) se definiše na sledeći način:

- $\mathcal{D}_v(\top) = 1$
- $\mathcal{D}_v(\perp) = 0$
- $\mathcal{D}_v(p) = p^{\mathcal{D}}$, ako je p iskazni atom iz $\mathcal{L}$
- $\mathcal{D}_v(p(t_1, \dots, t_n)) = 1$ akko je $(\mathcal{D}_v(t_1), \dots, \mathcal{D}_v(t_n)) \in p^{\mathcal{D}}$, ako je p predikat-ski simbol arnosti $n > 0$ iz $\mathcal{L}$
- $\mathcal{D}_v(\neg F) = 1$ akko $\mathcal{D}_v(F) = 0$
- $\mathcal{D}_v(F \wedge G) = 1$ akko je $\mathcal{D}_v(F) = 1$ i $\mathcal{D}_v(G) = 1$
- $\mathcal{D}_v(F \vee G) = 1$ akko je $\mathcal{D}_v(F) = 1$ ili $\mathcal{D}_v(G) = 1$
- $\mathcal{D}_v(F \Rightarrow G) = 1$ akko je $\mathcal{D}_v(F) = 0$ ili $\mathcal{D}_v(G) = 1$
- $\mathcal{D}_v(F \Leftrightarrow G) = 1$ akko je $\mathcal{D}_v(F) = \mathcal{D}_v(G)$
- $\mathcal{D}_v(\forall x.F) = 1$ akko za svaku valuaciju v' koja je dobijena od v samo eventualnom izmenom vrednosti promenljive x , važi $\mathcal{D}_{v'}(F) = 1$
- $\mathcal{D}_v(\exists x.F) = 1$ akko postoji valuacija v' koja je dobijena od v samo eventualnom izmenom vrednosti promenljive x , takva da je $\mathcal{D}_{v'}(F) = 1$.

Napomena 2.29. Iz prethodne definicije vidimo da se termovi interpretiraju elementima domena D , dok se formule (kao i u iskaznoj logici) interpretiraju kao *tačne* ili *netačne*.

Primer 2.22. Neka je dat jezik $\mathcal{L}$ koji sadrži konstante $0, 1, 2, \dots$, funkcijske simbole $+$ i $\cdot$ arnosti dva, kao i predikatske simbole $=$ i $\leq$ arnosti 2 (svi binarni simboli se zapisuju u operatorskoj notaciji, tj. infiksno). Neka je $\mathcal{D}$ standardna struktura prirodnih brojeva $\mathbb{N}$ sa uobičajenim interpretacijama gornjih simbola (koje ćemo, uz zloupotrebu notacije, označavati na isti način kao i same simbole). Neka je, dalje, valuacija v takva da je $v(x) = 3$ i $v(y) = 5$. Sada imamo:

- $\mathcal{D}_v(x + 1) = \mathcal{D}_v(x) + \mathcal{D}_v(1) = v(x) + 1 = 3 + 1 = 4$
- $\mathcal{D}_v(x \cdot y) = \mathcal{D}_v(x) \cdot \mathcal{D}_v(y) = v(x) \cdot \mathcal{D}(y) = 3 \cdot 5 = 15$
- $\mathcal{D}_v(y \leq 1) = 0$ (netačno), zato što $(\mathcal{D}_v(y), \mathcal{D}_v(1)) = (v(y), 1) = (5, 1) \notin$
- $\mathcal{D}_v(\exists y. x + y = 3) = 1$ (tačno), zato što postoji valuacija v' koja se od v može eventualno razlikovati na y takva da je $\mathcal{D}_{v'}(x + y = 3) = 1$. Zaista, možemo uzeti valuaciju v' za koju je $v'(y) = 0$. Sada je $\mathcal{D}_{v'}(x + y = 3) = 1$, jer je $(\mathcal{D}_{v'}(x + y), \mathcal{D}_{v'}(3)) = (v'(x) + v'(y), 3) = (3 + 0, 3) = (3, 3) \in$ (s obzirom da se simbol $=$ interpretira kao relacija $\{(x, x) \mid x \in \mathbb{N}\}$, tj. kao relacija *jednakosti* nad $\mathbb{N}$)
- $\mathcal{D}_v(\forall x. \neg(x + y \leq 2)) = 1$ (tačno), zato što za svaku valuaciju v' koja se od v može razlikovati eventualno na x važi $\mathcal{D}_{v'}(\neg(x + y \leq 2)) = 1$. Zaista, kako god da odaberemo $v'(x)$, važiće da *nije* $v'(x) + v'(y) \leq 2$ (jer je $v'(y) = v(y) = 5$).
- $\mathcal{D}_v(\forall xy. x + y = y + x) = 1$ (tačno), zato što za sve valuacije v' koje se od v mogu eventualno razlikovati na x i na y važi da $v'(x) + v'(y) = v'(y) + v'(x)$
- $\mathcal{D}_v(\forall xy. x \leq y \Rightarrow x + 1 \leq y + 1) = 1$ (tačno), sličnom analizom kao i u prethodnom primeru

Napomena 2.30. Primetimo da vrednost formule F u strukturi $\mathcal{D}$ i valuaciji v ne zavisi od vrednosti koje vezane promenljive formule F imaju u valuaciji v , već samo od vrednosti slobodnih promenljivih formule F . Na primer, u formuli $\exists y. x + y = 3$ bila nam je od značaja vrednost $v(x)$, jer je x slobodna promenljiva. U gornjem primeru, imali smo da je $v(x) = 3$, pa je formula bila tačna. Sa druge strane, da smo imali $v(x) = 10$, tada bi ova formula bila netačna, jer ne postoji vrednost iz $\mathbb{N}$ takva da sabrana sa 10 daje 3. Dakle, vrednost formule zavisi od valuacije $v(x)$ slobodne promenljive x . Sa druge strane, vrednost formule ne zavisi od vrednosti $v(y)$, jer je y vezana. Kod vezanih promenljivih, mi posmatramo valuacije v' kod kojih se vrednost $v'(y)$ može razlikovati od $v(y)$, što samu vrednost $v(y)$ čini irelevantnom.

Posledično, vrednost zatvorene formule (rečenice) ne zavisi uopšte od valuacije v . Na primer, formula $\forall xy. x \leq y \Rightarrow x + 1 \leq y + 1$ je u strukturi prirodnih brojeva tačna, nezavisno od odabrane valuacije v . Zbog toga ćemo za zatvorene formule obično pisati samo $\mathcal{D}(F)$ umesto $\mathcal{D}_v(F)$, jer nam valuacija nije relevantna.

Definicija 2.23. Za uređeni par $(\mathcal{D}, v)$ kažemo da *zadovoljava* formulu F (ili da je *model* formule F) ako je $\mathcal{D}_v(F) = 1$. Za formulu F kažemo da je *zadovoljiva* ako ima bar jedan model. Specijalno, rečenica F je zadovoljiva ako postoji struktura $\mathcal{D}$ takva da je $\mathcal{D}(F) = 1$ (takva struktura je model rečenice F).

Za formulu koja nije zadovoljiva kažemo da je *nezadovoljiva*.

Definicija 2.24. Formula F je *valjana* (ili *validna*) ako je tačna za svako $(\mathcal{D}, v)$. Specijalno, rečenica je valjana ako je tačna u svakoj strukturi $\mathcal{D}$.

Za formulu koja nije valjana kažemo da je *poreciva*.

Teorema 2.6. *Formula F je valjana akko je $\neg F$ nezadovoljiva. Formula F je zadovoljiva akko je $\neg F$ poreciva.*

Napomena 2.31. Pojmovi zadovoljivosti skupa formula, logičke posledice i logičke ekvivalencije se definišu na postpuno isti način kao i u iskaznoj logici.

Primer 2.23. Može se pokazati da je $\forall x.F \equiv \forall z.F[x \rightarrow z]$ gde je z nova promenljiva koja se ne pojavljuje u F . Slično važi i za egzistencijalni kvantifikator. Dakle, alfa konverzijom se dobija logički ekvivalentna formula.

Primer 2.24. Može se pokazati da važi: $\neg(\forall x.F) \equiv \exists x.\neg F$ i $\neg(\exists x.F) \equiv \forall x.\neg F$. Ove logičke ekvivalencije su poznate i kao *De Morganovi zakoni* za kvantifikatore.

U logici prvog reda takođe važi teorema kompaktnosti, kao i u iskaznoj logici. Ovu teoremu takođe navodimo bez dokaza.

Teorema 2.7 (Kompaktnost logike prvog reda). *Skup Δ formula prvog reda je zadovoljiv akko je zadovoljiv svaki njegov konačan potskup.*

Ekvivalentno, ako je Δ nezadovoljiv, tada postoji neki njegov konačan potskup koji je nezadovoljiv.

U logici prvog reda takođe važi teorema o zameni koju navodimo bez dokaza.

Teorema 2.8 (Teorema o zameni). *Ako je $G \equiv H$, tada je $F[G \rightarrow H] \equiv F$.*

Teorema o zameni se koristi kao opravdanje za *ekvivalentne transformacije* kojima se formula može svesti na pogodniji oblik, čuvajući pritom logičku ekvivalentnost.

Kada je u pitanju zamena promenljive termom, tu imamo sledeću teoremu.

Teorema 2.9. *Neka je E term ili formula. Za svaku strukturu $\mathcal{D}$ i valuaciju v važi da je $\mathcal{D}_v(E[x \rightarrow t]) = \mathcal{D}_{v'}(E)$, gde je v' valuacija koja se poklapa sa v svuda osim na x , gde važi da je $v'(x) = \mathcal{D}_v(t)$.*

Dokaz. Indukcijom po strukturi formule i termova. □

Napomena 2.32. Intuitivno, ova teorema kaže da kada zamenimo promenljivu x termom t , dobijena formula (ili term) se interpretira onako kako bi se polazna formula (ili term) interpretirala kada bismo tu promenljivu x interpretirali onako kako interpretiramo t .

Primetimo da je, kako bi gornja teorema važila, neophodno da se prilikom zamene x sa t zamenjuju samo *slobodna* pojavljivanja promenljive x , kao i da se tom prilikom ni jedna promenljiva iz t ne vezuje drugim kvantifikatorima u formuli. Ovo je zato što kvantifikatori menjaju semantiku promenljivih koje vezuju. Upravo zbog toga smo zamenili i definisali na takav način, jer smo želeli da ona ima semantički smisao iskazan gornjom teoremom.

2.2.3 Normalne forme u logici prvog reda

U logici prvog reda, pojmovi elementarne disjunkcije (klauze), elementarne konjunkcije, KNF-a i DNF-a se definišu na isti način kao u iskaznoj logici. Jedina razlika je u tome što literali više nisu samo iskazni atomi, već atomi prvog reda ili njihove negacije (npr. $x + y \leq 5$, $\neg(x \cdot y = 2)$ i sl.). Analogno teoremi 2.5, i u logici prvog reda se može pokazati da se bilo koja formula koja *ne sadrži* kvantifikatore može svesti na logički ekvivalentnu formulu u KNF-u ili DNF-u. Međutim, prisustvo kvantifikatora u formuli komplikuje proces transformacije formule. Može se pokazati sledeća teorema:

Teorema 2.10 (Prenex). *Za proizvoljnu formulu F postoji njoj logički ekvivalentna formula u prenex normalnoj formi, tj. formula oblika:*

$$Q_1x_1.Q_2x_2.\dots Q_nx_n.F'$$

gde je $Q_i \in \{\exists, \forall\}$ a formula F' ne sadrži kvantifikatore.

Dokaz. Da bi se formula transformisala u prenex normalnu formu, potrebno je „izvući” kvantifikatore ispred formule. Najpre se iz formule eliminišu implikacije i ekvivalencije na uobičajen način. Zatim se kvantifikatori izvlače primenom sledećih logičkih zakona:

$$\begin{aligned}\neg(\forall x.F) &\equiv \exists x.\neg F \\ \neg(\exists x.F) &\equiv \forall x.\neg F \\ (Qx.F) \wedge G &\equiv Qx.F \wedge G \\ (Qx.F) \vee G &\equiv Qx.F \vee G\end{aligned}$$

pri čemu je $Q \in \{\exists, \forall\}$, a formula G ne sadrži slobodna pojavljivanja promenljive x (ovo ne predstavlja ograničenje opštosti, s obzirom da se vezana promenljiva uvek može prethodno preimenovati u $Qx.F$, zadržavajući logičku ekvivalentnost). $\square$

Primer 2.25. Neka je data formula $\exists x.p(x) \Rightarrow (\forall x.p(x))$. Ova formula se transformiše u prenex normalnu formu tako što se najpre eliminiše implikacija, čime dobijamo formulu $\exists x.\neg p(x) \vee (\forall x.p(x))$. Dalje se kvantifikator $\forall x$ izvlači ispred disjunkcije. Pritom, kako se x pojavljuje kao slobodno u prvom disjunkt $\neg p(x)$, potrebno je najpre preimenovati vezanu promenljivu u $\forall x.p(x)$. Na primer, preimenujmo x u y . Sada imamo da je $\exists x.\neg p(x) \vee (\forall y.p(y)) \equiv \exists x.\forall y.\neg p(x) \vee p(y)$. Dobijena formula je u prenex normalnoj formi.

Primetimo da je bilo neophodno preimenovati promenljivu x prilikom izvlačenja univerzalnog kvantifikatora. Naime, da to nismo uradili, dobili bismo formulu $\exists x.\forall x.\neg p(x) \vee p(x)$, koja nije ekvivalentna polaznoj (sada su oba pojavljivanja promenljive x vezana univerzalnim kvantifikatorom, a u originalnoj formuli je prvo x bilo vezano egzistencijalnim kvantifikatorom). Dakle, bila bi promenjena semantika formule.

Glava 3

Izračunljivost

3.1 Intuitivni pojam algoritma

Definicija 3.1 (Intuitivni pojam algoritma). *Algoritam* predstavlja efektivni računski postupak za rešavanje nekog zadatog problema.

Napomena 3.1. *Problem* koji rešavamo može imati više različitih *instanci* koje imaju istu formu, ali se razlikuju po konkretnim vrednostima iz kojih se sastoje. Na primer, problem može biti *sortiranje niza*, a instanca ovog problema je bilo koji konačni niz brojeva koji treba sortirati. Algoritam treba da bude primenjiv na svaku instancu problema koji se rešava tim algoritmom. *Ulaz* algoritma je neka instanca problema, a *izlaz* je traženo rešenje, čija priroda i forma zavisi od samog problema. Na primer, u problemu sortiranja niza, izlaz je takođe niz – onaj koji se dobija sortiranjem ulaznog niza. U nekim drugim problemima, izlaz može biti jedan broj (npr. kod problema sumiranja elemenata niza), ili samo odgovor *da/ne* (npr. kada se pitamo da li se neka data vrednost nalazi u nizu ili ne).

Napomena 3.2. Iz definicije 3.1 sledi da kada govorimo o algoritmu, mislimo na *računski* postupak, tj. postupak koji se sastoji iz niza računskih operacija koje treba primeniti nad ulaznim podacima (koji su, samim tim, predstavljeni brojevima) da bi se dobio željeni rezultat (koji je opet predstavljen brojevima). Ovo je u skladu sa tradicionalnim shvatanjem algoritma kao postupka za rešavanje računskih problema koji se pre svega javljaju u matematici. U praksi, ovo nas previše ne ograničava, zato što se mnoge vrste informacija mogu *digitalizovati*, tj. predstaviti brojevima (npr. tekst, zvuk, slika, video, i sl.). Naravno, algoritam se može definisati i opštije – kao bilo koji precizno opisani postupak za rešavanje nekog problema (npr. postupak za pripremu nekog jela). Ipak, u ovom tekstu ćemo se pre svega fokusirati na računske postupke, jer su to postupci koji se mogu izvršavati na računarima.

Napomena 3.3. Definicija 3.1 navodi da je algoritam *efektivan* postupak. To znači sledeće:

- algoritam se sastoji iz konačno mnogo koraka. Samim tim, moguće ga je opisati konačnim sredstvima, sačuvati u konačnoj memoriji i sl.
- algoritam je precizno definisan u smislu da su njegovi koraci precizno zadati i njihov redosled je jasno određen. Samim tim, za izvršavanje algoritma

potrebno je samo slepo pratiti njegove korake, tj. nije potrebna nikakva kreativnost. Ovo znači da su algoritmi pogodni za izvršavanje od strane mašine.

- izvršavanje algoritma zahteva konačne resurse (vreme, prostor i sl.). Primetimo da to što se algoritam sastoji iz konačno mnogo koraka ne znači obavezno da će se izvršavati u konačnom vremenu, s obzirom da se isti korak može ponavljati više puta (npr. u petlji). Zbog toga je za korektnost algoritma obično neophodno obezbediti da se za svaki dopustivi ulaz algoritam *zaustavlja* nakon konačno mnogo koraka.
- za svaki dopustivi ulaz – *instancu* datog problema, algoritam daje odgovarajući izlaz. Dakle, algoritam zaista mora da rešava dati problem i da radi ispravno za svaki dopustivi ulaz. Naravno, mogu postojati ulazi koji nisu dopustivi (tj. koji ne predstavljaju legalne instance problema). Za takve ulaze, ponašanje algoritma može biti nepredvidivo (npr. može dati besmisleni rezultat ili se može izvršavati zauvek).

Napomena 3.4. Pojam algoritma mnogo je stariji od savremenih elektronskih računara na koje obično mislimo kada govorimo o algoritmima. Prvi algoritmi potiču još iz drevnih civilizacija, a u njih spadaju:

- algoritmi za izračunavanje aritmetičkih operacija u pozicionim brojevnim sistemima
- Euklidov algoritam za izračunavanje najvećeg zajedničkog delioca
- Gausov algoritam za rešavanje sistema linearnih jednačina
- ...



Slika 3.1: Persijski matematičar al-Horezmi

Sama reč *algoritam* nastala je kao vesternizovana varijanta prezimena persijskog matematičara *al-Horezmi* (*Muḥammad ibn Mūsā al-Khwārizmī*, slika 3.1), koji je u 9. veku nove ere napisao prve knjige o tada poznatim algoritmima, pre svega kada je u pitanju izvođenje računskih operacija u pozicionim brojevnim sistemima (*Knjiga o indijskim izračunavanjima* i *Sabiranje i oduzimanje u indijskoj aritmetici*).

3.2 Formalni pojam algoritma

Napomena 3.5. U ovom tekstu, pod skupom prirodnih brojeva $\mathbb{N}$ podrazumevamo skup $\{0, 1, 2, \dots\}$. Dakle, nulu smatramo prirodnim brojem i to dalje nećemo posebno naglašavati.

Definicija 3.2. *Parcijalna funkcija reda k ($k \in \mathbb{N}$) je bilo koja funkcija $f : A \rightarrow \mathbb{N}$, gde je $A \subseteq \mathbb{N}^k$. Skup A nazivamo *domen* funkcije f i označavamo ga sa $\text{dom}(f)$. Pritom, ako je $(x_1, \dots, x_k) \notin A$, kažemo da je funkcija f *nedefinisana* za $(x_1, \dots, x_k)$ i pišemo $f(x_1, \dots, x_k) = -$. Za parcijalnu funkciju reda k kažemo da je *totalna* ako je definisana za svako $(x_1, \dots, x_k) \in \mathbb{N}^k$, tj. ako je $\text{dom}(f) = \mathbb{N}^k$.*

Napomena 3.6. Primetimo da red k funkcije f može biti i nula. Takva funkcija nema argumente, pa uvek „vraća” istu vrednost, npr. $f() = c \in \mathbb{N}$. Takva funkcija se poistovećuje sa samim brojem $c \in \mathbb{N}$.

Da bi se parcijalna funkcija smatrala algoritmom, potrebno je da bude *efektivno izračunljiva*. Postoje razni načini da se definiše pojam *efektivne izračunljivosti*. U narednim odeljcima ilustrujemo neke najpoznatije formalizme kojima se opisuju efektivno izračunljive funkcije. Može se pokazati da su svi oni međusobno ekvivalentni.

3.2.1 μ -rekurzivne funkcije

μ -rekurzivne funkcije predstavljaju formalizam kod koga se parcijalne funkcije koje smatramo izračunljivim konstruišu konačnom primenom unapred datog skupa funkcijskih operatora. Svakim operatorom se od jednostavnijih funkcija grade složenije, kao što u programskim jezicima pomoću raspoloživih programskih konstrukcija od jednostavnijih programa gradimo složenije. μ -rekurzivne funkcije formalizovao je američki matematičar Stiven Klini (1909-1994, slika 3.2).



Slika 3.2: Američki matematičar Stiven Klini

Definicija 3.3. *Nula-funkcija reda k je funkcija definisana na sledeći način:*

$$Z_k(x_1, \dots, x_k) = 0$$

za svako $(x_1, \dots, x_k) \in \mathbb{N}^k$. *Funkcija sledbenik* je funkcija reda 1, definisana na sledeći način:

$$S(x) = x + 1$$

za svako $x \in \mathbb{N}$. *Funkcija i -te projekcije reda k ($i \leq k$)* je funkcija definisana na sledeći način:

$$P_k^i(x_1, \dots, x_k) = x_i$$

za svako $(x_1, \dots, x_k) \in \mathbb{N}^k$. Ove funkcije nazivamo *osnovne rekurzivne funkcije*.

Primer 3.1. Primetimo da je funkcija $P_1^1(x) = x$ zapravo *identička* funkcija. Označavaćemo je i sa $I(x)$.

Napomena 3.7. Intuitivno, osnovne rekurzivne funkcije predstavljaju elementarne algoritamske korake koje možemo da primenimo. Naime, s obzirom da je prirodan broj ili nula ili sledbenik nekog prirodnog broja, funkcije Z_k i S nam omogućavaju da izračunamo bilo koji prirodan broj, kao i da pomoću njih izrazimo uobičajene aritmetičke operacije. Dakle, intuitivno, sve što je izračunljivo se može na kraju svesti na uzastopnu primenu ove dve funkcije. Sa druge strane, funkcija projekcije nam omogućava da među argumentima funkcije izaberemo onaj koji nam treba da nad njim vršimo izračunavanje.

Definicija 3.4. Ako su date parcijalne funkcije $g_1, \dots, g_m$ reda k i parcijalna funkcija h reda m , tada je je parcijalna funkcija f reda k dobijena *supstitucijom* funkcija g_i u h (u oznaci $Sub(h, g_1, \dots, g_m)$) definisana na sledeći način:

$$f(x_1, \dots, x_k) = h(g_1(x_1, \dots, x_k), \dots, g_m(x_1, \dots, x_k))$$

za svako $(x_1, \dots, x_k)$ za koje je izraz na desnoj strani definisan (u suprotnom, funkcija f je nedefinisana za takvo $(x_1, \dots, x_k)$).

Napomena 3.8. Intuitivno, operator supstitucije omogućava kompoziciju algoritama. Funkcije $g_1, \dots, g_m$ možemo da razumemo kao „potprograme” kojima smo izračunali neke međurezultate, a funkcija h je glavni program koji se primenjuje na te međurezultate i daje konačan rezultat. Otuda je ceo program zapravo supstitucija $Sub(h, g_1, \dots, g_m)$. Intuitivno, kompozicijom algoritama dobijamo algoritam.

Primer 3.2. Neka je data proizvoljna parcijalna funkcija f reda 1. Posmatrajmo funkciju $g = \underbrace{Sub(f, Sub(f, \dots, Sub(f, I) \dots))}_n$, tj. funkciju:

$$g(x) = \underbrace{f(f(\dots f(I(x)) \dots))}_n$$

Ovo je zapravo funkcija koja se dobija kada n puta primenimo funkciju f na argument x . Označavaćemo je sa f^n .

Napomena 3.9. U prethodnom primeru, kao i u primerima koji slede, često ćemo imati dve notacije. Jedna notacija predstavlja *konstruktivni oblik* u kome formalno zapisujemo operatore koji se primenjuju prilikom konstrukcije funkcije. Na primer, $g = Sub(h, f, Sub(f, P_2^2, P_2^1))$. Istu funkciju možemo zapisati i u *eksplicitnom obliku*:

$$g(x, y) = h(f(x, y), f(y, x))$$

Konstruktivni oblik je formalniji i precizniji, jer nam govori tačno kako je funkcija konstruisana. Sa druge strane, eksplicitni zapis je obično čitljiviji, jer nam prikazuje šta funkcija izračunava, tj. koje operacije primenjuje nad svojim argumentima. Ipak, kada zapišemo funkciju u eksplicitnom obliku, tada nam nije jasno koji je argument koji po redu (npr. u zapisu $h(f(x, y), f(y, x))$) znamo da imamo dva argumenta, x i y , ali ne znamo koji je argument prvi, a koji drugi.

Primer 3.3. Ako u prethodnom primeru uzmemo da je $f = S$, dobijamo funkciju S^n . Važi da je $S^n(x) = x + n$ za svako $x \in \mathbb{N}$.

Primer 3.4. Posmatrajmo funkciju $C_k^n = \text{Sub}(S^n, Z_k)$, tj. funkciju za koju važi:

$$C_k^n(x_1, \dots, x_k) = S^n(Z_k(x_1, \dots, x_k))$$

Ova funkcija ima vrednost n za svako $(x_1, \dots, x_k)$. Zovemo je *konstantnom funkcijom*. Specijalno, za $k = 0$ ova funkcija predstavlja sam broj n .

Primer 3.5. Posmatrajmo funkciju $SP_k^{i,n} = \text{Sub}(S^n, P_k^i)$, tj. funkciju:

$$SP_k^{i,n}(x_1, \dots, x_k) = S^n(P_k^i(x_1, \dots, x_k))$$

Važi da je $SP_k^{i,n}(x_1, \dots, x_k) = x_i + n$ za svako $(x_1, \dots, x_k)$.

Napomena 3.10. Posmatrajmo sledeći pseudokod:

```

{
  x := f1(x);
  x := f2(x);
  ...
  x := fn(x);
}
return x;
```

Dakle, imamo jednu blok naredbu koja postupno računa rezultat tako što na ulazni podatak x primenjuje različite operacije. Ovaj algoritam se može predstaviti pomoću supstitucije:

$$g(x) = f_n(f_{n-1}(\dots f_1(x) \dots))$$

odnosno, u *konstruktivnom obliku*:

$$g = \text{Sub}(f_n, \text{Sub}(\dots, \text{Sub}(f_2, f_1) \dots))$$

Iz ovog primera vidimo da blok naredbe koje postoje u većini programskih jezika možemo predstaviti operatorom supstitucije.

Definicija 3.5. Ako su date parcijalne funkcije g reda k i h reda $k + 2$, tada je funkcija f reda $k + 1$ dobijena (*primitivnom*) *rekurzijom* od g i h (u oznaci $\text{Rec}(g, h)$) definisana na sledeći način:

$$\begin{aligned} f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\ f(y + 1, x_1, \dots, x_k) &= h(y, x_1, \dots, x_k, f(y, x_1, \dots, x_k)) \end{aligned}$$

za svako $y, x_1, \dots, x_k$ za koje su desne strane gornjih jednakosti definisane (u suprotnom, funkcija f je nedefinisana za takve ulaze).

Napomena 3.11. Operator rekurzije omogućava da se vrednost funkcije za neko $y + 1$ svede na izračunavanje vrednosti te iste funkcije za vrednost y (pri istim vrednostima ostalih argumenata $x_1, \dots, x_k$). Pritom, pretpostavljamo da se rekurzija uvek vrši po prvom argumentu. Intuitivno, da bismo neku funkciju implementirali primenom operatora rekurzije, potrebno je da znamo:

1. postupak za izračunavanje vrednosti funkcije za $y = 0$ (izlaz iz rekurzije). Ova vrednost zavisi samo od ostalih argumenata funkcije $x_1, \dots, x_k$, a postupak za njeno izračunavanje dat je funkcijom g reda k
2. postupak za izračunavanje vrednosti funkcije za $y + 1$, pod pretpostavkom da nam je poznata vrednost funkcije za y . Ovaj postupak može, pored vrednosti koju vraća rekurzivni poziv, koristiti i vrednosti ostalih argumenata $x_1, \dots, x_k$, ali i vrednost argumenta y . Otuda je on predstavljen funkcijom h reda $k + 2$.

Pritom, jasno je da će „dubina” rekurzije biti unapred poznata – imaćemo y rekurzivnih poziva, jer će u svakom od njih vrednost prvog argumenta biti manja za jedan u odnosu na prethodni. Na kraju ćemo imati izlaz iz rekurzije, za $y = 0$.

Postoji i drugačija intuicija vezana za operator rekurzije – možemo ga posmatrati kao način da se formalizuje pojam *petlje*. Naime, operator rekurzije odgovara sledećem pseudokodu:

```


$$\begin{aligned}
&z := 0; \\
&res := g(x_1, \dots, x_k); \\
&while(z < y) \\
&\{ \\
&\quad res := h(z, x_1, \dots, x_k, res); \\
&\quad z := z + 1; \\
&\} \\
&return \quad res;
\end{aligned}$$


```

Dakle, funkcija (algoritam) g se koristi inicijalizaciju rezultata res , a zatim se on dalje izračunava u petlji čije je telo predstavljeno funkcijom (algoritmom) h . Brojač petlje z ide od 0 do y . Dakle, u pitanju je brojačka petlja, pri čemu je broj njenih iteracija unapred poznat i određen argumentom y .

Definicija 3.6. Parcijalna funkcija f je *primitivno rekurzivna* ako se može dobiti od osnovnih elementarnih funkcija konačnom primenom operatora supstitucije i primitivne rekurzije.

Primer 3.6. Posmatrajmo funkciju $prev$ reda 1 definisanu na sledeći način:

$$\begin{aligned}
prev(0) &= 0 \\
prev(y + 1) &= y
\end{aligned}$$

Ovu funkciju možemo konstruisati pomoću operatora rekurzije – njen konstruktivni zapis je $Rec(Z_0, P_2^1)$. Zaista, kako je funkcija $prev$ reda 1, potrebna nam je funkcija reda 0 (konstanta) koja implementira izlaz iz rekurzije – to je funkcija Z_0 (jer je $Z_0() = 0$). Takođe, potrebna nam je funkcija reda 2 koja prihvata samo y i $prev(y)$ (jer funkcija $prev$ nema drugih argumenata) i vraća $prev(y + 1) = y$. To je upravo funkcija projekcije P_2^1 (jer je $P_2^1(y, prev(y)) = y$). Dakle, funkcija

prev je primitivno rekurzivna. Zovemo je *funkcija prethodnik*, jer za svaki prirodni broj y vraća njegovog prethodnika $y - 1$ (osim u slučaju nule, kada vraća nulu).

Napomena 3.12. Prethodni primer prikazuje jedan degenerisani slučaj primene operatora rekurzije u kome zapravo i nema rekurzije, s obzirom da nema rekurzivnog poziva. Naime, funkcija h može da ne zavisi od svog poslednjeg argumenta, tj. možemo imati da je:

$$h(y, x_1, \dots, x_k, z) = \bar{h}(y, x_1, \dots, x_k)$$

U prethodnom primeru, to smo postigli primenom funkcije projekcije, koja je vraćala samo svoj prvi argument, ignorišući drugi. U opštem slučaju, to znači da je funkcija $f = \text{Rec}(g, h)$ zapravo definisana ovako:

$$\begin{aligned} f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\ f(y + 1, x_1, \dots, x_k) &= \bar{h}(y, x_1, \dots, x_k) \end{aligned}$$

Intuitivno, ovo je ekvivalentno sa sledećim pseudokodom:

```

if (y = 0)
{
    return g(x1, ..., xk);
}
else
{
    return  $\bar{h}(y - 1, x_1, \dots, x_k)$ ;
}

```

Na ovaj način zapravo implementiramo *grananje*.

Napomena 3.13. Primetimo da kada funkcije implementirane operatorom rekurzije zapisujemo u eksplicitnom obliku, neophodno je navesti dva slučaja – jedan za izlaz iz rekurzije, a drugi za rekurzivni korak. Sa druge strane, konstruktivni zapis operatora rekurzije $\text{Rec}(f, h)$ je vrlo kompaktan i omogućava jednostavnu dalju konstrukciju složenijih funkcija.

Primer 3.7. Posmatrajmo funkciju $\text{prev}^n(x) = x - n$. Ova funkcija je primitivno rekurzivna, kao stepen primitivno rekurzivne funkcije. Primetimo da je $\text{prev}^n(x) = 0$ kad god je $n \geq x$.

Primer 3.8. Neka je data proizvoljna parcijalna funkcija f reda 1. Posmatrajmo funkciju g reda 2 definisanu na sledeći način:

$$\begin{aligned} g(0, x) &= x \\ g(y + 1, x) &= f(g(y, x)) \end{aligned}$$

Ova funkcija se može dobiti operatorom rekurzije od funkcije f : $g = \text{Rec}(I, \text{Sub}(f, P_3^3))$, s obzirom da je $I(x) = x$ i $\text{Sub}(f, P_3^3)(y, x, g(y, x)) = f(P_3^3(y, x, g(y, x))) = f(g(y, x))$. Ova funkcija zapravo radi tako što funkciju f primenjuje y puta na argument x . Na primer: $g(3, x) = f(g(2, x)) = f(f(g(1, x))) = f(f(f(g(0, x)))) = f(f(f(x)))$. Ovu funkciju ćemo označavati sa $\text{rep}(f)$. Ako je f primitivno rekurzivna, onda je i $\text{rep}(f)$ primitivno rekurzivna. Primetimo razliku funkcije $\text{rep}(f)$ u odnosu na f^n : funkcija f^n primenjuje

funkciju f na argument *konstantan broj puta* (n je unapred poznato). Funkcija $rep(f)$ primenjuje f na drugi argument onoliko puta koliko zahteva prvi argument (koji može biti proizvoljno zadat pri svakom pozivu funkcije). Otuda je za izračunavanje funkcije f^n dovoljno n puta primeniti f (nije potrebna petlja), dok je za izračunavanje funkcije $rep(f)$ potrebna brojačka petlja sa y iteracija.

Primer 3.9. Posmatrajmo funkciju $add = rep(S)$. Vrednost $add(y, x)$ dobijamo kada y puta primenimo S na x . Ovim se zapravo dobija zbir brojeva y i x . Na primer $add(3, 4) = S(S(S(4))) = S(S(5)) = S(6) = 7$.

Primer 3.10. Slično kao u prethodnom primeru, imamo da je $sub = rep(prev)$ funkcija oduzimanja, tj. $sub(y, x) = x - y$. Zaista, na primer, imamo da je $sub(3, 5) = prev(prev(prev(5))) = prev(prev(4)) = prev(3) = 2$. Primetimo da ako je $y \geq x$, tada je $sub(y, x) = 0$.

Primer 3.11. Neka je data proizvoljna parcijalna funkcija f reda 2. Posmatrajmo funkciju g reda 2 definisanu na sledeći način:

$$\begin{aligned} g(0, x) &= n \\ g(y + 1, x) &= f(x, g(y, x)) \end{aligned}$$

Ova funkcija se može konstruisati operatorom primitivne rekurzije: $g = Rec(C_1^n, Sub(f, P_3^2, P_3^3))$ (jer je $C_1^n(x) = n$, a $Sub(f, P_3^2, P_3^3)(y, x, g(y, x)) = f(P_3^2(y, x, g(y, x)), P_3^3(y, x, g(y, x))) = f(x, g(y, x))$). Razmotrimo šta ova funkcija radi. Na primer, imamo da je $g(4, x) = f(x, g(3, x)) = f(x, f(x, g(2, x))) = f(x, f(x, f(x, g(1, x)))) = f(x, f(x, f(x, f(x, g(0, x))))) = f(x, f(x, f(x, f(x, C_1^n(x))))) = f(x, f(x, f(x, f(x, n))))$. Dakle, polazeći od vrednosti n , ova funkcija akumulira rezultat primene funkcije f , y puta sa prvim argumentom x . Označavaćemo je sa $acc(n, f)$. Ako je f primitivno rekurzivna, onda je takva i $acc(n, f)$.

Primer 3.12. Sada imamo da je $mul = acc(0, add)$ funkcija množenja, tj. $mul(y, x) = y \cdot x$. Na primer:

$$\begin{aligned} mul(4, 5) &= \\ add(5, add(5, add(5, add(5, 0)))) &= \\ add(5, add(5, add(5, 5))) &= \\ add(5, add(5, 10)) &= \\ add(5, 15) &= 20 \end{aligned}$$

Primer 3.13. Funkcija $pow = acc(1, mul)$ izračunava stepen, tj. $pow(y, x) = x^y$. Na primer:

$$\begin{aligned} pow(3, 5) &= \\ mul(5, mul(5, mul(5, 1))) &= \\ mul(5, mul(5, 5)) &= \\ mul(5, 25) &= 125 \end{aligned}$$

Primer 3.14. Posmatrajmo funkciju:

$$\begin{aligned} fact(0) &= 1 \\ fact(x + 1) &= mul(S(x), fact(x)) \end{aligned}$$

tj. funkciju $fact = Rec(C_0^1, Sub(mul, Sub(S, P_2^1), P_2^2))$. Ovo je uobičajena rekurzivna definicija faktoriijela $x!$. Jasno je da je ova funkcija primitivno rekurzivna.

Primer 3.15. Posmatrajmo funkciju:

$$\begin{aligned} is_zero(0) &= 1 \\ is_zero(y+1) &= 0 \end{aligned}$$

Dakle, ova funkcija vraća 1 ako je argument jednak nuli, a u suprotnom vraća 0. Ova funkcija je primitivno rekurzivna ($is_zero = Rec(C_0^1, Z_2)$). Može se koristiti za testiranje da li je neka vrednost jednaka nuli.

Primer 3.16. Pomoću funkcija iz prethodnih primera možemo definisati logičke operatore. U tu svrhu, smatraćemo da je vrednost 0 logički *netačno*, dok je svaka vrednost različita od nule logički *tačno*. Sada imamo:

- $not(x) = is_zero(x)$
- $and(x, y) = mul(x, y)$
- $or(x, y) = add(x, y)$
- $xor(x, y) = or(and(not(x), y), and(x, not(y)))$

Primer 3.17. Posmatrajmo funkciju:

$$cond(y, x_1, x_2) = \begin{cases} x_2, & \text{za } y = 0 \\ x_1, & \text{za } y \neq 0 \end{cases}$$

Ova funkcija, ispituje logičku vrednost svog prvog argumenta – ako je ona logički tačna, tada vraća svoj drugi argument, a u suprotnom vraća treći argument. Intuitivno, ova funkcija omogućava *grananje* u našem programu. Ova funkcija se može predstaviti kao primitivno rekurzivna funkcija:

$$\begin{aligned} cond(0, x_1, x_2) &= x_2 \\ cond(y+1, x_1, x_2) &= x_1 \end{aligned}$$

tj. važi: $cond = Rec(P_2^2, P_4^2)$.

Primer 3.18. Posmatrajmo funkciju:

$$ge(x, y) = is_zero(sub(x, y))$$

Ova funkcija je primitivno rekurzivna, jer je dobijena supstitucijom od primitivno rekurzivnih funkcija ($ge = Sub(is_zero, sub)$). Primetimo da je $y - x = 0$ akko je $x \geq y$. Otuda ova funkcija vraća 1 ako je $x \geq y$, dok u suprotnom vraća 0. Čitaocu prepuštamo za vežbu da, koristeći ovu funkciju i logičke funkcije iz primera 3.16 implementira i ostale relacione operatore (*eq*, *le*, *gt*, *lt*).

Primer 3.19. Funkcije:

- $max(x, y) = cond(ge(x, y), x, y)$
- $min(x, y) = cond(le(x, y), x, y)$

su primitivno rekurzivne, s obzirom da su dobijene od primitivno rekurzivnih funkcija primenom supstitucije. Ove funkcije računaju, respektivno, maksimum i minimum dva broja.

Primer 3.20. Posmatrajmo funkciju:

$$\text{mod}(x, y) = \begin{cases} 0, & \text{za } x = 0 \\ 0, & \text{ako je } \text{mod}(x-1, y) + 1 = y \\ \text{mod}(x-1, y) + 1, & \text{inače} \end{cases}$$

Za $y \neq 0$ ova funkcija vraća ostatak pri deljenju x sa y (u suprotnom vraća x). Kako bismo je predstavili pomoću primitivne rekurzije, predstavimo je ovako:

$$\begin{aligned} \text{mod}(0, y) &= 0 \\ \text{mod}(x+1, y) &= \text{cond}(\text{eq}(\text{mod}(x, y) + 1, y), 0, \text{mod}(x, y) + 1) \end{aligned}$$

odnosno, važi da je:

$$\text{mod} = \text{Rec}(Z_1, \text{Sub}(\text{cond}, \text{Sub}(\text{eq}, \text{Sub}(S, P_3^3), P_3^2), Z_3, \text{Sub}(S, P_3^3)))$$

Slično:

$$\text{div}(x, y) = \begin{cases} 0, & \text{za } x = 0 \\ \text{div}(x-1, y) + 1, & \text{ako je } \text{mod}(x-1, y) + 1 = y \\ \text{div}(x-1, y), & \text{inače} \end{cases}$$

Ova funkcija za $y \neq 0$ izračunava celobrojni količnik x pri deljenju sa y , a u suprotnom vraća nulu. Da bismo je izrazili pomoću primitivne rekurzije, predstavimo je u sledećem obliku:

$$\begin{aligned} \text{div}(0, y) &= 0 \\ \text{div}(x+1, y) &= \text{cond}(\text{eq}(\text{mod}(x, y) + 1, y), \text{div}(x, y) + 1, \text{div}(x, y)) \end{aligned}$$

odnosno:

$$\text{div} = \text{Rec}(Z_1, \text{Sub}(\text{cond}, \text{Sub}(\text{eq}, \text{Sub}(S, \text{Sub}(\text{mod}, P_3^1, P_3^2)), P_3^2), \text{Sub}(S, P_3^3), P_3^3))$$

Teorema 3.1. *Svaka primitivno rekurzivna funkcija je totalna.*

Dokaz. Osnovne rekurzivne funkcije su totalne. Operatorima supstitucije i rekurzije se od totalnih funkcija dobijaju totalne funkcije. Otuda su sve primitivno rekurzivne funkcije totalne. $\square$

Primitivno rekurzivne funkcije omogućavaju nam da izražavamo aritmetičke operacije, linijske i razgranate strukture programa, kao i brojačke petlje, tj. petlje kod kojih je broj iteracija poznat unapred, u trenutku ulaska u petlju. Sa druge strane, petlje oblika „ponavljaj naredbu dokle god se neki uslov ne ispunji”, pri čemu ne znamo ni kada će ni da li će se uslov izlaska iz petlje ispuniti nije moguće predstaviti primitivno rekurzivnim funkcijama. Za tako nešto nam je potreban *operator minimizacije*.

Definicija 3.7. Ako je data parcijalna funkcija g reda $k+1$, tada je funkcija f reda k dobijena *minimizacijom* od funkcije g (u oznaci $\mu(g)$) definisana na sledeći način:

$$f(x_1, \dots, x_k) = y$$

gde je $y \in \mathbb{N}$ takvo da važi:

- za svako $z < y$, vrednost $g(z, x_1, \dots, x_k)$ je definisana i važi:

$$g(z, x_1, \dots, x_k) > 0$$

- vrednost $g(y, x_1, \dots, x_k)$ je definisana i važi:

$$g(y, x_1, \dots, x_k) = 0$$

Ako takvo y ne postoji, funkcija f je nedefinisana za takvo $x_1, \dots, x_k$.

Napomena 3.14. Oznaka $\mu(g)$ se koristi kada je g data u konstruktivnom obliku. Ako je g data u eksplicitnom obliku (kao funkcijski izraz $g(y, x_1, \dots, x_k)$), tada ćemo koristiti sledeću *eksplicitnu* notaciju:

$$\mu y[g(y, x_1, \dots, x_k) = 0]$$

Intuitivno, ovo znači: „*najmanje y takvo da je g(y, x₁, ..., x_k) jednako 0*”. Primetimo da je ovde neophodno eksplicitno navesti argument po kome se vrši minimizacija (μy). Naime, minimizacija se, prema definiciji, uvek vrši po prvom argumentu. Kada koristimo konstruktivnu notaciju, tada je jasno koji je argument prvi. Sa druge strane, kao što je ranije rečeno, u eksplicitnoj notaciji nije uvek jasno koji je argument koji po redu, pa je zato potrebno eksplicitno naglasiti po kom se argumentu vrši minimizacija.

Napomena 3.15. Operator minimizacije je intuitivno ekvivalentan sledećem pseudokodu:

```

y := 0;
while(g(y, x1, ..., xk))
{
    y := y + 1;
}
return y;

```

Dakle, funkciju $g(y, x_1, \dots, x_k)$ posmatramo kao logički izraz (uslov petlje), a rezultat je najmanje y za koje uslov taj uslov postaje netačan.

Pretpostavimo sada da želimo da u petlji izračunamo nešto drugo. Na primer:

```

y := 0;
res := h(x1, ..., xk);
while(g(y, x1, ..., xk))
{
    res := f(y, x1, ..., xk, res);
    y := y + 1;
}
return res;

```

Ovo je sada neka najopštija *while* petlja u kojoj izračunavamo neki rezultat *res*. Ako bismo unapred znali broj iteracija petlje, to bi onda bila primitivna rekurzija:

$$\begin{aligned}
u(0, x_1, \dots, x_k) &= h(x_1, \dots, x_k) \\
u(y + 1, x_1, \dots, x_k) &= f(y, x_1, \dots, x_k, u(y, x_1, \dots, x_k));
\end{aligned}$$

tj. $u = \text{Rec}(h, f)$. Kako broj iteracija ne znamo unapred, potrebno je najpre odrediti broj iteracija minimizacijom, a zatim u novoj brojačkoj petlji izračunati rezultat. Dakle, imamo funkciju (u konstruktivnoj notaciji):

$$v = \text{Sub}(u, \mu(g), P_k^1, \dots, P_k^k)$$

odnosno u eksplicitnoj formi:

$$v(x_1, \dots, x_k) = u(\mu y[g(y, x_1, \dots, x_k) = 0], x_1, \dots, x_k)$$

Definicija 3.8. Parcijalna funkcija f je μ -rekurzivna ako se može dobiti od osnovnih rekurzivnih funkcija konačnom primenom operatora supstitucije, rekurzije i minimizacije.

Napomena 3.16. Pojam μ -rekurzivne funkcije je opštiji od pojma *primitivno rekurzivne* funkcije, zato što pored supstitucije i rekurzije dozvoljavamo i operator minimizacije. Sada pored brojačkih petlji imamo na raspolaganju i petlje sa proizvoljnim uslovom izlaska. Intuitivno, smatramo da μ -rekurzivne funkcije predstavljaju potpun formalizam kojim su opisane sve parcijalne funkcije koje se efektivno mogu izračunati.

Napomena 3.17. Često ćemo μ -rekurzivne funkcije kraće nazivati samo *rekurzivne funkcije*.

Primer 3.21. Setimo se funkcije *div* iz primera 3.20. Ovu funkciju možemo predstaviti i na sledeći način:

$$\text{div}(x, y) = \mu q[\text{le}(\text{mul}(S(q), y), x) = 0]$$

odnosno, u konstruktivnom obliku:

$$\text{div} = \mu(\text{Sub}(\text{le}, \text{Sub}(\text{mul}, \text{Sub}(S, P_3^1), P_3^3), P_3^2))$$

Drugim rečima, vrednost $\text{div}(x, y)$ je najmanje q takvo da je vrednost izraza $\text{le}(\text{mul}(S(q), y), x)$ jednaka nuli, tj. najmanje q da je uslov $(q + 1) \cdot y \leq x$ logički netačan.

Napomena 3.18. Ponekad ćemo koristiti i oznaku:

$$\mu y[g(y, x_1, \dots, x_k) = n]$$

što znači „*najmanje y takvo da je g(y, x₁, ..., x_k) jednako n*“. Ovo proširenje notacije nije suštinsko, jer ovo uvek možemo ekvivalentno zapisati ako:

$$\mu y[\text{not}(\text{eq}(g(y, x_1, \dots, x_k), n)) = 0]$$

Na sličan način, možemo koristiti i druge relacione operatore u uslovu, npr.:

$$\mu y[g(y, x_1, \dots, x_k) \leq n]$$

uz značenje „*Najmanje y takvo da je g(y, x₁, ..., x_k) manje ili jednako n*“. Ovo je ekvivalentno sa:

$$\mu y[\text{not}(\text{le}(g(y, x_1, \dots, x_k), n)) = 0]$$

Tako smo u prethodnom primeru, umesto:

$$\text{div}(x, y) = \mu q[\text{le}(\text{mul}(S(q), y), x) = 0]$$

mogli smo da zapišemo:

$$\text{div}(x, y) = \mu q[(q + 1) \cdot y > x]$$

što je znatno čitljivije i intuitivnije („najmanje q takvo da je $(q + 1) \cdot y$ veće od x ”).

Napomena 3.19. Iz primera 3.20 i 3.21 vidimo da se ista parcijalna funkcija može predstaviti kao μ -rekurzivna funkcija na različite načine. Ovo je analogno činjenici da se isti rezultat može dobiti različitim računarskim programima. Za dve μ -rekurzivne reprezentacije kažemo da su *ekvivalentne*, ako izračunavaju istu parcijalnu funkciju.

Napomena 3.20. Sve funkcije koje smo prikazali u prethodnim primerima su bile primitivno rekurzivne. Slučaj funkcije *div* je zanimljiv jer smo nju predstavili na dva načina: prvi nije koristio minimizaciju, a drugi jeste. Ipak, funkcija *div* jeste primitivno rekurzivna, jer se *može* predstaviti bez korišćenja minimizacije. Dakle, dovoljno je da postoji jedna reprezentacija koja ne koristi minimizaciju, a koja realizuje neku funkciju, ta funkcija će biti primitivno rekurzivna. Ipak, ponekad operator minimizacije omogućava jednostavniju i intuitivniju konstrukciju.

Primer 3.22. Posmatrajmo funkciju:

$$\text{isqrt}(x) = \max\{y \mid y^2 \leq x\}$$

Ova funkcija vraća ceo deo kvadratnog korena broja x . Na primer $\text{isqrt}(36) = 6$, a $\text{isqrt}(50) = 7$. Ovu funkciju možemo predstaviti minimizacijom na sledeći način:

$$\mu y[(y + 1)^2 > x]$$

Dakle, tražimo najmanje y takvo da je $(y + 1)^2$ veće od x . Napomenimo da ova funkcija *jeste* primitivno rekurzivna, ali je njena primitivno rekurzivna formulacija znatno komplikovanija. Sa druge strane, prikazana realizacija pomoću operatora minimizacije je krajnje jednostavna i intuitivna.

Napomena 3.21. Primetimo da ako koristimo operator minimizacije, tada μ -rekurzivna funkcija više ne mora biti totalna. Naime, čak i da je funkcija $g(y, x_1, \dots, x_n)$ totalna, funkcija $\mu(g)$ to ne mora biti, jer se može dogoditi da za neko $x_1, \dots, x_n$ funkcija $g(y, x_1, \dots, x_n)$ nije jednaka nuli ni za jedno y .

Primer 3.23. Neka je data funkcija:

$$f(x, y) = \begin{cases} z, & \text{gde je } y \cdot z = x, \text{ ako takvo } z \text{ postoji} \\ -, & \text{inače} \end{cases}$$

Ova funkcija je μ -rekurzivna, jer se može predstaviti na sledeći način:

$$f(x, y) = \mu z[y \cdot z = x]$$

Ipak, ova funkcija nije primitivno rekurzivna, s obzirom da nije totalna.

Primer 3.24. Posmatrajmo funkciju:

$$\text{undef}(x) = -$$

za svako x . Ova funkcija je izračunljiva, jer se može prikazati npr. kao $\mu y[x + y + 1 = 0]$. Ova funkcija je nedefinisana za svako x . Intuitivno, odgovara programu koji sadrži beskonačnu petlju po y .

Iz prethodnog izlaganja bi se moglo pogrešno zaključiti da je μ -rekurzivna funkcija totalna ako i samo ako je primitivno rekurzivna. Ipak, ovde implikacija važi samo u jednom smeru. Naime, važi teorema.

Teorema 3.2. *Postoji totalna μ -rekurzivna funkcija koja nije primitivno rekurzivna.*

Umesto dokaza ove teoreme, navodimo jedan zanimljiv primer.

Primer 3.25. Čuvena *Akermanova funkcija*:

$$\begin{aligned} A(0, n) &= n + 1 \\ A(m + 1, 0) &= A(m, 1) \\ A(m + 1, n + 1) &= A(m, A(m + 1, n)) \end{aligned}$$

je totalna, a može se pokazati da je μ -rekurzivna, ali nije primitivno rekurzivna. Iako data definicija funkcije sadrži rekurziju, to nije primitivna rekurzija, jer se rekurzija vrši po oba argumenta, a ne samo po prvom. Naravno, postoje i drugi načini da se ova funkcija definiše, ali se može pokazati da ni jedan od njih neće biti u obliku primitivne rekurzije. Naime, važi da je:

$$\underbrace{A(m + 1, n) = A(m, A(m + 1, n - 1)) = A(m, A(m, A(m + 1, n - 2)) = \dots = A(m, A(m, \dots, A(m, A(m + 1, 0) \dots))}_{n} = \underbrace{A(m, A(m, \dots, A(m, 1) \dots))}_{n+1}$$

Intuitivno, potrebna nam je brojačka petlja sa $n + 1$ iteracija koje u sebi pozivaju funkciju A sa prvim argumentom m . Međutim, po istom principu, za izračunavanje $A(m, k)$ (za neko k) potrebna nam je petlja sa $k + 1$ iteracijom koja poziva funkciju A sa prvim argumentom $m - 1$. Dakle, imaćemo petlju u petlji. Dalje ćemo za izračunavanje funkcije $A(m - 1, k')$ (za neko k') opet imati petlju sa $k' + 1$ iteracijom koja poziva funkciju A sa prvim argumentom $m - 2$, pa ćemo imati trostruku petlju i td. Ukupan broj ugnježenih brojačkih petlji biće jednak $m + 1$. Za fiksirano m , funkcija:

$$f(n) = A(m + 1, n)$$

bi bila primitivno rekurzivna, jer bismo imali fiksirani broj ugnježenih brojačkih petlji, pa bismo kompozicijom $m + 1$ primitivno rekurzivne funkcije dobili primitivno rekurzivnu funkciju. Međutim, u Akermanovoj funkciji, prvi argument m može biti proizvoljno veliki. Otuda je nije moguće konstruisati konačnom primenom operatora primitivne rekurzije i supstitucije.

Najzad, nisu sve parcijalne funkcije μ -rekurzivne, o čemu govori sledeća teorema.

Teorema 3.3. *Postoji totalna funkcija koja nije μ -rekurzivna.*

Drugim rečima, nije moguće svaku funkciju efektivno izračunati, tj. nije moguće svaki računski problem algoritamski rešiti. Dokaz ove teoreme ostavićemo za kasnije.

3.2.2 URM programi

Pored rekurzivnih funkcija koje predstavljaju pogodnu matematičku notaciju za opisivanje algoritama, postoje i formalizmi koji su bliži realnim računarima i u većoj meri simuliraju proceduralni način razmišljanja uobičajen u programiranju. Jedan takav formalizam predstavljaju i *URM programi* koji predstavljaju apstraktnu varijantu jednog krajnje jednostavnog asemblerskog jezika.

Definicija 3.9 (URM program). Pretpostavimo da nam je na raspolaganju beskonačni niz registara $R_1, R_2, \dots$, pri čemu svaki registar može čuvati jedan prirodan broj. *URM program* (engl. *Unlimited Register Machine* (URM)) se sastoji iz konačnog niza *instrukcija* $I_1, I_2, \dots, I_r$, pri čemu je svaka od instrukcija nešto od sledećeg:

- $Z(n)$: ova instrukcija postavlja registar R_n na nulu ($R_n \leftarrow 0$)
- $S(n)$: ova instrukcija uvećava vrednost registra R_n za jedan ($R_n \leftarrow R_n + 1$)
- $T(m, n)$: ova instrukcija kopira vrednost registra R_m u registar R_n ($R_n \leftarrow R_m$)
- $J(m, n, k)$: ako su vrednosti registara R_m i R_n jednake, tada se vrši *skok* na instrukciju I_k . U suprotnom, instrukcija ne radi ništa.

Program se izvršava počev od instrukcije I_1 , pri čemu se instrukcije izvršavaju redom, osim u slučaju kada dođe do *skoka* u instrukciji J -tipa, kada se izvršavanje nastavlja od navedene instrukcije. Program se zaustavlja kada dođe do nepostojeće instrukcije (tj. instrukcije sa indeksom većim od r).

Definicija 3.10. Parcijalna funkcija f reda k koju *izračunava* URM program P definisana je na sledeći način:

- za svaku k -torku $(x_1, \dots, x_k) \in \mathbb{N}^k$, registri $R_1, \dots, R_k$ se inicijalizuju vrednostima $x_1, \dots, x_k$ redom, dok se ostali registri inicijalizuju nulom
- pokrenemo program počev od prve instrukcije
- ako se program zaustavi nakon konačno mnogo koraka, tada je vrednost funkcije $f(x_1, \dots, x_k)$ jednaka vrednosti registra R_1 na kraju izvršavanja programa
- u suprotnom, funkcija nije definisana za datu k -torku

Napomena 3.22. Jasno je da isti URM program P definiše po jednu parcijalnu funkciju za svako $k \geq 1$.

Definicija 3.11 (URM izračunljiva funkcija). Za parcijalnu funkciju f reda k kažemo da je *URM izračunljiva* ako postoji URM program koji je izračunava.

Primer 3.26. Jedan mogući program koji izračunava funkciju $add(x, y) = x + y$ bi bio sledeći:

- 1 : $J(3, 2, 5)$
- 2 : $S(1)$
- 3 : $S(3)$
- 4 : $J(1, 1, 1)$

Na početku je x u R_1 , a y u R_2 . Registar R_3 je inicijalno jednak nuli – on će biti brojač u petlji koja će se izvršavati y puta. Zato na početku petlje proveravamo da li je brojač R_3 jednak R_2 (tj. broju y). Ako jeste, skaćemo na nepostojeću instrukciju (instrukcija broj 5), čime se program završava. Ako nije, tada izvršavamo instrukciju broj 2 kojom se vrednost registra R_1 uvećava za 1 (tj. dodajemo jedan na x). Zatim uvećavamo brojač (instrukcijom 3) i skaćemo bezuslovno na početak petlje (instrukcija broj 1). Bezuslovni skok se postiže tako što upoređujemo registar R_1 sa samim sobom – time je jednakost uvek ispunjena, pa se skok uvek vrši. Kada se algoritam završi, u registru R_1 ostaje broj x uvećan y puta za 1 (tj. vrednost $x + y$).

Primer 3.27. Program u nastavku izračunava funkciju:

$$sub'(x, y) = \begin{cases} x - y, & \text{za } x \geq y \\ -, & \text{za } x < y \end{cases}$$

$$\begin{aligned} 1 : & J(1, 2, 5) \\ 2 : & S(2) \\ 3 : & S(3) \\ 4 : & J(1, 1, 1) \\ 5 : & T(3, 1) \end{aligned}$$

Registri R_1 i R_2 sadrže redom x i y . U registru R_3 biće izračunata razlika. Instrukcije 1 – 4 predstavljaju petlju u kojoj se uvećava R_2 dok ne postane jednako R_1 . Istovremeno, uvećava se i razlika u R_3 . Ako je na početku bilo $x \geq y$, tada će se nakon $x - y$ iteracija vrednost R_2 izjednačiti sa R_1 i iz petlje će se izaći, a u R_3 će se nakon izlaska iz petlje nalaziti razlika $x - y$. Ovu razliku prebacujemo u R_1 , čime se rad programa završava. Ako je na početku bilo $x < y$, tada R_2 nikada neće uvećanjem postati jednako R_1 (jer je u startu bilo veće). Zato će se petlja izvršavati zauvek, pa funkcija neće biti definisana.

Primer 3.28. Neka je data funkcija:

$$sub(x, y) = \begin{cases} x - y, & \text{za } x \geq y \\ 0, & \text{za } x < y \end{cases}$$

Ovu funkciju možemo izračunati sledećim URM programom:

$$\begin{aligned} 1 : & T(1, 3) \\ 2 : & T(2, 4) \\ 3 : & J(1, 4, 11) \\ 4 : & J(2, 3, 9) \\ 5 : & S(3) \\ 6 : & S(4) \\ 7 : & S(5) \\ 8 : & J(1, 1, 3) \\ 9 : & Z(1) \\ 10 : & J(1, 1, 12) \\ 11 : & T(5, 1) \end{aligned}$$

Ideja programa je da se u registrima R_3 i R_4 nakon k iteracija petlje nalaze, redom, vrednosti $x + k$ i $y + k$. Sama vrednost k nalaziće se u registru R_5 . Petlja je implementirana instrukcijama 3 – 8. Ako je R_1 jednako sa R_4 , znači da smo

pronašli k takvo da je $x = y + k$. Tada je k upravo razlika $x - y$, pa skaćemo na instrukciju 11 kojom se vrednost k iz R_5 kopira u R_1 , čime se program završava. U suprotnom, ako je R_2 jednako R_3 , to znači da smo pronašli k takvo da je $x + k$ jednako y . Tada je $y > x$, pa skaćemo na instrukciju 9 kojom se registar R_1 postavlja na nulu. Instrukcija 10 predstavlja beuslovni skok kojim se preskače instrukcija 11. U telu petlje uvećavamo registre R_3 , R_4 i R_5 , nakon čega bezuslovno skaćemo na početak petlje (instrukcija 8).

Napomena 3.23. Ponekad će biti zgodno da za implementaciju nekog složenijeg algoritma iskoristimo program P koji smo već imali ranije za realizaciju neke jednostavnije funkcije. U tom slučaju, biće nam potrebno da u neki program P' ugradimo program P kao njegov *potprogram*. Pritom, program P u okviru programa P' možda neće počinjati od prve instrukcije, već od instrukcije sa rednim brojem $k + 1$, za neko k . U tom slučaju, da bi program P i dalje radio korektno, biće potrebno da sve reference na instrukcije (u instrukcijama J tipa) u njemu uvećamo za k . Ovo ćemo nazivati *translacijom programa P za k instrukcija* i označavaćemo sa $P_{\rightarrow k}$.

Primer 3.29. Pretpostavimo da je data funkcija:

$$lt(x, y) = \begin{cases} 1, & \text{za } x < y \\ 0, & \text{za } x \geq y \end{cases}$$

Jasno je da je $x \geq y$ akko je $sub(y, x) = 0$. Otuda možemo iskoristiti program iz prethodnog primera. Označimo taj program sa P . Posmatrajmo program:

1 :	$T(1, 3)$
2 :	$T(2, 1)$
3 :	$T(3, 2)$
4 – 14 :	$P_{\rightarrow 3}$
15 :	$Z(2)$
16 :	$J(1, 2, 19)$
17 :	$Z(1)$
18 :	$S(1)$

Dakle, kako bismo pozvali potprogram P za argumente y i x , najpre je potrebno zameniti redosled argumenata u registrima R_1 i R_2 . Ovo se radi pomoću prve tri instrukcije koje implementiraju klasični *algoritam zamene vrednosti promenljivih*. Potprogram P smeštamo počev od instrukcije 4, pa je potrebno sve adrese u njemu translirati za 3 (jer je instrukcija koja je bila prva sada četvrta, druga je peta i td.). Nakon što se potprogram završi, u registru R_1 se nalazi vrednost koja je jednaka nuli akko je $x \geq y$. Samim tim, ako je R_1 jednak nuli, treba ga takvog i ostaviti. Ovo postizemo tako što registar R_2 postavimo na nulu, a zatim ga poredimo sa R_1 . Ako je uslov ispunjen, skaćemo na nepostojeću instrukciju 19, čime se program završava. Sa druge strane, ako R_1 nije jednak nuli, tada ga je potrebno postaviti na 1. Ovo radimo tako što ga najpre postavimo na nulu, a zatim ga uvećamo za jedan.

Primer 3.30. Funkcija $mul(x, y) = x \cdot y$ se može implementirati tako što iskoristimo program iz primera 3.26 koji izračunava zbir dva broja. Označimo taj

program sa P . Sada imamo program:

```

1 :       $T(1, 4)$ 
2 :       $T(2, 5)$ 
3 :       $J(5, 6, 13)$ 
4 :       $T(7, 1)$ 
5 :       $T(4, 2)$ 
6 – 9 :   $P_{\rightarrow 5}$ 
10 :      $T(1, 7)$ 
11 :      $S(6)$ 
12 :      $J(1, 1, 3)$ 
13 :      $T(7, 1)$ 

```

Argumenti programa x i y (koji se inicijalno nalaze u R_1 i R_2) će se čuvati u R_4 i R_5 . Registar R_6 će biti brojač u petlji, dok će R_7 akumulirati rezultat (inicijalno je 0). Na početku kopiramo x i y u R_4 i R_5 redom, a zatim ulazimo u petlju (instrukcije 3 – 12) u kojoj proveravamo da li je brojač R_6 jednak y . Ukoliko jeste izlazimo iz petlje i skačemo na instrukciju 13, dok u suprotnom ulazimo u petlju. U svakoj iteraciji petlje, potrebno je na tekuću sumu (registar R_7) dodati vrednost x iz registra R_4 . Zato ćemo R_7 i R_4 kopirati u R_1 i R_2 , kao argumente potprograma. Potprogram P je transliran za 5 instrukcija i zauzima instrukcije 6 – 9. Nakon njegovog završetka registar R_1 sadrži traženi zbir koji treba prebaciti u R_7 . Nakon toga uvećavamo brojač u R_6 , a zatim bezuslovno skačemo na početak petlje. Petlja će se završiti nakon što se izvrši y puta, tj. nakon što y puta dodamo x . Na kraju je samo potrebno izračunatu sumu iz R_7 kopirati u R_1 .

Teorema 3.4. *Funkcija je URM izračunljiva akko je μ -rekurzivna.*

Napomena 3.24. Da bismo dokazali da je svaka rekurzivna funkcija URM izračunljiva, potrebno je osnovne rekurzivne funkcije, kao i svaki od konstruktivnih koraka (*Sub*, *Rec* i μ) simulirati URM programom. Ovo nije teško, imajući u vidu intuitivna značenja ovih operatora koja su data u prethodnom odeljku. Ostavljamo čitaocu za vežbu da se u to i uveri.

Drugi smer je tehnički znatno zahtevniji, jer je potrebno proizvoljan URM program predstaviti kao μ -rekurzivnu funkciju. Ipak, to je moguće uraditi, a za detalje pogledati odgovarajuću literaturu.

3.2.3 Čerčova teza

U prethodna dva odeljka prikazali smo dva često korišćena formalizma za opis algoritama, tj. efektivno izračunljivih funkcija. Pored njih, postoje i mnogi drugi načini formalnog zasnivanja izračunljivosti, od kojih ćemo neke prikazati kasnije, u drugim kontekstima. Može se pokazati da su svi oni ekvivalentni, tj. da definišu istu klasu parcijalnih funkcija. Ipak, postavlja se pitanje da li ta klasa zaista odgovara intuitivnom pojmu algoritma o kome smo govorili u poglavlju 3.1. O tome govori sledeća:

Čerčova teza Klasa formalno izračunljivih funkcija poklapa se sa intuitivnim pojmom algoritma.

Ova teza povezuje formalni matematički pojam sa neformalnim, intuitivnim pojmom. Otuda ona nije teorema u klasičnom smislu reči i ne može se dokazivati. Ipak, Čerčova teza je nešto što je opšte prihvaćeno u računarstvu i u šta se u velikoj meri veruje.

3.3 Kodiranje

U prethodnom poglavlju smo algoritme razmatrali kao izračunljive parcijalne funkcije. To znači da smo podrazumevali da su njihovi argumenti prirodni brojevi. Sa druge strane, svesni smo da su ulazi naših algoritama u praksi često druge vrste podataka: tekstualni podaci, nizovi, grafovi, matrice, i sl. Samim tim, na prvi pogled deluje da takvi algoritmi nisu pokriveni našim formalnim pojmom izračunljivih funkcija, te da Čerčova teza u tom opštem slučaju ne važi. Na sreću, taj prvi utisak je pogrešan – ispostavlja se da se svi ti ulazni podaci mogu predstaviti prirodnim brojevima. Takva brojeva reprezentacija podatka naziva se njegovim *kôdom*. Postupak prevođenja podatka u brojeve nazivamo *kodiranje*, dok se obrnuti postupak dobijanja originalnog podatka iz njegovog kôda naziva *dekodiranje*.

Prilikom kodiranja, najpre je potrebno podatak predstaviti nizom ili uređenom torkom prirodnih brojeva. Ovo obično ne predstavlja problem:

- ako je ulazni podatak ceo broj, on se može kodirati npr. pomoću zapisa *znaka* i *apsolutne vrednosti*, tj. kao uređeni par prirodnih brojeva (znak 0 označava pozitivan, a 1 negativan broj). Alternativno, možemo koristiti zapis u *potpunom komplementu*, gde ćemo ponovo imati uređeni par brojeva: prvi broj će predstavljati broj pozicija u binarnom zapisu, a drugi će biti zapis u potpunom komplementu tumačen kao neoznačen ceo broj
- ako je ulazni podatak realni broj¹, on se obično predstavlja zapisom u *pokretnom zarezu*. Ova zapis se može kodirati pomoću tri prirodna broja koji predstavljaju redom *znak*, *mantisu* i *eksponent*
- ako je ulazni podatak niz brojeva $x_1, \dots, x_n$, onda ga možemo kodirati tako što najpre navedemo njegovu dužinu, a zatim i same elemente:

$$n, x_1, \dots, x_n$$

- ako je ulazni podatak konačan skup brojeva, tada ga možemo predstaviti rastućim nizom svojih elemenata
- ako je ulazni podatak matrica dimenzija $m \times n$:

$$\begin{array}{cccc} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{array}$$

¹Kada kažemo *realni broj*, mislimo na podatke realnog tipa u standardnim programskim jezicima (poput `float` ili `double` u C++-u). Matematički, u pitanju su racionalni brojevi, s obzirom da se uvek zapisuju sa konačno mnogo cifara.

tada se ona može predstaviti npr. tako što se najpre navedu njene dimenzije, a zatim se njene vrste poređaju u niz:

$$m, n, x_{11}, \dots, x_{1n}, x_{21}, \dots, x_{2n}, \dots, x_{m1}, \dots, x_{mn}$$

- ako je ulazni podatak tekst, tada se svaki karakter teksta kodira prirodnim brojem, pri čemu dobijamo niz brojeva. Kodiranje pojedinačnih karaktera se vrši na osnovu nekog unapred odabranog pridruživanja kôdova elemenata karakterskog skupa (poput ASCII, UTF-8, i sl.)
- graf se može predstaviti svojom *matricom povezanosti* – matricom dimenzije $n \times n$ (gde je n broj čvorova grafa), takve da je element na poziciji (i, j) jednak 1 ako je grana (i, j) prisutna u grafu, u suprotnom je jednak 0. Nakon toga se matrica može predstaviti nizom brojeva na ranije prikazan način. Alternativno, za retke grafove može biti efikasnije navesti najpre broj čvorova n i broj grana m , a zatim m parova brojeva koji predstavljaju grane
- zvučni zapis se predstavlja nizom *uzoraka* (*semplova*) koji predstavljaju brojeve koji opisuju nivo zvučnog signala u datom trenutku
- slika se predstavlja matricom *piksela* – brojeva koji opisuju boju slike u datoj tački. Ova matrica se predstavlja nizom brojeva kao što je ranije opisano
- video zapis se predstavlja nizom slika, pri čemu se svaka slika predstavlja nizom brojeva kao što je malopre opisano
- ...

Dakle, većina podataka sa kojima naši programi u praksi rade se prirodno kodiraju nizovima ili torkama prirodnih brojeva. Ipak, tehnički gledano, ako algoritam posmatramo kao parcijalnu funkciju, tada bi svaki od ulaznih argumenata trebalo da bude *jedan* broj, a ne niz brojeva. Da bismo to postigli, potrebno je da pronađemo način da proizvoljni niz ili torku prirodnih brojeva kodiramo jedinstvenim brojem. Takođe, potrebno je da omogućimo dekodiranje, tj. izdvajanje pojedinačnih elemenata polaznog niza ili torke brojeva iz odgovarajućeg kôda. Funkcija kodiranja bi trebalo da bude *bijektivna* – tj. da svakom nizu ili torci brojeva odgovara jedinstven kôd, kao i obrnuto. Najzad, i kodiranje i dekodiranje mora biti izračunljivo u formalnom smislu.

3.3.1 Kodiranje uređenih parova i torki

Uređeni par prirodnih brojeva (x, y) se može kodirati prirodnim brojem na sledeći način:

$$\pi(x, y) = 2^x \cdot (2y + 1) - 1$$

Jasno je da je funkcija π izračunljiva. Takođe, može se pokazati da je kodiranje π bijektivno. Intuitivno, $2y + 1$ predstavlja bijektivno preslikavanje skupa $\mathbb{N}$ u skup neparnih brojeva, tj. brojeva čiji binarni zapis ima 1 na najnižoj poziciji. Množenjem sa 2^x ovaj binarni zapis će dobiti x nula na najnižim pozicijama,

nakon čega će slediti binarni zapis broja $2y + 1$. Iz ovog zapisa možemo uvek jednoznačno odrediti x i y . Ipak, ovako dobijeni kôd nikada neće biti nula, tj. imaćemo bijektivno preslikavanje skupa $\mathbb{N} \times \mathbb{N}$ u skup $\mathbb{N} \setminus \{0\}$. Zbog toga na kraju oduzimamo 1, da bismo imali bijekciju $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

Dekodiranje se vrši na sledeći način:

- za izdvajanje prve komponente para iz koda c (što ćemo označavati sa $\pi_1(c)$), potrebno je najpre kôd uvećati za 1, a zatim odrediti najveće x takvo da je $\text{mod}(c+1, 2^x) = 0$. Lako se vidi da je ovo izračunljivo ($\pi_1(c) = \mu x[\text{mod}(c+1, 2^{x+1}) \neq 0]$)
- za izdvajanje druge komponente ($\pi_2(c)$), potrebno je najpre podeliti $c+1$ sa 2^x , a zatim odrediti celobrojni količnik sa 2: $\pi_2(c) = \text{div}(c+1, 2^{\pi_1(c)+1})$. Ovo je takođe izračunljivo.

Primer 3.31. Par $(2, 3)$ će biti kodiran sa $\pi(2, 3) = 2^2 \cdot (2 \cdot 3 + 1) - 1 = 27$. Sada se $\pi_1(27)$ dobija tako što u binarnom zapisu broja 28 prebrojimo nule na desnom kraju: $28 = 7 \cdot 4$, pa ima dve nule na kraju. Dalje je $\pi_2(27) = \text{div}(28, 2^3) = 3$.

Kodiranje uređenih torki možemo svesti na kodiranje uređenih parova. Na primer:

$$\sigma(x, y, z) = \pi(\pi(x, y), z)$$

Izračunljivost i bijektivnost funkcije σ slede iz izračunljivosti i bijektivnosti funkcije π . Dekodiranje se takođe svodi na dekodiranje funkcije π . Na sličan način možemo kodirati i n -torke za proizvoljno n .

Primer 3.32. Neka je dat realan broj $x = 2.5$. Binarni zapis ovog broja je 10.1 , što se u normalizovanom obliku može predstaviti kao $1.01 \cdot 2^1$. Znak ovog broja je $+$ (kodiran brojem 0), mantisa je 101 (odnosno 5 u dekadnom zapisu), a eksponent je $+1$. Najpre ćemo eksponent kodirati kao uređeni par $(0, 1)$ (znak i apsolutna vrednost). Važi da je $\pi(0, 1) = 2$. Sada je potrebno kodirati uređenu trojku $(0, 5, 2)$. Prema prethodnom, imamo:

$$\sigma(0, 5, 2) = \pi(\pi(0, 5), 2) = \pi(10, 2) = 1024 \cdot 5 - 1 = 5119$$

Primer 3.33. Pretpostavimo da naš algoritam na ulazu prihvata jedan argument koji predstavlja tačku u ravni, zadatu parom koordinata. Algoritam treba da odredi kvadrant u kom se tačka nalazi.

Tačku bismo kodirali tako što najpre njene koordinate x i y (koje su realni brojevi) kodiramo kao prirodne brojeve x' i y' (kao u prethodnom primeru). Zatim kodiramo odgovarajući uređni par brojeva (x', y') funkcijom π .

Algoritam bi interno dekodirao x' i y' , a zatim bi njihovim dekodiranjem odredio znakove brojeva x i y . Na osnovu tih znakova bi odredio kvadrant kome tačka pripada.

3.3.2 Kodiranje konačnih nizova brojeva

Ako nam ulaz predstavlja niz prirodnih brojeva proizvoljne konačne dužine, tada možemo koristiti sledeću formulu kodiranja:

$$\tau([x_1, \dots, x_k]) = 2^{x_1} + 2^{x_1+x_2+1} + \dots + 2^{x_1+\dots+x_k+(k-1)} - 1$$

Intuitivno, najpre niz $[x_1, \dots, x_k]$ transformišemo u niz $[u_1, \dots, u_k]$ takvih da su svi elementi u_i međusobno *različiti* (primetimo da to ne mora biti slučaj sa originalnim nizom). Ovo postizemo na sledeći način:

$$\begin{aligned} u_1 &= x_1 \\ u_2 &= x_1 + x_2 + 1 \\ u_3 &= x_1 + x_2 + x_3 + 2 \\ &\dots \\ u_k &= x_1 + \dots + x_k + (k-1) \end{aligned}$$

Nakon toga prosto saberemo stepene dvojke $2^{u_1} + \dots + 2^{u_k}$. Svi ovi stepeni dvojke su sada međusobno različiti, pa zbir odgovara binarnom broju u kome se jedinice nalaze na pozicijama $u_1, \dots, u_k$. Na kraju oduzimamo 1 kako bismo imali i 0 kao mogući kôd.

Dekodiranje kôda c vršimo tako što najpre odredimo binarni zapis broja $c + 1$, odakle rekonstruišemo redne brojeve pozicija $u_1, \dots, u_k$ na kojima se nalaze jedinice. Rešavanjem gornjeg trouganog sistema linearnih jednačina dobijamo originalnu sekvencu $[x_1, \dots, x_k]$. Intuitivno je jasno da je ovaj postupak izračunavanja efektivan.

Primer 3.34. Sekvenca $[2, 3, 1]$ će biti kodirana tako što najpre odredimo sekvencu $[u_1, u_2, u_3] = [2, 6, 8]$, a zatim izračunamo $2^2 + 2^6 + 2^8 - 1 = 323$. Za dekodiranje, najpre ćemo odrediti binarni zapis broja 324: to je 101000100. Iz ovoga utvrđujemo da su pozicije jedinica (brojano sa desna, počev od nule) redom 2, 6 i 8. Rešavajući linearni sistem:

$$\begin{aligned} 2 &= x_1 \\ 6 &= x_1 + x_2 + 1 \\ 8 &= x_1 + x_2 + x_3 + 2 \end{aligned}$$

dobijamo da je $x_1 = 2$, $x_2 = 6 - 1 - 2 = 3$, a $x_3 = 8 - 2 - 3 - 2 = 1$. Dakle, imamo $[x_1, x_2, x_3] = [2, 3, 1]$.

Alternativni način kodiranja konačnih nizova brojeva je primenom faktORIZACIJE. Neka je sa p_i označen i -ti prost broj (npr. $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, i td.). Sada se sekvenca $[x_1, \dots, x_k]$ može kodirati na sledeći način:

$$\tau'([x_1, \dots, x_k]) = p_1^{x_1} \cdot p_2^{x_2} \cdot \dots \cdot p_k^{x_k} - 1$$

pri čemu se ponovo oduzima 1 da bismo imali i 0 kao mogući kôd. Jasno je da će ova funkcija bijektivno preslikavati konačne sekvence prirodnih brojeva u $\mathbb{N}$, s obzirom na jednoznačnost faktORIZACIJE prirodnog broja. Takođe, funkcija τ' kao i njen inverz biće izračunljive (za to je dovoljno dokazati da je funkcija *prime(i)* koja vraća i -ti prost broj izračunljiva, što nije teško).

Primer 3.35. Posmatrajmo istu sekvencu $[2, 3, 1]$ kao i u prethodnom primeru. Imamo da je:

$$\tau'([2, 3, 1]) = 2^2 \cdot 3^3 \cdot 5^1 - 1 = 4 \cdot 27 \cdot 5 - 1 = 539$$

Dekodiranje se dobija faktORIZACIJOM broja $539 + 1 = 540$.

3.3.3 Kodiranje hijerarhijskih struktura

U računarstvu je često potrebno predstaviti hijerarhijske strukture, poput izraza, formula, dokaza i sl. Ovakve strukture se po pravilu predstavljaju *stablima*. Svaki čvor stabla ima 0 ili više *dece*. Pritom, svi čvorovi imaju tačno jednog roditelja, izuzev jednog izdvojenog čvora – *korena* stabla koji nema roditelja.

Da bismo kodirali stablo, najpre treba sadržaj svakog čvora kodirati prirodnim brojem. Npr. ako je u pitanju stablo izraza, tada čvorovi mogu sadržati operatore (npr. $+$, $-$, $\cdot$, $/$ i sl.). Svaki od operatora se kodira jedinstvenim prirodnim brojem. Ako čvor sadrži tekst, tada najpre tekst kodiramo sekvencom prirodnih brojeva (kôdovima karaktera), a zatim tu sekvencu kodiramo prirodnim brojem, kao u prethodnom paragrafu. Slično i u ostalim slučajevima.

Dakle, u nastavku možemo smatrati da je sadržaj svakog čvora stabla jedan prirodan broj. Funkciju kodiranja stabla $\rho(t)$ možemo definisati na sledeći način: posmatrajmo stablo t kao uređenu torku $t = (r, t_1, \dots, t_k)$, gde je r broj koji je sadržan u korenu, a $t_1, \dots, t_k$ su torke koje na analogan način kodiraju podstabla. Najpre ćemo rekursivno kodirati podstabla brojevima $c_i = \rho(t_i)$. Sada je još potrebno kodirati sekvencu $[r, c_1, \dots, c_k]$ funkcijom τ :

$$\rho((r, t_1, \dots, t_k)) = \tau([r, \rho(t_1), \dots, \rho(t_k)])$$

Ovo kodiranje je bijektivno, tj. svakom stablu odgovaraće jedinstven prirodan broj i obrnuto.

Primer 3.36. Posmatrajmo stablo $(5, (1, (0), (1)), (4))$. Ovo stablo ima 5 u korenu, levo podstablo ima u korenu 1 i kao svoju decu listove 0 i 1. Desno podstablo je samo list 4. Kodiranjem ovog stabla dobijamo:

$$\rho((5, (1, (0), (1)), (4))) = \tau([5, \rho((1, (0), (1))), \rho((4))])$$

Kako je:

$$\rho((1, (0), (1))) = \tau([1, \rho((0)), \rho((1))]) = \tau([1, \tau([0]), \tau([1])]) = \tau([1, 1, 2]) = 159$$

imamo da je:

$$\tau([5, \rho((1, (0), (1))), \rho((4))]) = \tau([5, 159, \tau([4])]) = \tau([5, 159, 16])$$

U pitanju je veoma veliki broj, ali svakako izračunljiv.

3.4 Enumeracija izračunljivih funkcija

3.4.1 Kodiranje URM programa

U prethodnom poglavlju smo razmatrali na koji način možemo da kodiramo različite podatke, tj. da ih predstavimo jednim prirodnim brojem. U ovom poglavlju ćemo pokušati da primenimo to znanje na same algoritme. Naime, pretpostavimo da su nam algoritmi predstavljeni URM programima. Svaki URM program je konačni niz instrukcija. Pritom, svaka instrukcija se može kodirati brojem:

- Instrukcija $Z(k)$ se kodira kao $4(k - 1)$

- Instrukcija $S(k)$ se kodira kao $4(k - 1) + 1$
- Instrukcija $T(m, n)$ se kodira kao $4\pi(m - 1, n - 1) + 2$
- Instrukcija $J(m, n, k)$ se kodira kao $4\sigma(m - 1, n - 1, k - 1) + 3$

Oduzimanje jedinice u parametrima je zato što numeracija instrukcija i registara kreće od 1. Prilikom dekodiranja, najpre se odredi tip instrukcije pomoću funkcije *mod* (ostatak pri deljenju sa 4). U slučaju instrukcija Z i S se celobrojnim deljenjem sa četiri dobija $k - 1$. Kod T instrukcije se celobrojnim deljenjem sa četiri odredi $\pi(m - 1, n - 1)$, a zatim se dekodiranjem funkcije π dobijaju m i n . Slično za J instrukcije.

Označimo opisanu funkciju kodiranja instrukcija sa ϕ_I . Program $P = I_1, \dots, I_n$ se kodira kao:

$$\phi(P) = \tau([\phi_I(I_1), \phi_I(I_2), \dots, \phi_I(I_n)])$$

Lako se vidi da je ovo kodiranje bijektivno i izračunljivo.

Primer 3.37. Setimo se programa $P = J(3, 2, 5), S(1), S(3), J(1, 1, 1)$ koji je računao zbir dva broja. Ovaj program možemo kodirati tako što najpre kodiramo njegove instrukcije:

- $\phi_I(J(3, 2, 5)) = 4\sigma(3, 2, 5) + 3 = 24189255811071$
- $\phi_I(S(1)) = 4 \cdot (1 - 1) + 1 = 1$
- $\phi_I(S(3)) = 4 \cdot (3 - 1) + 1 = 9$
- $\phi_I(J(1, 1, 1)) = 4\sigma(1, 1, 1) + 3 = 383$

Sada je:

$$\phi(P) = \tau([24189255811071, 1, 9, 383])$$

U pitanju je izuzetno veliki, ali izračunljiv broj.

Primer 3.38. Odredimo URM program čiji je kôd broj 20. Najpre ćemo odrediti sekvencu brojeva koja se funkcijom τ slika u 20. To radimo tako što broj 21 predstavimo kao zbir stepena dvojke: $21 = 2^4 + 2^2 + 2^0$. Dakle, imamo $[u_1, u_2, u_3] = [0, 2, 4]$. Otuda je u pitanju sekvenca $[x_1, x_2, x_3] = [0, 1, 1]$. Dakle, imamo program od tri instrukcije čiji su kodovi redom 0, 1 i 1. U pitanju su instrukcije $Z(0), S(0), S(0)$. Ovo je program koji izračunava broj 2.

Napomena 3.25. Na sličan, doduše tehnički složeniji, način je moguće kodirati i μ -rekurzivne funkcije. S obzirom da je rekurzivna funkcija jednoznačno zadata svojim konstruktivnim zapisom, koji se može predstaviti svojim apstraktnim stablom, logičan pristup je da se rekurzivne funkcije kodiraju po principima kodiranja stabala iz prethodnog odeljka. Ono što komplikuje situaciju je to što mnoga stabla neće predstavljati ispravne rekurzivne konstrukcije, s obzirom da će operatori u njima biti primenjeni na funkcije pogrešne arnosti. Otuda će mnogi kodovi odgovarati neispravnim konstrukcijama. Jedan način da se sa tim problemom izborimo je da kažemo da svaka neispravna rekurzivna konstrukcija predstavlja parcijalnu funkciju *undef* (tj. funkciju za koju je $undef(x) = -$; za svako x). Drugi način je da se definiše posebna funkcija kodiranja za svaku arnost. Za detalje pogledati literaturu.

3.4.2 Enumeracija URM programa

Imajući u vidu prethodno opisano kodiranje URM programa, sledi da se svi URM programi mogu efektivno poredati u niz:

$$P_0, P_1, \dots,$$

Dakle, i -ti program u nizu označavaćemo sa P_i , pri čemu ćemo broj i nazivati *kôdom* programa. Uz fiksirano kodiranje, svaki program će imati jedinstven kôd i obrnuto – svaki prirodan broj biće kôd nekog programa.

Kao što znamo, svaki URM program definiše po jednu parcijalnu funkciju za svaku arnost k . Funkciju arnosti k koju definiše program P_i označavaćemo sa $\phi_i^{(k)}$ (ukoliko je arnost 1, tada ćemo kraće pisati samo ϕ_i). Pritom, setimo se da postoje različiti URM programi koji izračunavaju istu parcijalnu funkciju. Otuda se može dogoditi da je $\phi_i^{(k)} = \phi_j^{(k)}$ za $i \neq j$. Drugim rečima, u nizu funkcija:

$$\phi_1^{(k)}, \phi_2^{(k)}, \dots$$

može biti jednakih funkcija, izračunatih na različite načine.

Napomena 3.26. Mogućnost enumeracije svih URM programa ima neke vrlo zanimljive posledice. Na primer, posmatrajmo funkciju definisanu na sledeći način:

$$f(x) = \begin{cases} \phi_x(x) + 1, & \text{ako je } \phi_x(x) \text{ definisano} \\ 0, & \text{inače} \end{cases}$$

Ova funkcija je totalna, jer je definisana za svako x . Međutim, ako bi ova funkcija bila izračunljiva, tada bi postojao URM program P_e koji je izračunava, tj. važno bi:

$$f(x) = \phi_e(x)$$

za svako x . Posebno, to bi značilo da je $\phi_e(e) = \phi_e(e) + 1$, s obzirom da je f totalna, pa je $f(e) = \phi_e(e)$ definisano. Ovo nije moguće. Otuda sledi da f nije izračunljiva. Dakle, upravo smo dokazali sledeću teoremu.

Teorema 3.5. *Postoji totalna funkcija koja nije URM izračunljiva.*

Imajući u vidu ekvivalentnost URM programa i μ -rekurzivnih funkcija (teorema 3.4), odatle sledi i da postoji totalna funkcija koja nije μ -rekurzivna. Time smo dokazali i teoremu 3.3.

Napomena 3.27. Postupak primenjen u prethodnom dokazu je poznat i kao *postupak dijagonalizacije*. Mnoge teoreme u teorijskom računarstvu i matematici se dokazuju na taj način (npr. neprebrojnost skupa realnih brojeva se tako dokazuje). Takođe, ovaj postupak je „odgovoran” i za otkrivanje mnogih paradoksa u teoriji skupova u 19. veku.

3.4.3 Univerzalni programi i funkcije

Posmatrajmo funkciju:

$$U^{(k)}(e, x_1, \dots, x_k) = \phi_e^{(k)}(x_1, \dots, x_k)$$

Intuitivno, ova funkcija najpre dekodira svoj prvi argument i odredi program P_e . Zatim izvršava program P_e nad ulazima $x_1, \dots, x_k$. Ovakvu funkciju ćemo nazivati *univerzalna funkcija*, s obzirom da ona može oponašati bilo koju URM izračunljivu funkciju, pod pretpostavkom da joj je na raspolaganju kôd odgovarajućeg programa.

Teorema 3.6. *Univerzalna funkcija $U^{(k)}$ je URM izračunljiva.*

Napomena 3.28. Dokaz ove teoreme izostavljamo zbog tehničke kompleksnosti. Ipak, ovde ćemo izneti intuitivno objašnjenje. Naime, jasno je da su funkcije kodiranja i dekodiranja programa URM izračunljive (ovo možemo zaključiti i na osnovu Čerčove teze, jer su svakako intuitivno izračunljive). Dakle, program P koji implementira funkciju $U^{(k)}$ bi mogao da najpre na osnovu kôda e odredi niz $[c_1, \dots, c_k]$ kôdova instrukcija programa P_e . Takođe, svaki kôd c_i možemo dekodirati i dobiti konkretnu instrukciju, koja je određena svojim tipom i svojim parametrima. Ključni problem je na koji način simulirati njihovo dejstvo? Na primer, recimo da smo mi dekodirali prvi kôd c_1 i dobili instrukciju $Z(k)$ za neko k . Naš program bi znao da je instrukcija Z tipa, a njen dekodirani parametar k bi bio sačuvan u nekom registru. Da bismo izvršili ovu instrukciju, potrebno je anulirati registar R_k . Tu nailazimo na problem: URM instrukcije isključivo koriste *direktno adresiranje*: ako želimo da anuliramo neki registar, moramo unapred (u fazi pisanja programa) znati koji registar želimo da anuliramo. Ne postoji mogućnost da anuliramo registar čiji se redni broj nalazi u nekom drugom registru (ovo bi bilo *indirektno adresiranje*, što nije moguće).

Rešenje za ovaj problem je u korišćenju *virtuelizacije*. Dekodirani program se neće izvršavati nad stvarnim registrima $R_1, R_2, \dots$ naše UR mašine. Umesto toga, imaćemo *virtuelnu* UR mašinu čije registre ćemo označavati sa $R'_1, R'_2, \dots$. Naš program P koji izračunava funkciju U će morati da održava *stanje* ove virtuelne mašine. To stanje je određeno indeksom tekuće instrukcije (*programski brojač*) i vrednostima registara. Pritom, u svakom trenutku izvršavanja URM programa, samo konačno mnogo registara su različiti od nule. Otuda se kodiranje stanja virtuelne mašine svodi na kodiranje konačne sekvence brojeva. Na primer, neka su u nekom trenutku svi registri virtuelne mašine počev od R'_{s+1} pa na dalje jednaki nuli. Tada sekvenca $[k, r_1, \dots, r_s]$ gde je k indeks tekuće instrukcije, a $r_1, \dots, r_s$ vrednosti registara $R'_1, \dots, R'_s$ određuje stanje virtuelnog UR računara. Ovu sekvencu možemo kodirati prirodnim brojem na uobičajen način i čuvati je u (stvarnom) registru R_1 . Naš program P bi mogao da interpretira instrukciju $Z(k)$ tako što dekodira stanje u registru R_1 , postavi vrednost registra r_k u dobijenoj sekvenci na nulu, a zatim ponovo kodira novo stanje i sačuva u R_1 . Slično važi i za ostale tipove instrukcija. Intuitivno, ovaj postupak je sada izračunljiv (čitalac bi bez većih problema mogao da napravi program u npr. C++-u koji simulira rad ovakve virtuelne URM mašine). Na osnovu Čerčove teze, svaki intuitivni algoritam se može predstaviti, između ostalog i URM programom. Otuda je funkcija $U^{(k)}$ URM izračunljiva.

Napomena 3.29. Izračunljivost funkcije $U^{(k)}$ je od izuzetnog teorijskog i praktičnog značaja. Program koji izračunava funkciju $U^{(k)}$ može simulirati rad proizvoljnog algoritma nad datim podacima. Ovo je teorijski osnov za pojam *interpretacije*. Naime, u teoriji programskih jezika, *interpretator* je program koji simulira rad drugih programa za zadati ulaz. Na primer, interpretator programskog jezika *Python* na ulazu ima *Python* program, kao i ulazne podatke nad

kojima taj program treba izvršiti. Interpretator tumači (dekodira) naredbe *Python* programa i izvršava njihov efekat nad datim podacima na računaru. Dakle, interpretator je program koji može izvršavati bilo koji algoritam napisan u programskom jeziku *Python*, tj. ponaša se upravo kao univerzalna funkcija.

Kao što znamo, interpretacija je na realnim računarima prisutna i na hardverskom nivou – moderni procesori su zapravo hardverski interpretatori za programe napisane na odgovarajućem *mašinskom jeziku*. Kontrolna jedinica procesora dohvata jednu po jednu instrukciju programa i simulira njen efekat uz pomoć aritmetičko logičke jedinice i registara procesora. Dakle, procesor može izvršavati bilo koji algoritam kodiran na mašinskom jeziku. Otuda je univerzalna funkcije $U^{(k)}$ teorijski osnov i za *računare sa uskladištenim programom* – program (tj. njegov kôd) se nalazi u memoriji i dekodira se od strane hardverskog interpretatora da bi se izvršavao. Sa druge strane, *računari sa fiksiranim programom* odgovaraju konkretnom, fiksiranom URM programu P – samo podaci se nalaze u memoriji, dok je program koji se nad njima izvršava fiksiran.

3.4.4 $s - m - n$ teorema

Teorema 3.7. *Za svako m i n postoji totalna izračunljiva funkcija s_n^m reda $m + 1$ takva da važi:*

$$\phi_e^{(m+n)}(x_1, \dots, x_{m+n}) = \phi_{s_n^m(e, x_1, \dots, x_m)}^{(n)}(x_{m+1}, \dots, x_{m+n})$$

za svako $e, x_1, \dots, x_{m+n}$.

Napomena 3.30. Prethodna teorema poznata je i kao $s - m - n$ teorema. Intuitivno, ona tvrdi da kada u nekoj izračunljivoj funkciji reda $m + n$ fiksiramo prvih m parametara, dobijamo funkciju po preostalim n parametara, i ta funkcija je takođe izračunljiva. Štaviše, moguće je efektivno odrediti program koji je izračunava (tj. moguće je efektivno izračunati njegov kôd). Taj kôd se može odrediti totalnom efektivno izračunljivom funkcijom s_n^m na osnovu vrednosti prvih m parametara.

Na primer, posmatrajmo program:

```
fun  f(x, y)
{
    return  x + y;
}
```

U pitanju je program koji izračunava zbir svoja dva argumenta. Ako bismo fiksirali da je $x = 3$, dobili bismo program:

```
fun  g(y)
{
    return  3 + y;
}
```

Dakle, ako fiksiramo prvi argument, dobijena funkcija po drugom argumentu je i dalje izračunljiva, a program koji je izračunava možemo efektivno odrediti u zavisnosti od izabrane vrednosti parametra x .

Dakle, suština je da je moguće na algoritamski način odrediti kako se transformiše program kada fiksiramo neke od parametara polaznog programa.

Napomena 3.31. U teoriji funkcionalnih programskih jezika postoji nešto što se zove *karing* (engl. *currying*):

$$f(x, y) = f(x)(y)$$

Dakle, da bismo izračunali funkciju f reda 2 za ulaz (x, y) , možemo najpre *parcijalno primeniti* funkciju f nad prvim parametrom, čime dobijamo *novu funkciju* $f(x)$ reda 1, koju dalje primenjujemo na preostali parametar y . Pritom, $s - m - n$ teorema nam daje teorijsku podlogu za ovu parcijalnu primenu – ona tvrdi da mi efektivno možemo da odredimo funkciju $f(x)$ polazeći od funkcije f na osnovu vrednosti prvog parametra x .

Termin *karing* je nastao u čast američkog naučnika Haskela Karija (engl. *Haskell Curry*) koji je dao neizmeran doprinos razvoju teorijskog računarstva. Programski jezik *haskell* je takođe dobio ime po njemu.

Napomena 3.32. Da bismo dokazali $s - m - n$ teoremu, možemo razmišljati u terminima URM programa. Naime, pretpostavimo da imamo program P_e koji izračunava funkciju ϕ_e^{m+n} . Neka su nam vrednosti $x_1, \dots, x_m$ fiksirane. Potrebno je konstruisati program koji izračunava funkciju g koja nastaje od ϕ_e^{m+n} fiksiranjem vrednosti prvih m argumenata. Takva funkcija bi kao argumente imala $x_{m+1}, \dots, x_{m+n}$ i oni bi, po pretpostavci, trebalo da budu smešteni u prvih n registara $R_1, \dots, R_n$. Odgovarajući program P' bi trebalo da najpre premesti ove argumente u registre $R_{m+1}, \dots, R_{m+n}$, a da zatim u $R_1, \dots, R_m$ upiše vrednosti $x_1, \dots, x_n$. Nakon toga bi trebalo da nastavi sa izvršavanjem programa P_e . Dakle, program P' bi izgledao ovako:

$$\begin{array}{l} T(n, m+n) \\ T(n-1, m+n-1) \\ \dots \\ T(1, m+1) \\ Z(1) \\ S(1) \left. \begin{array}{l} \dots \\ S(1) \end{array} \right\} x_1 \text{ puta} \\ Z(2) \\ S(2) \left. \begin{array}{l} \dots \\ S(2) \end{array} \right\} x_2 \text{ puta} \\ \dots \\ Z(m) \\ S(m) \left. \begin{array}{l} \dots \\ S(m) \end{array} \right\} x_m \text{ puta} \\ P_e \end{array}$$

Kôd ovog programa možemo efektivno odrediti na osnovu e i $x_1, \dots, x_m$: najpre dekodiramo e i dobijemo program P_e , zatim ga transformišemo na gore opisani način i onda dobijeni program ponovo kodiramo. Dobijeni rezultat je upravo ono što vraća funkcije $s_n^m(e, x_1, \dots, x_m)$. Iz intuitivne izračunljivosti opisanog postupka sledi i formalna izračunljivost funkcije s_n^m , prema Čerčovoj tezi.

Teorema 3.8. *Neka je data totalna izračunljiva funkcija f reda 1. Tada postoji neko $n \in \mathbb{N}$ za koje važi:*

$$\phi_{f(n)} = \phi_n$$

Dokaz. Posmatrajmo funkciju g reda 2 definisanu na sledeći način:

$$g(x, y) = \phi_{f(\phi_x(x))}(y)$$

Ova funkcija je intuitivno izračunljiva: treba dekodirati broj x i dobiti program P_x , zatim taj program treba primeniti na x . Kako je f izračunljiva, možemo izračunati $f(\phi_x(x))$ a zatim dekodirati dobijeni kôd i dobiti program $P_{f(\phi_x(x))}$. Dobijeni program treba zatim primeniti na y . Prema Čerčovoj tezi, sledi i formalna izračunljivost funkcije g . Samim tim, prema $s - m - n$ teoremi, postoji totalna izračunljiva funkcija s reda 1 takva da je:

$$g(x, y) = \phi_{s(x)}(y)$$

za svako x . Otuda imamo:

$$\phi_{s(x)}(y) = \phi_{f(\phi_x(x))}(y)$$

Ovo važi za svako x i y . Sa druge strane, kako je s izračunljiva, tada postoji neko e takvo da je $s = \phi_e$. Otuda je:

$$\phi_{\phi_e(x)}(y) = \phi_{f(\phi_x(x))}(y)$$

Ako u gornjoj jednakosti stavimo $x = e$, dobijamo:

$$\phi_{\phi_e(e)}(y) = \phi_{f(\phi_e(e))}(y)$$

za svako y . Otuda za $n = \phi_e(e)$ važi:

$$\phi_n(y) = \phi_{f(n)}(y)$$

za svako y , tj. važi:

$$\phi_n = \phi_{f(n)}$$

□

Napomena 3.33. Opisana teorema je poznata i kao *teorema o fiksnoj tački*. Intuicija je sledeća: funkcija f preslikava jedan kôd programa u drugi: $x \mapsto f(x)$. Otuda preslikavanje $\phi^{-1} \circ f \circ \phi$ transformiše program P_e u program $P_{f(e)}$. Dakle, imamo algoritam koji transformiše jedan program u drugi. Na primer, posmatrajmo sledeću transformaciju:

$$g \mapsto \text{fun } (x) \{ \begin{array}{l} \text{if}(x = 0) \text{ return } 1; \\ \text{return } x \cdot g(x - 1); \end{array} \}$$

Ovom transformacijom se funkcija g reda 1 transformiše u drugu funkciju reda 1 opisanu datim pseudokodom (koji iznutra poziva g , pa samim tim zavisi od g). Teorema o fiksnoj tački tvrdi da će za neku pogodno odabranu izračunljivu funkciju h reda 1 važiti:

$$h = fun \quad (x) \\ \{ \\ \quad if(x = 0) \quad return \quad 1; \\ \quad return \quad x \cdot h(x - 1); \\ \}$$

Dakle, kada transformišemo h na opisani način, ponovo dobijamo funkciju h . Drugim rečima, ova transformacija svodi h na samu sebe. Ovo opravdava definisanje funkcija pomoću rekursije, čak i u formalizmima koji nisu sami po sebi zasnovani na rekursiji (poput URM programa). Dakle, kada zadamo funkciju rekursivno, možemo da tvrdimo da je ona izračunljiva, na osnovu prethodne teoreme. Zato se ova teorema često naziva i *teoremom o rekursiji*.

Glava 4

Problemi odlučivanja

Problem odlučivanja (engl. *decision problem*) je problem kod koga se za svaku instancu kao odgovor očekuje „da” ili „ne”. Dakle, u pitanju su problemi kod kojih je potrebno odrediti da li ulazna instanca ima neko svojstvo ili zadovoljava neku relaciju.

Primer 4.1. Neki primeri problema odlučivanja:

- Da li je dati broj x paran?
- Da li je dati broj x veći od datog broja y ?
- Da li se dati broj x nalazi u datom nizu brojeva a ?
- Da li je data iskazna formula F zadovoljiva?
- Da li se dati program P zaustavlja za dati ulaz x ?
- Da li postoji put u grafu G koji povezuje dva data čvora u i v ?
- Da li u grafu G postoji ciklus?
- ...

Kod svih ovih problema, očekuje se odgovor „da” ili „ne” na izlazu.

U terminima formalne izračunljivosti, problemi odlučivanja predstavljaju *relacije* nad $\mathbb{N}$. Ako k -torka $(x_1, \dots, x_k) \in \mathbb{N}^k$ zadovoljava relaciju ρ arnosti k nad $\mathbb{N}$, tada ćemo pisati $\rho(x_1, \dots, x_k)$, u suprotnom, pisaćemo $\neg\rho(x_1, \dots, x_k)$. Intuitivno, torke koje zadovoljavaju relaciju odgovaraju instancama problema odlučivanja za koje je odgovor potvrđan, kao i obratno. U nastavku teksta, za termine *relacija* i *problem odlučivanja* smatraćemo da imaju isto značenje, pri čemu ćemo termin *relacija* koristiti pre svega u formalnim razmatranjima, dok će termin *problem odlučivanja* pretežno biti korišćen u intuitivnim objašnjenjima.

4.1 Odlučivost

Definicija 4.1. Neka je data proizvoljna relacija ρ arnosti k nad $\mathbb{N}$. Funkcija:

$$\chi_\rho(x_1, \dots, x_k) = \begin{cases} 1, & \text{ako } \rho(x_1, \dots, x_k) \\ 0, & \text{u suprotnom} \end{cases}$$

se naziva *karakteristična funkcija* relacije ρ .

Ključno pitanje je da li je za datu relaciju ρ njena karakteristična funkcija izračunljiva? Ovo nas vodi ka sledećoj definiciji.

Definicija 4.2. Za relaciju ρ kažemo da je *odlučiva* (engl. *decidable*), ako je njena karakteristična funkcija izračunljiva.

Napomena 4.1. Intuitivno, problem je odlučiv ako postoji efektivan postupak (algoritam) kojim se za ulaznu instancu utvrđuje da li je odgovor „da” ili „ne”. Takav algoritam se naziva i *procedura odlučivanja* (engl. *decision procedure*) za dati problem.

Primer 4.2. Primeri odlučivih problema su:

- Da li je dati broj x paran?
- Da li je broj x deljiv datim brojem y ?
- Da li je dati broj x veći od datog broja y ?
- Da li je dati broj x prost?
- Da li je dati graf G povezan?
- Da li u datom nizu a postoji element jednak datom broju x ?
- ...

Za neke od ovih problema smo ranije konstruisali μ -rekurzivne funkcije koje predstavljaju njihove karakteristične funkcije. Za neke druge možemo osmisлити programe na višim programskim jezicima (poput C++-a) koji bi predstavljali odgovarajuće procedure odlučivanja. Na osnovu Čerčove teze, odgovarajuće karakteristične funkcije su i formalno izračunljive.

Napomena 4.2. Složenije relacije možemo konstruisati od postojećih primenom logičkih veznika. Na primer, ako je ρ relacija, tada je i $\neg\rho$ takođe relacija koju zadovoljavaju tačno one torke koje *ne* zadovoljavaju relaciju ρ . Ovu relaciju nazivamo *komplementarnom relacijom* (ili *komplementarnim problemom*) relacije (problema) ρ . Slično, ako su ρ_1 i ρ_2 relacije arnosti k , tada su to i $\rho_1 \wedge \rho_2$ i $\rho_1 \vee \rho_2$, uz značenja koja odgovaraju uobičajenoj semantici konjunkcije i disjunkcije. Najzad, ako je $\rho(y, x_1, \dots, x_k)$ relacija arnosti $k+1$, tada su $\exists y. \rho(y, x_1, \dots, x_k)$ i $\forall y. \rho(y, x_1, \dots, x_k)$ relacije arnosti k , opet sa značenjem koje odgovara standardnoj semantici koju kvantifikatori imaju u logici.

Teorema 4.1. *Ako je relacija ρ odlučiva, tada je i $\neg\rho$ odlučiva. Ako su ρ_1 i ρ_2 odlučive, tada su i $\rho_1 \wedge \rho_2$ i $\rho_1 \vee \rho_2$ odlučive.*

Dokaz. Imamo da je $\chi_{\neg\rho}(x_1, \dots, x_k) = 1 - \chi_\rho(x_1, \dots, x_k)$ pa je $\neg\rho$ odlučiva. Slično, imamo da je $\chi_{\rho_1 \wedge \rho_2}(x_1, \dots, x_k) = \chi_{\rho_1}(x_1, \dots, x_k) \cdot \chi_{\rho_2}(x_1, \dots, x_k)$, pa je i $\rho_1 \wedge \rho_2$ odlučiva. Najzad, važi $\chi_{\rho_1 \vee \rho_2}(x_1, \dots, x_k) = \max(\chi_{\rho_1}(x_1, \dots, x_k), \chi_{\rho_2}(x_1, \dots, x_k))$, pa je i $\rho_1 \vee \rho_2$ odlučiva. $\square$

Napomena 4.3. Primetimo da iz odlučivosti relacije $\neg\rho$ sledi odlučivost relacije ρ (dakle, važi i obrnuta implikacija). Zaista, važi da je $\rho = \neg\neg\rho$, odakle zaključak sledi iz prethodne teoreme.

Napomena 4.4. Primetimo da ako je $\rho(y, x_1, \dots, x_k)$ odlučiva relacija, tada $\exists y. \rho(y, x_1, \dots, x_k)$ i $\forall y. \rho(y, x_1, \dots, x_k)$ ne moraju biti odlučive relacije. Primer ćemo videti kasnije.

Primer 4.3. Posmatrajmo sledeći problem:

- Da li se URM program P_x zaustavlja za ulaz y ?

Karakteristična funkcija ovog problema je:

$$f(x, y) = \begin{cases} 1, & \text{ako } \phi_x(y) \neq - \\ 0, & \text{u suprotnom} \end{cases}$$

Ako bi ova funkcija bila izračunljiva, tada bi bila izračunljiva i sledeća funkcija:

$$g(x) = \begin{cases} -, & \text{ako } \phi_x(x) \neq - \\ 0, & \text{u suprotnom} \end{cases}$$

Intuitivno, $g(x)$ možemo izračunati tako što najpre izračunamo $f(x, x)$, a zatim, ako je $f(x, x) = 1$, uđemo u beskonačnu petlju. Međutim, ovo znači da postoji neko n takvo da je $g = \phi_n$. Sada je $g(n) = \phi_n(n)$ pa imamo da je $\phi_n(n) = - \Leftrightarrow \phi_n(n) \neq -$. Iz ove kontradikcije sledi da funkcija f nije izračunljiva.

Napomena 4.5. Problem iz prethodnog primera poznat je i kao *problem zaustavljanja* (engl. *halting problem*). On nam govori da postoje fundamentalni problemi vezani za teoriju izračunljivosti („da li se dati program zaustavlja za dati ulaz?“) koji nisu odlučivi. Neodlučivost problema zaustavljanja dokazao je Alan Turing i tako postavio formalne granice izračunljivosti još i pre nego što su elektronski računari zvanično nastali.

Napomena 4.6. Možemo razmatrati i specijalne slučajeve problema zaustavljanja, poput:

- Da li program P_x staje za ulaz x ?
- Da li program P_x staje za fiksirani ulaz c ?

Dakle, za razliku od opšteg problema zaustavljanja gde na ulazu imamo dve vrednosti – kôd programa x i ulaz programa y , ovde imamo samo kôd programa x na ulazu. U prvom slučaju se pitamo da li program P_x staje kada mu je na ulazu njegov sopstveni kôd, dok u drugom slučaju imamo unapred fiksirani broj c i pitamo se da li program sa kôdom x staje za taj ulaz. Ponekad specijalni slučaj nekog problema može biti odlučiv, čak i kada opšti problem nije. Ipak, u ovom slučaju se može pokazati da su i ova dva specijalna slučaja problema zaustavljanja takođe neodlučiva. Dokazi se izvode vrlo slično kao i kod opšteg problema zaustavljanja, pa to ostavljamo čitaocu za vežbu.

Primer 4.4. Neka relacija $\rho(x, y, z)$ ima značenje: „program P_x se za ulaz y zaustavlja u najviše z koraka“. Ovaj problem je odlučiv. Intuitivno, treba dekodirati kôd x i dobiti program P_x , a zatim izvršiti najviše z njegovih koraka, kako bi se utvrdilo da li se program zaustavlja u tom broju koraka. Sa druge strane, relacija $\exists z. \rho(x, y, z)$ odgovara problemu zaustavljanja („da li postoji z takvo da se program P_x zaustavlja za ulaz y u najviše z koraka?“). Dakle, kao što smo ranije napomenuli, relacija $\exists z. \rho(x, y, z)$ ne mora biti odlučiva čak i ako $\rho(x, y, z)$ to jeste. Dalje, relacija $\neg \rho(x, y, z)$ je takođe odlučiva, a relacija

$\forall y. \neg \rho(x, y, z)$ to nije. Zaista, ako bi ova relacija bila odlučiva, onda bi to bila i relacija $\neg \forall y. \neg \rho(x, y, z)$, koja je ekvivalentna sa $\exists z. \rho(x, y, z)$, za koju smo dokazali da nije odlučiva. Dakle, univerzalni kvantifikator takođe može od odlučive relacije kreirati neodlučivu.

Primer 4.5. Za razliku od obične („neograničene”) kvantifikacije, *ograničena* kvantifikacija $\exists y < z. \rho(y, x_1, \dots, x_k)$ i $\forall y < z. \rho(y, x_1, \dots, x_k)$ od odlučivih relacija kreira nove odlučive relacije. Naime, neka je $\chi_\rho(y, x_1, \dots, x_k)$ karakteristična funkcija relacije ρ . Ako je ona izračunljiva, tada možemo posmatrati funkciju:

$$\begin{aligned} f(0, x_1, \dots, x_k) &= 0 \\ f(z+1, x_1, \dots, x_k) &= \max(\chi_\rho(z+1, x_1, \dots, x_k), f(z, x_1, \dots, x_k)) \end{aligned}$$

Funkcija f je dobijena primitivnom rekurzijom od χ_ρ , a predstavlja karakterističnu funkciju relacije $\exists y < z. \rho(y, x_1, \dots, x_k)$. Slično, funkcija:

$$\begin{aligned} g(0, x_1, \dots, x_k) &= 1 \\ g(z+1, x_1, \dots, x_k) &= \chi_\rho(z+1, x_1, \dots, x_k) \cdot f(z, x_1, \dots, x_k) \end{aligned}$$

predstavlja karakterističnu funkciju relacije $\forall y < z. \rho(y, x_1, \dots, x_k)$. Odavde sledi odlučivost relacija dobijenih ograničenom kvantifikacijom.

Primer 4.6. Posmatrajmo sledeći problem:

- Za dati prirodan broj x , da li je funkcija ϕ_x totalna?

Karakteristična funkcija odgovarajuće relacije je:

$$f(x) = \begin{cases} 1, & \text{ako je } \phi_x \text{ totalna} \\ 0, & \text{u suprotnom} \end{cases}$$

Ako je f izračunljiva, onda je izračunljiva i funkcija:

$$g(x) = \begin{cases} \phi_x(x) + 1, & \text{ako je } \phi_x \text{ totalna} \\ 0, & \text{u suprotnom} \end{cases}$$

Funkcija g je očigledno totalna, a njeno izračunavanje se svodi na određivanje da li je ϕ_x totalna (izračunavanjem funkcije f) i ako jeste, izračunavanje vrednosti $\phi_x(x) + 1$. Kako je g izračunljiva, važi da je $g = \phi_n$ za neko n . Kako je g totalna, važi $g(n) = \phi_n(n) + 1$. Dakle, imamo daje $\phi_n(n) = \phi_n(n) + 1$, što nije moguće. Iz ove kontradikcije sledi da f nije izračunljiva, pa dati problem nije odlučiv. Dakle, ne možemo za proizvoljan dati program odlučiti da li se zaustavlja za svaki ulaz.

Teorema 4.2 (Rajsova teorema). *Neka je $\mathcal{F}$ proizvoljni pravi neprazni potskup skupa svih izračunljivih funkcija reda 1. Tada je sledeći problem neodlučiv:*

- Za dato x , da li funkcija ϕ_x pripada $\mathcal{F}$?

Dokaz. Iz činjenice da je $\mathcal{F}$ pravi neprazni podskup skupa svih izračunljivih funkcija reda 1 sledi da postoje dve izračunljive funkcije ϕ_p i ϕ_q takve da je $\phi_p \in \mathcal{F}$ i $\phi_q \notin \mathcal{F}$.

Karakteristična funkcija ovog problema je:

$$f(x) = \begin{cases} 1, & \text{ako } \phi_x \in \mathcal{F} \\ 0, & \text{u suprotnom} \end{cases}$$

Ako bi ova funkcija bila izračunljiva, tada bi to bila i funkcija:

$$g(x) = \begin{cases} q, & \text{ako } \phi_x \in \mathcal{F} \\ p, & \text{u suprotnom} \end{cases}$$

Funkcija g je totalna, pa na osnovu teoreme o fiksnoj tački postoji neko n takvo da je $\phi_{g(n)} = \phi_n$. Postavlja se pitanje da li je $\phi_n \in \mathcal{F}$?

- ako je $\phi_n \in \mathcal{F}$, tada je $g(n) = q$, pa imamo da je $\phi_n = \phi_{g(n)} = \phi_q \notin \mathcal{F}$. Kontradikcija.
- ako je $\phi_n \notin \mathcal{F}$, tada je $g(n) = p$, pa imamo da je $\phi_n = \phi_{g(n)} = \phi_p \in \mathcal{F}$. Opet kontradikcija.

Dakle, u oba slučaja imamo kontradikciju. Otuda sledi da je polazna pretpostavka pogrešna, tj. funkcija f nije izračunljiva. $\square$

Napomena 4.7. Rajsova teorema je jedna od najznačajnijih „negativnih” teorema teorijskog računarstva. Ona tvrdi da ni za jedno ne-trivijalno *semantičko svojstvo* ne postoji efektivni način da se proveriti da li dati program ima to svojstvo ili ne. Pod semantičkim svojstvom podrazumevamo nešto što se odnosi na ponašanje programa u fazi izvršenja (tj. odnosi se na funkciju koju program izračunava). Dakle, radi se o *dinamičkim svojstvima* programa. Nasuprot tome, *sintaksna svojstva* programa tiču se strukture samog programa i mogu se ispitivati bez pokretanja programa (tzv. *statička svojstva*). Primeri sintaksnih svojstava bili bi:

- Da li program sadrži *while* petlju?
- Da li program ima više od 1000 linija koda?
- Da li program koristi rekurziju?
- ...

Problemi ispitivanja ovakvih sintaksnih svojstava programa su odlučivi (npr. treba proći kroz program i ispitati da li sadrži ključnu reč *while*). Sa druge strane, primeri semantičkih svojstava bili bi:

- Da li se program zaustavlja za dati ulaz c ?
- Da li se program uvek zaustavlja?
- Da li program izračunava funkciju $f(x) = x \cdot x$?
- Da li za svako x i y program izračunava vrednost koja se nalazi u intervalu $[x, y]$?
- Da li program za neki ulaz može prijaviti *grešku pristupa memoriji* (engl. *segmentation fault*)?

• ...

Prema Rajsovoj teoremi, svi ovi problemi su neodlučivi. Na primer, da bismo dokazali da ne možemo algoritamski utvrditi da li dati program izračunava vrednost $x \cdot x$, možemo posmatrati familiju funkcija $\mathcal{F} = \{f\}$, pri čemu je $f(x) = x \cdot x$ (dakle, familija koja sadrži samo tu jednu funkciju). Na osnovu Rajsove teoreme, problem $\phi_x \in \mathcal{F}$ je neodlučiv, što znači da nije moguće za proizvoljan program ispitati da li izračunava funkciju koja pripada $\mathcal{F}$, tj. da li izračunava funkciju f .

Uopšte, Rajsova teorema isključuje mogućnost da možemo da efektivno proverimo da li je implementacija nekog programa u skladu sa *specifikacijom*, odnosno da ispunjava zahteve u vezi onoga šta bi *trebalo* da radi. Na primer, nije moguće automatizovati pregledanje studentskih radova tako da se egzaktno utvrdi da li studentski programi rade ono što se traži u zadatku (moguće je testirati na ograničenom skupu ulaza, tzv. *test primera*, ali to ne garantuje potpunu korektnost).

Primer 4.7. Neka je dat program P_e . Problem:

- Da li je za dato x program P_x ekvivalentan sa P_e , tj. da li je $\phi_x = \phi_e$?

Prema Rajsovoj teoremi, ovaj problem je neodlučiv. Za to je dovoljno da posmatramo familiju $\mathcal{F} = \{\phi_e\}$. Sada je $\phi_x \in \mathcal{F}$ akko $\phi_x = \phi_e$.

Napomena 4.8. Intuicija iza prethodnog primera je sledeća: ako nastavnik napravi *referentnu implementaciju* P_e koja predstavlja rešenje ispitnog zadatka, ne postoji način da automatizuje proveru da li su studentski radovi ekvivalentni sa tom referentnom implementacijom.

Primer 4.8. Za date brojeve x i y , problem:

- da li je $\phi_x = \phi_y$?

je neodlučiv. Dakle, nije moguće algoritamski ispitati ekvivalentnost dva programa. Neodlučivost ovog problema sledi iz prethodnog primera, zato što je problem „ $\phi_x = \phi_e$?” (za unapred zadato fiksirano e) specijalni slučaj ovog problema (ako je specijalni slučaj neodlučiv, onda je opšti problem tim pre neodlučiv).

4.2 Rekurzivni skupovi

Problem odlučivanja možemo identifikovati sa skupom njegovih instanci za koje je odgovor potvrđan. Na primer, problem „Da li je broj paran?” se može identifikovati sa skupom parnih brojeva, dok se problem „Da li je dati graf povezan?” identifikuje sa skupom svih povezanih grafova. Ovo je u skladu sa uobičajenom definicijom pojma relacije u matematici – relacija ρ arnosti k nad $\mathbb{N}$ je proizvoljni *podskup* $\rho \subseteq \mathbb{N}^k$. Dakle, kada razmatramo da li neka k -torka zadovoljava neku relaciju, mi zapravo ispitujemo da li ona pripada nekom skupu k -torki koje ta relacija sadrži.

Važi i obratno: svaki skup $A \subseteq \mathbb{N}^k$ određuje jednu relaciju, tj. problem odlučivanja:

- Da li važi $(x_1, \dots, x_k) \in A$?

S tim u vezi, imamo sledeću definiciju.

Definicija 4.3. Za skup $A \subseteq \mathbb{N}^k$ kažemo da je *rekurzivan* ako je relacija $(x_1, \dots, x_k) \in A$ odlučiva, tj. ako je karakteristična funkcija skupa A :

$$\chi_A(x_1, \dots, x_k) = \begin{cases} 1, & \text{ako } (x_1, \dots, x_k) \in A \\ 0, & \text{u suprotnom} \end{cases}$$

izračunljiva.

Napomena 4.9. Intuitivno, skup je rekurzivan ako postoji efektivan metod (algoritam) kojim možemo da ispitamo da li proizvoljni element pripada tom skupu ili ne. Na primer, skup parnih brojeva je rekurzivan podskup skupa prirodnih brojeva, jer možemo efektivno proveriti da li neki prirodan broj pripada tom skupu ili ne.

Napomena 4.10. Većina ranije navedenih teorijskih rezultata se mogu formulirati u terminologiji rekurzivnih skupova. Na primer, možemo posmatrati skup $D = \{x \in \mathbb{N} \mid \phi_x(x) \neq -\}$ svih kôdova izračunljivih funkcija takvih da je $\phi_x(x)$ definisano. Ovak skup nije rekurzivan, zato što problem „ $\phi_x(x) \neq -$ ” nije odlučiv.

Napomena 4.11. Svaki konačan skup $A \subseteq \mathbb{N}^k$ je rekurzivan. Zaista, ako je $A = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ (gde je $\mathbf{x}_i = (x_{i1}, \dots, x_{ik})$), tada je $\mathbf{x} \in A$ ekvivalentno sa $\mathbf{x} = \mathbf{x}_1 \vee \mathbf{x} = \mathbf{x}_2 \vee \dots \vee \mathbf{x} = \mathbf{x}_n$. Jasno je da je vrednost ovog izraza izračunljiva, pa je skup rekurzivan.

4.3 Poluodlučivost

Definicija 4.4. *Polukarakteristična funkcija* relacije ρ arnosti k nad $\mathbb{N}$ je funkcija:

$$\bar{\chi}_\rho(x_1, \dots, x_k) = \begin{cases} 1, & \text{ako } \rho(x_1, \dots, x_k) \\ -, & \text{u suprotnom} \end{cases}$$

Za relaciju ρ kažemo da je *poluodlučiva* ako je njena polukarakteristična funkcija izračunljiva.

Napomena 4.12. Intuitivno, da bi neki problem bio poluodlučiv, potrebno je postojati efektivan postupak (algoritam) koji za sve instance za koje je odgovor „da” to potvrđuje u konačnom broju koraka, dok se za instance za koje je odgovor „ne” ne zaustavlja. Ovakav postupak se naziva i *procedura poluodlučivanja*.

Napomena 4.13. Jasno je da ako je relacija ρ odlučiva, ona je i poluodlučiva. Naime, ako je karakteristična funkcija χ_ρ relacije ρ izračunljiva, onda je i polukarakteristična funkcije $\bar{\chi}_\rho$ takođe izračunljiva. Intuitivno, ako je P program koji izračunava karakterističnu funkciju χ_ρ , tada je potrebno na njegov kraj dodati jednu beskonačnu petlju koja će se izvršavati ako je rezultat programa P jednak 0:

```

fun semi_decide(x)
{
    res = decide(x);
    if(res = 1) return res;
    while(1){ }
}

```

Obrnuto ne mora da važi, o čemu govore primeri u nastavku.

Primer 4.9. Ranije smo videli da je problem zaustavljanja neodlučiv. Međutim, problem zaustavljanja *jeste* poluodlučiv. Intuitivno, njegovu polukarakterističnu funkciju:

$$h(x, y) = \begin{cases} 1, & \text{ako } \phi_x(y) \neq - \\ -, & \text{u suprotnom} \end{cases}$$

možemo izračunati na sledeći način: dekodiramo x i odredimo URM program P_x . Zatim formiramo program $P' = P_x, Z(1), S(1)$. Ovaj program pokrenemo za ulaz y . Intuitivno, P' će najpre izvršiti program P_x , a zatim će (ako se P_x ikada završi) u registar R_1 upisati vrednost 1. Ako se P_x ne zaustavlja za y , ni program P' se neće zaustaviti. Prema Čerčovoj tezi, opisani intuitivni postupak se može pretočiti u formalni postupak izračunavanja.

Primer 4.10. Neka je $\rho(y, x_1, \dots, x_k)$ odlučiva relacija arnosti $k + 1$. Ranije smo videli da relacija $\exists y. \rho(y, x_1, \dots, x_k)$ ne mora biti odlučiva. Ipak, ova relacija će svakako biti poluodlučiva. Naime, ako je χ_ρ karakteristična funkcija funkcije ρ (koja je izračunljiva, po pretpostavci), tada je funkcija $C_1^1(\mu y[\chi_\rho(y, x_1, \dots, x_k) = 1])$ polukarakteristična funkcija relacije $\exists y. \rho(y, x_1, \dots, x_k)$. Ova funkcija je takođe izračunljiva, odakle sledi poluodlučivost relacije $\exists y. \rho(y, x_1, \dots, x_k)$. Intuitivno, u petlji ćemo uvećavati y dok ne pronađemo neku vrednost za koju važi relacija ρ . Ako takvo y ne postoji, petlja će se beskonačno izvršavati.

Primer 4.11. Dokažimo da važi i obrnuto: ako je relacija $\sigma(x_1, \dots, x_k)$ poluodlučiva, tada postoji neka odlučiva relacija $\rho(y, x_1, \dots, x_k)$ takva da je $\sigma(x_1, \dots, x_k) \Leftrightarrow \exists y. \rho(y, x_1, \dots, x_k)$. Naime, neka je P program koji izračunava polukarakterističnu funkciju relacije σ . Neka relacija $\rho(y, x_1, \dots, x_k)$ ima značenje „program P za ulaz $x_1, \dots, x_k$ se zaustavlja u najviše y koraka”. Intuitivno je jasno da je ova relacija odlučiva. Pritom, relacija $\sigma(x_1, \dots, x_k)$ je ekvivalentna sa $\exists y. \rho(y, x_1, \dots, x_k)$, s obzirom da se program P zaustavlja u konačno mnogo koraka ako i samo ako važi $\sigma(x_1, \dots, x_k)$.

Primer 4.12. Može se pokazati da je dovoljno da relacija $\rho(y, x_1, \dots, x_k)$ bude *poluodlučiva*, relacija $\exists y. \rho(y, x_1, \dots, x_k)$ će takođe biti poluodlučiva. Naime, ako je $\rho(y, x_1, \dots, x_k)$ poluodlučiva, tada prema prethodnom primeru postoji neka odlučiva relacija $\rho'(t, y, x_1, \dots, x_k)$ takva da je $\rho(y, x_1, \dots, x_k) \Leftrightarrow \exists t. \rho'(t, y, x_1, \dots, x_k)$. Sada je relacija $\exists y. \rho(y, x_1, \dots, x_k)$ ekvivalentna sa $\exists y. \exists t. \rho'(t, y, x_1, \dots, x_k)$. Posmatrajmo sada funkciju:

$$f(x_1, \dots, x_k) = C_1^1(\mu u[\chi_{\rho'}(\pi_1(u), \pi_2(u), x_1, \dots, x_k) = 1])$$

pri čemu je $\chi_{\rho'}$ karakteristična funkcija relacije ρ' (ova funkcija je izračunljiva, jer je ρ' odlučiva), a π_1 i π_2 su funkcije dekodiranja uređenih parova (odeljak 3.3.1). Ova funkcija je definisana samo za one $x_1, \dots, x_k$ za koje postoji neko u , pa samim tim i neko $t = \pi_1(u)$ i $y = \pi_2(u)$ za koje je $\rho'(t, y, x_1, \dots, x_k)$, tj. ako postoji y takvo da je $\rho(y, x_1, \dots, x_k)$. Za te vrednosti, funkcija f ima vrednost 1. Otuda, f je upravo polukarakteristična funkcija relacije $\exists y. \rho(y, x_1, \dots, x_k)$, pa je ona poluodlučiva.

Teorema 4.3 (Postova teorema). *Ako su relacije ρ i $\neg\rho$ poluodlučive, onda je ρ odlučiva.*

Dokaz. Neka su P i $\bar{P}$ programi koji izračunavaju funkcije poluodlučivanja za relacije ρ i $\neg\rho$ respektivno. Možemo napraviti program P' koji simulira rad programa P i $\bar{P}$ nad ulazom $(x_1, \dots, x_k)$ tako što naizmenično izvršava instrukcije ova dva programa sve dok se jedan od njih ne završi. Kako su relacije ρ i $\neg\rho$ poluodlučive, a za ulaz $(x_1, \dots, x_k)$ važi ili $\rho(x_1, \dots, x_k)$ ili $\neg\rho(x_1, \dots, x_k)$, u konačnom broju koraka će se završiti jedan od programa P ili $\bar{P}$ na osnovu čega ćemo odlučiti da li važi $\rho(x_1, \dots, x_k)$ ili ne. Ovaj intuitivni postupak se može pretočiti u formalni algoritam, na osnovu Čerčove teze. $\square$

Primer 4.13. Problem „ $\phi_x(y) = -$?” (tj. problem koji pita „Da li se program P_x ne zaustavlja za ulaz y ?”) nije poluodlučiv. Naime, kako znamo da je problem „ $\phi_x(y) \neq -$?” (problem zaustavljanja) poluodlučiv, iz poluodlučivosti njegovog komplementarnog problema „ $\phi_x(y) = -$?” bi, prema Postovoj teoremi, sledela odlučivost halting problema, što znamo da nije tačno. Otuda problem „ $\phi_x(y) = -$?” nije poluodlučiv.

Primer 4.14. Neka je $\rho(y, x_1, \dots, x_k)$ odlučiva relacija arnosti $k + 1$. Znamo da relacija $\forall y. \rho(y, x_1, \dots, x_k)$ ne mora biti odlučiva. Pitanje je da li mora biti poluodlučiva? Ova relacija je ekvivalentna sa $\neg \exists y. \neg \rho(y, x_1, \dots, x_k)$. Relacija $\neg \rho(y, x_1, \dots, x_k)$ je takođe odlučiva, pa je relacija $\exists y. \neg \rho(y, x_1, \dots, x_k)$ poluodlučiva, prema primeru 4.10. Ako bi i relacija $\neg \exists y. \neg \rho(y, x_1, \dots, x_k)$ bila poluodlučiva, tada bi relacija $\exists y. \neg \rho(y, x_1, \dots, x_k)$ morala biti odlučiva, prema Postovoj teoremi. Ipak, znamo da to ne mora biti slučaj. Otuda, relacija $\forall y. \rho(y, x_1, \dots, x_k)$ ne mora biti ni poluodlučiva u opštem slučaju.

Primer 4.15. Razmotrimo problem „Da li je ϕ_x totalna?”. Ovaj problem nije odlučiv (primer 4.6). Ispitajmo da li je ovaj problem poluodlučiv. Posmatrajmo sledeću funkciju:

$$f(x, y) = \begin{cases} 1, & \text{ako } P_x \text{ za ulaz } x \text{ radi duže od } y \text{ koraka} \\ -, & \text{u suprotnom} \end{cases}$$

Ova funkcija je izračunljiva. Intuitivno, potrebno je dekodirati program P_x , a zatim za ulaz x simulirati najviše prvih y koraka njegovog rada. Ako se ne završi u tom broju koraka, vraćamo 1. U suprotnom, ulazimo u beskonačnu petlju.

Kako je f izračunljiva, prema $s - m - n$ teoremi imamo da postoji totalna izračunljiva funkcija s reda 1 takva da važi:

$$f(x, y) = \phi_{s(x)}(y)$$

za svako $x, y \in \mathbb{N}$. Za svako fiksirano x , razmotrimo sada sledeća dva slučaja:

- P_x se ne zaustavlja za ulaz x . Tada će funkcija $f(x, y)$ za to x imati vrednost 1 za svako y , pa će funkcija $\phi_{s(x)}$ biti totalna.
- P_x se zaustavlja za ulaz x u y_0 koraka. Tada će funkcija $f(x, y)$ za to x imati vrednost 1 za $y < y_0$, a biće nedefinisana za $y \geq y_0$. Drugim rečima, $\phi_{s(x)}$ neće biti totalna.

Dakle, važi da je $\phi_{s(x)}$ totalna ako i samo ako $\phi_x(x) = -$. Sada bi iz poluodlučivosti problema totalnosti sledila i poluodlučivost problema „ $\phi_x(x) = -$?”. Međutim, za taj problem znamo da nije poluodlučiv (primer 4.13). Odavde sledi da problem totalnosti funkcije nije ni poluodlučiv.

Primer 4.16. Može se pokazati da ni komplementarni problem problema totalnosti (tj. problem „Da li funkcije ϕ_x nije totalna?”) nije poluodlučiv. Za to je dovoljno posmatrati funkciju:

$$f(x, y) = \begin{cases} 1, & \text{ako } y \neq 0 \\ \phi_x(x), & \text{ako } y = 0 \end{cases}$$

Ova funkcija je izračunljiva, pa po istom principu imamo totalnu funkciju s reda 1 takvu da je:

$$f(x, y) = \phi_{s(x)}(y)$$

na osnovu $s - m - n$ teoreme. Sada $\phi_{s(x)}$ nije totalna akko je $\phi(x) = -$, za šta znamo da nije ni poluodlučivo.

Napomena 4.14. Prethodna dva primera nam pokazuju da postoji problem takav da ni on ni njegov komplement nisu poluodlučivi. Ovakvi problemi su u izvesnom smislu najteži, jer nije moguće efektivno odrediti ni da li jeste ni da li nije ispunjeno dato svojstvo.

4.4 Polurekurzivni skupovi

Definicija 4.5. Skup $A \subseteq \mathbb{N}^k$ je *polurekurzivan* (ili *rekurzivno nabrojiv*) ako je njegova polukarakteristična funkcija:

$$\bar{\chi}_A(x_1, \dots, x_k) = \begin{cases} 1, & \text{ako } (x_1, \dots, x_k) \in A \\ -, & \text{u suprotnom} \end{cases}$$

izračunljiva.

Napomena 4.15. Iz prethodne definicije je jasno da je skup A polurekurzivan akko je problem „ $(x_1, \dots, x_k) \in A$?” poluodlučiv. Otuda se svi rezultati iz prethodnog poglavlja mogu primeniti kada se razmatra polurekurzivnost. Na primer, iz Postove teoreme sledi da ako su skupovi A i njegov komplement $\bar{A}$ polurekurzivni, tada je A i rekurzivan.

Primer 4.17. Iz prethodne definicije i ranijih primera sledi da je skup $\{x \in \mathbb{N} \mid \phi_x(x) \neq -\}$ polurekurzivan, dok skupovi $\{x \in \mathbb{N} \mid \phi_x(x) = -\}$ i $\{x \in \mathbb{N} \mid \phi_x \text{ je totalna}\}$ to nisu.

Jasno je da će svaki polurekurzivni skup biti domen neke izračunljive funkcije – na primer, upravo svoje polukarakteristične funkcije. Važi i obratno, o čemu govori sledeća teorema.

Teorema 4.4. Neka je data proizvoljna izračunljiva funkcija f reda k . Tada je njen domen $\text{dom}(f)$ polurekurzivan skup.

Dokaz. Funkcija:

$$g(x_1, \dots, x_k) = S(Z(f(x_1, \dots, x_k)))$$

ima vrednost 1 akko je $(x_1, \dots, x_k) \in \text{dom}(f)$, dok za $(x_1, \dots, x_k) \notin \text{dom}(f)$ nije definisana. Otuda je g upravo polukarakteristična funkcija skupa $\text{dom}(f)$. Kako je g izračunljiva, $\text{dom}(f)$ je polurekurzivan. $\square$

Iz prethodne teoreme sledi da se polurekurzivni skupovi poklapaju sa domenima izračunljivih funkcija. Na primer, u terminologiji URM programa, neki skup A je polurekurzivan akko postoji URM program koji se zaustavlja upravo za ulaze koji su elementi od A , dok se za ostale ulaze ne zaustavlja.

Teorema 4.5. *Neprazan skup $A \subseteq \mathbb{N}$ je polurekurzivan akko postoji totalna izračunljiva funkcija f reda 1 takva da je $f(\mathbb{N}) = A$ (tj. slika skupa $\mathbb{N}$ je ceo skup A).*

Dokaz. Pretpostavimo da postoji takva funkcija f . Sada se polukarakteristična funkcija skupa A može izraziti kao:

$$\bar{\chi}_A(x) = S(Z(\mu y[f(y) = x]))$$

Dakle, $\bar{\chi}_A$ je izračunljiva, pa je A polurekurzivan.

Obrnuto, pretpostavimo da je A neprazan i polurekurzivan. Neka je P program koji izračunava polukarakterističnu funkciju skupa A i neka je a neki fiksirani element od A . Sada možemo posmatrati sledeću funkciju:

$$g(x, y) = \begin{cases} x, & \text{ako se } P \text{ zaustavlja u najviše } y \text{ koraka} \\ a, & \text{u suprotnom} \end{cases}$$

Ova funkcija je izračunljiva i totalna. Pritom, važi $g(\mathbb{N}^2) = A$. Naime, za svako $x \in A$, program P će se završiti u konačnom broju koraka, pa će za neko y važiti $g(x, y) = x$. Sada možemo uzeti da je:

$$f(u) = g(\pi_1(u), \pi_2(u))$$

gde su π_1 i π_2 funkcije dekodiranja za uređene parove prirodnih brojeva (odeljak 3.3.1). Zaista, ova funkcija će biti izračunljiva i totalna i važiće $f(\mathbb{N}) = A$. $\square$

Napomena 4.16. Intuitivno, prethodna teorema nam kaže da je skup polurekurzivan akko se njegovi elementi mogu *efektivno* nabrojati, tj. poredati u niz:

$$f(0), f(1), f(2), \dots$$

Na primer, možemo efektivno nabrojati sve x takve da je $\phi_x(x) \neq -$. Sa druge strane, nije moguće efektivno nabrojati sve x takve da je ϕ_x totalna. Zbog toga se polurekurzivni skupovi često nazivaju i *rekurzivno nabrojivi skupovi*.

Dodatno, sam dokaz teoreme prikazuje zanimljiv intuitivni postupak kako se ova enumeracija može izvršiti. Potrebno je paralelno pokrenuti program P za sve ulaze x . Kako se koji program završi, taj ulaz dodajemo u niz. Na ovaj način će svi elementi skupa A pre ili kasnije biti dodati u niz. Paralelno izvršavanje programa P za različite ulaze možemo sekvencijalno simulirati na sledeći način. Ako sa $P[x, y]$ označimo y -ti korak programa P za ulaz x , tada možemo izvršavati korake u sledećem redosledu:

$$\begin{array}{l} P[0, 1] \\ P[0, 2], \quad P[1, 1] \\ P[0, 3], \quad P[1, 2], \quad P[2, 1] \\ P[0, 4], \quad P[1, 3], \quad P[2, 2], \quad P[3, 1] \\ \dots \end{array}$$

Dakle, u svakom ciklusu izvršavamo po jedan novi korak za sve do sada pokrenute programe i pokrećemo još jedan program. Na ovaj način će za svako x i za svako y biti izvršen y -ti korak $P[x, y]$ u nekom trenutku, pa će se za svako x za koje se P zaustavlja to i desiti pre ili kasnije.

4.5 Odlučivost u logici

U ovom poglavlju razmatramo najznačajnije teorijske rezultate kada je u pitanju odlučivost u iskaznoj logici i logici prvog reda. Svi rezultati biće predstavljeni u terminima intuitivne izračunljivosti, zbog čitljivosti. Ipak, treba imati u vidu da se formule, kao hijerarhijske strukture, uvek mogu kodirati brojevima. Isto važi i za sve konačne skupove formula. Samim tim, svi problemi odlučivanja i odgovarajuće (polu)karakteristične funkcije se mogu predstaviti i u formalnom obliku, kao parcijalne funkcije. Njihovu formalna izračunljivost će biti posledica Čerčove teze.

Teorema 4.6. *Problem zadovoljivosti iskazne formule je odlučiv.*

Dokaz. Odlučivost proističe iz konačnosti skupa valuacija koje treba razmotriti. Jedna moguća procedura odlučivanja bila bi metod istinitosnih tablica, kao što je razmatrano u glavi 2. $\square$

Posledica 4.1. *Problem tautologičnosti iskazne formule je odlučiv.*

Dokaz. Problem tautologičnosti formule A je ekvivalentan problemu nezadovoljivosti formule $\neg A$, što je komplementaran problem problemu zadovoljivosti formule $\neg A$. Kako je komplementaran problem odlučivog problema odlučiv, sledi da je i problem tautologičnosti formule odlučiv. $\square$

Napomena 4.17. Isti metod istinitosnih tablica se može primeniti i za odlučivanje o tautologičnosti iskazne formule. Ipak, u praksi se obično tautologičnost svodi na ispitivanje zadovoljivosti negacije, s obzirom da su za problem zadovoljivosti razvijeni efikasniji algoritmi.

Posledica 4.2. *SAT problem je odlučiv.*

Dokaz. SAT problem je specijalni slučaj zadovoljivosti iskaznih formula u KNF-u, pa je odlučiv. $\square$

Napomena 4.18. U praksi se obično opšti problem zadovoljivosti iskazne formule (pa i problem tautologičnosti) svodi na SAT problem, transformacijom polazne formule (ili njene negacije, u slučaju tautologičnosti) na KNF formulu. Ovo je zbog toga što najefikasniji postojeći algoritmi za ispitivanje zadovoljivosti rade nad formulama u KNF-u.

Posledica 4.3. *Zadovoljivost konačnog skupa iskaznih formula je odlučiv.*

Dokaz. Neka je $\Delta = \{A_1, \dots, A_n\}$ proizvoljan konačan skup iskaznih formula. Skup Δ je zadovoljiv ako postoji valuacija v koja istovremeno zadovoljava sve njegove formule. Kako je Δ konačan skup, moguće je konstruisati konjunkciju svih njegovih formula, tj. formulu $A_1 \wedge \dots \wedge A_n$. Sada valuacija v zadovoljava skup Δ akko zadovoljava ovu konjunkciju. Otuda se ispitivanje zadovoljivosti skupa Δ svodi na ispitivanje zadovoljivosti jedne formule, što je odlučivo na osnovu teoreme 4.6. $\square$

Posledica 4.4. *Ako je Δ konačan skup iskaznih formula i A proizvoljna iskazna formula, tada je problem $\Delta \models A$ odlučiv.*

Dokaz. Važi $\Delta \models A$ akko je skup $\Delta \cup \{\neg A\}$ nezadovoljiv. Kako je $\Delta \cup \{\neg A\}$ konačan skup, ispitivanje njegove (ne)zadovoljivosti je, odlučivo, prema posledici 4.3. $\square$

Situacija je znatno komplikovanija za beskonačne skupove iskaznih formula. O tome govori sledeća teorema, čiji dokaz izostavljamo, zbog složenosti. Pritom, napominjemo da u nastavku ovog poglavlja podrazumevamo da su svi beskonačni skupovi formula koje razmatramo rekursivni, tj. da je uvek moguće efektivno odrediti da li proizvoljna formula pripada tom skupu ili ne.

Teorema 4.7. *Problem zadovoljivosti beskonačnog skupa iskaznih formula je neodlučiv.*

Posledično, komplementarni problem *nezadovoljivosti* beskonačnog skupa iskaznih formula je takođe neodlučiv. Ispostavlja se da je ovaj problem poluodlučiv, o čemu govori sledeća teorema.

Teorema 4.8. *Problem nezadovoljivosti beskonačnog skupa iskaznih formula Δ (tj. pitanje „Da li je dati beskonačni skup iskaznih formula Δ nezadovoljiv?“) je poluodlučiv.*

Dokaz. Na osnovu teoreme o kompaktnosti, Δ je nezadovoljiv akko postoji njegov konačan nezadovoljiv potskup. Za svaki konačan potskup Δ' pitanje njegove (ne)zadovoljivosti je odlučivo. Pritom, kako je Δ po pretpostavci rekursivan, pa samim tim i rekursivno nabrojiv, moguće je efektivno nabrajati sve njegove elemente, pa samim tim i sve njegove konačne potskupove. Otuda je problem nezadovoljivosti skupa Δ poluodlučiv, jer je dobijen egzistencijalnom kvantifikacijom odlučivog problema (na osnovu primera 4.10). $\square$

Posledica 4.5. *Problem $\Delta \models A$, gde je Δ beskonačan skup iskaznih formula, a A je proizvoljna iskazna formula, je poluodlučiv.*

Razmotrimo sada fundamentalne semantičke probleme u logici prvog reda.

Teorema 4.9. *Problem valjanosti formule prvog reda je neodlučiv.*

Dokaz. Dokaz ćemo izvesti tako što ćemo svesti problem zaustavljanja (za koji znamo da je neodlučiv) na problem valjanosti formule prvog reda. Neka je dat proizvoljan URM program $P = I_1, \dots, I_m$. Konfiguracija programa P je data parom $([r_1, r_2, \dots], k)$, gde su $r_1, r_2, \dots$ redom vrednosti registara $R_1, R_2, \dots$, a k je indeks naredne instrukcije koju treba izvršiti. Bez ograničenja opštosti, možemo pretpostaviti da se program P može završiti jedino tako što indeks naredne instrukcije postane jednak $m + 1$, tj. kada dođe u konfiguraciju oblika $([r_1, r_2, \dots], m + 1)$. Neka je n indeks najvećeg registra koji se koristi u instrukcijama programa P .

Razmatraćemo signaturu $\mathcal{L}$ koja sadrži simbol konstante 0, funkcijski simbol S arnosti 1, predikatski simbol $=$ arnosti 2, kao i predikatski simbol R arnosti $n + 1$. Termovi $0, S(0), S(S(0)), \dots$ predstavljaju *numerales* koje ćemo kraće označavati sa $\bar{0}, \bar{1}, \bar{2}, \dots$, a koji intuitivno predstavljaju prirodne brojeve. Simbol $=$ intuitivno predstavlja jednakost nad brojevima. Ovakav smisao ovih simbola se obezbeđuje formulom A , koja predstavlja konjunkciju sledećih formula:

- $\forall x.x = x$
- $\forall xy.x = y \Rightarrow y = x$
- $\forall xyz.x = y \wedge y = z \Rightarrow x = z$
- $\forall xy.x = y \Rightarrow S(x) = S(y)$
- $\forall x_1 \dots x_n y u_1 \dots u_n z.x_1 = u_1 \wedge \dots \wedge x_n = u_n \wedge y = z \Rightarrow (R(x_1, \dots, x_n, y) \Leftrightarrow R(u_1, \dots, u_n, z))$
- $\forall x.\neg(0 = S(x))$
- $\forall xy.S(x) = S(y) \Rightarrow x = y$

Formula A obezbeđuje da relacija kojom se interpretira $=$ bude refleksivna, simetrična, tranzitivna i *kongruentna* (saglasna) sa S i R , kao i da se numerali interpretiraju različitim elementima domena.

Takođe, želimo da značenje atoma $R(x_1, \dots, x_n, y)$ bude „konfiguracija $([x_1, \dots, x_n, 0, 0, \dots], y)$ je dostižna prilikom izvršavanja programa P za ulaz a ”. Činjenica da je početna konfiguracija $([a, 0, 0, \dots], 1)$ dostižna je opisana atomičkom formulom $R(\bar{a}, \bar{0}, \dots, \bar{0}, \bar{1})$. Označimo ovu formulu sa T_0 .

Dalje, za svaku instrukciju I_k ($i = 1, \dots, m$) programa P formiramo formulu T_k na sledeći način:

- ako je $I_k = Z(r)$ ($r \leq n$), tada je T_k :

$$\forall x_1 \dots x_n. R(x_1, \dots, x_k, \dots, x_n, k) \Rightarrow R(x_1, \dots, 0, \dots, x_n, S(k))$$

- ako je $I_k = S(r)$ ($r \leq n$), tada je T_k :

$$\forall x_1 \dots x_n. R(x_1, \dots, x_k, \dots, x_n, k) \Rightarrow R(x_1, \dots, S(x_k), \dots, x_n, S(k))$$

- ako je $I_k = T(p, q)$ ($p, q \leq n$), tada je T_k :

$$\forall x_1 \dots x_n. R(x_1, \dots, x_p, \dots, x_q, \dots, x_n, k) \Rightarrow R(x_1, \dots, x_p, \dots, x_p, \dots, x_n, S(k))$$

- ako je $I_k = J(p, q, r)$, ($p, q \leq n$), tada je T_k :

$$\begin{aligned} \forall x_1 \dots x_n. \quad & R(x_1, \dots, x_p, \dots, x_q, \dots, x_n, k) \Rightarrow \\ & (x_p = x_q \Rightarrow R(x_1, \dots, x_p, \dots, x_q, \dots, x_n, r)) \wedge \\ & (\neg(x_p = x_q) \Rightarrow R(x_1, \dots, x_p, \dots, x_q, \dots, x_n, S(k))) \end{aligned}$$

Intuitivno, ove formule izvide zaključke o dostižnosti konfiguracija programa P na osnovu njegovih instrukcija. Označimo sa $T_{P,a}$ formulu:

$$A \wedge T_0 \wedge T_1 \wedge \dots \wedge T_m$$

Ova formula opisuje rad programa P za ulaz a , tj. njegove dostižne konfiguracije. Preciznije, neka je $\mathcal{D}$ proizvoljna struktura nad jezikom $\mathcal{L}$ koja zadovoljava formulu $T_{P,a}$. Tada se, zbog formule A , svi numerali $\bar{0}, \bar{1}, \bar{2}, \dots$ interpretiraju različitim elementima domena. Bez ograničenja opštosti, možemo pretpostaviti da domen strukture $\mathcal{D}$ sadrži skup $\mathbb{N}$, kao i da se numerali $\bar{0}, \bar{1}, \bar{2}, \dots$ interpretiraju odgovarajućim prirodnim brojevima $0, 1, 2, \dots$. Takođe, ako je ρ relacija

arnosti $n + 1$ kojom se interpretira simbol R , tada za svaku dostižnu konfiguraciju $([x_1, \dots, x_n, 0, 0, \dots], y)$ važi $\rho(x_1, \dots, x_n, y)$. Ovo se može dokazati indukcijom po rednom broju koraka izvršavanja programa, a na osnovu formula $T_0, T_1, \dots, T_m$. Primetimo da formula $T_{P,a}$ dopušta da i za neke nedostižne konfiguracije $([x_1, \dots, x_n, 0, \dots], y)$ važi da je $\rho(x_1, \dots, x_n, y)$. Dakle, formula $T_{P,a}$ definiše *minimalnu* relaciju ρ kojom se može interpretirati simbol R nad fiksnim domenom. Neka je $\mathcal{D}_0$ jedna takva minimalna struktura koja zadovoljava $T_{P,a}$, tj. struktura u kojoj $\rho(x_1, \dots, x_n, y)$ važi *samo* za dostižne konfiguracije.

Problem zaustavljanja programa P za ulaz a je sada ekvivalentan sa problemom valjanosti formule:

$$T_{P,a} \Rightarrow (\exists x_1 \dots x_n. R(x_1, \dots, x_n, S(m)))$$

Zaista, ako je ova formula valjana, tada i formula $\exists x_1 \dots x_n. R(x_1, \dots, x_n, S(m))$ mora biti tačna u svakoj strukturi $\mathcal{D}$ u kojoj je tačna i formula $T_{P,a}$. Specijalno, ona mora biti tačna u strukturi $\mathcal{D}_0$. To znači da postoji dostižna konfiguracija URM programa u kojoj je indeks naredne instrukcije jednak $m + 1$. Dakle, program P se zaustavlja za ulaz a .

Obrnuto, pretpostavimo da se program P zaustavlja za ulaz a . Tada postoji dostižna konfiguracija oblika $([x_1, \dots, x_n, 0, 0, \dots], m + 1)$, što znači da u svakoj strukturi $\mathcal{D}$ koja zadovoljava $T_{P,a}$ formula $R(x_1, \dots, x_n, S(m))$ mora biti tačna za neko $x_1, \dots, x_n$. Otuda u $\mathcal{D}$ važi $\exists x_1 \dots x_n. R(x_1, \dots, x_n, S(m))$. Samim tim, gornja formula je valjana.

Dakle, ako bi problem valjanosti formule prvog reda bio odlučiv, tada bi i problem zaustavljanja bio odlučiv. Zaista, za proizvoljan program P i ulaz a mogli bismo da formiramo gore opisanu formulu i da postojećom procedurom odlučivanja ispitamo njenu valjanost. Ipak, kako znamo od ranije da problem zaustavljanja nije odlučiv, sledi da ni problem valjanosti u logici prvog reda nije odlučiv. $\square$

Teorema 4.10. *Problem nezadovoljivosti formule prvog reda (tj. pitanje „da li je formula F prvog reda nezadovoljiva?“) je neodlučiv, ali jeste poluodlučiv.*

Dokaz. Opisani problem je ekvivalentan problemu valjanosti: formula F je nezadovoljiva akko je $\neg F$ valjana. Ako bi problem nezadovoljivosti bio odlučiv, tada bismo mogli ispitati valjanost proizvoljne formule F tako što utvrdimo da li je njena negacija nezadovoljiva. Otuda, problem nezadovoljivosti ne može biti odlučiv.

Sa druge strane, može se pokazati da se ispitivanje (ne)zadovoljivosti formule prvog reda može svesti na problem ispitivanja (ne)zadovoljivosti beskonačnog skupa iskaznih formula (dokazivanje te činjenice izlazi van okvira ovog teksta). Kako je problem nezadovoljivosti beskonačnog skupa iskaznih formula poluodlučiv, sledi da je i problem nezadovoljivosti formule prvog reda poluodlučiv. $\square$

Posledica 4.6. *Problem zadovoljivosti formule prvog reda (tj. pitanje „da li je formula F prvog reda zadovoljiva?“) nije ni poluodlučiv.*

Dokaz. Ako bi problem zadovoljivosti bio poluodlučiv, onda bi, na osnovu Postove teoreme, bio i odlučiv, s obzirom da je njegov komplementarni problem (problem *nezadovoljivosti*) poluodlučiv. Otuda bi i problem valjanosti bio odlučiv (jer je F valjana akko $\neg F$ nije zadovoljiva), što znamo da nije slučaj. Kontradikcija. $\square$

Teorema 4.11. *Problem $\Delta \models F$ u logici prvog reda nije odlučiv, ali jeste poluodlučiv.*

Dokaz. Neodlučivost sledi iz neodlučivosti specijalnog slučaja za $\Delta = \emptyset$ (tada je ovaj problem ekvivalentan sa valjanošću formule F).

Sa druge strane, problem $\Delta \models F$ je ekvivalentan sa problemom nezadovoljivosti skupa $\Delta \cup \{\neg F\}$. U slučaju konačnog skupa $\Delta = \{A_1, \dots, A_n\}$, ovo se svodi na nezadovoljivost formule $A_1 \dots A_n \wedge \neg F$, što je poluodlučivo. U slučaju beskonačnog skupa Δ , na osnovu teoreme kompaktnosti važi da je $\Delta \models F$ akko postoji konačan potskup $\Delta' \subseteq \Delta$ takav da $\Delta' \models F$. Pritom, iz rekurzivnosti skupa Δ sledi da je moguće efektivno nabrajati sve njegove konačne potskupove. Problem $\Delta' \models F$ je poluodlučiv, a na osnovu primera 4.12, egzistencijalnom kvantifikacijom poluodlučivog problema opet dobijamo poluodlučiv problem. Otuda je problem $\Delta \models F$ poluodlučiv i za beskonačni skup Δ . $\square$

Glava 5

Izračunljivost i formalna matematika

5.1 Deduktivni sistemi

Pod *dedukcijom* podrazumevamo izvođenje zaključaka na osnovu datih *pretpostavki*, korišćenjem jasno definisanih *pravila izvođenja*. U matematičkoj logici, dedukcija predstavlja mehanizam formalnog rezonovanja, tj. dokazivanja tvrdjenja. Pored logike, deduktivni sistemi se koriste i u drugim oblastima gde je potrebno formalno i rigorozno izvoditi zaključke o nekom sistemu i njegovim svojstvima. Sa nekim takvim primenama deduktivnih sistema ćemo se upoznati kasnije.

Svaki deduktivni sistem je zadat skupom pravila izvođenja, koja su oblika:

$$\frac{P_1 \quad P_2 \quad \dots \quad P_n}{Q}$$

gde su $P_1, \dots, P_n$ *premise* pravila, dok je Q *zaključak* pravila. Premise i zaključak pravila predstavljaju *tvrdjenja* – činjenice koje su predmet izvođenja u deduktivnom sistemu. Smisamo gornjeg pravila izvođenja je: „ako mogu izvesti $P_1, \dots, P_n$, tada mogu izvesti i Q ”.

U kontekstu logike, tvrdjenja se najčešće predstavljaju formulama, a izvođenje formula u deduktivnom sistemu predstavlja *dokazivanje* njihove tačnosti. U tom smislu, značenje gornjeg pravila izvođenja je: „ako mogu da dokažem formule $P_1, \dots, P_n$, tada mogu da dokažem i Q ”. Na primer, posmatramo pravilo izvođenja:

$$\frac{P \Rightarrow Q \quad P}{Q}$$

Ovo pravilo (poznato i kao *modus ponens*) kaže da, ako možemo da dokažemo neku formulu P , kao i formulu $P \Rightarrow Q$, tada možemo da dokažemo i formulu Q .

Pored pravila, deduktivni sistem po pravilu sadrži i skup *aksioma*, tj. tvrdjenja za koje pretpostavljamo da važe i da se ne moraju izvoditi. Aksiome se mogu formulisati kao pravila izvođenja sa praznim skupom premisa, npr.:

$$\overline{Q \vee \neg Q}$$

Ovo pravilo kaže da mogu da dokažem formulu $Q \vee \neg Q$ bez ikakvih prethodnih pretpostavki, tj. formulu $Q \vee \neg Q$ ne moram ni da dokazujem, već je uvek mogu smatrati već dokazanom.

Primitimo da su aksiome i pravila izvođenja zadata shematski, a njihove konkretne instance se mogu dobiti instanciranjem simbola konkretnim objektima koji čine tvrdjenje (u logici su to formule). Na primer, konkretna instanca pravila *modus ponens* može biti:

$$\frac{p \vee q \Rightarrow \neg r \quad p \vee q}{\neg r}$$

pri čemu je P instancirano sa $p \vee q$, a Q sa $\neg r$, gde su p, q, r konkretna iskazna slova. Slično, instanca aksioma $Q \vee \neg Q$ može biti formula $(p \Rightarrow r) \vee \neg(p \Rightarrow r)$.

Pod *izvođenjem* u deduktivnom sistemu podrazumevam niz tvrdjenja $A_1, A_2, \dots, A_n$ takav da je svako A_i u nizu ili instanca aksiome, ili je izvedeno primenom nekog pravila izvođenja koristeći neka tvrdjenja koje prethode A_i u nizu kao pretpostavke. Za tvrdjenje A_n kojim se izvođenje završava kažemo da je izvedeno datim izvođenjem.

U kontekstu logike, izvođenja nazivamo i *dokazima*, a tvrdjenja za koja postoji dokaz nazivamo *teoremama* datog deduktivnog sistema. Na primer, ako pored aksioma $Q \vee \neg Q$ postoji i aksioma $P \Rightarrow (Q \Rightarrow P)$, uz pravilo izvođenja *modus ponens*, možemo imati sledeći dokaz:

1. $p \vee \neg p$
2. $p \vee \neg p \Rightarrow (q \Rightarrow p \vee \neg p)$
3. $q \Rightarrow p \vee \neg p$
4. $(q \Rightarrow p \vee \neg p) \Rightarrow (p \Rightarrow (q \Rightarrow p \vee \neg p))$
5. $p \Rightarrow (q \Rightarrow p \vee \neg p)$

Prve dve formule u nizu su instance aksioma, dok je treća izvedena iz prve dve pravilom *modus ponens*. Četvrta je ponovo instanca aksiome, dok je peta izvedena iz treće i četvrte primenom pravila *modus ponens*. Otuda je formula $p \Rightarrow (q \Rightarrow p \vee \neg p)$ jedna teorema ovog jednostavnog deduktivnog sistema.

Zbog preglednosti, izvođenja (dokazi) se često umesto niza predstavljaju *stablom*, kako bi se jasno videlo koje je tvrdjenje izvedeno iz kojih pretpostavki, kao i koje je pravilo primenjeno. U listovima tog stabla se nalaze aksiome, dok se u unutrašnjim čvorovima nalaze izvedena tvrdjenja. Pritom, deca unutrašnjeg čvora odgovaraju pretpostavkama koje su primenjene prilikom izvođenja odgovarajućeg tvrdjenja (sam unutrašnji čvor obično sadrži i informaciju o primenjenom pravilu). U korenu stabla se nalazi tvrdjenje koje je izvedeno tim izvođenjem. Na primer, gornji dokaz se u obliku stabla može predstaviti na sledeći način:

$$\frac{\frac{p \vee \neg p \quad p \vee \neg p \Rightarrow (q \Rightarrow p \vee \neg p)}{q \Rightarrow p \vee \neg p} mp \quad (q \Rightarrow p \vee \neg p) \Rightarrow (p \Rightarrow (q \Rightarrow p \vee \neg p))}{p \Rightarrow (q \Rightarrow p \vee \neg p)} mp$$

Činjenicu da se neko tvrdjenje P može izvesti u deduktivnom sistemu R zapisivaćemo kao:

$$\vdash_R P$$

pri čemu se indeks R može izostaviti ako je jasno o kom se deduktivnom sistemu radi (tj. možemo pisati samo $\vdash P$). U kontekstu logike, $\vdash P$ značiće da je P teorema tog deduktivnog sistema.

Ponekad ćemo razmatrati izvođenja u kojima se, kao listovi u stablu, pored aksioma mogu nalaziti i dodatne *pretpostavke* iz nekog zadatog skupa tvrđenja Δ . U tom slučaju, kažemo da takvo izvođenje izvodi (dokazuje) neko tvrđenje P iz *pretpostavki* Δ i to zapisujemo kao:

$$\Delta \vdash P$$

Kažemo i da je P *deduktivna posledica* skupa formula Δ u datom deduktivnom sistemu.

Dedukcija predstavlja formalni, simbolički sistem manipulisanja diskretnim objektima i kao takva je lišena bilo kakvog značenja osim onog koji sama proizvodi. Ipak, svrha dedukcije je obično da formalno opravda neke zaključke o kojima mi imamo nekakvu predstavu u stvarnosti. Drugim rečima, mi deduktivnom sistemu želimo da pridružimo nekakav smisao, tj. neku semantiku. Konkretno, u slučaju logike, očekujemo da deduktivni sistem izvodi formule koje su logički valjane. Za takav deduktivni sistem kažemo da je *saglasan* (engl. *sound*). Saglasnost deduktivnog sistema je neophodan uslov da bi taj deduktivni sistem bio od bilo kakvog praktičnog značaja u formalizaciji logičkog rasuđivanja. Pored saglasnosti, poželjno je da logički deduktivni sistem ima i svojstvo *potpunosti* – tj. da omogućava da se svaka valjana formula može dokazati kao teorema u tom deduktivnom sistemu. Dakle, imamo:

- *saglasnost*: iz $\vdash F$ sledi $\models F$
- *potpunost*: iz $\models F$ sledi $\vdash F$
- *saglasnost i potpunost*: $\vdash F$ akko $\models F$

5.1.1 Prirodna dedukcija

U ovom poglavlju razmatramo logički deduktivni sistem poznat pod nazivom *prirodna dedukcija* (engl. *natural deduction*). Ovaj sistem se može koristiti kako u iskaznoj, tako i u logici prvog reda (a uz određena uopštenja, i u logikama višeg reda). Naziv *prirodna dedukcija* potiče od činjenice da njegova pravila izvođenja prate uobičajene načine izvođenja zaključaka koje matematičari koriste prilikom dokazivanja teorema. Prirodnu dedukciju je razvio nemački matematičar Gerhard Genten 1933. godine.

Prilikom izvođenja dokaza u prirodnoj dedukciji, moguće je u određenim delovima dokaza privremeno uvesti dodatne pretpostavke koje se u nekom trenutku mogu *osloboditi* primenom odgovarajućih pravila. Oslobođena pretpostavka se na dalje ne može koristiti u narednim koracima dokaza. To znači da tokom izvođenja dokaza moramo da vodimo evidenciju o tome koje su pretpostavke na snazi u svakom koraku. Zbog toga ćemo tvrđenja koja čine dokaz predstavljati u obliku $\Gamma \vdash A$, pri čemu je Γ konačan skup rečenica, sa značenjem „Formula A se može dokazati iz pretpostavki Γ ” (ili „Formulu A treba dokazati iz pretpostavki

Γ ” u zavisnosti u kom smeru posmatramo dokaz). Zapis $\Gamma \vdash A$ nazivamo *kontekst* ili *sekvent*. Iz konteksta za svaku formulu vidimo koje su nam pretpostavke na raspolaganju za izvođenje dokaza te formule.

U prirodnoj dedukciji, za svaki od logičkih veznika imamo dva pravila: *pravilo uvođenja* (označeno sa I) i *pravilo eliminacije* (označeno sa E). U nastavku razmatramo ova pravila za svaki od veznika.

U slučaju negacije, imamo sledeća pravila:

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg I \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \perp} \neg E$$

Pravilo uvođenja negacije ($\neg I$) kaže da ako se formula $\perp$ može dokazati iz skupa pretpostavki Γ, A (gde je Γ neki konačan skup pretpostavki, možda i prazan), tada je formulu $\neg A$ moguće dokazati iz pretpostavki Γ . Ovim se dokazuje formula $\neg A$, tj. uvodi se veznik $\neg$ u dokazanu formulu.

Primetimo da se ovo pravilo može tumačiti i „unazad”: da bismo dokazali $\neg A$ iz pretpostavki Γ , dovoljno je dokazati formulu $\perp$ iz pretpostavki Γ, A . Sva pravila prirodne dedukcije ćemo moći ovako dualno da tumačimo. Ova dva tumačenja proističu iz načina na koji posmatramo dokaze. Jedan način je da dokaz (predstavljen u vidu stabla) posmatramo od listova ka korenu: mi polazimo od pretpostavki i pravilima izvodimo nove činjenice. Tada je tumačenje tih pravila „ako sam dokazao premise, mogu dokazati i zaključak pravila”. Ovaj pogled na dokaz je krajnje intuitivan ako želimo da tumačimo postojeći dokaz. Drugi pogled na dokaz je od korena ka listovima: mi polazimo od tvrdjenja koje je potrebno dokazati (tzv. *cilj* dokaza) i primenom pravila „unazad” dolazimo do premisa koje je potrebno dokazati kako bi zaključak važio. Time se dokazivanje glavnog cilja svodi na dokazivanje *potciljeva* koji predstavljaju premise za primenu tog pravila. U takvom pogledu na dokaz, pravila se tumače u stilu „da bih dokazao cilj, dovoljno je dokazati ove potciljeve”. Ovakav pogled na dokaze i pravila je pogodan prilikom konstrukcije dokaza i tipično se koristi u automatskom dokazivanju teorema.

Pravilo eliminacije negacije ($\neg E$) kaže da ako iz pretpostavki Γ dokažemo A i $\neg A$, tada iz Γ možemo dokazati i $\perp$. Ili, inverzno: da bismo dokazali $\perp$ iz Γ , dovoljno je dokazati A i $\neg A$ iz Γ , za neku, proizvoljno odabranu formulu A . Ovim pravilom se negacija koja se nalazi u premisi $\neg A$ eliminiše, tj. ne postoji u zaključku.

Za konjunkciju imamo sledeća pravila:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E1 \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E2$$

Pravilo uvođenja konjunkcije ($\wedge I$) kaže da ako iz Γ možemo dokazati formule A i B , tada iz Γ možemo dokazati i $A \wedge B$. Inverzno: da bismo dokazali $A \wedge B$ iz Γ , dovoljno je iz istog skupa pretpostavki dokazati A i B . Ovim se dokazivanje cilja $A \wedge B$ svodi na dokazivanje dva potcilja, A i B .

Pravilo eliminacije konjunkcije dolazi u dve varijante: eliminacija po prvom konjunkt ($\wedge E1$) i po drugom konjunkt ($\wedge E2$). U prvom slučaju, pravilo kaže: ako možemo da dokažemo $A \wedge B$ iz pretpostavki Γ , tada je moguće dokazati i formulu A iz Γ . Ili inverzno: da bismo dokazali A iz Γ , dovoljno je dokazati $A \wedge B$ iz istih pretpostavki. Druga varijanta pravila je analogna, samo za drugi konjunkt B .

Za disjunkciju imamo sledeća pravila:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee I1 \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee I2$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee E$$

Ovoga puta, pravilo uvođenja disjunkcije dolazi u dva oblika, po prvom disjunkt (∨I1) i po drugom disjunkt (∨I2). Ovo pravilo kaže da ako možemo da iz Γ dokažemo jedan disjunkt (A ili B), tada je iz istog skupa pretpostavki moguće dokazati i disjunkciju $A \vee B$. Ili obrnuto: da bismo dokazali $A \vee B$ iz Γ , dovoljno je ili dokazati A iz Γ , ili B iz Γ .

Pravilo eliminacije disjunkcije (∨E) izgleda znatno kompleksnije. Ono kaže da, ako je moguće (1) iz pretpostavki Γ dokazati formulu $A \vee B$, (2) iz pretpostavki Γ, A dokazati neku formulu C i (3) iz pretpostavki Γ, B dokazati tu istu formulu C , tada je iz pretpostavki Γ moguće dokazati formulu C . Intuitivno, ovo pravilo predstavlja grananje po slučajevima koje matematičari često primenjuju u dokazima („ako sam dokazao da važi $A \vee B$ tada mogu razmatrati dva odvojena slučaja: da važi A i da važi B i u svakom od ta dva slučaja da posebno dokažem tvrđenje C ”). Tumačenje ovog pravila „unazad” bilo bi: da bih dokazao tvrđenje C iz pretpostavki Γ , dovoljno je dokazati $A \vee B$ iz Γ za neke proizvoljne formule A i B , a zatim dokazati C iz Γ, A i dokazati C iz Γ, B . Primitimo da ovo pravilo vrši oslobađanje pretpostavki – prilikom dokazivanja druge i treće premise dozvoljeno je koristiti A , odnosno B , kao pretpostavku, ali te pretpostavke se zatim oslobađaju i nisu dostupne u daljem toku dokaza.

Za implikaciju imamo sledeća pravila:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow I \quad \frac{\Gamma \vdash A \quad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B} \Rightarrow E$$

Pravilo uvođenja implikacije ($\Rightarrow I$) ima sledeći intuitivni smisao: ako iz pretpostavki Γ, A možemo dokazati neku formulu B , tada formulu $A \Rightarrow B$ možemo dokazati iz pretpostavki Γ . Dakle, ovde A imamo kao lokalnu pretpostavku koju koristimo da dokažemo B , a zatim je primenom ovog pravila oslobađamo (jer je ugrađujemo u levu stranu uvedene implikacije). Inverzno tumačenje pravila glasi: da bismo dokazali implikaciju $A \Rightarrow B$ iz nekog skupa pretpostavki Γ , dovoljno je dokazati B pod dodatnom pretpostavkom A .

Pravilo eliminacije implikacije ($\Rightarrow E$) je u stvari naše staro dobro pravilo *modus ponens*: ako iz Γ dokažemo A i $A \Rightarrow B$, tada možemo dokazati B iz istog skupa pretpostavki. Ili inverzno: da bismo dokazali neku formulu B , dovoljno je, za proizvoljno odabranu formulu A , dokazati A , kao i formulu $A \Rightarrow B$.

Za logičke konstante imamo sledeća pravila:

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} \perp E \quad \frac{}{\Gamma \vdash \top} \top I$$

Intuitivno, pravilo $\perp E$ kaže da ako je iz nekog skupa Γ moguće dokazati $\perp$, tada je iz Γ moguće dokazati bilo koju formulu A . Sa druge strane, pravilo $\top I$ kaže da je $\top$ moguće dokazati iz bilo kog skupa pretpostavki Γ . Kako ovo pravilo nema premisa, kontekst $\Gamma \vdash \top$ smatramo *aksiomom* prirodne dedukcije. Tumačeno unatraske, dokazivanje cilja oblika $\Gamma \vdash \top$ ne proizvodi nikakve potciljeve, pa se tu ta grana dokaza završava. Drugim rečima, kontekst $\Gamma \vdash \top$ se može nalaziti u listovima dokaza, budući da je aksioma.

Najzad, imamo pravilo izvođenja:

$$\frac{}{\Gamma, A \vdash A} \text{ ass}$$

Ovo pravilo takođe uvodi kao aksiomu kontekst oblika $\Gamma, A \vdash A$. Intuitivno, uvek je moguće dokazati formulu A iz skupa pretpostavki koji sadrži tu formulu. Samim tim, ovakvi konteksti se mogu nalaziti u listovima dokaza. Tumačeno „unazad”: ako se formula koja se dokazuje nalazi u skupu pretpostavki, onda je dokaz završen.

Za formulu A kažemo da je *teorema* prirodne dedukcije (u oznaci $\vdash A$) ako je u prirodnoj dedukciji moguće dokazati kontekst $\vdash A$. Slično, za formulu A kažemo da je *deduktivna posledica* skupa formula Γ (u oznaci $\Gamma \vdash A$), ako je moguće dokazati kontekst $\Gamma \vdash A$.

Primer 5.1. Dokažimo da je formula $\neg(A \vee B) \Rightarrow \neg A \wedge \neg B$ teorema prirodne dedukcije. Imamo sledeći dokaz konteksta $\vdash \neg(A \vee B) \Rightarrow \neg A \wedge \neg B$:

$$\frac{\frac{\frac{\neg(A \vee B), A \vdash \neg(A \vee B)}{\neg(A \vee B), A \vdash \neg(A \vee B)} \quad \frac{\neg(A \vee B), A \vdash A}{\neg(A \vee B), A \vdash A \vee B} \vee I1}{\frac{\neg(A \vee B), A \vdash \perp}{\neg(A \vee B) \vdash \neg A} \neg I} \neg E \quad \frac{\frac{\frac{\neg(A \vee B), B \vdash \neg(A \vee B)}{\neg(A \vee B), B \vdash \neg(A \vee B)} \quad \frac{\neg(A \vee B), B \vdash B}{\neg(A \vee B), B \vdash A \vee B} \vee I2}{\frac{\neg(A \vee B), B \vdash \perp}{\neg(A \vee B) \vdash \neg B} \neg I} \neg E$$

$$\frac{\frac{\neg(A \vee B) \vdash \neg A \wedge \neg B}{\vdash \neg(A \vee B) \Rightarrow \neg A \wedge \neg B} \Rightarrow I}{\vdash \neg(A \vee B) \Rightarrow \neg A \wedge \neg B} \wedge I$$

Ovaj dokaz se konstruiše od korena ka listovima, tako što se određuju potciljevi koje je potrebno dokazati da bi se odgovarajuće pravilo moglo primeniti. Na primer, da bismo dokazali polazni kontekst $\vdash \neg(A \vee B) \Rightarrow \neg A \wedge \neg B$, na osnovu pravila $\Rightarrow I$, dovoljno je dokazati kontekst $\neg(A \vee B) \vdash \neg A \wedge \neg B$. Otuda se ovaj čvor dodaje u stablo kao dete korenog čvora, a dokazivanje se nastavlja dokazivanjem ovog potcilja. Da bismo njega dokazali, dovoljno je, u skladu sa pravilom $\wedge I$, dokazati dva potcilja: $\neg(A \vee B) \vdash \neg A$ i $\neg(A \vee B) \vdash \neg B$. Odgovarajući čvorovi se dodaju u stablo dokaza kao deca čvora $\neg(A \vee B) \vdash \neg A \wedge \neg B$, a dokazivanje se nastavlja dokazivanjem ovih potciljeva u proizvoljnom redosledu. Dokazivanje svakog potcilja se završava kada se dođe do aksiome.

Napomena 5.1. Kao što je objašnjeno u prethodnom primeru, dokazi se tipično konstruišu „unazad”, počev od korena stabla. U tom smislu, pravila uvođenja veznika zapravo omogućavaju „eliminaciju” tog veznika, jer veznik obrnutom primenom pravila nestaje iz potciljeva. Pritom, formule koje se pojavljuju u dobijenim potciljevima su uvek *potformule* formula koje su se pojavljivale u polaznom kontekstu. Na primer, kada pravilo uvođenja konjunkcije primenimo unazad, tada umesto formule $A \wedge B$ dobijamo formule A i B , koje su potformule polazne formule $A \wedge B$. Ovo *svojstvo potformule* je veoma važno prilikom konstrukcije dokaza – u potciljevima baratamo samo sa potformulama koje se javljaju u polaznom cilju, pa nije potrebno „izmišljati” nikakve dodatne formule. Sa druge strane, pravila eliminacije nemaju svojstvo potformule. Na primer, primena pravila eliminacije negacije „unazad” u cilju dokaza formule $\perp$ zahteva da se „izmisli” neka nova formula A takva da se dalje dokazivanje svodi na potciljeve A i $\neg A$. U praksi, ta formula A se ne bira proizvoljno, već se bira tako da ju je moguće dokazati. Tipično, biramo neku formulu koja se već nalazi među pretpostavkama, poput formule $\neg(A \vee B)$ u prethodnom primeru.

Primer 5.2. Dokažimo da je formula $A \Rightarrow C$ deduktivna posledica skupa formula $A \Rightarrow B, B \Rightarrow C$. Dakle, potrebno je dokazati kontekst $A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C$. Imamo sledeći dokaz u prirodnoj dedukciji:

$$\frac{\frac{A \Rightarrow B, B \Rightarrow C, A \vdash A \quad A \Rightarrow B, B \Rightarrow C, A \vdash A \Rightarrow B}{A \Rightarrow B, B \Rightarrow C, A \vdash B} \Rightarrow E \quad \frac{A \Rightarrow B, B \Rightarrow C, A \vdash B \Rightarrow C}{A \Rightarrow B, B \Rightarrow C, A \vdash C} \Rightarrow E}{\frac{A \Rightarrow B, B \Rightarrow C, A \vdash C}{A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C} \Rightarrow I} \Rightarrow E$$

Dakle, da bismo dokazali $A \Rightarrow C$, dokazujemo formulu C pod dodatnom pretpostavkom A . Da bismo to dokazali, dovoljno je dokazati formulu B , s obzirom da formulu $B \Rightarrow C$ već imamo među pretpostavkama, pa je moguće izvesti C eliminacijom implikacije. Najzad, dokazivanje formule B se opet vrši primenom eliminacije implikacije, s obzirom da formule A i $A \Rightarrow B$ postoje među pretpostavkama.

Opisani sistem prirodne dedukcije barata sa iskaznim formulama, pa se može koristiti za rezonovanje u iskaznoj logici. Važi sledeća teorema.

Teorema 5.1. *Sistem prirodne dedukcije je saglasan za iskaznu logiku: ako je $\Gamma \vdash A$, tada je $\Gamma \models A$. Specijalno, ako je A teorema prirodne dedukcije, tada je A tautologija.*

Dokaz. Dokaz se izvodi tako što se pokaže saglasnost svakog pravila izvođenja. Da bismo dokazali saglasnost pravila izvođenja, potrebno je dokazati da ako saglasnost važi za premise, tada važi i za zaključak. Na primer, za pravilo $\wedge I$, pretpostavimo da saglasnost važi za premise $\Gamma \vdash A$ i $\Gamma \vdash B$. To znači da je $\Gamma \models A$ i $\Gamma \models B$. Sada u svakoj valuaciji u kojoj važe sve formule iz Γ važe i formule A i B , pa važi i formula $A \wedge B$. Otuda je $\Gamma \models A \wedge B$. Dakle, saglasnost važi i za zaključak. Slično se dokazuje i za ostala pravila. $\square$

Iz prethodne teoreme sledi da možemo utvrditi tautologičnost neke iskazne formule tako što konstruišemo njen formalni dokaz u prirodnoj dedukciji. Ovo je sasvim drugačiji pristup od, npr. utvrđivanja tautologičnosti metodom istinitosnih tablica. Naime, metod istinitosnih tablica je semantički metod: njime utvrđujemo tautologičnost tako što razmatramo sve moguće modele, tj. pretražujemo prostor valuacija. Sa druge strane, u sintaksno-deduktivnom pristupu, mi konstruišemo eksplicitan dokaz koji potvrđuje univerzalnu tačnost formule. Ovaj dokaz je značajan zato što nam pruža razumljiv argument zbog čega je neka formula valjana.

Prirodno se postavlja i pitanje potpunosti prirodne dedukcije za iskaznu logiku. Na žalost, ispostavi se da gore navedeni skup pravila ne obezbeđuje potpunost. Na primer, formulu $A \vee \neg A$ nije moguće dokazati pomoću do sada prikazanih pravila prirodne dedukcije. Sa druge strane, formula $A \vee \neg A$ je svakako tautologija. Dakle, nije moguće dokazati svaku tautologiju, pa potpunost ne važi.

Da bismo obezbedili potpunost, dovoljno je dodati još jedno pravilo u naš deduktivni sistem:

$$\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A} \text{ contr}$$

Ovo pravilo je poznato i kao *pravilo svodenja na kontradikciju* – ili što matematičari vole da kažu „pretpostavimo suprotno”: da bismo dokazali A iz pretpostavki Γ , dovoljno je da pretpostavimo da važi $\neg A$ i da iz te dodatne pretpostavke izvedemo kontradikciju, tj. dokažemo $\perp$.

Primer 5.3. Želimo da dokažemo da je formula $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$ teorema prirodne dedukcije. Da bismo to dokazali, potrebno je da konstruišemo dokaz u čijem korenu se nalazi kontekst $\vdash \neg(A \wedge B) \Rightarrow \neg A \vee \neg B$. Zbog nedostatka prostora, ovaj dokaz ćemo prikazati iz delova. Najpre, polazeći od polaznog konteksta u korenu, imamo sledeći deo dokaza:

$$\frac{\frac{\frac{\vdots}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash \neg(A \wedge B)}}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash \perp} \neg E}{\frac{\neg(A \wedge B) \vdash \neg A \vee \neg B}{\vdash \neg(A \wedge B) \Rightarrow \neg A \vee \neg B} \Rightarrow I} \text{contr}$$

Ovde smo primenom pravila uvođenja implikacije naš cilj sveli na potcilj $\neg(A \wedge B) \vdash \neg A \vee \neg B$. Zatim smo primenom pravila *contr* dokazivanje formule $\neg A \vee \neg B$ sveli na dokazivanje formule $\perp$ iz dodatne pretpostavke $\neg(\neg A \vee \neg B)$. Kako je formula $\neg(A \wedge B)$ među pretpostavka, na osnovu pravila $\neg E$, dovoljno je dokazati formulu $A \wedge B$ iz tekućeg skupa pretpostavki. Odgovarajući deo dokaza navodimo u nastavku.

$$\frac{\frac{\frac{\vdots}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash A}}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash A \wedge B} \wedge I}{\vdots}$$

Dakle, primenjujemo pravilo uvođenja konjunkcije kojim dokaz formule $A \wedge B$ svodimo na dokazivanje potciljeva A i B . Ovi delovi dokaza su dati u nastavku.

$$\frac{\frac{\frac{\frac{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg A \vdash \neg A}{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg A \vdash \neg A \vee \neg B} \vee I1}{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg A \vdash \perp} \neg E}{\frac{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg A \vdash \perp}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash A} \text{contr}} \vdots$$

$$\frac{\frac{\frac{\frac{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg B \vdash \neg B}{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg B \vdash \neg A \vee \neg B} \vee I2}{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg B \vdash \perp} \neg E}{\frac{\neg(A \wedge B), \neg(\neg A \vee \neg B), \neg B \vdash \perp}{\neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash B} \text{contr}} \vdots$$

Dakle, formulu A dokazujemo tako što primenjujemo pravilo *contr*, čime se dokaz svodi na dokazivanje formule $\perp$ pod dodatnom pretpostavkom $\neg A$ („pretpostavimo suprotno”). Sada se, na osnovu pravila eliminacije negacije, dokaz svodi na dokazivanje formule $\neg A \vee \neg B$, s obzirom da $\neg(\neg A \vee \neg B)$ imamo među pretpostavkama. Dokaz formule B je analogan.

Napomena 5.2. Prikazani sistem prirodne dedukcije bez pravila *contr* predstavlja sistem *intuicionističke iskazne logike*. Intuicionistička logika je nastala krajem 19. veka kao jedan od pokušaja da se matematika „očisti” od paradoksa koji su se u njoj javili u tom periodu. *Intuicionizam* predstavlja filozofski pravac u matematici koji insistira na tome da se u matematici može dokazati samo ono za šta je moguće pružiti *konstruktivistički* dokaz, tj. dokaz koji direktno konstruiše tu formulu. Zbog toga intuicionisti odbacuju svođenje na kontradikciju kao metod dokazivanja: time što smo dokazali da $\neg A$ ne može da važi i dalje nam ne daje za pravo da tvrdimo da važi A , jer mi nismo konstruisali dokaz za A . U suštini, intuicionisti odbacuju *zakon isključenja trećeg* – $A \vee \neg A$ i zahtevaju da se eksplicitno konstruiše dokaz ili za A ili za $\neg A$.

Ako pravilima prirodne dedukcije pridružimo i pravilo *contr*, tada dobijamo *klasičnu iskaznu logiku*. U klasičnoj iskaznoj logici je dozvoljeno koristiti svođenje na kontradikciju kao legitiman metod dokazivanja. Samim tim, u klasičnoj logici je moguće dokazati i $A \vee \neg A$, o čemu govori sledeći primer.

Primer 5.4. Dokažimo da je formula $A \vee \neg A$ teorema prirodne dedukcije za *klasičnu* logiku. Imamo sledeći dokaz (ponovo prikazan iz delova, zbog nedostatka prostora):

$$\frac{\frac{\vdots}{\neg(A \vee \neg A) \vdash A} \quad \frac{\vdots}{\neg(A \vee \neg A) \vdash \neg A}}{\neg(A \vee \neg A) \vdash \perp} \neg E \quad \frac{}{\vdash A \vee \neg A} \text{contr}$$

Prvi deo dokaza predstavlja koren u kome se nalazi kontekst $\vdash A \vee \neg A$ koji dokazujemo. Pravilom kontradikcije ovaj dokaz svodimo na dokazivanje kontradikcije iz $\neg(A \vee \neg A)$. Ovo se dokazuje eliminacijom negacije, po formuli A . To znači da je u nastavku potrebno dokazati dva potcilja: A i $\neg A$ iz iste pretpostavke $\neg(A \vee \neg A)$.

$$\frac{\frac{\neg(A \vee \neg A), \neg A \vdash \neg A}{\neg(A \vee \neg A), \neg A \vdash A \vee \neg A} \vee I2 \quad \frac{\neg(A \vee \neg A), \neg A \vdash \neg(A \vee \neg A)}{\neg(A \vee \neg A), \neg A \vdash \perp} \neg E}{\neg(A \vee \neg A) \vdash A} \text{contr}$$

$$\vdots$$

Drugi deo dokaza predstavlja dokaz formule A . Ovde ponovo primenjujemo pravilo *contr*, pri čemu dobijamo dodatnu pretpostavku $\neg A$ iz koje treba dokazati kontradikciju. Ovo dalje radimo eliminacijom negacije po formuli $\neg(A \vee \neg A)$ koju imamo među pretpostavkama. Preostali potcilj $A \vee \neg A$ je sada lako dokazati jer imamo $\neg A$ među pretpostavkama, pa možemo primeniti pravilo uvođenja disjunkcije.

$$\frac{\frac{\neg(A \vee \neg A), A \vdash A}{\neg(A \vee \neg A), A \vdash A \vee \neg A} \vee I1 \quad \frac{\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)}{\neg(A \vee \neg A), A \vdash \perp} \neg E}{\neg(A \vee \neg A) \vdash \neg A} \neg I$$

$$\vdots$$

Treći deo dokaza predstavlja dokaz formule $\neg A$. Ovde umesto pravila *ccontr* primenjujemo pravilo uvođenja negacije unazad, čime se među pretpostavkama pojavljuje dodatno i formula A . Dalji dokaz je sličan kao i u drugom delu dokaza.

Teorema 5.2. *Sistem prirodne dedukcije za klasičnu iskaznu logiku ima svojstvo potpunosti: ako je $\Gamma \models A$, tada je i $\Gamma \vdash A$. Specijalno, ako je formula A tautologija, tada je moguće dokazati formulu A u sistemu prirodne dedukcije.*

Napomena 5.3. Dokaz gornje teoreme izostavljamo zbog složenosti.

Teorema 5.3. *Neka je Γ konačan skup iskaznih formula, a A proizvoljna iskazna formula. Problem $\Gamma \vdash A$ je odlučiv.*

Dokaz. Da bismo utvrdili da li važi $\Gamma \vdash A$, dovoljno je da utvrdimo da li je $\Gamma \models A$, što je odlučivo na osnovu posledice 4.4. $\square$

Posledica 5.1. *Problem $\vdash A$, tj. pitanje „Da li je A teorema prirodne dedukcije za klasičnu iskaznu logiku?” je odlučivo.*

Napomena 5.4. Iz dokaza prethodne teoreme sledi da mi možemo ispitivati dokazivost nekog tvrdjenja u prirodnoj dedukciji (tj. utvrđivati da li dokaz postoji ili ne) bez konstruisanja samog dokaza. Dovoljno je semantičkim metodama utvrditi tautologičnost odgovarajuće formule. Ako se ispostavi da je formula tautologija, tada znamo da dokaz postoji, iako ga nismo efektivno konstruisali. Ipak, efektivna konstrukcija dokaza je često poželjna u praksi, jer ona omogućava jednostavnu verifikaciju dobijenog odgovora (složenim procedurama odlučivanja često ne možemo verovati, jer mogu sadržati greške).

Sistem prirodne dedukcije se može proširiti tako da se može primeniti i na formule u logici prvog reda. Za ovo je dovoljno dodati odgovarajuća pravila uvođenja i eliminacije za kvantifikatore.

Pravila za univerzalni kvantifikator su:

$$\frac{\Gamma \vdash A[x \rightarrow y]}{\Gamma \vdash \forall x.A} \forall I \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x \rightarrow t]} \forall E$$

Pravilo uvođenja univerzalnog kvantifikatora ($\forall I$) u premisi sadrži formulu $A[x \rightarrow y]$ koja nastaje tako što se promenljiva x u formuli A zameni sa y . Pritom, važno je da promenljiva y bude takva da se ne pojavljuje kao slobodna promenljiva ni u A ni u Γ . Intuitivno, to znači „neko y o kome nemamo nikakvih pretpostavki”, tj. „proizvoljno fiksirano y ”. Sada pravilo možemo razumeti na sledeći način: ako iz pretpostavki Γ možemo da dokažemo da formula A važi za proizvoljno fiksirano y , tada A važi za svako x pod pretpostavkama Γ . Obratno, možemo reći: da bismo dokazali $\forall x.A$ iz Γ , dovoljno je da dokažemo da A važi kada umesto x stavimo neko proizvoljno fiksirano y .

Pravilo eliminacije univerzalnog kvantifikatora ($\forall E$) kaže da ako je moguće dokazati $\forall x.A$ iz nekih pretpostavki Γ , tada je moguće dokazati i bilo koju instancu formule A , tj. formulu $A[x \rightarrow t]$ koja nastaje kada se x zameni proizvoljnim termom t . Pritom je jedino važno da term t ostane potpuno slobodan u $A[x \rightarrow t]$, tj. ni jedna promenljiva koja se javlja u t ne sme biti vezana nekim kvantifikatorom u A . Inverzno tumačenje pravila je: da bismo dokazali neku instancu formule A po promenljivoj x , dovoljno je dokazati da važi $\forall x.A$. Tipično, ovo pravilo primenjujemo kada imamo $\forall x.A$ među pretpostavkama.

Pravila za egzistencijalni kvantifikator su sledeća:

$$\frac{\Gamma \vdash A[x \rightarrow t]}{\Gamma \vdash \exists x.A} \exists I \quad \frac{\Gamma \vdash \exists x.A \quad \Gamma, A[x \rightarrow y] \vdash B}{\Gamma \vdash B} \exists E$$

Pravilo uvođenja egzistencijalnog kvantifikatora ($\exists I$) kaže da ako je moguće dokazati neku instancu (po x) formule A iz pretpostavki Γ , tada je moguće dokazati formulu $\exists x.A$ (tj. tada postoji neko x za koje važi A). Ili obratno, da bismo dokazali $\exists x.A$ iz pretpostavki Γ , dovoljno je dokazati neku instancu formule A iz istih pretpostavki. I ovde važi uslov da term t mora biti slobodan u $A[x \rightarrow t]$, tj. ni jedna njegova promenljiva ne sme biti vezana kvantifikatorom u A .

Prva premisa pravila eliminacije egzistencijalnog kvantifikatora ($\exists E$) je da se iz pretpostavki Γ može dokazati $\exists x.A$. U drugoj premisi, među pretpostavkama imamo formulu $A[x \rightarrow y]$ u kojoj umesto x stoji y koje je opet „proizvoljno fiksirano y “, tj. y koje se ne pojavljuje ni u jednoj od pretpostavki iz Γ , kao ni u formulama A i B . Smisao pravila je sledeći: ako smo dokazali $\exists x.A$ i ako možemo da, pod pretpostavkom da A važi za neko proizvoljno fiksirano y , dokažemo formulu B , tada je iz Γ moguće dokazati B . Intuitivno, to „proizvoljno fiksirano y “ je upravo ono x za koje važi A , a za koje smo dokazali da postoji. Posmatranjem pravila unazad, možemo ga razumeti ovako: da bismo dokazali formulu B , dovoljno je dokazati $\exists x.A$ za neku formulu A , a onda pod pretpostavkom da za neko proizvoljno y važi A dokazati formulu B .

U logici prvog reda takođe možemo razmatrati sistem prirodne dedukcije sa i bez pravila *contr*, čime dobijamo klasičnu i intuicionističku logiku prvog reda. Kao i u iskaznoj logici, u logici prvog reda važi sledeća teorema u *saglasnosti*.

Teorema 5.4. *Sistem prirodne dedukcije za klasičnu logiku prvog reda je saglasan, tj. ako je $\Gamma \vdash A$, tada je i $\Gamma \models A$. Specijalno, ako je formula A teorema prirodne dedukcije za klasičnu logiku, onda je A valjana formula.*

Dokaz. Kao i u slučaju iskazne logike, potrebno je dokazati saglasnost svakog od pravila. Saglasnost pravila za iskazne veznike je već dokazana ranije, pa je potrebno još dokazati saglasnot pravila za kvantifikatore. Ilustracije radi, razmotrimo pravilo uvođenja univerzalnog kvantifikatora. Pretpostavimo da je premisa saglasna, tj. da iz $\Gamma \vdash A[x \rightarrow y]$ sledi $\Gamma \models A[x \rightarrow y]$. Ovo znači da za svaku strukturu $\mathcal{D}$ i valuaciju v koje zadovoljavaju sve formule iz Γ važi da je $\mathcal{D}_v(A[x \rightarrow y]) = 1$. Pritom, kako se y ne pojavljuje kao slobodna promenljiva u Γ , sledi da vrednost formula iz Γ u strukturi $\mathcal{D}$ ne zavisi od vrednosti valuacije za promenljivu y . To znači da su formule iz Γ tačne u strukturi $\mathcal{D}$ i za svaku valuaciju v_y koja se od v razlikuje samo na promenljivoj y . Otuda će i formula $A[x \rightarrow y]$ biti tačna za svaku takvu valuaciju v_y , kao logička posledica formula iz Γ . Samim tim će formula $\forall y.A[x \rightarrow y]$ biti tačna u strukturi $\mathcal{D}$ i valuaciji v . Odavde sledi da će i formula $\forall x.A$ biti tačna u strukturi $\mathcal{D}$ i valuaciji v , s obzirom da su formule $\forall x.A$ i $\forall y.A[x \rightarrow y]$ logički ekvivalentne (druga je dobijena od prve preimenovanjem vezane promenljive, tj. alfa konverzijom koja je korektno izvedena, s obzirom da se y ne pojavljuje ni u A kao slobodna). Time smo dokazali da važi $\Gamma \models \forall x.A$, tj. da je i zaključak takođe saglaasn. Čitalac za vežbu može dokazati saglasnost ostalih pravila za kvantifikatore. $\square$

Sledeća teorema, koju navodimo bez dokaza, govori o poptunosti sistema prirodne dedukcije za klasičnu logiku prvog reda.

Teorema 5.5. *Sistem prirodne dedukcije za klasičnu logiku prvog reda je potpun, tj. važi da iz $\Gamma \models A$ sledi da je $\Gamma \vdash A$. Specijalno, svaka valjana formula prvog reda je i teorema sistema prirodne dedukcije za klasičnu logiku prvog reda.*

Napomena 5.5. Prethodna teorema se pripisuje nemačkom matematičaru Kurtu Godelu i poznata je pod nazivom *Gedelova teorema o potpunosti*. Ova teorema, iako veoma važna, često se nalazi u senci druge, mnogo poznatije *Gedelove teoreme o nepotpunosti aritmetike* koju izlažemo kasnije.

Primer 5.5. Dokažimo da je formula $(\exists x.\forall y.p(x, y)) \Rightarrow (\forall y.\exists x.p(x, y))$ teorema prirodne dedukcije za logiku prvog reda. Imamo sledeći dokaz, sa kontekstom $\vdash (\exists x.\forall y.p(x, y)) \Rightarrow (\forall y.\exists x.p(x, y))$ u korenu:

$$\frac{\frac{\frac{\frac{\forall y.p(x', y) \vdash \forall y.p(x', y)}{\forall y.p(x', y) \vdash p(x', y')}{\forall y.p(x', y) \vdash \exists x.p(x, y')}{\exists x.\forall y.p(x, y) \vdash \exists x.\forall y.p(x, y)} \quad \frac{\frac{\frac{\forall y.p(x', y) \vdash \forall y.p(x', y)}{\forall y.p(x', y) \vdash p(x', y')}{\forall y.p(x', y) \vdash \exists x.p(x, y')}{\forall y.p(x', y) \vdash \forall y.\exists x.p(x, y)} \quad \frac{\exists x.\forall y.p(x, y) \vdash \exists x.\forall y.p(x, y)}{\exists x.\forall y.p(x, y) \vdash \forall y.\exists x.p(x, y)} \quad \frac{\exists x.\forall y.p(x, y) \vdash \forall y.\exists x.p(x, y)}{\vdash (\exists x.\forall y.p(x, y)) \Rightarrow (\forall y.\exists x.p(x, y))} \Rightarrow I$$

Dakle, polazeći od korena stabla, najpre primenjujemo pravilo uvođenja implikacije, čime dobijamo podcilj $\exists x.\forall y.p(x, y) \vdash \forall y.\exists x.p(x, y)$, tj. dokazujemo formulu $\forall y.\exists x.p(x, y)$ pod pretpostavkom $\exists x.\forall y.p(x, y)$. Kako imamo formulu sa vodećim egzistencijalnim kvantifikatorom među pretpostavkama, prirodno je da primenimo pravilo eliminacije egzistencijalnog kvantifikatora, čime nam preostaje da dokažemo formulu $\forall y.\exists x.p(x, y)$ pod pretpostavkom da formula $\forall y.p(x', y)$ važi za neko proizvoljno fiksirano x' . Ovo se dalje svodi na dokazivanje formule $\exists x.p(x, y')$ za proizvoljno fiksirano y' (pravilo uvođenja univerzalnog kvantifikatora). Da bismo dokazali formulu $\exists x.p(x, y')$, dovoljno je dokazati formulu $p(x, y')$ za neku instancu po x . Mi ćemo odabrati baš instancu $x \rightarrow x'$, s obzirom da imamo $\forall y.p(x', y)$ među pretpostavkama, odakle jednostavno sledi $p(x', y')$ (eliminacija univerzalnog kvantifikatora).

Primer 5.6. Dokažimo da je formula $(\neg \exists x.p(x)) \Rightarrow (\forall y.\neg p(y))$ teorema prirodne dedukcije za logiku prvog reda.

$$\frac{\frac{\frac{\neg \exists x.p(x), p(z) \vdash p(z)}{\neg \exists x.p(x), p(z) \vdash \exists x.p(x)} \quad \frac{\neg \exists x.p(x), p(z) \vdash \neg \exists x.p(x)}{\neg \exists x.p(x), p(z) \vdash \perp} \quad \frac{\neg \exists x.p(x), p(z) \vdash \perp}{\neg \exists x.p(x) \vdash \neg p(z)} \quad \frac{\neg \exists x.p(x) \vdash \neg p(z)}{\neg \exists x.p(x) \vdash \forall y.\neg p(y)} \quad \frac{\neg \exists x.p(x) \vdash \forall y.\neg p(y)}{\vdash (\neg \exists x.p(x)) \Rightarrow (\forall y.\neg p(y))} \Rightarrow I$$

Dakle, najpre pravilom uvođenja implikacije, dokaz teoreme svodimo na potcilj $\neg \exists x.p(x) \vdash \forall y.\neg p(y)$. Da bismo dokazali $\forall y.\neg p(y)$, dovoljno je dokazati da važi $\neg p(z)$ za neko proizvoljno fiksirano z (uvođenje univerzalnog kvantifikatora). Dalje, da bismo dokazali $\neg p(z)$, dovoljno je dokazati kontradikciju ($\perp$) pod pretpostavkom $p(z)$ (uvođenje negacije). Kako među pretpostavkama imamo formulu $\neg \exists x.p(x)$, prirodno je da pokušamo primenu pravila eliminacije negacije po ovoj formuli. Time se dokaz svodi na potcilj $\exists x.p(x)$, za šta je dovoljno dokazati da $p(x)$ važi za neki proizvoljan term. Ovo trivijalno sledi, s obzirom da $p(z)$ imamo među pretpostavkama.

Teorema 5.6. *Problem $\Delta \vdash F$ u sistemu prirodne dedukcije za klasičnu logiku prvog reda je neodlučiv, ali jeste poluodlučiv.*

Dokaz. Problem $\Delta \vdash F$ je ekvivalentan sa $\Delta \models F$, pa teorema sledi iz teoreme 4.11. $\square$

Posledica 5.2. *Problem $\vdash F$, tj. pitanje „Da li je F teorema u sistemu prirodne dedukcije za klasičnu logiku prvog reda?” je neodlučiv, ali je poluodlučiv.*

Napomena 5.6. Iz dokaza prethodne teoreme sledi da mi možemo utvrditi da je neka formula valjana (semantičkim metodama) i na osnovu toga zaključiti da postoji dokaz u prirodnoj dedukciji, iako ga nismo efektivno konstruisali. Ipak, u logici prvog reda, čak i više nego u iskaznoj logici, postoji potreba za efektivnom konstrukcijom dokaza, kako bismo mogli da verifikujemo dobijeni rezultat (proverom ispravnosti dokaza).

Jedan način, ne preterano efikasan, je da se sistematski nabrajaju svi ispravni dokazi (kao hijerarhijske strukture, odeljak 3.3.3). Ako je formula F teorema, pre ili kasnije ćemo doći do dokaza u čijem se korenu nalazi F . Ukoliko F nije teorema, takav dokaz ne postoji, pa se ovaj postupak nikada neće završiti. Otuda je opisani postupak jedna procedura poluodlučivanja.

U računarstvu se razmatraju i znatno efikasniji postupci pronalaženja i konstrukcije dokaza.

5.2 Logičke teorije

U ovom poglavlju podrazumevaćemo da je fiksiran neki deduktivni sistem $\mathcal{R}$ u logici prvog reda koji ima svojstvo saglasnosti i potpunosti (poput prirodne dedukcije). Otuda će oznaka $\vdash$ značiti $\vdash_{\mathcal{R}}$ i to nećemo dalje posebno naglašavati.

Definicija 5.1. *Teorija prvog reda $\mathcal{T}$ zadata skupom aksioma $\mathcal{A}_{\mathcal{T}}$ je skup svih rečenica koje se mogu dokazati iz $\mathcal{A}_{\mathcal{T}}$ u datom deduktivnom sistemu, tj. $\mathcal{T} = \{F \mid \mathcal{A}_{\mathcal{T}} \vdash F\}$. Ako je $F \in \mathcal{T}$, tada kažemo da je F teorema teorije $\mathcal{T}$ i označavamo sa $\vdash_{\mathcal{T}} F$.*

Napomena 5.7. Skup aksioma $\mathcal{A}_{\mathcal{T}}$ teorije $\mathcal{T}$ može biti konačan ili prebrojivo beskonačan.

Napomena 5.8. Jasno je da je svaka aksioma teorije $\mathcal{T}$ ujedno i njena teorema, tj. $\mathcal{A}_{\mathcal{T}} \subseteq \mathcal{T}$. Pored toga, teorija sadrži i sve rečenice koje se iz aksioma mogu dokazati. Drugim rečima, teorija $\mathcal{T}$ je *deduktivno zatvorenje* svog skupa aksioma.

Napomena 5.9. Skup aksioma $\mathcal{A}_{\mathcal{T}}$ teorije $\mathcal{T}$ ne treba mešati sa skupom aksioma deduktivnog sistema. Aksiome deduktivnog sistema određuju koje se formule mogu dokazati u deduktivnom sistemu bez ikakvih dodatnih pretpostavki, tj. koje su formule *teoreme deduktivnog sistema*. Sa druge strane, aksiome teorije su, posmatrano iz ugla deduktivnog sistema, dodatne pretpostavke koje možemo koristiti prilikom izvođenja dokaza. Samim tim, *teoreme teorije $\mathcal{T}$* će, pored teorema deduktivnog sistema, biti i mnoge druge formule koje se ne mogu izvesti samo iz aksioma deduktivnog sistema, ali se mogu izvesti uz pomoć aksioma teorije kao dodatnih pretpostavki.

Napomena 5.10. Specijalno, skup aksioma $\mathcal{A}_{\mathcal{T}}$ može biti i prazan. Tada je $\vdash_{\mathcal{T}} F$ ekvivalentno sa $\vdash F$, tj. skup teorema teorije $\mathcal{T}$ se poklapa sa skupom teorema

deduktivnog sistema. Ovakva „prazna” teorija se često naziva i *čist predikatski račun prvog reda*.

Definicija 5.2. Rečenica F je *deduktivna posledica* skupa rečenica Γ u teoriji $\mathcal{T}$ (u oznaci $\Gamma \vdash_{\mathcal{T}} F$) ako važi $\mathcal{A}_{\mathcal{T}}, \Gamma \vdash F$.

Napomena 5.11. Deduktivna posledica u teoriji je uopštenje pojma deduktivne posledice u deduktivnom sistemu, gde sada možemo, pored formula iz Γ , koristiti i aksiome teorije $\mathcal{T}$ kao dodatne pretpostavke prilikom izvođenja dokaza.

Definicija 5.3. Teorija $\mathcal{T}$ zadata skupom aksioma $\mathcal{A}_{\mathcal{T}}$ je *konzistentna* (ili *neprotivrečna*) ako ne postoji rečenica A takva da je i $A \in \mathcal{T}$ i $\neg A \in \mathcal{T}$. U suprotnom, teorija je *nekonzistentna*.

Napomena 5.12. Ako bi postojala rečenica takva da je $A \in \mathcal{T}$ i $\neg A \in \mathcal{T}$, to bi značilo da je moguće konstruisati dokaze formula A i $\neg A$ koji koriste aksiome $\mathcal{A}_{\mathcal{T}}$ kao pretpostavke. Međutim, iz ova dva dokaza bi se mogao konstruisati dokaz formule $\perp$, tj. izvesti kontradikcija (npr. u prirodnoj dedukciji bismo to uradili primenom pravila eliminacije negacije iz A i $\neg A$). Slično, ako se iz aksioma teorije $\mathcal{T}$ može izvesti $\perp$, tada se dalje može izvesti bilo koja formula A (npr. u prirodnoj dedukciji primenom pravila $\perp E$), tj. sve formule bi bile dokazive u teoriji $\mathcal{T}$. Specijalno, za proizvoljnu formulu A mogli bismo da dokažemo i A i $\neg A$. Dakle, teorija je nekonzistentna akko važi $\vdash_{\mathcal{T}} \perp$. Ponekad se i ovo uzima za definiciju nekonzistentnosti.

Definicija 5.4. *Model* teorije $\mathcal{T}$ zadate skupom aksioma $\mathcal{A}_{\mathcal{T}}$ je bilo koja $\mathcal{L}$ -struktura $\mathcal{D}$ koja zadovoljava sve aksiome teorije $\mathcal{A}_{\mathcal{T}}$.

Narednu poznatu teoremu navodimo bez dokaza.

Teorema 5.7. *Teorija $\mathcal{T}$ je konzistentna akko ima bar jedan model.*

Definicija 5.5. Formula F je *zadovoljiva u teoriji $\mathcal{T}$* ako je tačna u bar jednom modelu teorije $\mathcal{T}$. Formula F je *valjana u teoriji $\mathcal{T}$* (u oznaci $\models_{\mathcal{T}} F$) ako je tačna u svim modelima teorije $\mathcal{T}$. Formula F je *logička posledica skupa formula Γ u teoriji $\mathcal{T}$* (u oznaci $\Gamma \models_{\mathcal{T}} F$) ako je $\mathcal{A}(\mathcal{T}), \Gamma \models F$. Formule F i G su *logički ekvivalentne u teoriji $\mathcal{T}$* (u oznaci $F \equiv_{\mathcal{T}} G$) ako je $F \models_{\mathcal{T}} G$ i $G \models_{\mathcal{T}} F$, tj. ako imaju iste vrednosti u svim modelima teorije $\mathcal{T}$.

Teorema 5.8. *Važi $\Gamma \vdash_{\mathcal{T}} F$ akko $\Gamma \models_{\mathcal{T}} F$. Specijalno, F je teorema teorije $\mathcal{T}$ akko je logički valjana u teoriji $\mathcal{T}$, tj. $\vdash_{\mathcal{T}} F$ akko $\models_{\mathcal{T}} F$.*

Prethodna teorema sledi iz saglasnosti i potpunosti deduktivnog sistema, kao i gornjih definicija.

Na kraju navodimo još jednu teoremu bez dokaza koja govori o izražajnosti logike prvog reda, u smislu (ne)mogućnosti da se aksiomama fiksira kardinalnost domena modela.

Teorema 5.9 (Skolem-Lovenhajmova teorema). *Neka je $\mathcal{T}$ proizvoljna teorija prvog reda. Ako je $\mathcal{T}$ konzistentna, tada ona ima model bilo koje beskonačne kardinalnosti $\mathcal{K}$.*

Posledica 5.3. *Svaka konzistentna teorija prvog reda ima prebrojiv model.*

Posledica 5.4. *Teorija $\mathcal{T}$ ne može biti kategorična, tj. ne može fiksirati jedinstven model, do na izomorfizam.*

Dakle, u logici prvog reda nije moguće fiksirati interpretaciju aksiomatski zadate logičke teorije na jednu, unapred određenu strukturu. Uvek će postojati suštinski različite, međusobno neizomorfne strukture (čak različitih kardinalnosti) koje zadovoljavaju date aksiome. Najviše što se može postići je da te strukture makar budu *elementarno ekvivalentne*. Za dve strukture kažemo da su elementarno ekvivalentne ako je proizvoljna rečenica A tačna u jednoj strukturi akko je tačna u drugoj. To znači da se te strukture, iako neizomorfne, logički ponašaju isto, tj. slažu se po pitanju *istinitosti* tvrđenja koja se na jeziku $\mathcal{L}$ mogu izraziti.

Definicija 5.6. Teorija $\mathcal{T}$ nad jezikom $\mathcal{L}$ je *potpuna* akko za svaku rečenicu A važi da ili je $\vdash_{\mathcal{T}} A$ ili je $\vdash_{\mathcal{T}} \neg A$.

Napomena 5.13. Pojam *potpunosti* u smislu potpune teorije ne treba mešati sa pojmom potpunosti deduktivnog sistema. Potpunost deduktivnog sistema označava da je u tom deduktivnom sistemu moguće dokazati svaku valjanu formulu. Dakle, ovaj pojam predstavlja spoj između dedukcije i semantike. Sa druge strane, potpunost teorije je čisto deduktivni pojam koji govori o deduktivnoj moći teorije, tj. deduktivnog sistema „ojačanog” skupom aksioma teorije.

Primer 5.7. Čist predikatski račun nije potpuna teorija, s obzirom da npr. ni formula $\forall x.p(x)$ ni formula $\neg(\forall x.p(x))$ nisu dokazive (jer nisu valjane, a pretpostavljamo da je deduktivni sistem saglasan, tj. može dokazati samo valjane formule).

Dodavanjem aksioma teorije se povećava broj rečenica koje se mogu dokazati u deduktivnom sistemu. Pitanje je da li je dati skup aksioma dovoljan da za svaku rečenicu A možemo dokazati ili A ili $\neg A$ (ali nikako ne oba, jer bi tada teorija bila nekonzistentna)? Dakle, želimo da upotpunimo deduktivnu moć teorije, a da pritom ostanemo konzistentni. O važnost potpunosti teorije prvog reda govori sledeća teorema.

Teorema 5.10. *Teorija $\mathcal{T}$ je potpuna akko su joj svi modeli međusobno elementarno ekvivalentni.*

Dokaz. Ako je teorija potpuna, tada se za svaku rečenicu A može dokazati ili A ili $\neg A$. Pretpostavimo, određenosti radi, da je A dokazivo u teoriji. Na osnovu teoreme 5.8, sledi da ono što je dokazivo mora biti tačno u svim modelima teorije. Dakle, rečenica A je tačna u svim modelima teorije, a rečenica $\neg A$ je netačna u svim modelima teorije. Dakle, svaka rečenica ima istu interpretaciju u svim modelima teorije, pa su svi modeli elementarno ekvivalentni.

Obratno, ako su svi modeli elementarno ekvivalentni, onda za proizvoljnu rečenicu A važi ili da je ona tačna u svim modelima teorije, ili da je njena negacija $\neg A$ tačna u svim modelima teorije. Na osnovu teoreme 5.8, tada je ili A ili $\neg A$ dokaziva u teoriji $\mathcal{T}$. Dakle, teorija je potpuna. $\square$

Imajući u vidu ranije napomene, iz prethodne teoreme zaključujemo da je potpunost teorije najviše što možemo očekivati od teorije prvog reda. Na žalost, ispostavlja se da ovo nije uvek moguće, tj. nije moguće svaki skup aksioma dopuniti do potpune teorije (a da se ne naruši konzistentost). O tome ćemo detaljnije govoriti kasnije.

Definicija 5.7. Teorija $\mathcal{T}$ je *odlučiva* ako je problem $\vdash_{\mathcal{T}} F$, gde je F proizvoljna formula, odlučiv.

Primetimo da teorija može biti odlučiva, iako sama logika prvog reda nije odlučiva. Ovo je zato što je problem $\vdash_{\mathcal{T}} F$, odnosno $\mathcal{A}_{\mathcal{T}} \vdash F$ potproblem opšteg problema $\Delta \vdash F$ (jer je $\mathcal{A}_{\mathcal{T}}$ fiksiran, unapred odabran skup aksioma, dok je u opštem slučaju skup Δ proizvoljan skup). Potproblem često može biti jednostavniji od opšteg problema.

Definicija 5.8. Za teoriju $\mathcal{T}$ kažemo da je *aksiomska* ako je njen skup aksioma $\mathcal{A}_{\mathcal{T}}$ rekurzivan.

Rekurzivnost skupa aksioma znači da za proizvoljnu formulu možemo da utvrdimo da li je aksioma teorije ili ne. Ukoliko je skup aksioma konačan, on je svakako rekurzivan, pa će teorija sa konačnim brojem aksioma uvek biti aksiomska. Sa druge strane, kod beskonačnih skupova aksioma, neophodno je da postoji mogućnost efektivnog utvrđivanja da li je nešto aksioma ili ne. Tipično, beskonačni skupovi aksioma se zadaju kao *scheme* aksioma, tj. obrasci koji definišu oblik i strukturu formule koja se smatra aksiomom. Otuda je moguće efektivno ispitati da li se data formula uklapa u date obrasce, tj. da li je *instanca* aksiomske sheme ili ne.

Teorema 5.11. *Ako je teorija $\mathcal{T}$ aksiomska i potpuna, ona je odlučiva.*

Dokaz. Jedna jednostavna intuitivna procedura odlučivanja bi se sastojala u rekurzivnom nabranjanju svih dokaza sa pretpostavkama iz skupa $\mathcal{A}_{\mathcal{T}}$. Ovo je moguće zato što je skup $\mathcal{A}_{\mathcal{T}}$ rekurzivan, pa je i sam rekurzivno nabrojiv. Kako je teorija potpuna, sledi da ili postoji dokaz rečenice A ili dokaz njene negacije $\neg A$. Otuda ćemo pre ili kasnije naići na jedan od ta dva dokaza, odakle ćemo utvrditi da li je $\vdash A$ ili ne. $\square$

5.2.1 Peanova aritmetika

U ovom odeljku dajemo primer jedne logičke teorije koja je od fundamentalnog značaja kako za formalno zasnivanje matematike, tako i za teorijsko računarstvo. U pitanju je teorija aritmetike koja logički opisuje uobičajenu, intuitivnu strukturu prirodnih brojeva.

Neka je data signatura $\mathcal{L}$ koja sadrži simbol konstante 0, funkcijski simbol S arnosti 1, funkcijske simbole $+$ i $\cdot$ arnosti 2, kao i predikatski simbol $=$ arnosti 2. *Penaova aritmetika prvog reda* $\mathcal{P}$ je logička teorija zadata sledećim skupom aksioma:

1. $\forall x. x = x$
2. $\forall xy. x = y \Rightarrow y = x$
3. $\forall xyz. x = y \wedge y = z \Rightarrow x = z$
4. $\forall xy. x = y \Rightarrow S(x) = S(y)$
5. $\forall x_1 x_2 y_1 y_2. x_1 = x_2 \wedge y_1 = y_2 \Rightarrow x_1 + y_1 = x_2 + y_2$
6. $\forall x_1 x_2 y_1 y_2. x_1 = x_2 \wedge y_1 = y_2 \Rightarrow x_1 \cdot y_1 = x_2 \cdot y_2$

7. $\forall x. \neg(0 = S(x))$
8. $\forall xy. S(x) = S(y) \Rightarrow x = y$
9. $\phi(0) \wedge (\forall x. \phi(x) \Rightarrow \phi(S(x))) \Rightarrow (\forall x. \phi(x))$, za svaku formulu $\phi(x)$
10. $\forall x. x + 0 = x$
11. $\forall xy. x + S(y) = S(x + y)$
12. $\forall x. x \cdot 0 = 0$
13. $\forall xy. x \cdot S(y) = x \cdot y + x$

Napomena 5.14. Aksiome 1-6 predstavljaju *aksiome jednakosti*. Ovim se zahteva da za simbol jednakosti ($=$) važi refleksivnost, simetričnost i tranzitivnost, kao i saglasnost (*kongruentnost*) sa funkcijskim simbolima S , $+$ i $\cdot$. Aksiome 7 i 8 zahtevaju da različito konstruisani objekti budu različiti: 0 nije jednaka $S(x)$ ni za jedno x , a ako su x i y različiti, tada su i $S(x)$ i $S(y)$ različiti. Element $S(x)$ nazivaćemo *sledbenik od x* . Aksioma 9 predstavlja tzv. *aksiomatsku shemu indukcije*. Formula $\phi(x)$ je formula koja u sebi sadrži slobodna pojavljivanja promenljive x , pri čemu je $\phi(0)$ kraći zapis za $\phi(x)[x \rightarrow 0]$, a $\phi(S(x))$ je kraći zapis za $\phi(x)[x \rightarrow S(x)]$. Intuitivno, $\phi(x)$ predstavlja neko *svojstvo* koje neki element x ispunjava. Aksioma 9 kaže da ako 0 ima to svojstvo, i ako iz činjenice da x ima to svojstvo sledi da i $S(x)$ ima to svojstvo, onda svi elementi imaju to svojstvo. Ovo odgovara uobičajenom principu matematičke indukcije u skupu prirodnih brojeva. Aksiome 10 i 11 definišu operaciju sabiranja: $x + 0$ je uvek jednako x , dok se dodavanje sledbenika od y svodi na dodavanje y i određivanje sledbenika tako dobijenog zbira. Slično, aksiome 12 i 13 definišu operaciju množenja: proizvod $x \cdot 0$ je uvek nula, dok se proizvod x sa sledbenikom od y definiše tako što pomnožimo sa y , a zatim dodamo x .

Primetimo da je aksioma 9 zapravo aksiomatska shema iz koje možemo generisati prebrojivo mnogo instanci ove aksiome (za svaku formulu $\phi(x)$ nad jezikom $\mathcal{L}$ sa slobodnom promenljivom x po jednu). Ipak, uvek je moguće efektivno utvrditi da li je data formula instanca aksiome 9 ili ne. Otuda je skup aksioma rekurzivan, pa je Peanova aritmetika aksiomatska teorija.

Napomena 5.15. Uvodimo oznake $\bar{0} = 0$, $\bar{1} = S(0)$, $\bar{2} = S(1)$, $\bar{3} = S(2)$, $\dots$, koje nazivamo *numeralima*.

Primer 5.8. Prisetimo da iz aksioma jednakosti uvek možemo zaključiti da iz npr. $s_1 = s_2$ sledi $s_1 + t = s_2 + t$ za proizvoljne termine s_1 , s_2 i t . Slično važi i za funkcijske simbole S i $\cdot$. Evo dokaza u prirodnoj dedukciji:

$$\frac{\frac{s_1 = s_2 \vdash_{\mathcal{P}} s_1 = s_2 \quad \frac{s_1 = s_2 \vdash_{\mathcal{P}} \forall x. x = x}{s_1 = s_2 \vdash_{\mathcal{P}} t = t} \vee E}{s_1 = s_2 \vdash_{\mathcal{P}} s_1 = s_2 \wedge t = t} \wedge I \quad \frac{s_1 = s_2 \vdash_{\mathcal{P}} \forall x_1 x_2 y_1 y_2. x_1 = x_2 \wedge y_1 = y_2 \Rightarrow x_1 + y_1 = x_2 + y_2}{s_1 = s_2 \vdash_{\mathcal{P}} s_1 = s_2 \wedge t = t \Rightarrow s_1 + t = s_2 + t} \vee E}{s_1 = s_2 \vdash_{\mathcal{P}} s_1 + t = s_2 + t} \Rightarrow E$$

pri čemu $\vdash_{\mathcal{P}}$ podrazumeva da se među pretpostavkama sa leve strane pored $s_1 = s_2$ nalaze i sve aksiome Peanove aritmetike. Dakle, „jednako se može zameniti jednakim”. Slično, može se dokazati da iz $s = t$ i $t = r$ sledi $s = r$:

$$\frac{\frac{s = t, t = r \vdash_{\mathcal{P}} s = t \quad s = t, t = r \vdash_{\mathcal{P}} t = r}{s = t, t = r \vdash_{\mathcal{P}} s = t \wedge t = r} \wedge I \quad \frac{s = t, t = r \vdash_{\mathcal{P}} \forall xyz. x = y \wedge y = z \Rightarrow x = z}{s = t, t = r \vdash_{\mathcal{P}} s = t \wedge t = r \Rightarrow s = r} \forall E}{s = t, t = r \vdash_{\mathcal{P}} s = r} \Rightarrow E$$

Ova dva tvrđenja nam opravdavaju tzv. *zamenske dokaze*: da bismo dokazali da važi neka jednakost $s = t$, možemo da krenemo od s i da u njemu sukcesivno zamenjujemo neke podtermove njima jednakim termovima sve dok ne stignemo do terma t .

Primer 5.9. Dokažemo da je $\bar{2} + \bar{2} = \bar{4}$, tj. da važi $S(S(0)) + S(S(0)) = S(S(S(S(0))))$. Zaista, imamo zamenski dokaz: $S(S(0)) + S(S(0)) = S(S(S(0)) + S(0)) = S(S(S(S(0)) + 0)) = S(S(S(S(0))))$. Pritom, kao opravdanja zamena u prethodnom nizu koristili smo instance aksioma $\forall xy. x + S(y) = S(x + y)$ i $\forall x. x + 0 = x$.

Napomena 5.16. Izraz $s \neq t$ gde su s i t proizvoljni termovi je kraći zapis za $\neg(s = t)$.

Primer 5.10. Dokažimo da važi $\forall x. x \neq S(x)$. Za dokaz ovog tvrđenja korišćićemo aksiomu indukcije, tj. aksiomu 9. Gornje tvrđenje možemo zapisati u obliku $\forall x. \phi(x)$, gde je $\phi(x)$ formula $x \neq S(x)$. Važi $\phi(0)$, odnosno $0 \neq S(0)$, na osnovu aksiome 7, eliminacijom univerzalnog kvantifikatora za $x \rightarrow 0$. Pretpostavimo da za neko proizvoljno x važi $\phi(x)$, odnosno $x \neq S(x)$. Ako bi važilo $S(x) = S(S(x))$, primenom aksiome 8, eliminacijom univerzalnog kvantifikatora za $y \rightarrow S(x)$, a zatim eliminacijom implikacije mogli bismo da dokažemo $x = S(x)$, odakle iz $x \neq S(x)$ (induktivna pretpostavka) sledi kontradikcija. Oдавde možemo uvođenjem negacije da izvedemo $S(x) \neq S(S(x))$, tj. $\phi(S(x))$. Kako smo za proizvoljno x pod pretpostavkom $\phi(x)$ dokazali $\phi(S(x))$, uvođenjem implikacije i univerzalnog kvantifikatora dobijamo $\forall x. \phi(x) \Rightarrow \phi(S(x))$. Sada uvođenjem konjunkcije imamo $\phi(0) \wedge (\phi(x) \Rightarrow \phi(S(x)))$, pa iz aksiome 9, eliminacijom implikacije sledeći $\forall x. \phi(x)$, tj. $\forall x. x \neq S(x)$.

Primer 5.11. Dokažimo da važi $\forall x. 0 + x = x$. Ovo ćemo dokazati pomoću aksiome 9 (aksioma indukcije). Neka je formula $\phi(x)$ upravo formula $0 + x = x$. Svakako važi $\phi(0)$, tj. $0 + 0 = 0$, na osnovu aksiome $\forall x. x + 0 = x$. Pretpostavimo da važi $\phi(x)$ za neko x , tj. važi $0 + x = x$. Treba dokazati da važi formula $\phi(S(x))$, tj. da je $0 + S(x) = S(x)$. Kako je $0 + S(x) = S(0 + x) = S(x)$ (jer je $0 + x = x$ na osnovu induktivne pretpostavke), sledeći da važi $\forall x. \phi(x) \Rightarrow \phi(S(x))$. Prema aksiomi 9, imamo da je $\forall x. 0 + x = x$.

Primer 5.12. Posledica prethodnog primera je da važi $\forall x. 0 + x = x + 0$. Naime, za proizvoljno fiksirano y važi $y + 0 = y$ (pravilom eliminacije univerzalnog kvantifikatora iz aksiome 10), kao i $0 + y = y$ na osnovu prethodnog primera. Na osnovu simetričnosti imamo da je $y = y + 0$, a na osnovu tranzitivnosti da je $0 + y = y + 0$. Kako je y proizvoljno i fiksirano, možemo primeniti pravilo uvođenja univerzalnog kvantifikatora, tj. važi $\forall x. 0 + x = x + 0$.

Primer 5.13. Dokažimo da važi $\forall xy. S(y) + x = S(y + x)$. Zapišimo ovo tvrđenje u obliku $\forall x. \phi(x)$, gde je $\phi(x)$ formula $\forall y. S(y) + x = S(y + x)$. Dokažimo ovo tvrđenje indukcijom, tj. pomoću aksiome 9. Formula $\phi(0)$, tj. $\forall y. S(y) + 0 = S(y + 0)$ važi, jer za proizvoljno fiksirano y važi $S(y) + 0 = S(y)$ (na osnovu aksiome 10, eliminacijom univerzalnog kvantifikatora), a iz $y + 0 = y$ sledi $S(y + 0) = S(y)$, pa na osnovu simetričnosti i tranzitivnosti jednakosti važi $S(y) + 0 = S(y + 0)$. Otuda, uvođenjem univerzalnog kvantifikatora

važi $\forall y.S(y) + 0 = S(y + 0)$. Dalje, pretpostavimo da važi $\phi(x)$ za neko x , tj. da važi $\forall y.S(y) + x = S(y + x)$. Imamo da za proizvoljno fiksirano y važi $S(y) + S(x) = S(S(y) + x) = S(S(y + x))$ na osnovu induktivne pretpostavke. Dalje je $S(S(y + x)) = S(y + S(x))$, jer je $S(y + x) = y + S(x)$ (na osnovu aksiome 11, eliminacijom univerzalnog kvantifikatora i primenom simetričnosti). Dakle, imamo da je $S(y) + S(x) = S(y + S(x))$ za proizvoljno fiksirano y . Otuda, uvođenjem univerzalnog kvantifikatora imamo $\forall y.S(y) + S(x) = S(y + S(x))$, tj. važi $\phi(S(x))$. Dakle, za proizvoljno x , iz pretpostavke $\phi(x)$ smo dokazali $\phi(S(x))$, odakle uvođenjem implikacije i univerzalnog kvantifikatora dokazujemo $\forall x.\phi(x) \Rightarrow \phi(S(x))$. Na osnovu aksiome 9, sada zaključujemo da važi $\forall x.\phi(x)$, tj. $\forall xy.S(y) + x = S(y + x)$.

Primer 5.14. Dokažimo da važi $\forall xy.x + y = y + x$. Ovo je isto što i $\forall x.\forall y.x + y = y + x$, odnosno $\forall x.\phi(x)$, gde je $\phi(x)$ formula $\forall y.x + y = y + x$, sa slobodnom promenljivom x . U dokazu se ponovo služimo indukcijom. $\phi(0)$ je isto što i $\forall y.0 + y = y + 0$, što je dokazano u primeru 5.12. Pretpostavimo da za neko proizvoljno x važi $\phi(x)$, tj. važi $\forall y.x + y = y + x$. Za proizvoljno y važi $S(x) + y = S(x + y)$ na osnovu primera 5.13. Prema induktivnoj pretpostavci i na osnovu simetričnosti važi $y + x = x + y$, pa je $S(x) + y = S(x + y) = S(y + x) = y + S(x)$ (poslednja jednakost sledi iz aksiome 11, primenom simetričnosti). Kako je y proizvoljno, uvođenjem univerzalnog kvantifikatora imamo $\forall y.S(x) + y = y + S(x)$, tj. važi $\phi(S(x))$. Kako smo za proizvoljno x iz $\phi(x)$ dokazali $\phi(S(x))$, važi $\forall x.\phi(x) \Rightarrow \phi(S(x))$, pa na osnovu aksiome 9 imamo da važi $\forall x.\phi(x)$, tj. važi $\forall xy.x + y = y + x$.

Uočimo strukturu čiji je domen skup $\mathbb{N} = \{0, 1, \dots\}$ *prirodnih brojeva*. Pretpostavimo da ova struktura simbole iz signature interpretira na sledeći način:

- 0 se interpretira kao prirodan broj *nula* (tj. kao 0).
- = se interpretira kao *relacija jednakosti* nad $\mathbb{N}$, tj. kao relacija $= = \{(x, x) \mid x \in \mathbb{N}\}$.
- S se interpretira kao funkcija *sledbenika* u skupu $\mathbb{N}$, tj. kao funkcija $s : \mathbb{N} \rightarrow \mathbb{N}$ definisana sa $s(x) = x + 1$ za svako x .
- + se interpretira kao operacija *sabiranja* + nad skupom prirodnih brojeva
- · se interpretira kao operacija *množenja* · nad skupom prirodnih brojeva.

Može se pokazati da ova struktura zadovoljava sve aksiome teorije $\mathcal{P}$. Nazivamo je i *standardni model aritmetike* i označavamo ga takođe sa $\mathbb{N}$. U njemu će se numeral $\bar{n}$ interpretirati kao prirodan broj n . Iz postojanja ovog modela sledi sledeća teorema.

Teorema 5.12. *Peanova aritmetika $\mathcal{P}$ je konzistentna.*

Standardni model $\mathbb{N}$ je, naravno, prebrojiv. Kao posledica Skolem-Lovenhajmove teoreme, postojeće i *nestandardni modeli* Peanove aritmetike čiji će domeni biti neprebrojivo beskonačni.

Napomena 5.17. U prethodnom razmatranju, konzistentnost Peanove aritmetike smo zaključili na osnovu postojanja standardnog modela $\mathbb{N}$. Ovaj model se konstruiše u okviru *teorije skupova*: broj 0 se identifikuje sa praznim skupom $\emptyset$, a sledbenik broja n se identifikuje sa skupom $n \cup \{n\}$ (tj. broj 1 se identifikuje sa

skupom $\{0\}$, broj 2 sa skupom $\{0, 1\}$, broj 3 sa skupom $\{0, 1, 2\}$ i sl.). Uopšte, dokazivanje konzistentnosti teorije $\mathcal{T}$ na osnovu postojanja modela uvek podrazumeva da je taj model konstruisan u okviru neke druge teorije $\mathcal{T}'$, za koju pretpostavljamo da je konzistentna. Dakle, konzistentnost teorije $\mathcal{T}$ je relativna, do na konzistentnost teorije $\mathcal{T}'$. Da bismo dokazali apsolutnu konzistentnost teorije $\mathcal{T}$, potrebno bi bilo da možemo da dokažemo konzistentnost teorije $\mathcal{T}$ u okviru nje same. O ovom problemu u slučaju Peanove aritmetike govorili smo u narednom poglavlju.

5.3 Gedelove teoreme

5.3.1 Izrazivost izračunljivih funkcija na jeziku aritmetike

Neka je f proizvoljna parcijalna funkcija reda k . Pod *grafom* funkcije f podrazumevamo relaciju $\rho_f \subseteq \mathbb{N}^k \times \mathbb{N}$ definisanu sa $\rho_f = \{(x_1, \dots, x_k), y \mid f(x_1, \dots, x_k) = y\}$. Za funkciju f kažemo da je *izraziva* na jeziku Peanove aritmetike $\mathcal{P}$ ako postoji formula $R_f(x_1, \dots, x_k, y)$ sa slobodnim promenljivama $x_1, \dots, x_k, y$ nad signaturom teorije $\mathcal{P}$ takva da svaki izbor brojeva $n_1, \dots, n_k, m \in \mathbb{N}$ važi da je $R_f(\bar{n}_1, \dots, \bar{n}_k, \bar{m})$ tačna u strukturi $\mathbb{N}$ akko $(n_1, \dots, n_k, m) \in \rho_f$.

Primer 5.15. Funkcija $z = \text{mod}(x, y)$ koja izračunava ostatak pri deljenju x sa y se može izraziti na sledeći način:

$$\exists q. x = y \cdot q + z \wedge z < y$$

Ova formula ima slobodne promenljive x, y, z i u standardnom modelu aritmetike je tačna akko je $z = \text{mod}(x, y)$.

Napomena 5.18. U prethodnom primeru, koristili smo simbol $<$ koji, formalno, nije deo signature teorije $\mathcal{P}$. Ipak, relaciju $z < y$ je lako izraziti na jeziku Peanove aritmetike:

$$\exists k. z + S(k) = y$$

Ova formula je tačna u $\mathbb{N}$ akko je $z < y$ tačno u $\mathbb{N}$. Na sličan način se mogu izraziti i drugi relacioni operatori ($\leq, >, \geq$). U nastavku ćemo, zbog jednostavnosti, uvek koristiti ove relacione simbole kao da su deo jezika, uz napomenu da ih uvek možemo eliminisati na opisan način i svesti formulu na „čistu Peanovu aritmetiku”.

Primer 5.16. Slično prethodnom primeru, funkciju $z = \text{div}(x, y)$ možemo izraziti kao:

$$\exists r. x = y \cdot z + r \wedge r < y$$

Ova formula je tačna za neko x, y, z akko je $z = \text{div}(x, y)$.

Lema 5.1. *Neka su date funkcije h reda m i $g_1, \dots, g_m$ reda k . Ako su ove funkcije izrazive na jeziku aritmetike, tada je to i funkcije $f = \text{Sub}(h, g_1, \dots, g_m)$.*

Dokaz. Neka je $H(u_1, \dots, u_m, y)$ formula koja izražava funkciju $y = h(u_1, \dots, u_m)$ i neka su $G_i(x_1, \dots, x_k, z)$ formule koje izražavaju funkcije $z = g_i(x_1, \dots, x_k)$. Formula:

$$\exists u_1 \dots u_m. H(u_1, \dots, u_m, y) \wedge g_1(x_1, \dots, x_k, u_1) \wedge \dots \wedge g_m(x_1, \dots, x_k, u_m)$$

izražava funkciju f . □

Primer 5.17. Funkcija definisana sa:

$$\beta(x_1, x_2, x_3) = \text{mod}(x_1, 1 + (x_3 + 1) \cdot x_2)$$

je izraziva na jeziku aritmetike, s obzirom da se može dobiti supstitucijom izrazive funkcije $1 + (x_3 + 1) \cdot x_2$ u funkciji mod za koju smo pokazali da je izraziva. Dakle, postoji formula $R_\beta(x_1, x_2, x_3, y)$ na jeziku aritmetike koja je tačna akko je $y = \beta(x_1, x_2, x_3)$.

Funkcija iz prethodnog primera je u literaturi poznata kao *Gedelova β funkcija*. Jedno njeno zanimljivo svojstvo opisano je sledećom lemom.

Lema 5.2. *Neka je data proizvoljna sekvenca prirodnih brojeva $a_0, \dots, a_n$. Tada postoje s i t takvi da važi:*

$$\beta(s, t, i) = a_i$$

za svako $i \in \{0, \dots, n\}$.

Dokaz. Dokaz se zasniva na činjenici da postoji t takvo da su brojevi $1 + t, 1 + 2 \cdot t, 1 + 3 \cdot t, \dots, 1 + (n + 1) \cdot t$ uzajamno prosti (što ovde nećemo dokazivati). Otuda, na osnovu *Kineske teoreme o ostacima* moguće je pronaći s takvo da važi:

$$\begin{aligned} s \bmod (1 + t) &= a_0 \\ s \bmod (1 + 2 \cdot t) &= a_1 \\ \dots \\ s \bmod (1 + (n + 1) \cdot t) &= a_n \end{aligned}$$

□

Prethodna lema je značajna zato što omogućava da *kodiramo* proizvoljne sekvence brojeva $a_0, \dots, a_n$ parom brojeva s i t , pri čemu je to kodiranje izrazivo na jeziku aritmetike. Ovo omogućava da se dokaže sledeća lema.

Lema 5.3. *Neka su date funkcije g reda k i h reda $k + 2$. Ako su g i h izrazive na jeziku aritmetike, tada je i funkcija $f = \text{Rec}(g, h)$ takva.*

Dokaz. Neka je $G(x_1, \dots, x_k, y)$ formula koja izražava da je $y = g(x_1, \dots, x_k)$, a neka je formula $H(y, x_1, \dots, x_k, u, z)$ formula koja izražava da je $z = h(y, x_1, \dots, x_k, u)$. Imamo da je:

$$\begin{aligned} f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\ f(y + 1, x_1, \dots, x_k) &= h(y, x_1, \dots, x_k, f(y, x_1, \dots, x_k)) \end{aligned}$$

Da bismo izrazili da je $z = f(y, x_1, \dots, x_k)$, posmatrajmo sekvencu vrednosti:

$$\begin{aligned} a_0 &= f(0, x_1, \dots, x_k) \\ a_1 &= f(1, x_1, \dots, x_k) \\ \dots \\ a_y &= f(y, x_1, \dots, x_k) \end{aligned}$$

Prema prethodnoj lemi, postoje s i t takvi da je: $\beta(s, t, i) = a_i$ za svako $i \leq y$. Neka je data formula:

$$\exists st. \left(\begin{array}{l} G(x_1, \dots, x_k, \beta(s, t, 0)) \\ \forall i. i < y \Rightarrow H(i, x_1, \dots, x_k, \beta(s, t, i), \beta(s, t, S(i))) \\ z = \beta(s, t, y) \end{array} \right) \wedge$$

Ova formula izražava da je $z = f(y, x_1, \dots, x_k)$. □

Napomena 5.19. U dokazu prethodne leme smo u formuli koristili funkcijski simbol β koji ne postoji u signaturi aritmetike. Ovo smo uradili da bismo pojednostavili notaciju. To nije „varanje”, zato što smo uvek mogli da preformulišemo formulu tako da se koristi formula $R_\beta(s, t, i, u)$ kojom se izražava da je $u = \beta(s, t, i)$, umesto terma $\beta(s, t, i)$. Uopšte, za proizvoljnu izrazivu funkciju f , formulu:

$$A(x_1, \dots, x_n, f(x_1, \dots, x_n))$$

uvek možemo izraziti i kao:

$$\exists u. R_f(x_1, \dots, x_n, u) \wedge A(x_1, \dots, x_n, u)$$

gde je $R_f(x_1, \dots, x_n, u)$ formula koja izražava da je $u = f(x_1, \dots, x_n)$. Na ovaj način se možemo osloboditi funkcijskog simbola f i izraziti formulu na jeziku „čiste” aritmetike. Mi ćemo u praksi uglavnom koristiti funkcijsku notaciju zbog jednostavnosti, ali ćemo uvek imati u vidu da se ta notacija može lako transformisati na opisani način.

Iz prethodnih lema sledi važna teorema.

Teorema 5.13. *Svaka primitivno rekurzivna funkcija je izraziva na jeziku Peanove aritmetike.*

Da bismo izrazili proizvoljnu μ -rekurzivnu funkciju na jeziku aritmetike, potrebno je još izraziti i operator minimizacije. O ovome govori sledeća lema.

Lema 5.4. *Ako je funkcije $g(y, x_1, \dots, x_k)$ izraziva na jeziku Peanove aritmetike, tada je takva i funkcija:*

$$f(x_1, \dots, x_k) = \mu y [g(y, x_1, \dots, x_k) = 0]$$

Dokaz. Pretpostavimo da je $R_g(y, x_1, \dots, x_k, z)$ aritmetička formula koja izražava da je $z = g(y, x_1, \dots, x_k)$. Razmotrimo formulu:

$$R_g(y, x_1, \dots, x_k, 0) \wedge (\forall y'. y' < y \Rightarrow (\exists u. R_g(y', x_1, \dots, x_k, u) \wedge u \neq 0))$$

Ova formula je tačna za neko $x_1, \dots, x_k, y$ akko je $y = f(x_1, \dots, x_k)$. $\square$

Sada imamo sledeću teoremu.

Teorema 5.14. *Svaka μ -rekurzivna funkcija se može izraziti na jeziku Peanove aritmetike.*

Na kraju, povežimo izrazivost izračunljivih funkcija sa *dokazivošću* u Peanovoj aritmetici.

Teorema 5.15. *Neka je f bilo koja μ -izračunljiva funkcija reda k i neka za neke konkretne vrednosti $n_1, \dots, n_k, m \in \mathbb{N}$ važi da je $m = f(n_1, \dots, n_k)$. Tada važi:*

$$\vdash_{\mathcal{P}} R_f(\bar{n}_1, \dots, \bar{n}_k, \bar{m})$$

gde je $R_f(x_1, \dots, x_k, y)$ formula koja izražava da je $y = f(x_1, \dots, x_k)$.

Dokaz. (skica) Intuitivno, formula $R_f(\bar{n}_1, \dots, \bar{n}_k, \bar{m})$ će sadržati aritmetičke operacije $S, +$ i $\cdot$ primenjene nad konkretnim numeralima, nad kojima je jednostavno dokazati odgovarajuće identitete (setimo se kako smo dokazivali da je $\bar{2} + \bar{2} = \bar{4}$). Sa druge strane, formula može sadržati i egzistencijalne kvantifikatore, pri čemu se takvi delovi formule mogu dokazati uvođenjem konkretnog numerala (pravilo uvođenja egzistencijalnog kvantifikatora, primenjeno unatraške). Najzad, iz dokaza prethodnih lema vidimo da se univerzalni kvantifikatori uvek pojavljuju u ograničenoj formi (npr. $\forall k. k < n \Rightarrow \dots$, gde je n parametar koji se zamenjuje konkretnim numeralom). To znači da se dokazivanje takvih formula može svesti na dokazivanje konačno mnogo slučajeva. $\square$

Dokaz da je $\vdash_{\mathcal{P}} R_f(\bar{n}_1, \dots, \bar{n}_k, \bar{m})$ ćemo nazivati *dokaz izračunavanja* $m = f(n_1, \dots, n_m)$.

Primer 5.18. Dokažimo da je $\text{mod}(5, 2) = 1$. Potrebno je dokazati da važi:

$$\exists q. \bar{5} = \bar{2} \cdot q + \bar{1} \wedge (\exists k. \bar{1} + S(k) = \bar{2})$$

Da bismo dokazali ovu egzistencijalno kvantifikovanu formulu, dovoljno je dokazati potformulu za neko konkretno q . Zamenimo q sa $\bar{2}$, tj. dokažimo formulu:

$$\bar{5} = \bar{2} \cdot \bar{2} + \bar{1} \wedge (\exists k. \bar{1} + S(k) = \bar{2})$$

Prvi konjunkt se dokazuje jednostavno, sledećim zamenama zasnovanim na aksiomama Peanove aritmetike:

$$\begin{aligned} S(S(0)) \cdot S(S(0)) + S(0) &= \\ S(S(0)) \cdot S(0) + S(S(0)) + S(0) &= \\ S(S(0)) \cdot 0 + S(S(0)) + S(S(0)) + S(0) &= \\ 0 + S(S(0)) + S(S(0)) + S(0) &= \\ S(S(0)) + S(S(0)) + S(0) &= \\ S(S(S(0)) + S(0)) + S(0) &= \\ S(S(S(S(0))) + S(0)) &= \\ S(S(S(S(S(0)))) + 0) &= \\ S(S(S(S(S(S(0)))))) &= \end{aligned}$$

Za drugi konjunkt možemo ponovo primeniti pravilo uvođenja egzistencijalnog kvantifikatora, dokazujući potformulu za konkretan numeral 0:

$$\bar{1} + S(0) = \bar{2}$$

Ovo se opet lako dokazuje, s obzirom da su u pitanju konkretni numerali:

$$S(0) + S(0) = S(S(0) + 0) = S(S(0))$$

5.3.2 Kodiranje termova, formula i dokaza

Kao i sve hijerarhijske strukture, termovi, formule i dokazi u Peanovoj aritmetici se mogu kodirati prirodnim brojevima. Pritom, funkcije kodiranja su efektivno izračunljive i mogu se izraziti na jeziku aritmetike, kao što smo prikazali u prethodnom odeljku. Specijalno, sami kôdovi se mogu predstaviti odgovarajućim numeralima. Ovo nam daje vrlo zanimljivu mogućnost: da tvrđenja

o samoj aritmetici (tzv. *metatvrđenja*, poput „formula A je dokaziva”, „teorija je konzistentna” i sl.) izražavamo u obliku tvrđenja *u samoj aritmetici*.

Formalno, neka je sa $\lceil t \rceil$ označen numeral koji odgovara kôdu terma t nad jezikom Peanove aritmetike. Slično, neka $\lceil A \rceil$ predstavlja kôd formule A , a $\lceil D \rceil$ kôd dokaza D . Na jeziku aritmetike možemo izraziti tvrđenje: „dokaz sa kôdom y dokazuje formulu čiji je kôd x ”. Intuitivno, potrebno je dekodirati y , a zatim odrediti kôd formule u korenu dobijenog dokaza i uporediti ga sa x . Iz ove intuitivne izračunljivosti, prema Čerčovoj tezi sledi i formalna izračunljivost, a samim tim i izrazivost u Peanovoj aritmetici. Označimo formulu koja kodira ovo tvrđenje sa $proof(x, y)$. Sada formula $\exists y. proof(x, y)$ govori da „postoji dokaz formule čiji je kôd x ”, odnosno „formula čiji je kôd x je dokaziva”. Označimo ovu formulu sa $pr(x)$.

Može se dokazati sledeća lema.

Lema 5.5. *Za svaku rečenicu Peanove aritmetike važi da je $\vdash_P A$ akko $\vdash_P pr(\lceil A \rceil)$.*

Dokaz. Pretpostavimo da je $\vdash_P A$. To znači da postoji dokaz D_A formule A u Peanovoj aritmetici. Potrebno je dokazati formulu $pr(\lceil A \rceil)$, tj. formulu $\exists y. proof(\lceil A \rceil, y)$. Za ovo je dovoljno pronaći neki numeral $\bar{n}$ takav da je moguće dokazati $proof(\lceil A \rceil, \bar{n})$ (nakon toga ćemo samo primeniti pravilo uvođenja egzistencijalnog kvantifikatora). Uzmimo da je $\bar{n} = \lceil D_A \rceil$. Sada se dekodiranjem broja n može efektivno utvrditi da je kôd formule koju dokazuje dokaz kodiran sa n upravo jednak $\lceil A \rceil$. Dokaz ovog izračunavanja je upravo dokaz formule $proof(\lceil A \rceil, \lceil D_A \rceil)$.

Obratno, neka je $\vdash_P pr(\lceil A \rceil)$, tj. važi da je $\exists y. proof(\lceil A \rceil, y)$ dokazivo. S obzirom na saglasnost deduktivnog sistema, to znači da je formula $\exists y. proof(\lceil A \rceil, y)$ tačna u strukturi prirodnih brojeva $\mathbb{N}$, kao modelu Peanove aritmetike. Imajući u vidu značenje formule $proof(\lceil A \rceil, y)$ u ovoj strukturi, to znači da postoji neko y koje kodira dokaz formule A , tj. $\vdash_P A$. $\square$

Lema 5.6. *Za svake dve rečenice A i B nad jezikom Peanove aritmetike važi da ako je $\vdash_P pr(\lceil A \Rightarrow B \rceil)$, tada je i $\vdash_P pr(\lceil A \rceil) \Rightarrow pr(\lceil B \rceil)$.*

Dokaz. Na osnovu prethodne leme, iz $\vdash_P pr(\lceil A \Rightarrow B \rceil)$ sledi $\vdash_P A \Rightarrow B$. Ako bismo imali dokaz formule A , tada bismo eliminacijom implikacije mogli dokazati formulu B . Dakle, iz dokazivosti A sledi dokazivost B . Formalno, pretpostavimo da važi $pr(\lceil A \rceil)$, tj. formula $\exists y. proof(\lceil A \rceil, y)$. Primenimo pravilo eliminacije egzistencijalnog kvantifikatora – pretpostavimo da imamo neko z za koje važi $proof(\lceil A \rceil, z)$. Polazeći od tog z možemo efektivno izračunati kôd dokaza z' koji se dobija od dokaza z primenom pravila eliminacije implikacije (tj. dokaz formule B). Time smo dokazali $proof(\lceil B \rceil, z')$, odakle uvođenjem egzistencijalnog kvantifikatora dobijamo $\exists y. proof(\lceil B \rceil, y)$, tj. $pr(\lceil B \rceil)$. Dakle, iz pretpostavke $pr(\lceil A \rceil)$ smo dokazali $pr(\lceil B \rceil)$, odakle uvođenjem implikacije dobijamo $\vdash_P pr(\lceil A \rceil) \Rightarrow pr(\lceil B \rceil)$. $\square$

Lema 5.7. *Neka je $\Psi(x)$ proizvoljna formula koja sadrži slobodnu promenljivu x . Tada postoji formula Φ takva da je $\vdash_P \Phi \Rightarrow \Psi(\lceil \Phi \rceil)$ i $\vdash_P \Psi(\lceil \Phi \rceil) \Rightarrow \Phi$.*

Dokaz. Neka je sa $sub(m, n)$ označen kôd formule koja se dobija tako što u formuli čiji je kôd m sve slobodne promenljive zamenimo numeralom $\bar{n}$. Funkcija sub je efektivno izračunljiva i može se izraziti na jeziku Peanove aritmetike.

Specijalno, za bilo koju formulu $A(x)$ sa jednom slobodnom promenljivom x važi da funkcija $sub(\lceil A(x) \rceil, \bar{n})$ izračunava kôd $\lceil A(\bar{n}) \rceil$. Otuda je u Peanovoj aritmetici dokazivo:

$$\vdash_{\mathcal{P}} sub(\lceil A(x) \rceil, \bar{n}) = \lceil A(\bar{n}) \rceil \quad (5.1)$$

Neka je sa $\Delta(x)$ označena formula $\Psi(sub(x, x))$. Neka je n_0 kôd formule $\Delta(x)$. Neka je $\Phi = \Delta(\bar{n}_0) = \Psi(sub(\bar{n}_0, \bar{n}_0)) = \Psi(sub(\lceil \Delta(x) \rceil, \bar{n}_0))$. S obzirom na 5.1, imamo da je:

$$\vdash_{\mathcal{P}} \Psi(sub(\lceil \Delta(x) \rceil, \bar{n}_0)) \Rightarrow \Psi(\lceil \Delta(\bar{n}_0) \rceil)$$

i

$$\vdash_{\mathcal{P}} \Psi(\lceil \Delta(\bar{n}_0) \rceil) \Rightarrow \Psi(sub(\lceil \Delta(x) \rceil, \bar{n}_0))$$

odnosno:

$$\vdash_{\mathcal{P}} \Phi \Rightarrow \Psi(\lceil \Phi \rceil)$$

i

$$\vdash_{\mathcal{P}} \Psi(\lceil \Phi \rceil) \Rightarrow \Phi$$

□

Poslednja lema je poznata i kao *lema o dijagonalizaciji*. Primenom ove leme na formulu $\neg pr(x)$, možemo tvrditi da postoji rečenica Φ za koju je $\vdash_{\mathcal{P}} \neg pr(\lceil \Phi \rceil) \Rightarrow \Phi$ kao i $\vdash_{\mathcal{P}} \Phi \Rightarrow \neg pr(\lceil \Phi \rceil)$. Intuitivno, formula Φ je takva da je ekvivalentna sa sopstvenom nedokazivošću. Odavde sledi sledeća važna teorema.

Teorema 5.16 (Gedelova teorema o nepotpunosti aritmetike). *Neka je Φ formula takva da važi $\vdash_{\mathcal{P}} \neg pr(\lceil \Phi \rceil) \Rightarrow \Phi$ kao i $\vdash_{\mathcal{P}} \Phi \Rightarrow \neg pr(\lceil \Phi \rceil)$. Ako je Peanova aritmetika konzistentna, tada u njoj ni Φ ni $\neg\Phi$ nije dokazivo.*

Dokaz. Ako bi Φ bilo dokazivo, tada bi i $pr(\Phi)$ bilo dokazivo, na osnovu leme 5.5. Sa druge strane, iz $\vdash_{\mathcal{P}} \Phi \Rightarrow \neg pr(\lceil \Phi \rceil)$ sledi da bismo mogli da dokažemo i $\neg pr(\lceil \Phi \rceil)$ (primenom eliminacije implikacije). Odavde sledi da je teorija $\mathcal{P}$ nekonzistentna, suprotno pretpostavci. Dakle, Φ ne može biti dokazivo u $\mathcal{P}$.

Obratno, ako bi $\neg\Phi$ bilo dokazivo, tada bi bilo dokazivo i $pr(\lceil \neg\Phi \rceil)$. Kako je $\vdash_{\mathcal{P}} \neg\Phi \Rightarrow (\Phi \Rightarrow \perp)$, sledi da je i $\vdash_{\mathcal{P}} pr(\lceil \neg\Phi \Rightarrow (\Phi \Rightarrow \perp) \rceil)$ (na osnovu leme 5.5), pa samim tim i $\vdash_{\mathcal{P}} pr(\lceil \neg\Phi \rceil) \Rightarrow pr(\lceil \Phi \Rightarrow \perp \rceil)$ (lema 5.6). Sada iz dokazivosti $pr(\lceil \neg\Phi \rceil)$ sledi $\vdash_{\mathcal{P}} pr(\lceil \Phi \Rightarrow \perp \rceil)$. Dalje, na osnovu leme 5.6, imamo da je $\vdash_{\mathcal{P}} pr(\lceil \Phi \rceil) \Rightarrow pr(\lceil \perp \rceil)$, odnosno $pr(\lceil \Phi \rceil) \vdash_{\mathcal{P}} pr(\lceil \perp \rceil)$. Kako iz $\vdash_{\mathcal{P}} pr(\lceil \perp \rceil)$ sledi $\vdash_{\mathcal{P}} \perp$, imamo da je $pr(\lceil \Phi \rceil) \vdash_{\mathcal{P}} \perp$. Odavde je $\vdash_{\mathcal{P}} \neg pr(\lceil \Phi \rceil)$. Iz svojstva $\vdash_{\mathcal{P}} \neg pr(\lceil \Phi \rceil) \Rightarrow \Phi$ sledi $\vdash_{\mathcal{P}} \Phi$. Kako je teorija konzistentna, ne može biti istovremeno i $\vdash_{\mathcal{P}} \neg\Phi$ i $\vdash_{\mathcal{P}} \Phi$. Iz ove kontradikcije zaključujemo da ni $\neg\Phi$ nije dokazivo. □

Gedelova teorema nam govori da postoje aritmetička tvrđenja koja se ne mogu ni dokazati ni opovrgnuti. Štaviše, za takvu rečenicu A važi da je ili A ili $\neg A$ tačno u standardnom modelu prirodnih brojeva $\mathbb{N}$. Dakle, postoje tvrđenja koja su tačna u intuitivnoj aritmetici, ali se ne mogu formalno dokazati u Peanovoj aritmetici. Štaviše, umesto Peanove aritmetike mogli smo uzeti bilo koju drugu aksiomatizaciju koja definiše „dovoljno” aritmetike da se izrazi kodiranje

formula i dokaza. Jedan primer takve minimalističke aritmetičke teorije je *Robinsonova aritmetika*¹. Isto važi i za *teoriju skupova* (npr. *Cermelo-Frenkelova teorija* (ZF)) koja takođe dopušta izražavanje aritmetike na svom jeziku.

Neko bi mogao pomisliti da se ovaj problem jednostavno može rešiti tako što formulu Φ iz teoreme 5.16 dodamo u aksiome i time je učinimo dokazivom. Međutim, tako proširena teorija $\mathcal{P}'$ će takođe zadovoljavati potrebne uslove za primenu Gedelove teoreme, pa će i u njoj postojati neka druga rečenica Φ' koja će biti tačna u strukturi prirodnih brojeva, a nedokaziva. Dakle, nije moguće dopuniti skup aksioma tako da teorija postane potpuna, a da ostane konzistentna.

Kao što znamo iz teoreme 5.11, potpunost aritmetike bi povlačila njenu odlučivost. Međutim, čak i kada teorija nije potpuna, ona može biti odlučiva. To nam daje mogućnost da makar prepoznamo tvrđenja koja su u teoriji ostala „neresena”, tj. koja nije moguće ni dokazati ni opovrgnuti (prosto bismo procedurom odlučivanja mogli utvrditi da ni A ni $\neg A$ nije teorema u toj teoriji). Na žalost, nekoliko godina nakon što je Gedel prezentovao svoju teoremu o nepotpunosti aritmetike, dokazano je da aritmetika nije ni odlučiva, o čemu govori sledeća teorema.

Teorema 5.17. *Peanova aritmetika je neodlučiva: ne postoji algoritam koji za proizvoljnu formulu A utvrđuje da li je $\vdash_{\mathcal{P}} A$.*

Dokaz. Dokaz ove teoreme je vrlo sličan dokazu teoreme 4.9 koji u sebi koristi minimalnu aritmetičku teoriju dovoljnu da se izraze instrukcije URM programa. Dakle, na jeziku aritmetike možemo izraziti problem zaustavljanja proizvoljnog URM programa za koji znamo da je neodlučiv. $\square$

Teorema 5.18 (Druga Gedelova teorema o nepotpunosti). *Konzistentnost Peanove aritmetike nije moguće dokazati u okviru nje same, tj. $\nvdash_{\mathcal{P}} \neg pr(\lceil \perp \rceil)$.*

Dokaz. Za ovo je dovoljno dokazati da iz $\vdash_{\mathcal{P}} \neg pr(\lceil \perp \rceil)$ sledi $\vdash_{\mathcal{P}} \Phi$, gde je Φ formula iz teoreme 5.16 za koju znamo da nije dokaziva. Kao u dokazu teoreme 5.16, pokazuje se da važi $pr(\lceil \Phi \rceil) \vdash_{\mathcal{P}} pr(\lceil \perp \rceil)$, odnosno $\vdash_{\mathcal{P}} pr(\lceil \Phi \rceil) \Rightarrow pr(\lceil \perp \rceil)$. Kontrapozicijom, može se dokazati $\vdash_{\mathcal{P}} \neg pr(\lceil \perp \rceil) \Rightarrow \neg pr(\lceil \Phi \rceil)$. Zbog svojstva $\vdash_{\mathcal{P}} \neg pr(\lceil \Phi \rceil) \Rightarrow \Phi$ formule Φ , sledi da važi $\vdash_{\mathcal{P}} \neg pr(\lceil \perp \rceil) \Rightarrow \Phi$, pa iz dokazivosti formule $\neg pr(\lceil \perp \rceil)$ sledi dokazivost formule Φ . Kako za nju znamo da je nedokaziva, sledi da ni $\neg pr(\lceil \perp \rceil)$ nije dokaziva, tj. nije moguće dokazati nedokazivost kontradikcije u Peanovoj aritmetici u okviru same Peanove aritmetike. $\square$

Dakle, u okviru same aritmetike nije moguće rezonovati o konzistentnosti same aritmetike.

¹Robinsonova aritmetika je konačno aksiomatizovana, jer ne uključuje aksiomatsku shemu indukcije koja nam za dokaz Gedelovih teorema nije neophodna.

Glava 6

Složenost izračunavanja

U prethodnim glavama bavili smo se problemom izračunljivosti, gde smo utvrdili da postoje problemi koji su algoritamski nerešivi, tj. za koje ne postoji nada da će ikada biti konstruisan algoritam koji ih rešava. Sa druge strane, za probleme koji *jesu* algoritamski rešivi, postavlja se pitanje *cene* tog rešavanja. Ova cena se izražava u količini potrebnih resursa da se odgovarajuće izračunavanje obavi. Glavni resurs koji se razmatra je *vreme izvršavanja*, koje predstavlja broj koraka koje algoritam izvršava za dati ulaz. Ovaj broj, naravno, zavisi od konkretne ulazne instance. Pritom, za očekivati je da će za „veće” ulazne instance taj broj koraka biti veći. Otuda nam je cilj da vreme izvršavanja izrazimo u funkciji od *veličine ulaza* n (tj. kao $f(n)$).

Tu se javljaju dva problema. Prvi je na koji način definisati veličinu ulaza. Cilj je da veličina ulaza n na neki način opisuje količinu memorijskog prostora koji se koristi za smeštanje ulaznog podatka u memoriju, tj. prostora potrebnog za kodiranje ulazne instance. U zavisnosti od izabrane reprezentacije ulaznog podatka, ovaj prostor može biti manji ili veći. Na primer, ako prirodan broj x na ulazu predstavimo kao niz od $x + 1$ jedinica, tada će veličina ulaza biti $n = x + 1$. Sa druge strane, ako broj x predstavimo u binarnom zapisu (pomoću nula i jedinica) na standardni način, tada će broj potrebnih binarnih cifara biti jednak $n = \lfloor \log_2 x \rfloor + 1$.

Drugi problem je to što čak i za fiksiranu veličinu ulaza n može postojati više različitih instanci problema zadate veličine n . Za različite instance iste veličine broj koraka algoritma može biti različit. Najčešće nas zanima gornja granica potrebnog vremena izvršavanja, tj. zelimo da uspostavimo garanciju da će se za sve instance date veličine n izvršavanje završiti u najviše $f(n)$ koraka. Ovo znači da razmatramo *najgori slučaj* tj. „najtežu” instancu veličine n . Ovakvu gornju granicu nazivamo *vremenska složenost* datog algoritma.

Drugi najznačajniji resurs jeste *prostor*, tj. količina memorije koju algoritam konzumira tokom svog rada za dati ulaz. I ovde se potrebni memorijski prostor izražava kao funkcija od veličine ulaza n , tj. veličine memorijskog prostora koji konzumira sam ulazni podatak. Takođe, po pravilu ćemo razmatrati najgori mogući slučaj za datu veličinu ulaza n , tj. uspostavljamo gornju granicu $g(n)$ za prostor potreban da se izvrši algoritam za proizvoljnu instancu veličine n . Ovakva gornja granica se naziva *prostorna složenost* datog algoritma.

Dati problem se često može rešiti na više načina, tj. postoji više različitih algoritama koji rešavaju isti problem. Složenosti ovih algoritama mogu biti raz-

ličite, a obično smo zainteresovani za najefikasnije moguće rešenje, tj. za algoritam najmanje moguće složenosti. U tom smislu, probleme svrstavamo u *klase složenosti* u zavisnosti od složenosti najefikasnijeg algoritma kojim se mogu rešiti. Oblast teorijskog računarstva koja razmatra klase složenosti i odnose među njima naziva se *složenost izračunavanja*. U ovoj glavi iznosimo neke najpoznatije rezultate iz ove oblasti.

Naglasimo da postoji i srodna oblast računarstva koja se naziva *analiza algoritama*. Ona se bavi preciznim razmatranjem složenosti konkretnih algoritama i njihovim međusobnim poređenjem. Pritom, analiza koja se tu sprovodi je obično znatno preciznija, jer nam daje odgovor na pitanje da li je, na primer, nekom algoritmu potrebno n^2 ili $n \log n$ koraka za izvršavanje za ulaze veličine n . Takođe, prilikom praktične analize algoritama, često razmatramo i *prosečno vreme izvršavanja* za ulaze date veličine n . U ovom tekstu, takvom analizom se nećemo baviti.

Sa druge strane, klase složenosti koje razmatramo u okviru složenosti izračunavanja su obično znatno grublje. Na primer, pitamo se da li za neki problem postoji algoritam *polinomske složenosti*, tj. algoritam takav da postoji polinom $p(n)$ takav da je broj koraka $f(n)$ koji je potreban za izvršavanje algoritma nad ulazom veličine n u najgorem slučaju manji ili jednak od $p(n)$. Pritom, nebitno je da li je, na primer, $p(n) = n^2$ ili $p(n) = n^{10}$. Dakle, cilj složenosti izračunavanja je da se problemi relativno grubo klasifikuju u one koji su *praktično rešivi* i one koji to nisu, iako se formalno mogu algoritamski rešiti. U praktično rešive probleme obično ubrajamo one za koje postoji algoritam polinomske složenosti. Intuitivni razlog za tako nešto je sledeći: za algoritme polinomske složenosti možemo očekivati da ćemo uz razumno povećanje datih resursa moći značajno da povećamo veličinu instanci problema koje možemo rešavati. Na primer, ako je složenost algoritma n^2 , tada ćemo, ako se računar ubrza 100 puta (što je razumno očekivanje, s obzirom na razvoj tehnologije), moći da za isto dato vreme rešimo instance koje su 10 puta veće. Sa druge strane, ako je složenost algoritma 2^n (tzv. *eksponencijalna složenost*), tada ćemo sa 100 puta bržim računarom za isto zadato vreme moći da rešimo instance koje su svega za 6 veće (jer je već $2^7 = 128$). Zbog toga je za očekivati da će daljim razvojem tehnologije problemi rešivi algoritmima polinomske složenosti biti sve pristupačniji za rešavanje pomoću realnih računara, dok to nije slučaj za probleme koji nisu rešivi algoritmima polinomske složenosti (za takve probleme se obično kaže da su *neukrotivi*, engl. *intractable*). Zato nam je od najvećeg značaja da izučimo klasu polinomske rešivih problema, kao i kako se ta klasa odnosi sa drugim klasama složenosti.

6.1 Tjuringova mašina

Da bismo formalno definisali pojam složenosti algoritma, potrebno je da unapred fiksiramo formalni model izračunavanja. U dosadašnjem razmatranju, izučili smo dva takva formalizma – μ -rekurzivne funkcije i URM programe. Na žalost, ni jedan od ta dva formalizma nije pogodan za definisanje pojma složenosti izračunavanja. Naime, oba ova modela izračunavanja su definisani u terminima aritmetičkih operacija nad prirodnim brojevima, pri čemu se pretpostavlja da resursi za izvođenje pojedinačnih operacija nad brojevima (poput Z i S operacija) ne zavise od vrednosti samog broja nad kojim se primenjuju. Na primer, kod URM programa, pretpostavljamo da svaki registar može da čuva

bilo koji prirodan broj. Ovo prilično odudara od realnih računara, kod kojih ni jedan registar ne može čuvati bilo koji prirodan broj, već samo prirodne brojeve iz nekog unapred datog konačnog skupa (npr. 8-bitni registar može čuvati samo brojeve od 0 do 255). S obzirom da znamo da se svaki podatak može na kraju kodirati jednim prirodnim brojem, sledi da se kod URM programa svaki podatak može smestiti u jedan registar, što nam ne omogućava da realno odredimo veličinu ulaza. Takođe, URM program pretpostavlja da se svaki prirodan broj može uvećati za 1 pomoću jedne S instrukcije. U praksi, mi brojeve predstavljamo nizom njihovih cifara, pri čemu aritmetičke operacije nad njima mogu zahtevati transformisanje više cifara u zapisu. Na primer, uvećanje broja 199 za 1 zahteva promenu tri cifre, dok uvećanje broja 1999999 za 1 zahteva promenu sedam cifara. Dakle, URM programi ne mogu realno opisati vremensku složenost postupka izračunavanja. Najzad, URM program izračunava zbir dva broja x i y tako što uzastopno uvećava broj x y puta. Realni računari ovo rade znatno efikasnije, s obzirom da operišu sa pozicionim zapisima brojeva. Slično važi i za ostale osnovne aritmetičke operacije.

Da bismo prevazišli nabrojane probleme, uvodimo još jedan formalizam za opis algoritama – *Tjuringove mašine*. U pitanju je jedan od najstarijih formalnih modela izračunavanja koji je Alan Tjuring osmislio kako bi dokazao neodlučivost logike prvog reda. Princip rada Tjuringove mašine podseća na rad pisaaće mašine i spada u formalizme najnižeg nivoa, s obzirom da ne barata sa brojevima, već sa simbolima pomoću kojih se podaci zapisuju. Zbog toga su Tjuringove mašine naročito pogodne za razmatranje složenosti izračunavanja, jer omogućavaju da se realno ocene veličine podataka sa kojima se radi, kao i vreme koje je potrebno za njihovu obradu.

Da bismo definisali pojam Tjuringove mašine, najpre uvodimo definiciju azbuke, reči i jezika.

Definicija 6.1. *Azbuka (ili alfabet) $\Sigma = \{a_1, \dots, a_n\}$ je konačan skup simbola. Reč nad azbukom Σ je bilo koji konačan niz $a_{i_1}a_{i_2}\dots a_{i_k}$ simbola iz Σ . Dužina reči $w = a_{i_1}\dots a_{i_k}$ (u oznaci $|w|$) je jednaka broju simbola u njoj. Specijalno, prazna reč ε je reč koja ne sadrži ni jedan simbol azbuke i njena dužina je $|\varepsilon| = 0$. Skup svih reči nad Σ označavamo sa Σ^* . Jezik L nad azbukom Σ je bilo koji podskup od Σ^* .*

Primer 6.1. Neka je data azbuka $\Sigma = \{0, 1\}$. Reči nad ovom azbukom su binarne niske, poput 01101 ili 111011. Pritom, imamo da je $|01101| = 5$ i $|111011| = 6$. Reč ε je prazna niska i ona je takođe reč nad Σ .

Skup $L = \{w \in \Sigma^* \mid w \text{ sadrži paran broj nula}\}$ je jedan jezik nad Σ . Na primer, $01101 \in L$, a $111011 \notin L$. Prazna reč ε takođe pripada L , s obzirom da ne sadrži ni jednu nulu.

Definicija 6.2. *Tjuringova mašina sa k traka je uređena sedmorka $(\Gamma, b, \Sigma, Q, q_0, F, \delta)$, gde je:*

- Γ je azbuka trake
- $\square \in \Gamma$ je simbol praznog polja
- $\Sigma \subseteq (\Gamma \setminus \{\square\})$ je ulazna azbuka
- Q je konačan skup stanja

- $q_0 \in Q$ je početno stanje
- $F \subseteq Q$ je skup završnih stanja
- $\delta : (Q \setminus F) \times \Gamma^k \rightarrow Q \times (\Gamma \times \{\leftarrow, \rightarrow, \downarrow\})^k$ je funkcija prelaska

Konfiguracija Tjuringove mašine se može opisati kao $(k + 1)$ -torka $(q, \square^\omega u_1 \hat{a}_1 w_1 \square^\omega, \dots, \square^\omega u_k \hat{a}_k w_k \square^\omega)$, gde je $q \in Q$ trenutno stanje, $\square^\omega u_i \hat{a}_i w_i \square^\omega$ je konfiguracija i -te trake ($u_i, w_i \in \Gamma^*$, $a_i \in \Gamma$, $\square^\omega = \dots \square \square \square \dots$, a oznaka $\hat{}$ označava poziciju glave za čitanje/pisanje i -te trake). Početna konfiguracija za dati ulaz $w \in \Sigma^*$ je data sa $(q_0, \square^\omega \hat{a} u \square^\omega, \square^\omega \hat{} \square^\omega, \dots, \square^\omega \hat{} \square^\omega)$, gde je $w = au$ ($a \in \Sigma$, $w \in \Sigma^*$). Ukoliko je tekuća konfiguracija $(q, \square^\omega u_1 b_1 \hat{a}_1 c_1 w_1 \square^\omega, \dots, \square^\omega u_k b_k \hat{a}_k c_k w_k \square^\omega)$ ($q \in Q \setminus F$, $u_i, w_i \in \Gamma^*$, $a_i, b_i, c_i \in \Gamma$) i važi $\delta(q, a_1, \dots, a_k) = (q', ((e_1, d_1), \dots, (e_k, d_k)))$, gde je $q' \in Q$, $e_i \in \Gamma$, $d_i \in \{\leftarrow, \rightarrow, \downarrow\}$, tada će sledeća konfiguracija mašine biti $(q', \square^\omega u_1 b_1 e_1 c_1 w_1 \square^\omega, \dots, \square^\omega u_k b_k e_k c_k w_k \square^\omega)$, pri čemu će pozicija i -te glave biti iznad e_i ako je $d_i = \downarrow$, iznad b_i ako je $d_i = \leftarrow$, ili iznad c_i ako je $d_i = \rightarrow$. Konfiguracija $(q, \square^\omega u_1 \hat{a}_1 w_1 \square^\omega, \dots, \square^\omega u_k \hat{a}_k w_k \square^\omega)$ je završna, ako $q \in F$. Tjuringova mašina staje za dati ulaz $w \in \Sigma^*$ ako polazeći iz početne konfiguracije koja odgovara tom ulazu nakon konačno mnogo koraka stiže u neku završnu konfiguraciju. Izlaz Tjuringove mašine je niska $w_{out} \in \Sigma^*$ koja se dobija kada se iz niske $u_k a_k w_k \in \Gamma^*$ zapisane na k -toj traci u završnoj konfiguraciji mašine obrišu svi simboli koji ne pripadaju azbuci Σ .

Intuitivno, Tjuringova mašina ima k beskonačnih memorijskih traka koje se sastoje iz polja u koja možemo upisivati simbole iz Γ . Pritom, u svakom trenutku samo konačan broj polja svake trake nije prazno, tj. sadrži simbole iz Γ koji nisu $\square$. Svaka traka ima glavu za čitanje/pisanje koja je pozicionirana na nekom polju trake. Prva traka je *ulazna traka* i na njoj se inicijalno nalazi ulazna niska $w \in \Sigma^*$. Poslednja traka je *izlazna traka* i sa nje se na kraju očitava izlaz. Program Tjuringove mašine je određen funkcijom prelaska δ . Ona nam govori na koji način menjamo konfiguraciju u svakom koraku. Promena konfiguracije se vrši na osnovu simbola koji se trenutno nalaze ispod glava za čitanje, kao i na osnovu trenutnog stanja programa (stanje $q \in Q$). Svaki korak menja konfiguraciju mašine tako što menja stanje q mašine u $q' \in Q$, zamenjuje simbol a_i ispod svake od glava za čitanje/pisanje novim simbolom e_i , a zatim glavu eventualno pomera za jedno mesto u levo ili desno. Rad programa se završava kada se dođe u neko završno stanje. Izlaz se sa poslednje trake očitava tako što se sa nje uzmu samo simboli koji pripadaju Σ , dok se simboli iz $\Gamma \setminus \Sigma$ preskaču. Naime, Σ predstavlja ulazno/izlaznu azbuku pomoću koje mašina komunicira sa spoljnom sredinom. Sa druge strane, radna azbuka Γ može pored simbola iz Σ koristiti i razne pomoćne simbole koje zapisuje na trakama, ali oni neće biti prisutni na izlazu nakon završetka rada mašine.

Primer 6.2. Neka je data azbuka $\Sigma = \{a, b\}$. Konstruišimo Tjuringovu mašinu koja proizvoljnu nisku $w \in \Sigma^*$ transformiše tako što slova a stavlja na početak, a slova b na kraj. Na primer, za ulaznu nisku $abbaabbab$, izlaz će biti $aaaabbbbb$. Azbuka Γ će u ovom primeru biti jednaka $\Sigma \cup \{\square\}$, jer nam, osim blanko-simbola $\square$ drugi pomoćni simboli nisu potrebni. Imaćemo dve trake, ulaznu i izlaznu. Na ulaznoj traci će biti inicijalno smeštena ulazna niska, a glava će biti pozicionirana nad početnim karakterom niske. Izlazna traka će inicijalno biti prazna, a glava

će biti pozicionirana iznad proizvoljnog praznog polja. Program će se sastojati iz sledećih „naredbi”:

- $\delta(q_0, a, \square) = (q_0, (a, \rightarrow), (a, \rightarrow))$
- $\delta(q_0, b, \square) = (q_0, (b, \rightarrow), (\square, \downarrow))$
- $\delta(q_0, \square, \square) = (q_1, (\square, \leftarrow), (\square, \downarrow))$
- $\delta(q_1, a, \square) = (q_1, (a, \leftarrow), (\square, \downarrow))$
- $\delta(q_1, b, \square) = (q_1, (b, \leftarrow), (b, \rightarrow))$
- $\delta(q_1, \square, \square) = (q_F, (\square, \downarrow), (\square, \downarrow))$

Pritom, q_0 je početno, a q_F završno stanje mašine. Intuitivno, program radi tako što najpre prolazi kroz ulaznu nisku sa leva na desno i sve simbole a prepisuje na izlaznu traku, dok simbole b preskače. Kada stigne do kraja niske, mašina prelazi u stanje q_1 u kome se kreće sa desna u levo i tom prilikom sve simbole b prepisuje na izlaznu traku, dok simbole a preskače. Na primer, za nisku $abbaa$ bi iz početne konfiguracije $(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{\square} \square^\omega)$ imali sledeći niz konfiguracija:

$$\begin{aligned}
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{\square} \square^\omega) &\implies \\
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_0, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_1, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies \\
(q_F, \square^\omega \hat{a} b b a a \square^\omega, \square^\omega \hat{a} \square^\omega) &\implies
\end{aligned}$$

Odbacivanjem simbola $\square$ iz završne konfiguracije izlazne trake, dobijamo izlaznu nisku $aaabb$, kao što je i očekivano.

Napomena 6.1. Tjuringova mašina može imati proizvoljan, unapred fiksiran broj traka. Broj traka može biti i jedan – u tom slučaju imamo Tjuringovu mašinu sa jednom trakom koja će ujedno biti i ulazna i izlazna. Originalna definicija Tjuringove mašine je podrazumevala da postoji samo jedna traka. Može se pokazati da veći broj traka ne utiče na moć izračunavanja, jer se svaka Tjuringova mašina sa k traka može simulirati pomoću Tjuringove mašine sa jednom trakom. Ipak, postojanje više traka može značajno da olakša opis postupka izračunavanja, jer nam omogućava da pristupamo većem broju podataka istovremeno.

Primer 6.3. Rešimo isti zadatak koji smo imali u prethodnom primeru, ali ovoga puta pomoću Tjuringove mašine sa jednom trakom. Ideja je da izlaz upisujemo ispred ulaza na istoj traci. Zato ćemo u azbuci Γ pored simbola a, b i $\square$ imati još jedan simbol $|$ koji predstavlja *separator* između ulaza i izlaza. Program će sada izgledati ovako:

- $\delta(q_0, a) = (q_0, (a, \leftarrow))$
- $\delta(q_0, b) = (q_0, (b, \leftarrow))$
- $\delta(q_0, \square) = (q_b, (|, \rightarrow))$

Dakle, u stanju q_0 se pomeramo levo od ulaza i ispred ulaza upisujemo separator $|$ i prelazimo u stanje q_b .

Sada je ideja da prođemo kroz ulaznu nisku i sve simbole b prepisemo ispred separatora $|$. Da bismo vodili evidenciju koje smo simbole b prepisali, a koje nismo, uvodimo pomoćni simbol B u azbuku Γ koji predstavlja *marker* prepisanih slova b . U stanju q_b se krećemo sa leva u desno, preskačemo slova a i markere B i stajemo kod prvog neprepisanog slova b . Njega zamenjujemo markerom B i prelazimo u stanje q_{bw} u kome nam je cilj da to b prepisemo na izlaz. Ako ne pronađemo ni jedno b do kraja ulazne niske, prelazimo u stanje q_r .

- $\delta(q_b, a) = (q_b, (a, \rightarrow))$
- $\delta(q_b, B) = (q_b, (B, \rightarrow))$
- $\delta(q_b, b) = (q_{bw}, (B, \leftarrow))$
- $\delta(q_b, \square) = (q_r, (\square, \leftarrow))$

Stanje q_{bw} služi da se pronađeni simbol b upiše levo od separatora.

- $\delta(q_{bw}, *) = (q_{bw}, (*, \leftarrow))$
- $\delta(q_{bw}, \square) = (q_{sb}, (b, \rightarrow))$

U ovom stanju za sve simbole koji nisu $\square$ (što je gore označeno sa $*$, da ne bismo ponavljali istu naredbu za sve simbole), se samo pomeramo u levo, a kada stignemo do $\square$, zamenjujemo ga sa b i prelazimo u stanje q_{sb} . U ovom stanju je cilj da ponovo dođemo na početak ulazne niske i da nastavimo sa pretragom sledećeg simbola b .

- $\delta(q_{sb}, *) = (q_{sb}, (*, \rightarrow))$
- $\delta(q_{sb}, |) = (q_b, (|, \rightarrow))$

pri čemu je sa $*$ označen bilo koji simbol osim separatora $|$.

Kada stignemo u stanje q_r , znači da smo prepisali sve b simbole na izlaz. Sada je potrebno levo od njih na izlazu prepisati sva slova a iz ulazne niske. Najpre je potrebno vratiti se na početak ulaza:

- $\delta(q_r, *) = (q_r, (*, \leftarrow))$
- $\delta(q_r, |) = (q_a, (|, \rightarrow))$

pri čemu ovde $*$ označava sve simbole osim separatora $|$. U stanju q_a tražimo prvi simbol a iz ulazne niske:

- $\delta(q_a, B) = (q_a, (B, \rightarrow))$
- $\delta(q_a, A) = (q_a, (A, \rightarrow))$

- $\delta(q_a, a) = (q_{aw}, (A, \leftarrow))$
- $\delta(q_a, \square) = (q_c, \leftarrow)$

Kao i ranije, uvodimo marker A u abukvu Γ da označimo one simbole a koji su već prepisani na izlaz. Njih preskačemo, kao i markere B . Kada naiđemo na simbol a , zamenjujemo ga markerom A i prelazimo u stanje q_{aw} koje služi za upis pročitane simbole a na izlaz. Ako nije preostalo ni jedno neprepisano a , tada ćemo stići do praznog polja. U tom slučaju prelazimo u stanje q_c čiji je zadatak da obriše sa trake sve osim onoga što treba da ostane na izlazu.

- $\delta(q_{aw}, *) = (q_{aw}, (*, \leftarrow))$
- $\delta(q_{aw}, \square) = (q_{sa}, (a, \rightarrow))$

Ovde ponovo $*$ označava sve osim $\square$. Kada naiđemo na prvo prazno polje na levom kraju, u njega upisujemo a i zatim prelazimo u stanje q_{sa} koje služi da se vratimo na početak ulaza i tražimo sledeće a .

- $\delta(q_{sa}, *) = (q_{sa}, (*, \rightarrow))$
- $\delta(q_{sa}, |) = (q_a, (|, \rightarrow))$

pri čemu $*$ označava sve osim $|$. Najzad, u stanju q_c radimo sledeće:

- $\delta(q_c, *) = (q_c, (\square, \leftarrow))$
- $\delta(q_c, |) = (q_F, (\square, \downarrow))$

Dakle, sve simbole osim $|$ brišemo i pomeramo se dalje u levo. Kada stignemo do $|$, njega takođe brišemo i prelazimo u završno stanje q_F . Za nisku $abbaa$ imaćemo sledeći niz konfiguracija:

Ovim delom programa vršimo kopiranje prvog sabirka na drugu traku (stanje q_0), a zatim premotavanje prve trake na desni kraj drugog sabirka (stanje q_1). Na kraju stižemo u stanje q_n u kome započinjemo sabiranje.

- $\delta(q_n, 0, 0, \square) = (q_n, (0, \leftarrow), (0, \leftarrow), (0, \leftarrow))$
- $\delta(q_n, 0, 1, \square) = (q_n, (0, \leftarrow), (1, \leftarrow), (1, \leftarrow))$
- $\delta(q_n, 1, 0, \square) = (q_n, (1, \leftarrow), (0, \leftarrow), (1, \leftarrow))$
- $\delta(q_n, 1, 1, \square) = (q_c, (1, \leftarrow), (1, \leftarrow), (0, \leftarrow))$
- $\delta(q_c, 0, 0, \square) = (q_n, (0, \leftarrow), (0, \leftarrow), (1, \leftarrow))$
- $\delta(q_c, 0, 1, \square) = (q_c, (0, \leftarrow), (1, \leftarrow), (0, \leftarrow))$
- $\delta(q_c, 1, 0, \square) = (q_c, (1, \leftarrow), (0, \leftarrow), (0, \leftarrow))$
- $\delta(q_c, 1, 1, \square) = (q_c, (1, \leftarrow), (1, \leftarrow), (1, \leftarrow))$

Ovaj deo programa vrši sabiranje binarnih cifara i pomeranje u levo na sledeće cifre u zapisima brojeva. Stanje q_n znači da ne postoji prenos sa prethodne pozicije, dok stanje q_c znači da taj prenos postoji. Ostaje da se obradi slučaj kada jedan od sabiraka ima manje cifara od drugog – tada ćemo u nekom trenutku stići do praznog polja u jednom sabirku, dok će drugi imati još cifara.

- $\delta(q_n, \square, 0, \square) = (q_n, (\square, \downarrow), (0, \leftarrow), (0, \leftarrow))$
- $\delta(q_n, 0, \square, \square) = (q_n, (0, \leftarrow), (\square, \downarrow), (0, \leftarrow))$
- $\delta(q_n, \square, 1, \square) = (q_n, (\square, \downarrow), (1, \leftarrow), (1, \leftarrow))$
- $\delta(q_n, 1, \square, \square) = (q_n, (1, \leftarrow), (\square, \downarrow), (1, \leftarrow))$
- $\delta(q_c, \square, 0, \square) = (q_n, (\square, \downarrow), (0, \leftarrow), (1, \leftarrow))$
- $\delta(q_c, 0, \square, \square) = (q_n, (0, \leftarrow), (\square, \downarrow), (1, \leftarrow))$
- $\delta(q_c, \square, 1, \square) = (q_c, (\square, \downarrow), (1, \leftarrow), (0, \leftarrow))$
- $\delta(q_c, 1, \square, \square) = (q_c, (1, \leftarrow), (\square, \downarrow), (0, \leftarrow))$

Na kraju, kada više nema cifara ni u jednom sabirku, završavamo algoritam:

- $\delta(q_n, \square, \square, \square) = (q_F, (\square, \downarrow), (\square, \downarrow), (\square, \downarrow))$
- $\delta(q_c, \square, \square, \square) = (q_F, (\square, \downarrow), (\square, \downarrow), (1, \leftarrow))$

pri čemu je sa q_F označeno jedino završno stanje. Čitaocu za veću prepuštamo demonstraciju rada programa na primeru ulazne niske 11|101.

Primer 6.5. Čitaocu ostavljamo za vežbu da osmisli programe Tjuringovih mašina koje obavljaju aritmetičke operacije oduzimanja, množenja i celobrojnog deljenja nad brojevima zapisanim u binarnom zapisu. *Uputstvo:* simulirati standardne postupke računanja u binarnom sistemu.

Definicija 6.3. *Parcijalna funkcija nad Σ^** je funkcija $f : L \subseteq \Sigma^*$ gde je $L \subseteq \Sigma^*$ neki jezik nad Σ koji nazivamo *domen* funkcije f i označavamo ga sa $\text{dom}(f)$. Pritom, za nisku $w \notin L$ kažemo da je funkcija f *nedefinisana* za w i pišemo $f(w) = -$. Za parcijalnu funkciju f nad Σ^* kažemo da je *totalna* ako je definisana za svaku nisku $w \in \Sigma^*$.

Definicija 6.4. Parcijalna funkcija f_T nad Σ^* koju *izračunava* Tjuringova mašina T je definisana na sledeći način:

- ako za ulaz $w \in \Sigma^*$ Tjuringova mašina T staje i na izlazu daje $u \in \Sigma^*$, tada je $f_T(w) = u$
- u suprotnom, ako za ulaz $w \in \Sigma^*$ Tjuringova mašina T ne staje, tada je funkcija f_T nedefinisana za w .

Za parcijalnu funkciju f nad Σ^* kažemo da je *Tjuring izračunljiva* (ili *T-izračunljiva*) ako postoji Tjuringova mašina koja je izračunava.

Dakle, za razliku od URM programa i μ -rekurzivnih funkcija, Tjuringova mašina definiše funkcije nad rečima neke proizvoljne azbuke Σ , a ne nad prirodnim brojevima. Ovo na prvi pogled znači da ne možemo direktno upoređivati klase URM izračunljivih (tj. μ -rekurzivnih) i Tjuring izračunljivih funkcija, s obzirom da nisu definisane nad istim domenima. Ipak, lako se može videti da se svakoj parcijalnoj funkciji nad Σ^* može pridružiti parcijalna funkcija nad $\mathbb{N}$. Za ovo je dovoljno omogućiti da se proizvoljna niska $w \in \Sigma^*$ kodira prirodnim brojem. Ovo možemo uraditi tako što najpre simbolima azbuke pridružimo prirodne brojeve $0, 1, \dots, n-1$, gde je $n = |\Sigma|$, čime se svaka niska $w \in \Sigma^*$ transformiše u konačnu sekvencu prirodnih brojeva. Ova sekvencu se može kodirati jednim prirodnim brojem na uobičajen način prikazan ranije. Ako sa $\alpha : \Sigma^* \rightarrow \mathbb{N}$ označimo ovu funkciju kodiranja, a sa f proizvoljnu parcijalnu funkciju nad Σ^* , tada će funkcija $g = \alpha \circ f \circ \alpha^{-1}$ biti parcijalna funkcija nad $\mathbb{N}$ reda 1. Sada važi sledeća teorema.

Teorema 6.1. *Za svaku Tjuring izračunljivu funkciju f njoj pridružena parcijalna funkcija $g = \alpha \circ f \circ \alpha^{-1}$ nad $\mathbb{N}$ je URM izračunljiva.*

Dokaz. (skica) Intuitivno, dati broj $x \in \mathbb{N}$ treba dekodirati funkcijom α^{-1} i dobiti niz kodova simbola neke niske $w \in \Sigma^*$. Dalje treba simulirati rad Tjuringove mašine nad niskom w URM programom. Ovo je moguće, zato što je sadržaj trake Tjuringove mašine uvek konačan niz simbola (ne računajući simbole $\square$), pa je sadržaj trake moguće kodirati brojevima i čuvati ih u registrima URM mašine. Poseban registar će se koristiti za čuvanje stanja Tjuringove mašine. Sada je svaku naredbu Tjuringove mašine moguće simulirati tako što se dekodiraju sadržaji trake, na osnovu kodova očitanih simbola i trenutnog stanja se odrede prelazi i zatim se tako izmenjeni sadržaji trake ponovo kodiraju i upišu u odgovarajuće registre. Ovaj intuitivni postupak se može prevesti u konkretan URM program. Najzad, dobijeni izlaz Tjuringove mašine, predstavljen kao sekvencu kodova simbola na poslednjoj traci, se može funkcijom α kodirati, čime se dobija rezultat funkcije g . Dakle, funkcija g je URM izračunljiva. □

Razmotrimo sada obrnut slučaj. Neka je f parcijalna funkcija nad $\mathbb{N}$ reda k . Neka je sa $\gamma : \mathbb{N}^k \rightarrow \Sigma^*$ ($\Sigma = \{0, 1, |\}$) definisana funkcija kodiranja koja k -torku

brojeva $(x_1, \dots, x_k)$ prevodi u nisku $w_1|w_2|\dots|w_k \in \Sigma^*$, gde je w_i binarni zapis broja x_i , a simbol $|$ se koristi kao separator. Tada će funkcija $g = \gamma \circ f \circ \gamma^{-1}$ biti parcijalna funkcija nad Σ^* . Sada važi sledeća teorema.

Teorema 6.2. *Za svaku URM izračunljivu funkciju f , njoj pridružena parcijalna funkcija $g = \gamma \circ f \circ \gamma^{-1}$ nad Σ^* je T -izračunljiva.*

Dokaz. (skica) Intuitivno, ako URM program P izračunava funkciju f , tada je potrebno konstruisati Tjuringovu mašinu koja će simulirati izvršavanje instrukcija programa P nad binarnom reprezentacijom podataka. Indeks tekuće instrukcije će se predstavljati stanjem Tjuringove mašine. Za svaki registar R_i koji koristi URM program P imaćemo posebnu traku na kojoj se nalazi binarna reprezentacija vrednosti u tom registru. Sada je samo potrebno omogućiti simulaciju operacija Z , S , T i J , što nije posebno teško. $\square$

Iz prethodne dve teoreme sledi da su Tjuringove mašine suštinski ekvivalentne po izražajnosti sa druga dva formalizma (URM mašine i μ -rekurzivne funkcije) koja smo ranije upoznali, s obzirom da izračunavanje svake URM izračunljive funkcije možemo simulirati Tjuringovom mašinom i obrnuto, izračunavanje svake Tjuring izračunljive funkcije možemo simulirati URM programom.

6.2 Definicija složenosti

Definicija 6.5. *Vreme izvršavanja* Tjuringove mašine T za dati ulaz $w \in \Sigma^*$ (u oznaci $time(T(w))$) je jednako potrebnom broju koraka da se od početnog stanja stigne do završnog stanja pri ulazu w . *Prostor izvršavanja* Tjuringove mašine T za dati ulaz $w \in \Sigma$ (u oznaci $space(T(w))$) jednak je $\max_{j=1}^t (\sum_{i=1}^k |w_{ij}|)$, gde je t broj koraka pri izvršavanju za ulaz w , a $\square^w w_{ij} \square^w$ je sadržaj i -te trake u koraku j .

Primetimo da je prostor izvršavanja, prema gornjoj definiciji, jednak maksimalnom zauzeću memorije na svim trakama mašine tokom njenog rada.

Pod veličinom ulaza w Tjuringove mašine podrazumevamo njegovu dužinu $|w|$, kada ga posmatramo kao nisku nad ulaznom azbukom Σ . Sada možemo definisati vremensku i prostornu složenost date Tjuringove mašine u funkciji od veličine ulaza.

Definicija 6.6. *Vremenska složenost* Tjuringove mašine T je funkcija $t_T(n)$ za koju važi:

$$t_T(n) = \max_{|w|=n} (time(T(w)))$$

Prostorna složenost Tjuringove mašine T je funkcija $s_T(n)$ za koju važi:

$$s_T(n) = \max_{|w|=n} (space(T(w)))$$

Dakle, u oba slučaja računamo maksimum potrebnih resursa za sve instance čija je ukupna veličina jednaka n . Dakle, uzimamo najgori slučaj među svim instancama date veličine.

Definicija 6.7. Neka su date dve funkcije $f, g : \mathbb{N} \rightarrow \mathbb{N}$. Tada je $f(n) = O(g(n))$ ako postoji konstanta $C \in \mathbb{N}$ i vrednost $n_0 \in \mathbb{N}$ takvi da je $f(n) \leq C \cdot g(n)$ za svako $n > n_0$.

Prethodna definicija uvodi uobičajenu „veliko-O” notaciju. Pritom, naglasimo da $O(g(n))$ nije jedna funkcija, već označava čitavu klasu funkcija koje asimptotski ne rastu brže od funkcije g . Oznaka $f(n) = O(g(n))$ znači da je f jedna takva funkcija. Na primer, važi da je $n^2 + n + 4 = O(n^2)$, jer za, na primer, $C = 2$ za svako $n \geq 3$ važiće da je $n^2 + n + 4 \leq 2n^2$.

Veliko-O notacija će nam omogućiti da uvedemo klase složenosti problema. Pritom, „problem” će ovde biti bilo koja totalna izračunljiva funkcija. Kao što znamo, izračunljiva funkcija f se može izračunati potencijalno pomoću različitih Tjuringovih mašina, ali nas zanima ona koja ima najmanju složenost.

Definicija 6.8. Neka je data totalna izračunljiva funkcija f reda k . Za f kažemo da pripada klasi vremenske (prostorne) složenosti $O(g(n))$ za neku funkciju $g : \mathbb{N} \rightarrow \mathbb{N}$ ako postoji Tjuringova mašina T koja izračunava funkciju f takva da je $t_T(n) = O(g(n))$ ($s_T(n) = O(g(n))$).

Dakle, problem pripada klasi vremenske složenosti $O(g(n))$ ako se vreme izvršavanja može asimptotski ograničiti sa $g(n)$. Iz ove definicije sledi da isti problem može pripadati većem broju klasa, jer njegova vremenska složenost može imati više različitih gornjih granica. Na primer, ako imamo mašinu T koja izračunava funkciju f takva da je $t_T(n) = n^2 + n + 4$, tada je problem f u klasi vremenske složenosti $O(n^2)$, ali i u klasi $O(n^3)$, kao i u klasi $O(2^n)$, jer važi $t_T(n) = O(n^2)$, $t_T(n) = O(n^3)$ i $t_T(n) = O(2^n)$. Takođe, jasno je da neka klasa problema može biti potskup neke druge klase problema. Na primer, klasa $O(n^2)$ je potskup klase $O(n^3)$, jer svaki problem koji pripada klasi $O(n^2)$ je i u klasi $O(n^3)$. Iste napomene važe i za prostornu složenost.

Primer 6.6. Setimo se Tjuringove mašine iz primera 6.4 koja je sabirala dva prirodna broja u binarnom zapisu. Ako su n_1 i n_2 redom dužine zapisa prvog i drugog sabirka, tada je program najpre prolazio jednom kroz celu ulaznu nisku da bi iskopirao prvi sabirak na drugu traku i „premotao” glavu prve trake na desni kraj drugog sabirka ($n_1 + n_2 + 1$ koraka), a zatim je vršio sabiranje standardnim postupkom ($\max(n_1, n_2) + 1$ koraka). Ako je $n = n_1 + n_2 + 1$ veličina ulaza, jasno je da je ukupan broj koraka linearno ograničen u odnosu na n , tj. vremenska složenost ovog algoritma je $O(n)$.

Primer 6.7. Čitaocu ostavljamo za vežbu da se uveri da je složenost operacije oduzimanja takođe $O(n)$, dok je složenost operacija množenja i celobrojnog deljenja kvadratna – $O(n^2)$. Pritom, podrazumevamo standardne računске postupke u binarnom brojevnom sistemu.

Napomena 6.2. Složenost aritmetičkih operacija koje smo diskutovali u prethodnim primerima se odnose na Tjuringove mašine sa više traka. Ako bismo se ograničili na jednu traku, tada je složenost nešto veća. Na primer, prilikom sabiranja, morali bismo oba sabirka, kao i zbir, da držimo na istoj traci, kao i da stalno premotavamo glavu od cifara jednog sabirka ka ciframa drugog sabirka i zbira. Ovo znači da ćemo za svaku binarnu poziciju prilikom sabiranja morati da izvršimo $O(n)$ operacija pomeranja glave. Kako je broj pozicija koje se sabiraju jednak $\max(n_1, n_2) + 1 = O(n)$, sledi da će ukupan broj koraka biti $O(n^2)$. Uopšte, može se pokazati da se svaki algoritam koji zahteva $O(f(n))$ koraka na Tjuringovoj mašini sa više traka može simulirati Tjuringovom mašinom sa jednom trakom u $O(f(n)^2)$ koraka. Dakle, postoji kvadratno usporenje.

Napomena 6.3. Iako smo složenost izračunavanja formalno definisali u terminima Tjuringovih mašina, u praksi je možemo razmatrati i manje formalno,

ali na suštinski isti način, u kontekstu realnih računara. U tom kontekstu, pod veličinom ulaza se smatra memorijski prostor koji ulazni podatak zauzima. Veličina ovog prostora je obično izražena u bajtovima, ali može biti izražena i u bitovima ili npr., 32-bitnim rečima. Izbor konkretne jedinice nije od suštinske važnosti za složenost, s obzirom da se sve ove jedinice međusobno razlikuju za konstantan faktor (npr. jedan bajt je osam bitova), a konstantni faktori ne utiču na složenost (u smislu „veliko-O” notacije). Broj koraka algoritma predstavlja broj izvršenih aritmetičko logičkih operacija i operacija transfera podataka. Prostorna složenost algoritma će odgovarati maksimalnom ukupnom zauzeću memorije tokom rada programa.

Ovako definisana složenost realnih računarskih programa će u dobroj meri odgovarati složenosti odgovarajućih algoritama implementiranih pomoću Tjuringove mašine sa više traka. Na primer, složenost sabiranja binarnih brojeva u realnim računarima je linearna, baš kao i kod sabiranja binarnih brojeva pomoću Tjuringove mašine sa više traka. Ipak, postoje i slučajevi gde će realni računari moći da implementiraju neke postupke efikasnije nego što bi to bilo moguće na Tjuringovim mašinama. Na primer, algoritam binarne pretrage implementiran na realnom računaru zahteva $O(\log_2(n))$ koraka pretrage, gde je n broj elemenata niza. Svaki korak pretrage se obavlja u konstantnom vremenu, zato što je moguće neposredno pristupiti elementu niza sa proizvoljnim indeksom. Zato je i ukupna složenost ovog algoritma $O(\log_2(n))$. Tjuringova mašina bi za svaki korak pretrage morala da pomera glavu ka odabranoj poziciji u nizu, što zahteva $O(n)$ koraka. Otuda će složenost binarne pretrage na Tjuringovoj mašini biti $O(n \cdot \log_2(n))$.

Napomena 6.4. S obzirom da konstantni faktori ne utiču na složenost, u praksi nije neophodno precizno izbrojati bajtove koje neki ulaz zauzima. Dovoljno je da za veličinu ulaza n uzmemo neku veličinu koja je proporcionalna stvarnom broju bajtova. Na primer, grafovi se u računarima često predstavljaju kao liste listi – imamo listu čvorova, pri čemu je svakom čvoru pridružena lista grana koje iz tog čvora izlaze. Veličina ove strukture podataka je proporcionalna zbiru broja čvorova i broja grana $|V| + |E|$, pa se ovaj zbir može uzeti za veličinu ulaznog podatka n . Stvarni broj bajtova može biti nekoliko puta veći, u zavisnosti od veličine celobrojnih tipova podataka kojima predstavljamo čvorove i grane, kao i zbog eventualnog prisustva pokazivača u čvorovima listi. Ipak, složenost će ostati asimptotski ista, uz nešto drugačiji konstantni faktor.

Isto važi i kada je u pitanju prebrojavanje koraka programa. Na primer, izvršavanje naredbe $x = y + z$ nad podacima celobrojnog (*int*) tipa može zahtevati nekoliko mašinskih instrukcija (učitavanje podataka y i z u registre, sabiranje u ALU jedinici i upis rezultata u promenljivu x), ali je taj broj koraka konstantan i unapred fiksiran. Zato ovu naredbu možemo brojati kao jedan korak algoritma. Sa takvom računicom će se broj koraka algoritma smanjiti za konstantan faktor, ali to neće uticati na složenost u asimptotskom smislu.

Napomena 6.5. Složenost izračunavanja se uvek izražava asimptotski – razmatramo koliko brzo raste funkcija složenosti $t_T(n)$ (ili $s_T(n)$) kada n teži beskonačnosti. To znači da uvek moramo pretpostaviti da se veličina ulaza može neograničeno uvećavati. U realnim programima, ako nam je na ulazu, na primer, jedan podatak x tipa *int*, tada se on ne može neograničeno uvećavati, jer je njegova veličina ograničena veličinom tog tipa podatka (32 bita na većini računara). Da bismo analizirali složenost ovakvih algoritama, neophodno je da pretpostavi-

mo da je tip *int* zamenjen nekim celobrojnim tipom čija veličina u bitovima nije unapred fiksirana i može se uvećavati po potrebi.¹ Tek tada možemo pristupiti realnoj analizi složenosti algoritma.

Primer 6.8. Neka je dat sledeći program u jeziku C++:

```
i = 0;
while(i*i <= x) i++;
i--;
```

Ovaj program za ulaz x izračunava $\lfloor \sqrt{x} \rfloor$. Dominatni deo ovog algoritma je *while* petlja. Broj iteracija petlje je jednak $\lfloor \sqrt{x} \rfloor$. Da bismo asimptotski izrazili složenost, moramo pretpostaviti da je x nekog celobrojnog tipa čija veličina nije ograničena, te da broj bitova $n = \lfloor \log_2(x) \rfloor + 1$ neophodan za zapis podatka x može neograničeno da raste. U svakoj iteraciji petlje dominira množenje čija je složenost $O(n^2)$. Kako je $x = O(2^n)$, asimptotsku složenost gornjeg algoritma možemo izraziti u funkciji od n kao $O(n^2 \cdot \sqrt{2^n}) = O(2^{\log_2(n^2)} \cdot 2^{\frac{n}{2}}) = O(2^{2 \cdot \log_2(n) + \frac{n}{2}})$. Dakle, algoritam je *eksponencijalne* složenosti u odnosu na veličinu ulaza.

6.3 Složenost problema odlučivanja

Posebna pažnja u proučavanju složenosti izračunavanja se posvećuje problemima odlučivanja, tj. problemima čiji je izlaz uvek odgovor „da” ili „ne”. Ukoliko ulazne instance problema posmatramo kao niske nad nekom azbukom Σ , tada skup svih ulaznih niski $w \in \Sigma^*$ za koje je odgovor „da” čine jedan jezik L nad azbukom Σ . Problem koji rešavamo ćemo u nastavku poistovećivati sa upravo sa tim jezikom L .

Definicija 6.9. *Karakteristična funkcija* jezika $L \subseteq \Sigma^*$ je funkcija $\chi_L : \Sigma^* \rightarrow \{0, 1\}$, definisana na sledeći način:

$$\chi_L(w) = \begin{cases} 1, & \text{za } w \in L \\ 0, & \text{za } w \notin L \end{cases}$$

Za jezik L kažemo da je *odlučiv* ili *rekurzivan* ako je χ_L izračunljiva.

U nastavku ove glave podrazumevamo da radimo sa odlučivim jezicima, tj. problemima.

Tjuringova mašina koja izračunava karakterističnu funkciju jezika L bi na izlazu trebalo da daje 0 ili 1. To znači da bismo azbuku Σ morali da proširimo specijalnim simbolima kojima označavamo ova dva moguća izlaza, uz restrikciju da se oni ne mogu nalaziti na ulazu. Kako bismo izbegli tehničke komplikacije ovog tipa, obično se Tjuringove mašine za rešavanje problema odlučivanja definišu tako da uopšte ne proizvode izlaz – umesto toga, podrazumevamo da postoje dva *završna stanja* – q_{da} i q_{ne} u kojima se rad Tjuringove mašine završava u zavisnosti od toga da li ulazna instanca pripada jeziku L ili mu ne pripada. Mi ćemo se u ovom tekstu takođe držati ove konvencije.

¹Takvi tipovi obično ne postoje kao ugrađeni tipovi u programskim jezicima, ali su često podržani kao korisnički definisani tipovi u različitim bibliotekama za rad sa *velikim brojevima* (poput *GNU Multiprecision Library* (GMP) biblioteke).

Primer 6.9. Razmotrimo problem odlučivanja „Da li je dati broj x paran?“. Ako broj x predstavimo svojim binarnim zapisom, Tjuringova mašina koja odlučuje o ovom problemu bi prosto premotala glavu na desni kraj ulaza i proverila vrednost najnižeg bita. Ako je taj bit jednak 1, tada bi mašina prešla u stanje q_{ne} , a ako je jednak 0, tada bi prešla u stanje q_{da} . U oba slučaja, u konačnom broju koraka dobijamo odgovor na postavljeno pitanje. Dakle, ovaj problem je odlučiv. Složenost ove procedure odlučivanja je linearna, jer zahteva jedan prolaz kroz niz bitova.

Primer 6.10. Razmotrimo problem odlučivanja „Da li je broj x deljiv brojem y ?“. Ovaj problem bi se mogao rešiti tako što konstruišemo Tjuringovu mašinu koja vrši celobrojno deljenje i računa količnik i ostatak pri deljenju. Nakon što izračuna ostatak, potrebno je uporediti ga sa nulom – ako je jednak nuli, tada prelazimo u stanje q_{da} , a u suprotnom u q_{ne} . Složenost postupka celobrojnog deljenja je $O(n^2)$, dok poređenje sa nulom zahteva jedan prolaz kroz binarni zapis ostatka (što je $O(n)$). Dakle, ceo postupak je kvadratne složenosti.

6.4 Klasa P

U ovom poglavlju razmatramo jednu od najznačajnijih klasa složenosti – klasu *polinomske vremenske složenosti* koju označavamo sa P (ili PTIME).

Definicija 6.10. Klasu problema polinomske složenosti P čine svi problemi odlučivanja za koje postoji procedura odlučivanja čija je složenost $O(p(n))$ za neki polinom $p(n)$.

Primer 6.11. Sledeći problemi pripadaju klasi P:

- „Da li je broj x paran?“
- „Da li je broj x deljiv brojem y ?“
- „Da li je $x < y$?“
- „Da li binarni zapis broja x sadrži dve uzastopne nule?“
- „Da li je dati broj x stepen dvojke?“
- „Da li u datom nizu brojeva $x_1, \dots, x_n$ postoji par brojeva (x_i, x_j) takvi da je njihov proizvod $x_i \cdot x_j$ jednak datom broju x ?“

itd. Za neke od ovih problema smo već razmotrili procedure odlučivanja i odredili njihovu složenost. Za ostale probleme ostavljamo čitaocu da to uradi za vežbu.

Primer 6.12. Razmotrimo problem „Da li je dati neusmereni graf $G = (V, E)$ povezan?“ Graf je povezan ako za svaka dva čvora $u, v \in V$ postoji put $u = v_0, v_1, \dots, v_k = v$, pri čemu je $(v_i, v_{i+1}) \in E$ za svako $i \in \{0, 1, \dots, k-1\}$. Ovaj problem možemo rešiti tako što izvršimo obilazak grafa iz proizvoljnog početnog čvora i označimo sve čvorove koji su tom prilikom dostignuti. Ako su na kraju svi čvorovi dostignuti, znači da je graf povezan, dok u suprotnom nije. Procedura obilaska grafa (poput, npr. *obilaska u dubinu*) zahteva da se svaki čvor i svaka grana obiđe po jednom, što znači da je složenost $O(|V| + |E|)$, tj. $O(n)$ ako uzmemo da je veličina grafa $n = |V| + |E|$. Dakle, ovaj problem pripada klasi P.

Primer 6.13. Problem „Da li dati usmereni graf $G = (V, E)$ sadrži ciklus?“, tj. da li postoji put $v_0, v_1, \dots, v_k$ takav da je $(v_i, v_{i+1}) \in E$ za $i \in \{0, \dots, k-1\}$ i $v_0 = v_k$ se može, kao i prethodni, svesti na obilazak grafa. Ukoliko se tokom obilaska u dubinu nađe na čvor koji je već označen, znači da imamo ciklus, u suprotnom ciklus ne postoji. Dakle, složenost ove procedure je takođe linearna u odnosu na veličinu grafa, pa i ovaj problem pripada klasi P.

Primer 6.14. Razmotrimo problem „Da li je dati broj x potpun kvadrat?“. Ovaj problem je ekvivalentan sa ispitivanjem da li je $\lfloor \sqrt{x} \rfloor^2 = x$. U primeru 6.8 smo razmotrili jedan algoritam za izračunavanje $\lfloor \sqrt{x} \rfloor$ koji je bio eksponencijalne složenosti. Ipak, postoje i efikasniji algoritmi da se ovo izračuna. Ideja je da posmatramo interval $I = \{1, 2, \dots, x\}$ i da broj y takav da je $y \cdot y = x$ tražimo binarnom pretragom. Ako sa n označimo broj bitova koji zauzima zapis broja x , tada je $x = O(2^n)$, tako da interval I ima $O(2^n)$ elemenata. Binarnom pretragom je potrebno $O(\log_2(2^n)) = O(n)$ koraka da se u ovom intervalu locira traženi element y , ako postoji, ili da se utvrdi da takav element ne postoji. Za računanje središnjeg elementa tekućeg podintervala potrebno je vreme $O(n)$ (sabiranje i deljenje sa dva). Za njegovo kvadriranje je potrebno $O(n^2)$ koraka, a za upoređivanje sa x još $O(n)$ koraka. Otuda je složenost celog algoritma $O((n + n^2 + n) \cdot n) = O(n^3)$. Dakle, problem pripada klasi P.

Primer 6.15. Razmotrimo problem „Da li je niska $u \in \Sigma^*$ faktor niske $w \in \Sigma^*$?“. Za nisku u kažemo da je faktor niske w ako se niska w može zapisati u obliku $w = \alpha u \beta$, gde su $\alpha, \beta \in \Sigma^*$. Ovaj problem se može rešiti na sledeći način. Neka je $w = a_1 a_2 \dots a_m$, a $u = b_1 \dots b_k$. Ako je $m < k$, onda je odgovor definitivno „ne“. U suprotnom, najpre možemo uporediti niske $a_1 \dots a_k$ i $b_1 \dots b_k$. Ako su ove niske jednake, vraćamo „da“. U suprotnom, treba uporediti $a_2 \dots a_{k+1}$ sa $b_1 \dots b_k$, zatim $a_3 \dots a_{k+2}$ sa $b_1 \dots b_k$ i tako dalje, zaključno sa $a_{m-k+1} \dots a_m$. Ako ni jedna od ovih niski nije jednaka $b_1 \dots b_k$, vraćamo „ne“, u suprotnom vraćamo „da“. Složenost opisanog postupka je jednaka $k \cdot (m - k + 1)$. Ako je $n = m + k$ ukupna veličina ulaza, tada je $k = O(n)$ i $m = O(n)$, pa je složenost $O(n^2)$ u najgorem slučaju. Dakle, ovaj problem je takođe u klasi P. Napomenimo da postoje i efikasniji algoritmi koji rešavaju ovaj problem (npr. *Knut-Moris-Prat* (KMP) algoritam, čija je složenost linearna).

Primer 6.16. Za problem „Da li je dati broj x prost?“ dugo nije bio poznat ni jedan algoritam polinomske složenosti, tako da se nije znalo da li pripada klasi P ili ne. Tek 2002. godine je konstruisan prvi algoritam odlučivanja za koji je pokazano da je polinomske složenosti. Složenost ovog algoritma (poznatog i kao *AKS (Agrawal-Kayal-Saxena) test primalnosti*) uz sva poboljšanja koja su naknadno usledila, je $O(n^6)$, gde je n broj cifara broja x .

Napomena 6.6. Kada utvrđujemo da li je problem u klasi P, tada nam nije važan tačan stepen polinoma kojim je vreme izvršavanja programa ograničeno. Zbog toga analiza može biti dosta „gruba“ i lišena mnogih nepotrebnih detalja. Na primer, ne moramo se truditi da pronađemo najefikasniji mogući algoritam – bilo koji algoritam polinomske složenosti je dovoljan da utvrdimo da je problem u klasi P. Takođe, veličinu ulaza možemo zameniti bilo kojom veličinom n koja je *polinomski povezana* sa stvarnom veličinom ulaza N , tj. za koju važi da postoje polinomi p i q takvi da je $n = O(p(N))$ i $N = O(q(n))$. Na primer, kada analiziramo grafovske probleme, za veličinu grafa možemo uzeti broj čvorova grafa $n = |V|$. Iako znamo da je veličina grafa proporcionalna sa $N = |V| + |E|$, broj grana $|E|$ je uvek $O(n^2)$, pa će važiti da je $N = |V| + |E| = n + O(n^2) =$

$O(n^2)$, dok je $n \leq N$, tj. važi $n = O(N)$. Zato možemo koristiti n kao veličinu ulaza. Ovo može dovesti do toga da rezultujući polinom kojim ograničavamo vreme izvršavanja bude drugačijeg stepena, ali će i dalje biti polinom. Na primer, procedure zasnovane na obilasku grafa (kao u primerima 6.12 i 6.13) će sada biti kvadratne složenosti u odnosu na n , ali to je i dalje polinomsko ograničenje.