PROGRAMIRANJE I

TJURINGOVE MAŠINE I IZRAČUNLJIVE FUNKCIJE

1. UVODNA RAZMATRANJA

1.1. INTUITIVNI POJAM ALGORITMA

Na početku definišemo jednostavne pojmove azbuka i riječ. Neka je $n \in N$. Razmotrimo skup od n članova $A = \{a_1, \ldots, a_n\}$. Za skup A se kaže da je azbuka, a za njegove članove se kaže da su slova. Riječ w u azbuci A definiše se kao konačan niz slova te azbuke, gdje je moguće i da niz bude prazan. Na primjer, $A = \{0, 1, \ldots, 9\}$, $w = 314$, $|w| = 3$. Dužina riječi w, u oznaci $|w|$, definiše se kao broj članova tog niza, kao broj slova te riječi. Tako da je $|w| \geq 0$. Riječ čija je dužina $|w| = 0$ označava se kao $w = \square$, to je tzv. prazna riječ. Sa $\Omega(A)$ označavaćemo skup svih riječi u azbuci A.

Pogledajmo slučaj minimalne ili jednočlane azbuke, tj. slučaj $n = 1$. Razmotrimo azbuku $A_1 = \{a_1\}$. Pogledajmo skup svih riječi u toj azbuci. Očito je $\Omega(A_1) = \{\square, a_1, a_1a_1, a_1a_1a_1, \ldots\}$. Jednom članu skupa $\Omega(A_1)$ odgovara jedan član skupa N_0.

Slovo ili znak ili simbol a_k zauzima jedno polje na Tjuringovoj traci. A moguće je i da je polje na Tjuringovoj traci nepopunjeno. Tada se kaže da u tom polju piše znak ili simbol a_0.

Šta je to algoritam? Najprostije se kaže da je to jasno pravilo koje se sastoji iz jednostavnih koraka. Sada ćemo pokušati da intuitivnu predstavu o tom pojmu izrazimo malo jasnije. Algoritam $\mathcal{A}$ je uređena petorka $\mathcal{A} = (\mathcal{P}, E, A, B, n)$. Ovdje je $\mathcal{P}$ pravilo ili postupak. Imamo tri azbuke: E – ulazna azbuka, A – radna azbuka i B – izlazna azbuka. Ulazni podaci izraženi su u azbuci E, pravilo $\mathcal{P}$ zapisano je u azbuci A, dok azbuka B služi za izlazne podatke. Uzima se da je $E \subset A$ i $B \subset A$. Ulazni podaci sastoje se od n riječi $w_1, \ldots, w_n$, gdje $w_k \in \Omega(E)$. Izlazni podatak ili rezultat jeste neka riječ $r \in \Omega(B)$.

Uređena n–torka riječi $(w_1, \ldots, w_n)$ predstavlja polazni objekat prilikom izvršavanja operacija po pravilu $\mathcal{P}$. Kaže se da se $\mathcal{A}$ primjenjuje na tu n–torku riječi. Za riječ $r \in \Omega(A)$ kaže se da predstavlja rezultat ako se niz operacija (saglasno $\mathcal{P}$) obustavlja i ako tada imamo saopštenu jednu riječ r, s tim da još $r \in \Omega(B)$.

Navesti jedan primjer algoritma. Euklidov algoritam za nalaženje NZD dva prirodna broja temelji se na sljedećem iskazu: ako je $n_1 > n_2$ onda je $\mathrm{NZD}(n_1, n_2) = \mathrm{NZD}(n_2, n_1 \bmod n_2)$.

Pogledajmo neka svojstva intuitivnog pojma algoritma.

1 Tokom rada algoritma, operacije se izvršavaju jedna za drugom. Svaka operacija ukazuje koja će biti sljedeća operacija.

2 Pravilo $\mathcal{P}$ je jednoznačno u svim svojim detaljima.

3 Algoritam ima osobinu određenosti (determinisanosti). To znači da će algoritam, ma koliko puta se on izvršavao, uvijek proći kroz jedan te isti niz koraka, naravno za fiksirani ulazni podatak. Ne smije se nekoj promjenljivoj dodijeliti vrijednost x na osnovu rezultata bacanja novčića (bilo bi $x = 0$ ili $x = 1$) ili na osnovu bacanja kocke (bilo bi $1 \leq x \leq 6$). Drugim riječima, stohastičke algoritme ne smatramo algoritmima.

4 Pravilo $\mathcal{P}$ treba da bude sastavljeno tako da njegovo izvršavanje ne zahtijeva nikakvih spoljašnjih informacija, odnosno da su dovoljne informacije sadržane u ulaznoj n–torci riječi i u samom pravilu $\mathcal{P}$.

5 Pravilo $\mathcal{P}$ treba da bude konačno. Njegova dužina je ma koliko velika, ali ipak konačna.

6 Broj koraka potrebnih da se $\mathcal{P}$ izvrši treba da bude konačan.

7 $\mathcal{P}$ zahtijeva samo konačan memorijski prostor.

Nabrojana svojstva pretvaraju se u zahtjeve koje treba da zadovoljava formalna (precizna) definicija algoritma.

Šta računar može da uradi, a šta ne može? Pogledajmo istorijsku perspektivu i strateška pitanja. Znamo da izvršavanje algoritma ima čisto mehanički karakter i da nije potrebna bilo kakva vještina za to izvršavanje. Ako za neki matematički zadatak postoji algoritam za njegovo rješavanje onda je taj zadatak riješen do kraja, on je riješen na najbolji mogući način, postupak njegovog rješavanja je trivijalan. U prošlosti je prećutno vladalo uvjerenje da za bilo koji zadatak postoji algoritamsko rješenje, do jednog trenutka. Gedel je 1931. godine dokazao algoritamsku nerješivost nekih zadataka! Na jeziku programera: za te zadatke ne postoji program, misli se program za računar.

U tom trenutku postalo je jasno sljedeće. Treba jasno postaviti granicu šta je algoritam a šta nije, pa se tek nakon toga može ozbiljno govoriti o algoritamskoj rješivosti ili nerješivosti nekog zadatka. Drugim riječima, intuitivni pojam je nedovoljan i treba da bude zamijenjen preciznim (matematičkim) pojmom. Predloženo je niz matematičkih definicija pojma algoritma, od kojih navodimo Gedelovu definiciju iz 1931. godine, Čerčovu iz 1936. Tjuringovu iz 1936. i Postovu iz 1936. Dokazano je da su sve predložene definicije međusobno ekvivalentne: ovaj postupak je algoritam u smislu $\text{def}_1 \Leftrightarrow$ on je algoritam u smislu def_2. Kada se u nastavku kaže "matematička definicija algoritma" onda se ima u vidu jedna, bilo koja, od međusobno ekvivalentnih definicija tog pojma.

Pogledajmo kako glasi čuvena Čerčova teza ili Čerčova hipoteza. Ona tvrdi da se intuitivni pojam algoritma poklapa sa matematičkim pojmom algoritma. Ne može se govoriti o eventualnom dokazu Čerčove teze, budući da ona stavlja znak jednakosti između dva pojma, od kojih je jedan unutar matematike, a drugi je van matematike. Smatra se da je Čerčova teza dobro utemeljena i zato je ona praktično opšte–prihvaćena.

Sličan iskaz: računar = fizička realizacija Tjuringove mašine. Drugi sličan iskaz: ako je neki zadatak nerješiv za Tjuringovu mašinu onda je taj zadatak nerješiv i za bilo kakvu računsku mašinu, kao i obrnuto.

1.2. POJAM IZRAČUNLJIVE FUNKCIJE I POJAM RJEŠIVOG SKUPA

Poznata je oznaka $A \times B$ za Dekartov proizvod dva skupa, kao i oznaka $\underbrace{A \times \ldots \times A}_{n \text{ puta}} = A^n$. Razmotrimo funkciju $f\colon \Omega^n(E) \to \Omega(B)$ čiji je domen (oblast definisanosti) očito skup $\text{Dom}\, f = \underbrace{\Omega(E) \times \ldots \times \Omega(E)}_{n \text{ puta}}$. Za funkciju f kaže se da je potpuna ili totalna. Razmatraćemo i tzv. djelimične ili parcijalne funkcije. Njihov domen je $\text{Dom}\, f \subset \Omega^n(E)$. Dakle, za funkciju $f\colon \text{Dom}\, f \to \Omega(B)$ kaže se da je djelimična funkcija iz skupa $\text{Dom}\, f$ u skup $\Omega(B)$ ako je njen domen $\text{Dom}\, f \subset \Omega^n(E)$. Kada je $\text{Dom}\, f \subset \Omega^n(E)$ onda je naravno moguće i da bude $\text{Dom}\, f = \Omega^n(E)$.

Polazimo od jednog algoritma $\mathcal{A} = (\mathcal{P}, E, A, B, n)$. Na sasvim prirodan način, algoritmu se pridružuje jedna djelimična funkcija, označavaćemo je kao f ili eventualno kao $f_{\mathcal{A}}$. Upravo, neka n–torka riječi $\mathcal{M} = (w_1, \ldots, w_n) \in \Omega^n(E)$ posluži kao ulazni podatak algoritma $\mathcal{A}$. Dobiće se rezultat (to znači da $\mathcal{M} \in \text{Dom}\, f$) ili se neće dobiti rezultat (to znači da $\mathcal{M} \notin \text{Dom}\, f$). Znamo da "dobiće se rezultat" znači: $\mathcal{A}$ će raditi samo konačan broj koraka i $\mathcal{A}$ će nam saopštiti riječ r, s tim da $r \in \Omega(B)$. Tada je naravno $f(\mathcal{M}) = r$. Vidimo da se pomoću algoritma određuje domen funkcije, a takođe i njene vrijednosti u tačkama iz domena.

Maločas smo posmatrali u smjeru od algoritma prema funkciji. Prosto se razumije da je funkcija $f_{\mathcal{A}}$ izračunljiva. Pogledajmo sada iz suprotnog ugla. Praktično, neka je data cjelobroj-

na funkcija od više cjelobrojnih promjenljivih, bilo da je djelimična ili totalna. Ona je data formulom ili riječima ili slično. Ima li algoritma (programa) za saznavanje njenih vrijednosti u pojedinim tačkama?

Definicija. Razmotrimo djelimičnu funkciju $f\colon \mathrm{Dom}\, f \to \Omega(B)$ ($\mathrm{Dom}\, f \subset \Omega^n(E)$). Za funkciju f kaže se da je izračunljiva ako postoji algoritam $\mathcal{A} = (\mathcal{P}, E, A, B, n)$ takav da je $f = f_{\mathcal{A}}$. Tada se za algoritam $\mathcal{A}$ kaže da predstavlja računsku proceduru za funkciju f.

Egzotični primjer. Razmotrimo funkciju f definisanu sa:

$$f(n) = \begin{cases} 0, \text{ ako je Rimanova hipoteza istinita} \\ 1, \text{ u suprotnom slučaju} \end{cases}$$

gdje je $n = 0, 1, 2, \ldots$. Da li je funkcija f izračunljiva?

Odgovor: jeste. Makar što mi danas nismo u stanju da napišemo jedan algoritam i onda da kažemo: evo ovo je računska procedura.

Zapaziti da je funkcija f konstanta. Neka bude: A_0: učitaj broj n i onda odštampaj 0 i A_1: učitaj n i onda odštampaj 1. Jedan od ta dva algoritma je računska procedura. Dakle, računska procedura postoji.

Umjesto n može se pisati $\underbrace{a_1 \ldots a_1}_{n \text{ puta}}$. Znamo da je Rimanova hipoteza jedan čuveni iskaz čija istinitost je pod znakom pitanja. Rimanova hipoteza odnosi se na pitanje o tome gdje se nalaze nule Rimanove funkcije $\zeta = \zeta(z)$, gdje je $\zeta(z) = \sum_{n=1}^{\infty} \frac{1}{n^z}$, $\mathrm{Re}\, z > 1$, a zatim se analitički produži. Završen egzotični primjer.

Prelazimo na pojam rješivog skupa. Moguće je da funkcija bude totalna i da ima samo dvije moguće vrijednosti.

Neka su $\mathcal{M}_1$ i $\mathcal{M}_2$ dva skupa i neka je $\mathcal{M}_1 \subset \mathcal{M}_2$. Kaže se da je $\mathcal{M}_1$ rješiv u odnosu na $\mathcal{M}_2$ ako postoji algoritam koji može da se primijeni na bilo koji član skupa $\mathcal{M}_2$ i koji u tom slučaju daje odgovor na pitanje – da li taj član pripada $\mathcal{M}_1$ ili ne pripada.

Počinju primjeri koji se tiču diofantskih jednačina.

Primjer 1. Ruski matematičar Matijasevič riješio je 1971. godine jedan čuveni matematički problem. On je riješio tzv. 10. Hilbertov problem. Dokazao je da je pitanje koje se tiče tzv. opšte diofantske jednačine – algoritamski nerješivo. Mali primjer: da li jednačina $x_1^6 x_2^4 x_3 - 4x_1^2 x_3^2 + 8x_1^2 - 12x_3 + 7 = 0$ ima cjelobrojno rješenje $(x_1, x_2, x_3) \in Z \times Z \times Z$? Pogledajmo postavku zadatka, a dokaz se izostavlja.

Neka je $p = p(x_1, \ldots, x_k)$ polinom ma kog stepena od $k \geq 1$ promjenljivih čiji koeficijenti su cijeli brojevi. Za jednačinu $p(x_1, \ldots, x_k) = 0$ kaže se da je opšta diofantska jednačina. Da li jednačina ima cjelobrojno rješenje, ima rješenje koje pripada skupu Z^k? Program treba da učita podatke o polinomu p i onda da odštampa "ima cjelobrojno rješenje" ili da odštampa "nema cjelobrojnog rješenja".

Takav program ne postoji.

Mali primjer od maločas: "ima", $(x_1, x_2, x_3) = (1, 1, 1)$. Mali primjer: $x_1^4 x_2^4 + 4x_1^2 x_3^2 + 6x_1^2 + 8x_4^2 + 10 = 0$, "nema".

Jedino se može sastaviti program koji, zavisno od ulaznog podatka, štampa "ima" (odgovor je tačan) ili "nema" (odgovor je tačan) ili "za ovu jednačinu ne znam".

Mogli smo uvesti oznake $\mathcal{M}_1 \subset \mathcal{M}_2$, $\mathcal{M}_2 = \{p(x_1, \ldots, x_k) = 0\}$, $\mathcal{M}_1 = \{\text{"ima"}\}$.

Primjer 2. Neka je $\mathcal{M}_2$ skup svih jednačina oblika $p(x) = 0$, gdje je $p = p(x)$ polinom ma kog stepena od jedne promjenljive sa cijelim koeficijentima, a $\mathcal{M}_1$ je podskup kome pripadaju jednačine koje imaju bar jedno cjelobrojno rješenje.

Skup $\mathcal{M}_1$ je rješiv u odnosu na $\mathcal{M}_2$.

Poznata su sljedeća dva pravila. Na osnovu drugog pravila lako se sastavi algoritam (program).

Pravilo 1. Razmotrimo jednačinu $x^n + a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0 = 0$, gdje je $a_k \in Z$ za $0 \leq k \leq n-1$ i gdje je $n \geq 1$. Među svim cijelim brojevima, jedino su činioci slobodnog sabirka a_0 kandidati za rješenje postavljene jednačine.

Mali primjer: jedino brojevi $x = \pm 1, \pm 2, \pm 3, \pm 6$ dolaze u obzir da budu rješenja jednačine $x^3 - 2x^2 + 7x - 6 = 0$.

Pravilo 2. Razmotrimo jednačinu $a_nx^n + a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0 = 0$, gdje je $a_k \in Z$ za $0 \leq k \leq n$, $a_n \neq 0$ i $n \geq 1$. Među svim racionalnim brojevima, jedino su brojevi oblika $\pm p/q$ kandidati za rješenje polazne jednačine, gdje je p činilac slobodnog sabirka a_0, a q je činilac najstarijeg koeficijenta a_n.

Mali primjer: jedino brojevi $x = \pm \frac{1,2,3,6}{1,2}$ dolaze u obzir da budu racionalna rješenja jednačine $2x^3 - 2x^2 + 7x - 6 = 0$.

Tako da algoritam koji saopštava "da" ili "ne", u smislu: jednačina iz skupa jednačina $\mathcal{M}_2 = \{a_nx^n + \ldots + a_0 = 0\}$ pripada ili ne pripada skupu $\mathcal{M}_1 = \{\text{"ima"}\}$, glasi kako slijedi, u grubim crtama. Pošto se učitaju podaci o polinomu $p(x) = a_nx^n + \ldots + a_0$ (to su $n, a_n, \ldots, a_0$), sastavi se spisak svih racionalnih kandidata $x = \pm p/q$ (gdje $p \mid a_0$ i $q \mid a_n$). Spisak se redukuje time što se iz spiska uklone brojevi koji nisu cijeli. Zatim se redom vrši supstitucija jednog po jednog člana spiska u polinom, sve dok se ne pokaže da neki član zadovoljava jednačinu (štampati "pripada" i zaustaviti se) ili dok se spisak ne potroši (štampati "ne pripada" i zaustaviti se).

Primjer 3. Neka je $\mathcal{M}_2$ skup svih jednačina oblika $a_1x_1 + a_2x_2 + \ldots + a_kx_k = b$, gdje $a_1, \ldots, a_k, b \in Z$ i $k \geq 1$. Razmotrimo skup $\mathcal{M}_1 \subset \mathcal{M}_2$. Jednačina pripada skupu $\mathcal{M}_1$ ako i samo ako ona ima bar jedno cjelobrojno rješenje, tj. ako postoje brojevi $x_1 \in Z$, $x_2 \in Z$, ..., $x_k \in Z$ takvi da je $a_1x_1 + a_2x_2 + \ldots + a_kx_k = b$.

Mali primjer: jednačina $2x_1 + 3x_2 = 28$ pripada skupu $\mathcal{M}_1$. Zaista, $(x_1, x_2) = (5, 6)$. Mali primjer: jednačina $2x_1 - 2x_2 + 4x_3 = 7$ ne pripada $\mathcal{M}_1$. Zaista, ma kakvi da su $x_1 \in Z$, $x_2 \in Z$ i $x_3 \in Z$, na lijevoj strani jednačine je paran broj, a na desnoj strani je neparan broj 7.

Poznato je sljedeće pravilo. Na osnovu tog pravila lako se može sastaviti odgovarajući algoritam. Tako da je skup $\mathcal{M}_1$ algoritamski rješiv u odnosu na skup $\mathcal{M}_2$.

Pravilo. Neophodan i dovoljan uslov da jednačina pripada skupu $\mathcal{M}_1$ glasi $\text{NZD}(a_1, \ldots, a_k) \mid b$, tj. slobodan sabirak b može da se podijeli bez ostatka sa $\text{NZD}(a_1, \ldots, a_k)$. $a \preceq b$ ili $a \mid b$ znači $ak = b$ za neko k. Npr. $2 \mid 6$. Umjesto oznake Z, moglo bi D za skup cijelih brojeva.

Završeni su primjeri koji se tiču diofantskih jednačina.

Definicija. Neka je $\text{card}(E) \geq 1$ i $n \geq 1$. Razmotrimo skup $\mathcal{S} \subset \Omega^n(E)$. Za skup $\mathcal{S}$ kaže se da je rješiv u odnosu na skup $\Omega^n(E)$ ako postoji algoritam $\mathcal{A} = (\mathcal{P}, E, A, B, n)$ takav da: kada se algoritam $\mathcal{A}$ primijeni na bilo koju uređenu n–torku riječi $\mathcal{M} \in \Omega^n(E)$ onda se taj algoritam zaustavlja (tj. njegovo izvršavanje uvijek traje samo konačan broj taktova) i saopštava kao rezultat riječ $\square$ ili riječ a_1, s tim da saopštava $\square$ kada $\mathcal{M} \in \mathcal{S}$, odnosno saopštava a_1 u slučaju da $\mathcal{M} \notin \mathcal{S}$. Za svaki takav algoritam kaže se da predstavlja rješavajuću proceduru za $\mathcal{S}$ u odnosu na $\Omega^n(E)$. Oznaka: $\text{card}(E)$ – kardinalni broj skupa E.

Sljedeća teorema svodi pitanje rješivosti skupa na pitanje izračunljivosti funkcije. Neka je $\mathcal{S} \subset \Omega^n(E)$. Znamo da se karakteristična funkcija skupa $\mathcal{S}$ u odnosu na skup $\Omega^n(E)$ obično označava kao χ ili eventualno kai $\chi_{\mathcal{S}}$ i da se definiše sljedećom relacijom:

$$\chi(x) = \begin{cases} \square \text{ ako } x \in \mathcal{S} \\ a_1 \text{ ako } x \notin \mathcal{S} \end{cases}$$

gdje $x \in \Omega^n(E)$. Zapažamo da je karakteristična funkcija svuda definisana: ona je definisana na čitavom skupu $\Omega^n(E)$. Njene vrijednosti $\chi(x)$ pripadaju skupu $\Omega(E)$.

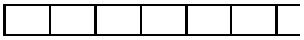
Teorema. Skup $\mathcal{S}$ je rješiv u odnosu na skup $\Omega^n(E)$ ako i samo ako je karakteristična funkcija χ skupa $\mathcal{S}$ u odnosu na skup $\Omega^n(E)$ – izračunljiva.

Dokaz. Pretpostavimo da je $\mathcal{S}$ rješiv u odnosu na $\Omega^n(E)$. To znači da postoji rješavajuća procedura (algoritam) $\mathcal{A} = (\mathcal{P}, E, A, B, n)$. Skup vrijednosti karakteristične funkcije je $\{\Box, a_1\}$, a bilo koja azbuka E sadrži bar jedno slovo a_1, tako da se može smatrati da je $B = E$. Vidi se da je $\mathcal{A}$ jedna računska procedura za χ (da je $f_{\mathcal{A}} = \chi$), čime je pokazano da je χ jedna izračunljiva funkcija.

Što se tiče dokaza u suprotnom smjeru (χ izračunljiva $\Rightarrow$ $\mathcal{S}$ rješiv), on je također veoma jednostavan i svodi se na prosto upoređivanje jedne i druge definicije (izračunljivosti i rješivosti). Teorema je dokazana.

2. PREGLEDAN OPIS I DEFINICIJA TJURINGOVE MAŠINE

2.1. DEFINICIJA TJURINGOVE MAŠINE



Pogledajmo prvo prebrojivu traku. Traka je sa lijeve strane ograničena, a sa desne nije, a podijeljena je na polja. U jednom polju je upisano jedno slovo ili je polje nepopunjeno (tada se kaže da polje sadrži znak a_0). Također se uvodi zahtjev da je samo konačno mnogo polja popunjeno u trenutku kada počinje rad mašine.

Naredbe koje čine program mogu da budu: izmjena sadržaja jednog polja trake, traženje susjednog polja i zaustavljanje računanja. Polje na koje se odnosi naredba koja se u određenom taktu izvršava naziva se tekućim radnim poljem. Svaka naredba treba da ukaže i – koja se naredba izvršava nakon nje, osim ako ona naređuje zaustavljanje.

Tjuringova mašina sastoji se od: **operacionog uređaja** koji može da se nalazi u jednom od diskretnih **stanja** $q_0, q_1, \ldots, q_s$ koja zajedno čine jedan konačan skup $Q = \{q_0, q_1, \ldots, q_s\}$, **glave** koja može da čita i piše, **prebrojive trake** opisane maločas i **mehanizma** za pomijeranje trake. Stanje q_0 naziva se **početnim stanjem** mašine: mašina se nalazi u stanju q_0 kada njen rad počinje. Polja trake numerisana su slijeva udesno brojevima $0, 1, 2, \ldots$. U svakom datom taktu, glava se nalazi iznad određenog polja trake – tekućeg **radnog polja**. Pomoću mehanizma za pomijeranje trake, bilo koje od dva susjedna polja može da bude dovedeno ispod glave za čitanje i pisanje. U tom slučaju kaže se da se radno polje pomjerilo za jedno mjesto udesno ili ulijevo.

Glava može da čita slova azbuke $A = \{a_1, \ldots, a_t\}$ i slovo a_0, ona može da izbriše slovo ili da ga upiše. Za A se kaže da je **radna azbuka** mašine. Svako polje trake u svakom taktu sadrži slovo iz skupa $A \cup \{a_0\}$. Pri tome su skoro sva polja zauzeta slovom a_0.

Lampa MZ (mašinsko zaustavljanje) upali se prilikom izvršavanja naredbe o zaustavljanju (naredbe s). Lampa PT (prelazak iza kraja trake) upali se kada je zaustavljanje izazvano time što se pod glavom nalazi početno polje trake ($m = 0$), a zahtijeva se pomijeranje radnog polja ulijevo (zahtijeva se da se izvrši naredba l). U jednom i drugom slučaju, MZ i PT, mašina se zaustavila.

Razmotrimo sada jednu konkretnu Tjuringovu mašinu T. Neka je njena radna azbuka $\{a_1\}$. Neka ona ima tri radna stanja q_0, q_1 i q_2.

Redosljed radnji u stanju q_0: nezavisno od sadržaja radnog polja (tj. nezavisno od toga da li je u radnom polju znak a_0 ili a_1), mašina T pomijera radno polje udesno i zatim prelazi u stanje q_1. Redosljed radnji u stanju q_1: mašina T zamjenjuje slovo koje stoji upisano u radnom polju (to je a_0 ili a_1) slovom a_1 i zatim prelazi u stanje q_2. Redosljed radnji u stanju q_2: mašina T se zaustavlja, nezavisno od sadržaja radnog polja.

Očito je da mašina T radi sljedeće: nakon postavljanja glave iznad nekog polja trake (stanje q_0), mašina pomijera radno polje udesno, briše sadržaj tog polja, tu upisuje a_1 i zaustavlja se.

Rad mašine T može da se opiše sljedećom tablicom:

q_0	a_0	r	q_1
q_0	a_1	r	q_1
q_1	a_0	a_1	q_2
q_1	a_1	a_1	q_2
q_2	a_0	s	q_2
q_2	a_1	s	q_2

Napominje se da simbol q_2 koji stoji na kraju dva posljednja reda tablice nema nikakvog značaja: kada se mašina zaustavila onda njeno unutrašnje stanje više nije značajno.

Na redu je definicija Tjuringove mašine, na redu je precizna definicija tablice (ili tablice rada ili programa) Tjuringove mašine T.

Neka je $t \geq 1$ i neka je $s \geq 1$. To znači da je radna azbuka $A = \{a_1, \ldots, a_t\}$ i da je skup stanja $Q = \{q_0, q_1, \ldots, q_s\}$. Tablica mašine je jedna matrica koja ima četiri stupca i $(s+1)(t+1)$ redova. Pogledajmo prvi stubac: prvih $t+1$ redova počinje sa q_0, sljedećih $t+1$ redova počinje sa q_1, itd. U drugom stupcu, u prvih $t+1$ redova upisano je redom $a_0, a_1, \ldots, a_t$ a slično dalje. Opišimo sada sadržaj trećeg stupca. Ako je upisano r onda se radno polje pomijera za jedno mjesto udesno. Ako je upisano l onda se radno polje pomijera za jedno mjesto ulijevo; ovo ne važi u slučaju da je radno polje $m = 0$ jer je tražena radnja nemoguća, nego se tada mašina zaustavlja i u tom slučaju kažemo da je mašina izašla preko kraja trake (PT). Ako je upisano s onda se mašina takođe zaustavlja i u tom slučaju kažemo da je došlo do mašinskog zaustavljanja (MZ). Na kraju, ako je upisano slovo $v \in A \cup \{a_0\}$ onda glava briše sadržaj radnog polja i upisuje u to polje slovo v. Na kraju, u četvrtom stupcu naznačeno je sljedeće stanje mašine.

Stanje q_0 je početno stanje mašine. Završena definicija.

Vidimo da pojedini red tablice ima oblik $qavq'$, gdje q pripada skupu stanja Q, dok a pripada proširenom skupu slova $A \cup \{a_0\}$. Prve dvije komponente q i a imaju "ulazni" karakter, dok treća i četvrta komponenta v i q' imaju "izlazni" smisao. Dakle, pojedini red tablice treba čitati kao: ako q i a onda v i q'. Ili detaljnije: ako je mašina u stanju q i ako radno polje sadrži slovo a onda neka mašina izvrši radnju $v \in A \cup \{a_0, r, l, s\}$ i zatim neka ona pređe u stanje $q' \in Q$.

2.2. RAČUNAR NASPRAM TJURINGOVE MAŠINE

Prosto rečeno, mogućnosti računara poklapaju se sa mogućnostima Tjuringove mašine. Kaže se da je računar = fizička realizacija Tjuringove mašine. Zato što je računar jedan stvarni (fizički) uređaj, dok je Tjuringova mašina jedan zamišljeni ili matematički pojam. Dva uređaja, računar i Tjuringova mašina, poklapaju se u pogledu mogućnosti računanja vrijednosti funkcije (v. ranije definiciju izračunljive funkcije), u pogledu mogućnosti raspoznavanja skupa (v. ranije $\mathcal{M}_1 \subset \mathcal{M}_2$), itd. Za vježbu, dokazati sljedeće: onu obradu koju može da izvrši računar zasnovan na procesoru (recimo) Intel 8086 može i Tjuringova mašina da ostvari. Kao i obrnuto.

Ipak, da bi dokaz mogao da bude sproveden, treba izvršiti dvije sasvim prirodne korekcije, kako slijedi, tiču se memorijskog prostora. Umjesto "traka Tjuringove mašine je polu–beskonačna" treba da stoji: na početku rada mašine, traka se sastoji od jednog konačnog broja polja, s tim da tokom rada mašine mogu da se po mjeri potrebe dodaju nova polja (na desnoj strani trake). Umjesto "memorija računara sastoji se od konačnog broja memorijskih riječi (bajta)" treba da stoji: na početku ima konačno mnogo memorijskih riječi, s tim da, tokom rada računara, raspoloživi memorijski prostor može da se povećava (mogu da se memoriji dodaju nove memorijske riječi, po mjeri potrebe).

Uobičajeni su izrazi aktuelna i potencijalna beskonačnost. Traka je polu–beskonačna – to znači da je memorijski prostor sa kojim raspolaže Tjuringova mašina aktuelno beskonačan.

Traka je konačna, s tim da kako vrijeme prolazi tako traci mogu da se dodaju nova polja – to znači da je memorijski prostor sa kojim raspolaže Tjuringova mašina samo potencijalno beskonačan.

2.3. POJMOVI KONFIGURACIJA MAŠINE I POZICIJA MAŠINE

U ovom naslovu uvodi se nekoliko običnih oznaka i termina.

Koji je trenutni sadržaj svih polja $m = 0$, $m = 1$, ... Zapis na traci ili kratko **zapis** u oznaci $\mathcal{J}$ shvatamo kao funkciju $\mathcal{J}\colon N_0 \to A \cup \{a_0\}$. Nejednakost $J(i) \neq \star$ istinita je samo za konačno mnogo i, kako je ranije rečeno. Isto je a_0 ili $\star$.

Želimo da snimimo trenutno stanje mašine. Pod konfiguracijom C mašine T podrazumijeva se uređena trojka $(m, \mathcal{J}, q)$, gdje je $m \geq 0$ redni broj tekućeg radnog polja, $\mathcal{J}$ je tekući zapis a $q \in Q$ je tekuće stanje mašine. Tako da je $\mathcal{J}(m)$ – sadržaj radnog polja. Početnom konfiguracijom naziva se konfiguracija čija je treća komponenta jednaka q_0.

Neka je $C = (m, \mathcal{J}, q)$ – neka konfiguracija mašine. Uočimo red tablice čije su prve dvije komponente q i $\mathcal{J}(m)$ tj. uočimo red tablice koji glasi $q\ \mathcal{J}(m)\ v\ q'$, treću komponentu u redu označili smo sa v a četvrtu sa q'. Kao što znamo, ako je $v \neq s$ i nije ($v = l$ i $m = 0$) onda izvršavanje radnje $v \in A \cup \{a_0, r, l, s\}$ dovodi do novog radnog polja m' (gdje je $m' = m - 1$ ili $m' = m$ ili $m' = m + 1$) i do novog zapisa $\mathcal{J}'$ (s tim da je $\mathcal{J}'(k) = \mathcal{J}(k)$ za svako $k \geq 0$ osim možda za jedno k) i još mašina zatim prelazi u novo stanje q'. Jednoznačno definisanu konfiguraciju $C' = (m', \mathcal{J}', q')$ nazivamo **konfiguracijom koja slijedi poslije** C. Ako je (suprotno) $v = s$ ili ($v = l$ i $m = 0$) onda sljedeća konfiguracija ne postoji. U tom slučaju C nazivamo **završnom konfiguracijom** mašine.

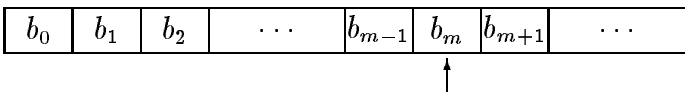
Očito je da na rad mašine utiču pored njene tablice i početni zapis i indeks početnog radnog polja. Kažemo da se mašina T **primjenjuje** na zapis $\mathcal{J}$ u radnom polju m ako je za početnu konfiguraciju izabrana konfiguracija $C = (m, \mathcal{J}, q_0)$.

Kada se mašina pusti da radi onda se konfiguracije smjenjuju iz takta u takt.

Konfiguracija $C_0 = (m, \mathcal{J}, q_0)$ na jednoznačan način generiše konačan ili beskonačan niz konfiguracija $C_0, C_1, C_2, \ldots$. Uvodi se prirodna terminologija. Kaže se da je C_i konfiguracija poslije i taktova (poslije i koraka). Kaže se da u i–tom koraku mašina prelazi od C_{i-1} na C_i. Uzmimo da je niz $C_0, C_1, C_2, \ldots$ konačan i da je C_i posljednji član tog niza. Tada se kaže da se poslije i koraka mašina zaustavlja. I kaže se da se mašina zaustavila poslije konačnog broja koraka. Niz je beskonačan – mašina radi vječno.

Ako je $C = (m, \mathcal{J}, q)$ konfiguracija onda je $S = (m, \mathcal{J})$ **pozicija**. Pozicije su pogodno sredstvo za opisivanje procesa računanja, ako nas ne interesuju stanja mašine. Na pozicije se skoro doslovno prenosi terminologija uvedena za konfiguracije.

Za jasno prikazivanje C koristi se oznaka $b_0 b_1 \ldots b_m^q \ldots$ što znači: u i–tom polju piše slovo b_i ($i \geq 0$), m–to polje je radno i mašina se nalazi u stanju q. Za jasno prikazivanje S koristi se oznaka $b_0 b_1 \ldots \underset{\uparrow}{b_m} \ldots$ što znači: u i–tom polju piše slovo b_i ($i \geq 0$) i m–to polje je radno. Ponekad se piše ($b_i \in A$ ili $b_i = a_0$):



Povezani dio zapisa koji nas ne interesuje ili svejedno niz od nekoliko uzastopnih polja trake čiji nam sadržaj nije poznat označava se kao $\sim$. Umjesto $\sim$ može se pisati – ako smo sigurni da se taj povezani dio zapisa sastoji od tačno jednog polja. Početak trake označava se sa], kada je bitno da početak trake bude prikazan. Ako na kraju zapisa stoji simbol $\star\ldots$ onda to

znači da dalje na traci dolaze sve znaci $\star$. Znamo da je $\star$ zamjenica za a_0, a $|$ za a_1. Primjeri $\underset{\uparrow}{\star} \sim \underset{\uparrow}{|} - \star$ ili $] \star\star \underset{\uparrow}{|} \star |$ ili $] \star | \underset{\uparrow}{\star} \ldots$ gdje je $m = 2$, $\mathcal{J}(0) = a_0$, $\mathcal{J}(1) = a_1$, $\mathcal{J}(2) = a_0$, ...

Želimo da sagledamo ukupan učinak rada jedne Tjuringove mašine T. Neka su S i S' dvije moguće pozicije mašine T. Oznaka $S \overset{T}{\Rightarrow} S'$ ima sljedeći smisao: ako je S početna pozicija mašine T onda se mašina T zaustavlja poslije konačnog broja koraka i to je S' njena završna pozicija. Primjer $] \star w \underset{\uparrow}{\star} \ldots \overset{T}{\Rightarrow}] - \star | \underset{\uparrow}{\star}$, ovdje je w neka riječ tj. $w \in \Omega(A)$.

Još neke definicije. **Primijeniti T na zapis poslije riječi w nad A** znači da se kao početna pozicija mašine uzima pozicija $] \star w \underset{\uparrow}{\star} \ldots$, gdje $w \in \Omega(A)$. Primijeniti T na zapis poslije n–torke riječi $w_1, \ldots, w_n$ nad A znači da se kao početna pozicija mašine uzima pozicija $] \star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$, gdje $w_k \in \Omega(A)$ za $1 \leq k \leq n$. Primijeniti T na zapis ispred riječi w nad A znači da se kao početna uzima pozicija $] \underset{\uparrow}{\star} w \star \ldots$, gdje $w \in \Omega(A)$. Primijeniti T na zapis ispred n–torke riječi $w_1, \ldots, w_n$ znači da se kao početna uzima pozicija $] \underset{\uparrow}{\star} w_1 \star \ldots \star w_n \star \ldots$, gdje $w_k \in \Omega(A)$ za $1 \leq k \leq n$. Primijeniti T na praznu traku znači da se kao početna uzima pozicija $] \underset{\uparrow}{\star} \ldots$.

Neka $w \in \Omega(A)$. Kaže se da se mašina T **zaustavila poslije riječi** w ako završna pozicija mašine T glasi $] \sim \star w \underset{\uparrow}{\star}$. Kaže se da se mašina T zaustavila ispred riječi w ako završna pozicija mašine T glasi $] \sim \underset{\uparrow}{\star} w \star$. Ne može se ništa reći o sadržaju polja (u završnoj poziciji) koja se nalaze desno od drugog napisanog znaka $\star$, misli se na slučaj zaustavila se poslije riječi a isto tako i na slučaj zaustavila se ispred riječi.

Primjedba. Neka se mašina zaustavila poslije neke riječi, tj. neka bude $S = (m, \mathcal{J})$ i $S \overset{T}{\Rightarrow}] \sim \star w \underset{\uparrow}{\star}$. Da li je razlog njenog zaustavljanja MZ ili PT? Razlog je svakako MZ (mašinsko zaustavljanje). Zaista, znamo da u slučaju PT (prelazak preko kraja trake) mora da bude $m = 0$, dok je u našem slučaju svakako $m \geq 1$. Vidimo da je u našem slučaju $m = 1$ samo ako je $\sim = \square$ i $w = \square$. Sa m smo označili indeks (redni broj) radnog polja u završnoj poziciji Tjuringove mašine.

2.4. POJAM FUNKCIJE IZRAČUNLJIVE PO TJURINGU

U ovom naslovu uvode se dva pojma: funkcija izračunljiva po Tjuringu (skraćeno i. T.) i funkcija normalno izračunljiva po Tjuringu.

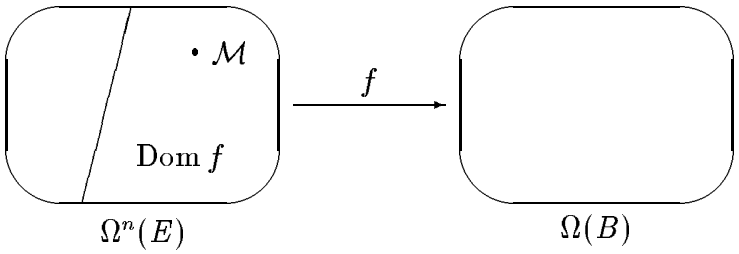
Neka je T T. m. sa radnom azbukom A. Neka je $E \subset A$, $B \subset A$ i $n \geq 1$. Mašina T definiše jednu funkciju f od n promjenljivih, iz $\Omega^n(E)$ ili nekog njegovog dijela u $\Omega(B)$, po sljedećem pravilu. Uređena n–torka $\mathcal{M} = (w_1, \ldots, w_n) \in \Omega^n(E)$ pripada domenu $\mathrm{Dom}\, f$ ako i samo ako se mašina T, primijenjena na zapis poslije $\mathcal{M}$, zaustavlja poslije riječi $w \in \Omega(B)$. Ta riječ $w = f(w_1, \ldots, w_n)$ predstavlja vrijednost funkcije f u tački $\mathcal{M}$. Prema tome, za $(w_1, \ldots, w_n) \in \mathrm{Dom}\, f$ važi $] \star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \overset{T}{\Rightarrow}] \sim \star f(w_1, \ldots, w_n) \underset{\uparrow}{\star}$.

Definicija. Djelimična funkcija f: $\mathrm{Dom}\, f \to \Omega(B)$ ($\mathrm{Dom}\, f \subset \Omega^n(E)$) naziva se izračunljivom po Tjuringu ako postoji T. m. T čija je radna azbuka A (gdje je $E \subset A$ i $B \subset A$) takva da se funkcija od n promjenljivih definisana na skupu $\Omega^n(E)$ (ili na dijelu skupa) u skup $\Omega(B)$ definisana mašinom T poklapa sa f. Za svaku takvu mašinu T kaže se da računa funkciju f.

Prepričajmo definiciju. Uzmimo da je f djelimična funkcija iz $\mathrm{Dom}\, f$ u $\Omega(B)$, pri čemu je $\mathrm{Dom}\, f \subset \Omega^n(E)$, koja je izračunljiva po Tjuringu i uzmimo da je T T. m. koja računa tu funkciju. Neka se sada T primjenjuje na $\mathcal{M} \in \Omega^n(E)$. Sada su moguća tri ishoda: T se uopšte

ne zaustavlja ili T izlazi preko kraja trake ili dolazi do mašinskog zaustavljanja mašine T. Ako je nastupio treći ishod i ako se može reći da se T zaustavila poslije neke riječi (označimo tu riječ sa w) i ako, osim toga, $w \in \Omega(B)$ **onda** $\mathcal{M} \in \mathrm{Dom}\, f$ i $f(\mathcal{M}) = w$. U svim ostalim slučajevima $\mathcal{M} \notin \mathrm{Dom}\, f$.

Slika za funkciju izračunljivu po Tjuringu:



U slučaju $\mathcal{M} \in \mathrm{Dom}\, f$: u završnoj poziciji: radno polje sadrži $\star$, lijevo od radnog polja ima bar jedna $\star$, $f(\mathcal{M})$ je riječ između te dvije zvijezde, ta riječ $\in \Omega(B)$. Tako da radno polje nije početno polje trake. U poljima desno od radnog polja – upisano je bilo šta.

Uvjereni smo da pojam funkcije izračunljive po Tjuringu predstavlja adekvatnu formalizaciju intuitivnog pojma funkcije koja može da se izračuna. Zato se umjesto funkcija izračunljiva po Tjuringu može jednostavno kazati izračunljiva funkcija.

Tehnički je značajan pojam funkcije koja je **normalno** izračunljiva po Tjuringu.

Definicija. Za djelimičnu funkciju f: $\mathrm{Dom}\, f \to \Omega(B)$, pri čemu je $\mathrm{Dom}\, f \subset \Omega^n(E)$, kaže se da je normalno izračunljiva po Tjuringu ako postoji T. m. T koja računa f i još ispunjava sljedeća dva zahtjeva:

a) ako $\mathcal{M} \in \Omega^n(E) \setminus \mathrm{Dom}\, f$ onda se mašina T nikad ne zaustavlja poslije primjene na $\mathcal{M}$,

b) ako pak $\mathcal{M} = (w_1, \ldots, w_n) \in \mathrm{Dom}\, f$ onda

$$]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \overset{T}{\Rightarrow}]\star w_1 \star \ldots \star w_n \star f(\mathcal{M}) \underset{\uparrow}{\star} \ldots .$$

Prepričajmo posljednju definiciju. Uzmimo da je f djelimična funkcija iz $\mathrm{Dom}\, f$ u $\Omega(B)$ koja je normalno izračunljiva po Tjuringu i uzmimo da je T T. m. koja normalno računa tu funkciju. Neka se sada T primjenjuje na $]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$, gdje $w_k \in \Omega(E)$ za $k \in \{1, \ldots, n\}$. U negativnom slučaju tj. u slučaju $\mathcal{M} = (w_1, \ldots, w_n) \notin \mathrm{Dom}\, f$, mašina T se neće zaustaviti. A u pozitivnom slučaju tj. u slučaju $\mathcal{M} \in \mathrm{Dom}\, f$, važiće sljedeće:

$$]\sim\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \overset{T}{\Rightarrow}]\sim\star w_1 \star \ldots \star w_n \star w \underset{\uparrow}{\star} \ldots ,$$

gdje je $w = f(\mathcal{M}) = f(w_1, \ldots, w_n)$, pri čemu tokom rada mašine ona nikad neće preći iza lijeve zvijezde. Dakle, sadržaj dijela trake koji je označen sa $\sim$ ne utiče na rad mašine T i ne mijenja se u toku računanja.

Slika za funkciju normalno izračunljivu po Tjuringu, kada je $n = 1$. Ovdje riječ $x = x_1 x_2 \ldots x_i$ pripada skupu $\Omega(E)$. Na slici je prikazan slučaj kada ta riječ x pripada domenu funkcije (oblasti definisanosti funkcije). Ovdje riječ $y = y_1 y_2 \ldots y_j$ pripada skupu $\Omega(B)$ (y je riječ u azbuci B). Vidimo da je $f(x) = y$. Posebno je označeno radno polje:

početni zapis:

$\star$	x_1	x_2	$\cdots$	x_i	$\star$	sve $\star$

završni zapis:

$\star$	x_1	x_2	$\cdots$	x_i	$\star$	y_1	y_2	$\cdots$	y_j	$\star$	sve $\star$

Očito je da su normalno izračunljive funkcije po Tjuringu – izračunljive po Tjuringu, jer su dodata dva zahtjeva (a) i (b). Kasnije ćemo dokazati da važi i obrnut iskaz.

REZIME o izračunljivoj funkciji i rješivom skupu.

Kao primjer, funkcija $f(x,y) = x+2y+3$ $(x,y \in Z)$ je izračunljiva, treba prikazati brojeve x, y i $z = f(x,y)$ kao riječi u fiksiranoj azbuci. Programer bi kzao: funkcija f: $Z^n \to Z$ je izračunljiva ako postoji program na jeziku C čiji je ulazni podatak n–torka brojeva $x_1, \ldots, x_n$ i čiji je rezultat broj y, gdje je $y = f(x_1, \ldots, x_n)$, očito imamo u vidu totalnu funkciju od n promjenljivih ($\mathrm{Dom}\, f = Z^n$), gdje je $n \geq 1$.

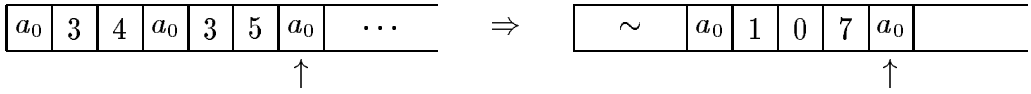
Kao što je rađeno, za skup $S \subset \Omega^n(E)$ kaže se da je rješiv (E je azbuka, $n \in N$) ako postoji mašina T sa svojstvom: ako je početna pozicija $a_0\, w_1\, a_0 \ldots a_0\, w_n\, \underset{\uparrow}{a_0} \ldots$ $(w_i \in \Omega(E))$ onda će završna pozicija biti $\sim a_0\, \underset{\uparrow}{a_0}$ kada $(w_1, \ldots, w_n) \in S$ odnosno $\sim a_0\, a_1\, \underset{\uparrow}{a_0}$ kada $(w_1, \ldots, w_n) \notin S$. Drugim riječima, završna pozicija će biti $\sim a_0\, y\, \underset{\uparrow}{a_0}$, gdje je $y = \square$ (tako da je $|y| = 0$) ili $y = a_1$ (tako da je $|y| = 1$). Na primjer, u slučaju $n = 1$, "skup S je rješiv" ekvivalentno je sa karakteristična funkcija skupa S je izračunljiva što znači da postoji mašina T sa svojstvom: $a_0\, w\, \underset{\uparrow}{a_0} \ldots \Rightarrow \sim a_0\, \underset{\uparrow}{a_0}$ kada $w \in S$ i $a_0\, w\, \underset{\uparrow}{a_0} \ldots \Rightarrow \sim a_0\, a_1\, \underset{\uparrow}{a_0}$ kada $w \notin S$, gdje svakako $w \in \Omega(E)$. Najprostije je da završna pozicija bude $a_0\, \underset{\uparrow}{a_0} \ldots$. ili $a_0\, a_1\, \underset{\uparrow}{a_0} \ldots$. Konvencija: sa $\square$ saopštava se da $\in$, a sa a_1 da $\notin$. Recimo, skup prostih brojeva je rješiv u odnosu na skup N. Slično, skup parnih brojeva je rješiv u odnosu na skup Z. Isto tako, skup $S = \{(a,b) |\ a^2+b^2 \equiv 0 \,(\mathrm{mod}\, 2015)\}$ (znači $S \subset Z \times Z$) je rješiv u odnosu na $Z \times Z$. Skup Diofantovih jednačina sa cijelim rješenjima nije rješiv u odnosu na skup Diofantovih jednačina. Programer bi kazao: skup $S \subset Z^n$ je rješiv ako postoji program na jeziku C čiji je ulazni podatak $(x_1, \ldots, x_n)$ i čiji je rezultat "Y" ako $(x_1, \ldots, x_n) \in S$ odnosno "N" ako $(x_1, \ldots, x_n) \notin S$, yes ili no.

Negacija od "skup je rješiv" glasi skup nije rješiv (skup je nerješiv). Volja je kazati rješiv skup ili odlučiv skup, engl. a decidable set.

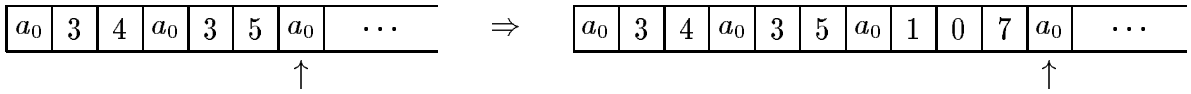
Umjesto rješiv skup ponekad kažemo algoritamski rješiv skup.

REZIME o izračunljivoj i normalno izračunljivoj funkciji.

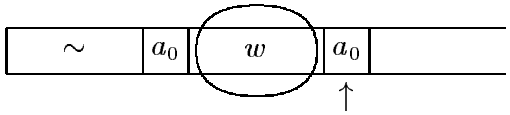
Poslužimo se primjerom $f(x,y) = x + 2y + 3$ $(x,y \in N)$ i neka se brojevi prikazuju kao riječi u azbuci $\{a_1, \ldots, a_{10}\} = \{0, \ldots, 9\}$. Ako imamo mašinu koja računa funkciju onda imamo recimo



dok u slučaju mašine koja normalno računa tu funkciju imamo recimo



Uvijek, rezultat rada mašine je jedna riječ w i w se čita iz završne pozicije. Riječ w sastavljena je od slova $a_1, \ldots, a_t$, odnosno $w \in \Omega(A)$. U završnoj poziciji, ona se na traci nalazi neposredno ispred radnog polja. Sa lijeve i desne strane, oivičena je sa dva polja koja sadrže a_0. Koliko polja na traci zauzima w, koliko ona ima slova, kolika je njena dužina, čemu je jednako $|w|$? Važi $|w| \geq 0$. U slučaju $|w| = 0$, dva polja gdje piše a_0 su susjedna na traci. Slika za vrijednost funkcije w: završna pozicija Tjuringove mašine ima oblik



Završna pozicija – to je pozicija nakon zaustavljanja mašine.

3. PRIMJERI TJURINGOVIH MAŠINA. TJURINGOVI DIJAGRAMI

3.1. ELEMENTARNE MAŠINE

U ovom naslovu definišu se tri tzv. elementarne Tjuringove mašine. Neka je $t \geq 1$. Sve tri elementarne mašine a_i, r i l koriste radnu azbuku $A = \{a_1, \ldots, a_t\}$. Rezultat njihove primjene na proizvoljnu poziciju jeste izvjesna "elementarna" promjena.

a) Razmotrimo T. m. a_i, gdje je $0 \leq i \leq t$. Kada se ta mašina primijeni na proizvoljnu poziciju, ona u radno polje upisuje slovo a_i i onda se zaustavlja na tom polju ne mijenjajući ostale zapise. Očito je da tako funkcioniše T. m. sa tablicom .... Dakle, u prvom koraku se u radno polje (nezavisno od njegovog sadržaja) upisuje slovo a_i i nakon tog koraka mašina se zaustavlja.

b) Razmotrimo Tjuringovu mašinu r. Kada se mašina r primijeni na proizvoljnu poziciju, ona pomjeri radno polje za jedno mjesto **udesno** i zatim se zaustavlja ne mijenjajući sadržaj trake. Tako funkcioniše T. m. sa tablicom ....

c) Sljedeća tablica definiše T. m. l. Kada se mašina l primijeni na proizvoljnu poziciju, ona pomijera radno polje za jedno mjesto **ulijevo** i zatim se zaustavi ne mijenjajući sadržaj trake. Na primjer, $] a_4\ a_4\ a_2\ a_0 \ldots \overset{l}{\Rightarrow}] a_4\ a_4\ a_2\ a_0 \ldots$. Maločas rečeno ne važi u jednom slučaju. Upravo, ako je u početnoj poziciji $m = 0$ (radno polje je nulto tj. početno) onda će u rezultatu primjene mašine l doći do zaustavljanja zbog izlaska preko kraja trake (PT). Zaustavljanje mašine nije izazvano izvršavanjem radnje s, tj. nije se desilo mašinsko zaustavljanje (MZ). U završnoj poziciji, radno polje ima isti indeks $m = 0$ i sadržaj trake nije promijenjen. Na primjer, $] a_4\ a_4\ a_2\ a_0 \ldots \overset{l}{\Rightarrow}] a_4\ a_4\ a_2\ a_0 \ldots$.

Tablica a_i:	Tablica r:	Tablica l:
$q_0\ a_0\ a_i\ q_1$	$q_0\ a_0\ r\ q_1$	$q_0\ a_0\ l\ q_1$
...	...	...
$q_0\ a_t\ a_i\ q_1$	$q_0\ a_t\ r\ q_1$	$q_0\ a_t\ l\ q_1$
$q_1\ a_0\ s\ q_1$	$q_1\ a_0\ s\ q_1$	$q_1\ a_0\ s\ q_1$
...	...	...
$q_1\ a_t\ s\ q_1$	$q_1\ a_t\ s\ q_1$	$q_1\ a_t\ s\ q_1$

3.2. DVA PRIMJERA TJURINGOVIH MAŠINA

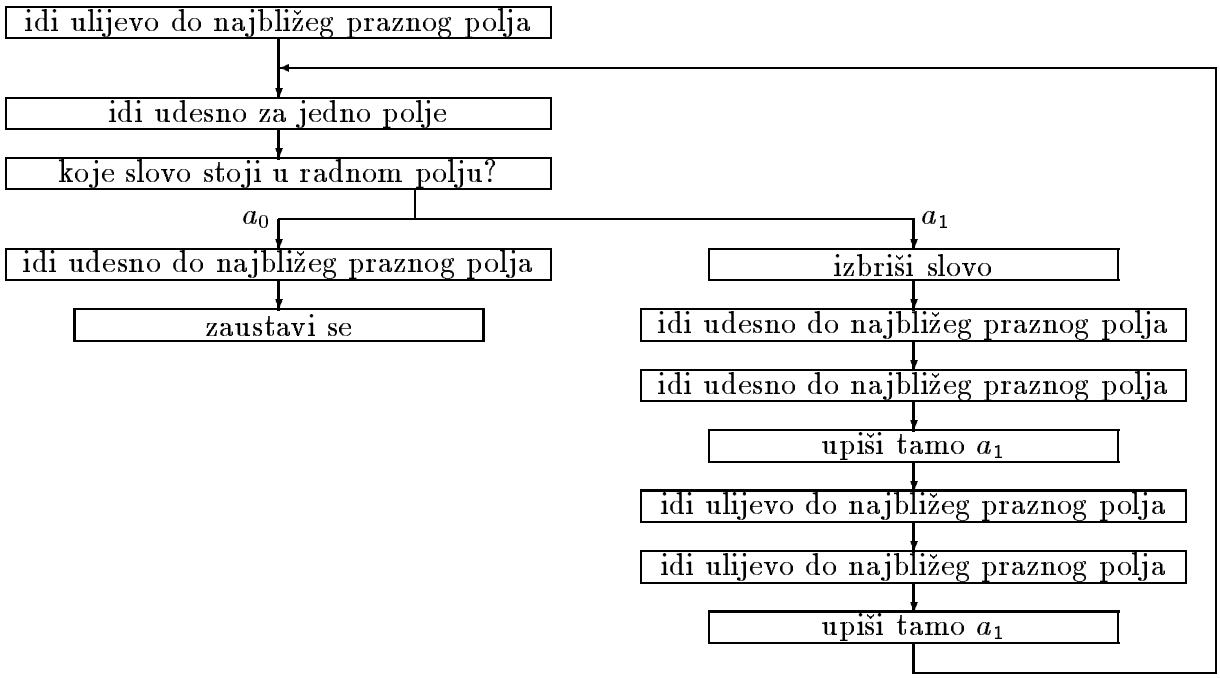
Dvije mašine T_1 i T_2 biće definisane svojim tablicama.

a) Mašina T_1 ima radnu azbuku $A = \{a_1, \ldots, a_t\}$, gdje je $t \geq 1$. V. tablicu. Mašina T_1, nezavisno od početne pozicije, pomijera radno polje za tri mjesta udesno, zatim upisuje a_0 u tekuće radno polje i nakon toga se zaustavlja ne unoseći druge promjene u zapis.

$q_0\ a_0\ r\ q_1$	$q_3\ a_0\ a_0\ q_4$
...	...
$q_0\ a_t\ r\ q_1$	$q_3\ a_t\ a_0\ q_4$
$q_1\ a_0\ r\ q_2$	$q_4\ a_0\ s\ q_4$
...	...
$q_1\ a_t\ r\ q_2$	$q_4\ a_t\ s\ q_4$
$q_2\ a_0\ r\ q_3$	
...	
$q_2\ a_t\ r\ q_3$	

b) Mašina za kopiranje T_2 ima radnu azbuku $A_1 = \{a_1\}$. Neka $w \in \Omega(A_1)$ tj. neka je w proizvoljna riječ u azbuci A_1. To znači da je $w = \square$ ili $w = a_1$ ili $w = a_1a_1$ ili $w = a_1a_1a_1$ ili .... Mašina T_2 prevodi početnu poziciju $\sim\star w\star\ldots$ u završnu poziciju $\sim\star w\star w\star\ldots$. Pored toga, ova mašina tokom svog rada ne smije da izađe na segment zapisa koji je označen sa $\sim$ i (specijalno) ona ne smije da izmijeni taj segment.

Od želje prema mašini, tj. kako sastaviti mašinu da vrši predviđenu obradu (kopiranje). Prvo ćemo nacrtati blok–šemu. Blok–šema prikazuje glavne korake, odnosno predstavlja grubo rješenje. V. blok–šemu. Vidi se da mašina kopira slovo po slovo. Sada, grubi koraci treba da budu rastavljeni na detalje. Drugim riječima, programer polazi od blok–šeme i pomoću nje sastavi rješenje (sastavi tablicu mašine). Mi samo dajemo gotovo rješenje – v. tablicu.



$q_0 \star l\ q_1$	$q_4 \star r\ q_5$	$q_8 \star \mid q_2$
$q_0 \mid l\ q_1$	$q_4 \mid l\ q_1$	$q_8 \mid l\ q_8$
$q_1 \star \star\ q_2$	$q_5 \star r\ q_6$	$q_9 \star s\ q_9$
$q_1 \mid l\ q_1$	$q_5 \mid r\ q_5$	$q_9 \mid r\ q_9$
$q_2 \star r\ q_3$	$q_6 \star \mid q_7$	
$q_2 \mid r\ q_3$	$q_6 \mid r\ q_6$	
$q_3 \star r\ q_9$	$q_7 \star l\ q_8$	
$q_3 \mid \star\ q_4$	$q_7 \mid l\ q_7$	

Testirati tablicu na nekoliko karakterističnih slučajeva ulaznog podatka $w = \underbrace{a_1a_1\ldots a_1}_{n \text{ puta}}$ ($n \geq 0$). Uzmimo da je $w = a_1$ tj. uzmimo da početna pozicija mašine glasi $\begin{array}{|c|c|c|c|}\hline a_0 & a_1 & a_0 & \cdots \\ \hline \end{array}$ $\uparrow$. Upravljajući se po blok–šemi ili po tablici, uvjeriti se da će tada završna pozicija mašine biti $\begin{array}{|c|c|c|c|c|c|}\hline a_0 & a_1 & a_0 & a_1 & a_0 & \cdots \\ \hline \end{array}$ $\uparrow$. Drugi karakteristični primjer. Zanimljivo je pogledati kako se mašina ponaša u krajnjem slučaju, kada je ulazni podatak $w = \square$. Ako je početna pozicija $\begin{array}{|c|c|c|}\hline a_0 & a_0 & \cdots \\ \hline \end{array}$ $\uparrow$

onda će završna pozicija biti $\boxed{a_0}\boxed{a_0}\boxed{a_0}\boxed{\cdots}$ ($\uparrow$ pod trećim a_0). Treći karakteristični primjer. Neka bude $w = a_1a_1$ tj. neka je početna pozicija $\boxed{a_0}\boxed{a_1}\boxed{a_1}\boxed{a_0}\boxed{\cdots}$ ($\uparrow$ pod drugim a_1). Tada će se mašina zaustaviti u poziciji

$\boxed{a_0}\boxed{a_1}\boxed{a_1}\boxed{a_0}\boxed{a_1}\boxed{a_1}\boxed{a_0}\boxed{\cdots}$ ($\uparrow$ pod zadnjim a_0).

Pokušati posljednji slučaj: polazeći iz $]\, a_0\, a_1\, a_1\, \underset{\uparrow}{a_0} \ldots$ stiže se do $]\, a_0\, a_1\, a_1\, a_0\, a_1\, a_1\, \underset{\uparrow}{a_0} \ldots$. Uvjeriti se da je zaista tako, prateći iz koraka u korak kako se smjenjuju pozicije (sadržaj trake i indeks radnog polja). U drugim oznakama: $]\star\,|\,|\underset{\uparrow}{\star}\ldots \Rightarrow\ ]\star\,|\,|\star\,|\,|\underset{\uparrow}{\star}\ldots$. Možda je za pisanje pogodnije $]\,0\,1\,1\,\underset{\uparrow}{0}\ldots \Rightarrow\ ]\,0\,1\,1\,0\,1\,1\,\underset{\uparrow}{0}\ldots$.

3.3. DEFINICIJA TJURINGOVOG DIJAGRAMA

Zašto je potreban Tjuringov dijagram? U prpmjeru 3.2.(a) mogli smo mašinu T_1 da prikažemo kao uniju tri mašine r i jedne mašine a_0. U primjeru 3.2.(b) bilo bi korisno da se male mašine uslovno povezuju (ako u radnom polju stoji a_0 onda ...). Prikaz mašine koji se dobija pomoću dijagrama kraći je od njenog prikaza pomoću tablice.

Definicija Tjuringovog dijagrama. Razmotrimo Tjuringove mašine $M_1, \ldots, M_k$ sa zajedničkom radnom azbukom $A = \{a_1, \ldots, a_t\}$. Za M_i se kaže da je mala mašina ili da je komponenta. Tjuringov dijagram D sastoji se od simbola $M_1, \ldots, M_k$ (neki od simbola mogu više puta da učestvuju u dijagramu) i od simbola $\cdot$ (tačka). Pored toga, simboli su međusobno spojeni strelicama, na strelici piše neko slovo. Pri tome se zahtijeva da:

(1) U dijagramu D simbol $\cdot$ (tačka) pojavljuje se samo jednom. On ukazuje gdje počinje rad mašine.

(2) Za svaki simbol i i za svako $j \in \{0, 1, \ldots, t\}$, iz tog simbola polazi najviše jedna strelica na kojoj piše a_j. To znači da treba da bude jednoznačno definisano koja mašina treba da radi (treba da preuzme upravljanje), nakon mašinskog zaustavljanja MZ prethodne mašine, s tim da je a_j stajalo u njenom posljednjem radnom polju.

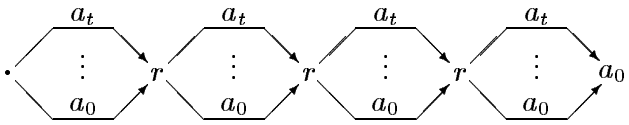
Napominje se da strelica može da vodi od simbola ka njemu samom.

Određivanje redosljeda rada mašine definisane dijagramom. Dijagram D definiše jednu Tjuringovu mašinu T. Dosad je izložena sintaksa (gramatika) i samo djelimično semantika (smisao), pa pogledajmo da upotpunimo.

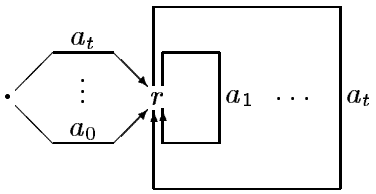
Neka se T primjenjuje na izvjesnu poziciju i neka radno polje sadrži slovo a. Ako iz tačke ne izlazi strelica sa slovom a onda dolazi do mašinskog zaustavljanja mašine T. Ako pak (suprotno) iz tačke izlazi strelica sa slovom a onda je to (prema uslovu (2)) samo jedna strelica. Ona vodi ka nekom simbolu M_i. Sada T funkcioniše kao M_i. Mašina M_i počinje da radi od početka svog programa (svoje tablice), od svog stanja q_0. Sada su moguća tri ishoda, kako slijedi: M_i radi vječno – tada i ukupna mašina T radi vječno ili tokom rada M_i dolazi do prelaska preko kraja trake – tada i T prelazi preko kraja trake (PT) i (znači) zaustavlja se ili dolazi do mašinskog zaustavljanja (MZ) male mašine M_i. Tada se upravljanje (uopšte uzev) predaje nekoj komponenti M_j, u zavisnosti od sadržaja (a_k) tekućeg radnog polja i od prisustva odgovarajuće strelice u dijagramu. Odgovarajuća strelica je $\xrightarrow{a_k}$. Ako ima $M_i \xrightarrow{a_k} M_j$ u dijagramu, M_j će početi da radi od početka svog programa. Itd.

Pogledajmo dva primjera Tjuringovih dijagrama. Radna azbuka je $A = \{a_1, \ldots, a_t\}$ u oba primjera.

Primjer a). V. sljedeći dijagram. Vidimo da se ustvari radi o mašini T_1 iz prethodnog naslova 3.2. pod (a).



Primjer b). V. sljedeći dijagram. Za razmatranu mašinu koristi se oznaka R. Mašina R vrši sljedeću obradu: $]\sim\underset{\uparrow}{-}w\star\sim \overset{R}{\Rightarrow}]\sim-w\underset{\uparrow}{\star}\sim$, gdje $w\in\Omega(A)$.



Dva primjera upućuju da se izvrši dogradnja definicije Tjuringovog dijagrama (dogradnja oznaka i pravila), kako slijedi.

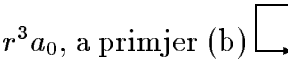
(3) Ako od jednog simbola ka drugom simbolu vodi strelica sa slovom a_j, za svako $j\in\{0,1,\ldots,t\}$ onda sve te strelice mogu da budu zamijenjene jednom strelicom **bez slova**.

(4) Ako su među strelicama koje vode od jednog simbola ka drugom simbolu prisutne sve moguće osim jedne (recimo osim a_k) onda sve te strelice mogu da budu zamijenjene jednom na kojoj piše $\neq a_k$. Slično ako nedostaju dvije strelice. Itd.

(5) Ako od tačke vode sve moguće strelice ($j=0,1,\ldots,t$), i to sve one vode ka jednom te istom simbolu (recimo ka simbolu M_1) onda tačka može da se izostavi, s tim što se taj simbol (M_1) označi kao **početni simbol** – time što se zaokruži ili se napiše lijevo od svih ostalih.

(6) Na kraju, može da se izostavi strelica na kojoj ništa ne piše.

Pisaćemo M^n umjesto $\underbrace{M\ldots M}_{n\text{ puta}}$.

Poslije dogradnje oznaka i pravila, primjer (a) dobija oblik r^3a_0, a primjer (b) 

.

Još jedna dogradnja, posljednja. Pogledajmo radne azbuke komponenti i ukupne mašine. Neka je za dijagram D izabrana radna azbuka $A_t=\{a_1,a_2,\ldots,a_t\}$. Neka je M_i simbol mašine čija je radna azbuka manja tj. čija je radna azbuka $A_s=\{a_1,\ldots,a_s\}$, gdje je $s<t$, tako da je $A_s\subset A_t$. Tada se (u dijagramu) pod mašinom M_i podrazumijeva mašina koja nastaje kada se tablica mašine M_i dopuni sa redovima $qasq$ za svako $a\in A_t\setminus A_s$ i svako stanje q mašine M_i. Prosto rečeno, M_i će se zaustaviti (radnja s) ako naiđe na slovo a_{s+1} ili ... ili a_t koje je za nju nepoznato.

3.4. KONSTRUKCIJA TABLICE PO ZADATOM DIJAGRAMU

Tjuringova mašina data je dijagramom D u kome učestvuju simboli malih mašina $M_1, \ldots, M_k$. Poznata je tablica svake male mašine. U svakoj maloj tablici, stanja su numerisana kao $q_0, q_1, \ldots$. Razmotrimo sljedeći zadatak. Konstruisati tablicu te Tjuringove mašine.

Moguće je da se neki simbol M_i pojavljuje u dijagramu dva puta ili više puta. Da bi se proces konstrukcije uprostio, smatramo da se svaki simbol pojavljuje samo jednom. Dakle, u pripremnom koraku, svakom simbolu u dijagramu pridruži se njegova tablica. Kao da se broj k "poveća".

Prepišimo sve male tablice jednu za drugom, u ma kom redosljedu, uvodeći pritom novu–opštu numeraciju stanja, ta opšta numeracija počinje od stanja q_1. Tako je nastala jedna velika tablica. Dakle, stari brojevi stanja su promijenjeni, da u velikoj tablici ne bi bilo preklapanja brojeva. Stanje q_0 male tablice koje odgovara nekom simbolu M zamijenjeno je u velikoj tablici stanjem q_M. Drugim riječima, po prethodnoj numeraciji, to je bilo stanje q_0 (za M), a po novoj numeraciji to je sada postalo stanje q_M.

U velikoj tablici treba izvršiti dodavanja i promjene, da bi se ona pretvorila u ukupnu tablicu, da bi se ona pretvorila u rješenje postavljenog zadatka.

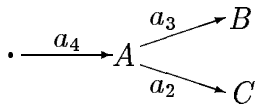
Ilustracija. Dati su dijagram i tablice komponenti. Prikazan je prvi oblik velike tablice. U velikoj tablici vršiće se dopune i promjene. Vidimo da je u prikazanom primjeru $q_A = q_1$, $q_B = q_5$, $q_C = q_{11}$:

A: $q_0 a_0 r q_1 \quad \ldots \quad q_3 a_t \ldots$

B: $q_0 a_0 \ldots \quad \ldots \quad q_5 a_t \ldots$

C: $q_0 a_0 \ldots \quad \ldots \quad q_5 a_t l q_4$

prvi oblik: $q_1 a_0 r q_2 \quad \ldots \quad q_4 a_t \ldots \quad q_5 a_0 \ldots \quad \ldots \quad q_{10} a_t \ldots \quad q_{11} a_0 \ldots \quad \ldots \quad q_{16} a_t l q_{15}$



Velikoj tablici dodaju se novi redovi tri tipa, kako slijedi. Vidjećemo da novi redovi zavise od strelica koje polaze od simbola tačka u datom dijagramu (ili takvih strelica nema u datom dijagramu). Za svako $a \in \{a_0, a_1, \ldots, a_t\}$ kome odgovara strelica koja vodi od tačke ka tački samoj, dodaje se red $q_0 a a q_0$. Za svako a za koje u dijagramu postoji strelica koja vodi od tačke ka simbolu M neke male mašine, dodaje se red $q_0 a a q_M$. Zašto? U polaznom dijagramu $\bullet \xrightarrow{a} M$ govori: na početku rada, predaj upravljanje komponenti M ako u radnom polju piše a. Zato u tablici treba red $q_0 a a q_M$ čiji je smisao: na početku rada, ako u radnom polju piše a, izvrši radnju a (upiši a u radno polje) i pređi u stanje q_M.

Za svako slovo a takvo da u dijagramu ne postoji strelica (na kojoj piše a) koja bi polazila od tačke, dodaje se red $q_0 a s q_0$.

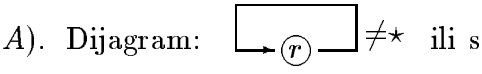
Još ostaje da se u tablici izvrše sljedeće promjene. Ako su dva simbola M i N u dijagramu spojeni strelicom sa slovom a ($M \xrightarrow{a} N$) onda u tablici treba svaki red oblika $qasq'$ (ovaj red potiče od male tablice simbola M) prepraviti da sada glasi $qaaq_N$. Smisao: neka je došlo do mašinskog zaustavljanja MZ komponente M. Tada se ukupna mašina ne zaustavlja, uopšte uzev. Već se upravljanje predaje komponenti N, pod uslovom da slovo a stoji u radnom polju.

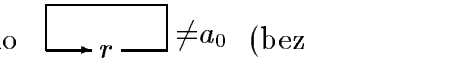
Sada je tablica konstruisana. Postavljeni zadatak je riješen. Vidi se da je procedura (pretvaranja dijagrama u tablicu) jednoznačna do na permutaciju oznaka $q_1, q_2, \ldots$. Vidi se da dijagram i tablica definišu jednu te istu Tjuringovu mašinu.

3.5. DALJI PRIMJERI TJURINGOVIH MAŠINA

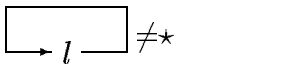
U ovom naslovu navodi se nekoliko primjera Tjuringovih mašina. Mašina se definiše svojim dijagramom. Funkcionisanje mašine prikazano je na običnom slučaju u kome se mašina upotrebljava, kaže se njena radnja ili obrada koju ona vrši, $\Rightarrow$. Dati su još i oznaka ili skraćenica za mašinu (recimo R), kao i njen naziv (udesno za jednu riječ).

Neka je $t \geq 1$ i $A = \{a_1, \ldots, a_t\}$. Ulazna i radna azbuka svake mašine jeste A, osim gdje je posebno rečeno drukčije.

(1) Simbol mašine ili oznaka za mašinu: R. Radnja (njeno funkcionisanje): $]\sim \underset{\uparrow}{\star} w \star \sim \Rightarrow \;]\sim \star w \underset{\uparrow}{\star} \sim$, gdje $w \in \Omega(A)$. Dijagram: 

ili svejedno 

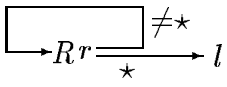
(bez zaokruživanja r). Ova mašina je već rađena ranije u naslovu 3.3.

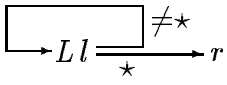
(2) Simbol L idi ulijevo za jednu riječ (idi ulijevo do prvog znaka $\star$). Radnja $]\sim \star w \underset{\uparrow}{\star} \sim \Rightarrow \;]\sim \underset{\uparrow}{\star} w \star \sim$, gdje $w \in \Omega(A)$. Dijagram 

Kod sljedeće dvije mašine, sa X je označen niz **nepraznih** riječi u azbuci A. Dakle, svaka riječ mora da ima bar jedno slovo. Riječi su razdvojene jedna od druge znacima $\star$. Niz riječi sastoji se od $n \geq 0$ riječi. Dakle, dopušta se da bude $n = 0$ kada se X svodi na prazan niz riječi. Dakle, $X = w_1 \star w_2 \star \ldots \star w_n$, gdje je $|w_k| \geq 1$ za svako $k \in \{1, 2, \ldots, n\}$. Dužina riječi w (broj slova riječi w) označava se sa $|w|$. Jedno slovo – jedno polje na traci. Recimo, $|\square| = 0$, $|a_4| = 1$, $|a_2 a_2 a_4 a_1| = 4$ i slično.

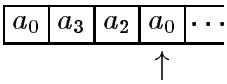
Na primjer, $X = w_1 \star w_2$ gdje je $|w_1| \geq 1$ i $|w_2| \geq 1$, $n = 2$. Ili $X = w_1$ gdje je $|w_1| \geq 1$ ili svejedno $w_1 \neq \square$, $n = 1$. Ili $X = \square$, $n = 0$.

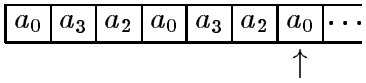
Već smo rekli da $w_k \in \Omega(A)$ za svako k.

(3) $\mathcal{R}$ za niz riječi udesno
$]\sim \underset{\uparrow}{\star} X \star \star \sim \;\Rightarrow\;]\sim \star X \underset{\uparrow}{\star} \star \sim$ 

(4) $\mathcal{L}$ za niz riječi ulijevo
$]\sim \star \star X \underset{\uparrow}{\star} \sim \;\Rightarrow\;]\sim \star \underset{\uparrow}{\star} X \star \sim$ 

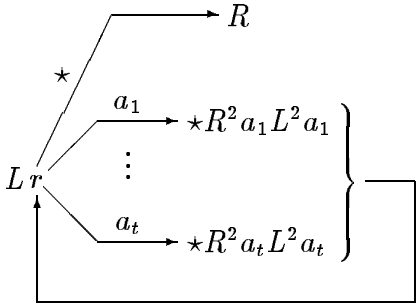
(5) K kopira jednu riječ
$]\sim \star w \underset{\uparrow}{\star} \ldots \;\Rightarrow\;]\sim \star w \star w \underset{\uparrow}{\star} \ldots$, gdje $w \in \Omega(A)$

Pokušati: ako se mašina K primijeni na početnu poziciju 

onda će se ona zaustaviti u završnoj poziciji 

.

Specijalan slučaj $t = 1$ je već rađen ranije u naslovu 3.2.



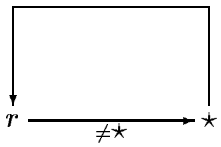
3.6. DALJI PRIMJERI TJURINGOVIH MAŠINA (NASTAVAK)

(6) W_r briše desnu riječ

$]\sim\underset{\uparrow}{\star}w\star\sim \Rightarrow]\sim\star\ldots\underset{\uparrow}{\star}\sim$

primjer $]\underset{\uparrow}{\star}\,a_4\,a_4\,a_1\star\ldots \Rightarrow]\star\star\star\star\underset{\uparrow}{\star}\ldots$

(mašina "$\star$" – upisuje $\star$)

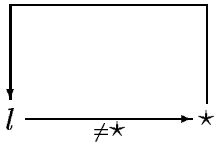


(7) W_l briše lijevu riječ

briše riječ koja je lijevo od početnog radnog polja

$]\sim\star w\underset{\uparrow}{\star}\sim \Rightarrow]\sim\underset{\uparrow}{\star}\ldots\star\sim$

primjer $]\star\,a_4\,a_4\,a_1\,a_3\underset{\uparrow}{\star}\ldots \Rightarrow]\underset{\uparrow}{\star}\star\star\star\star\star\ldots$



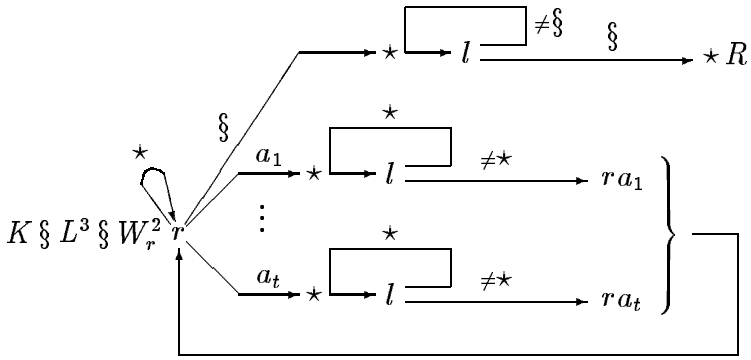
(8) Ulazna azbuka mašine V je $A_t = \{a_1, \ldots, a_t\}$. Međutim, njena radna azbuka je $A_{t+1} = \{a_1, \ldots, a_t, a_{t+1}\}$. Kaže se da mašina koristi jedno pomoćno ili dodatno slovo. Umjesto a_{t+1} obično se piše § (paragraf) u dijagramima. Mašina V prevodi početnu poziciju

★	w_1	★	w_2	★	…
				↑	

u završnu poziciju

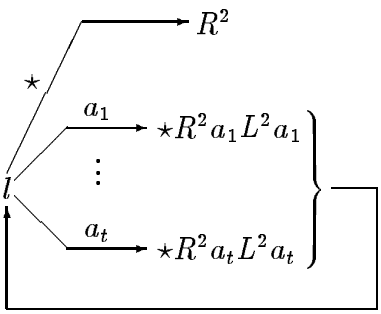
★	w_2	★	…
		↑	

gdje $w_1, w_2 \in \Omega(A_t)$. Dakle, prva riječ w_1 nestaje a druga riječ w_2 ostaje. Simbol mašine: V, naziv: isijeca (mašina koja isijeca dio trake), radnja: $]\sim\star w_1\star w_2\underset{\uparrow}{\star}\ldots \Rightarrow]\sim\star w_2\underset{\uparrow}{\star}\ldots$. Na primjer $]\,a_0\,a_2\,a_5\,a_0\,a_3\,\underset{\uparrow}{a_0}\ldots \Rightarrow]\,a_0\,a_3\,\underset{\uparrow}{a_0}\ldots$. Analiza rada mašine: vidi se da se pomoću § označi početak i kraj područja koje nas interesuje. Dijagram:



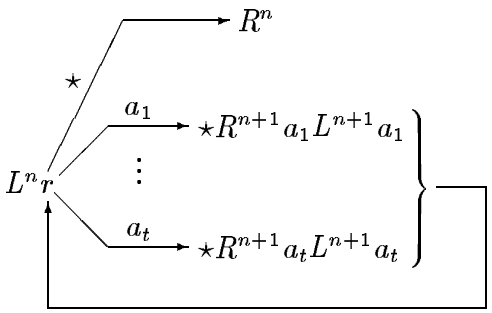
(9) I stvara inverznu riječ $]\sim\star w\underset{\uparrow}{\star}\ldots \Rightarrow]\sim\star w\star w^{-1}\underset{\uparrow}{\star}\ldots$, gdje $w \in \Omega(A)$. Na primjer $]\,a_0\,a_2\,a_6\,\underset{\uparrow}{a_0}\ldots \Rightarrow]\,a_0\,a_2\,a_6\,a_0\,a_6\,a_2\,\underset{\uparrow}{a_0}\ldots$.

Koristi se oznaka w^{-1} za riječ koja je inverzna riječi w. Riječ w^{-1} nastaje od w inverzijom redosljeda slova. Recimo $(a_3a_4a_1)^{-1} = a_1a_4a_3$. Posebno, stavlja se da je $\square^{-1} = \square$.



(10) Neka bude $n \geq 1$. Mašina K_n – kopira prvu riječ od n riječi.
Radnja $]\sim\star w_1\star\ldots\star w_n\underset{\uparrow}{\star}\ldots \Rightarrow]\sim\star w_1\star\ldots\star w_n\star w_1\underset{\uparrow}{\star}\ldots$,
gdje $w_k \in \Omega(A)$ za $k \in \{1,\ldots,n\}$. Na primjer, mašina K_3 vrši sljedeću obradu:
$]\star w_1\star w_2\star w_3\underset{\uparrow}{\star}\ldots \Rightarrow]\star w_1\star w_2\star w_3\star w_1\underset{\uparrow}{\star}\ldots$,
gdje $w_1, w_2, w_3 \in \Omega(A)$. Vidi se da je $K_1 = K$, ranije pod (5) je bilo rađeno K.

Dijagram mašine K_n:



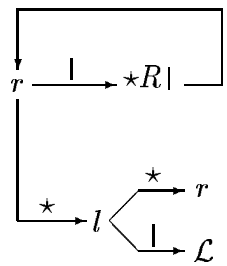
3.7. DALJI PRIMJERI TJURINGOVIH MAŠINA (NASTAVAK I KRAJ)

Definišimo još dvije Tjuringove mašine. Njihova ulazna i radna azbuka je $A_1 = \{a_1\}$. Tako da je sada $X = \square$ ili $X = w_1$ ili $X = w_1\star w_2$ ili $X = w_1\star w_2\star w_3$ ili ..., gdje $w_i \in \Omega(A_1)$ i $|w_i| \geq 1$.

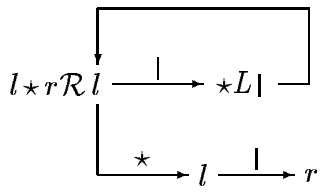
Simbol **Radnja** **Dijagram**

(11) T_r translacija udesno za jedno mjesto
$]\sim\underset{\uparrow}{\star}X\star\ldots \Rightarrow]\sim\star\underset{\uparrow}{\star}X\star\ldots$

Pogledati na primjeru $X = |||\star||$, odnosno uvjeriti se da zaista $]\underset{\uparrow}{\star}|||\star||\star\ldots \Rightarrow]\star\underset{\uparrow}{\star}||$ $|\star||\star\ldots$. Pogledati i na primjeru X – prazan niz ($X = \square$). Dijagram mašine:



(12) T_l translacija ulijevo za jedno mjesto
$]\sim||\underset{\uparrow}{\star}X\star\ldots \Rightarrow]\sim|\underset{\uparrow}{\star}X\star\ldots$



3.8. MAŠINA ZA MNOŽENJE DVA BROJA

Razmotrimo funkciju od dvije promjenljive f: $N \times N \to N$ ili f: $N_0 \times N_0 \to N_0$ definisanu relacijom $f(n_1, n_2) = n_1n_2$ za $n_1 \geq 0$, $n_2 \geq 0$, proizvod. Radi jednostavnosti, uzmimo da se

broj $n \in N_0$ prikazuje u tzv. unarnom obliku, tj. da se prikazuje kao $\underbrace{a_1 \ldots a_1}_{n \text{ puta}}$. Drugim riječima, razmotrimo funkciju $f\colon \Omega(A_1) \times \Omega(A_1) \to \Omega(A_1)$ definisanu relacijom $f(w_1, w_2) = \underbrace{a_1 \ldots a_1}_{n_1 n_2 \text{ puta}}$ kada je $w_1 = \underbrace{a_1 \ldots a_1}_{n_1 \text{ puta}}$ i $w_2 = \underbrace{a_1 \ldots a_1}_{n_2 \text{ puta}}$, gdje je $n_1 \geq 0$, $n_2 \geq 0$. Recimo, ako je $w_1 = a_1 a_1$ i $w_2 = a_1 a_1 a_1$ onda izlazi $f(w_1, w_2) = a_1 a_1 a_1 a_1 a_1 a_1$ u smislu $2 \cdot 3 = 6$. U ovom naslovu biće pokazano da je funkcija f izračunljiva. Drugim riječima, biće konstruisana mašina za množenje P koja računa funkciju f. Dakle, P vrši sljedeću obradu:

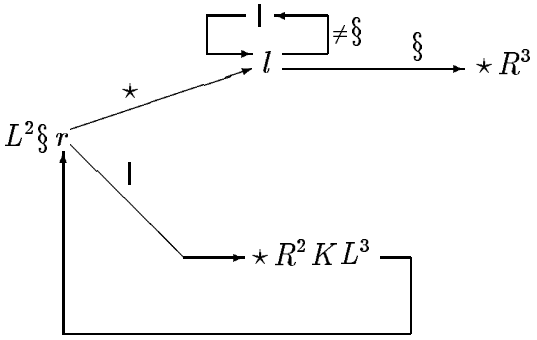
$$] \sim \star w_1 \star w_2 \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad] \sim \star w_1 \star w_2 \star w \underset{\uparrow}{\star} \ldots$$

gdje je $w = f(w_1, w_2)$. Vidi se da P ustvari **normalno** računa f. Sljedeći dijagram definiše mašinu P, odnosno predstavlja rješenje postavljenog zadatka. Treba se u to uvjeriti, odnosno treba izvršiti analizu predloženog dijagrama. Ulazna azbuka mašine P je $A_1 = \{a_1\}$. Njena radna azbuka je $A_2 = \{a_1, a_2\}$. Drugim riječima, ona koristi jedno dopunsko (pomoćno) slovo $a_{t+1} = a_2$ za koje se (kao i obično) koristi oznaka § u dijagramu.

U okviru analize, lako se vidi da važi relacija:

$$] \sim \star w_2 \underset{\uparrow}{\star} w_3 \star \ldots \quad \overset{K}{\Rightarrow} \quad] \sim \star w_2 \star w_3 w_2 \underset{\uparrow}{\star} \ldots$$

($w_3 w_2$ – nadovezivanje). Recimo, ako je $w_3 = a_1 a_1$ i $w_2 = a_1 a_1 a_1$ onda je $w_3 w_2 = a_1 a_1 a_1 a_1 a_1$ u smislu $2 + 3 = 5$. Tokom rada mašine, uzastopno se kopira riječ w_2. Ukupno, ona će biti iskopirana n_1 puta.



3.9. KONVERZIONA FUNKCIJA $\gamma_{t,s}$

Neka je $t \geq 1$ i $s \geq 1$. U ovom naslovu biće konstruisana funkcija koja preslikava skup $\Omega(A_t)$ u skup $\Omega(A_s)$. Budući da su jedan i drugi skup prebrojivi (imaju jedan te isti kardinalni broj) to sigurno postoji funkcija koja posjeduje sljedeća tri dobra teorijsko–skupovna svojstva. 1) Funkcija je svuda definisana, tj. funkcija je definisana na čitavom skupu $\Omega(A_t)$. 2) Funkcija je "**na**" skup $\Omega(A_s)$, tj. njen skup vrijednosti je čitav skup $\Omega(A_s)$. 3) Funkcija je uzajamno jednoznačna, isto se kaže da je funkcija "1–1". Međutim, mi želimo da funkcija posjeduje još dva dobra svojstva koja se tiču izračunljivosti, kako slijedi. 4) Funkcija je izračunljiva po Tjuringu. 5) Njena inverzna funkcija je izračunljiva po Tjuringu. U nastavku će biti definisana jedna funkcija $\gamma_{t,s}$ i biće dokazano da ona posjeduje svih pet dobrih svojstava 1)–5).

Na početku našeg rada, izvršimo jednu redukciju. Pretpostavimo da smo riješili postavljeni zadatak kada je $s = 1$. To znači da smo za svako $t \geq 1$ konstruisali funkciju $\gamma_{t,1}$ koja posjeduje svih pet dobrih svojstava 1)–5). Tada imamo i rješenje opšteg slučaja proizvoljnog s. Zaista, za $s > 1$, mi ćemo definisati

$$\gamma_{t,s}(w) = \gamma_{s,1}^{-1}(\gamma_{t,1}(w)) \text{ za } w \in \Omega(A_t). \tag{1}$$

Slijedi:

$$\gamma_{t,s}^{-1}(w) = \gamma_{t,1}^{-1}(\gamma_{s,1}(w)) \text{ za } w \in \Omega(A_s), \tag{2}$$

na osnovu poznate relacije za inverznu funkciju kompozicije $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$. Po pretpostavci: $\gamma_{t,1}$ (gdje je $t \geq 1$) je svuda definisana, "na" i "1–1". Odatle slijedi da je i funkcija $\gamma_{t,s}$: $\Omega(A_t) \to \Omega(A_s)$ definisana sa (1) svuda definisana, "na" i "1–1", što se lako vidi. Po pretpostavci: $\gamma_{t,1}$ i $\gamma_{t,1}^{-1}$ su i. T. Odatle slijedi da su i funkcije $\gamma_{t,s}$ i $\gamma_{t,s}^{-1}$, relacije (1) i (2), i. T. Da se ovo posljednje dokaže, treba da se iskoristi sljedeće tvrđenje: kompozicija (superpozicija) dvije funkcije koje su izračunljive po Tjuringu – jeste funkcija koja je izračunljiva po Tjuringu. To tvrđenje biće dokazano kasnije, u naslovu 4. Redukcija na slučaj $s = 1$ je izvršena.

Umjesto $\gamma_{t,1}$ odsad ćemo pisati samo γ_t. Preostaje da se riješi redukovani zadatak koji glasi kako slijedi. Za svako $t \geq 1$ konstruisati funkciju γ_t: $\Omega(A_t) \to \Omega(A_1)$ da je svuda definisana, "na" i "1–1" i da su ona i njena inverzna funkcija γ_t^{-1}: $\Omega(A_1) \to \Omega(A_t)$ izračunljive po Tjuringu.

Ako je $t = 1$ onda identičko preslikavanje rješava zadatak. Neka je odsad $t > 1$. Sljedeća formula definiše $\gamma_t(w)$ za svako $w \in \Omega(A_t)$:

$$\begin{cases} \gamma_t(\square) = 0, \\ \gamma_t(wa_i) = t \cdot \gamma_t(w) + i \text{ za } 1 \leq i \leq t. \end{cases} \tag{3}$$

Recimo $\gamma_t(a_4a_4a_1a_3) = t \cdot \gamma_t(a_4a_4a_1) + 3$.

Budući da je $\gamma_t(w) \in \Omega(A_1)$ to ima smisla pisati $\cdot$ i $+$ jer članove skupa $\Omega(A_1) = \{\square, a_1, a_1a_1, \ldots\}$ poistovjećujemo sa članovima skupa $N_0 = \{0, 1, 2, \ldots\}$.

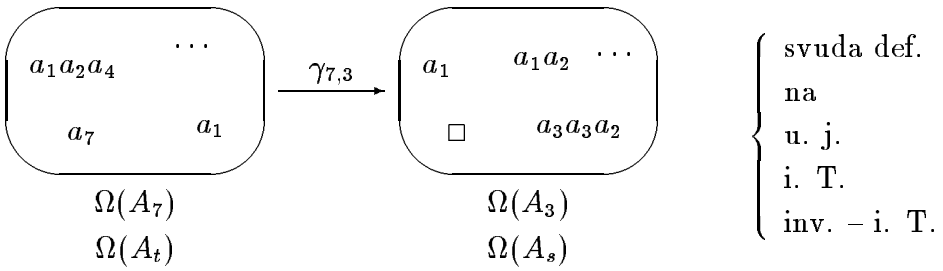
$N_0 = \{0, 1, 2, \ldots\}$ prošireni skup prirodnih brojeva, $N = \{1, 2, \ldots\}$ skup prirodnih brojeva

Analizom relacije (3) uvjeriti se da je funkcija γ_t zaista svuda definisana, "na" i "1–1". Sada bi po redu došla dva dijagrama, jedan za funkciju γ_t i drugi za funkciju γ_t^{-1}, naravno da treba da budu saglasni sa relacijom (3). Ta dva dijagrama pokazuju da su funkcije γ_t i γ_t^{-1} izračunljive po Tjuringu. Oni se izostavljaju jer su veoma složeni. Time je dokaz završen.

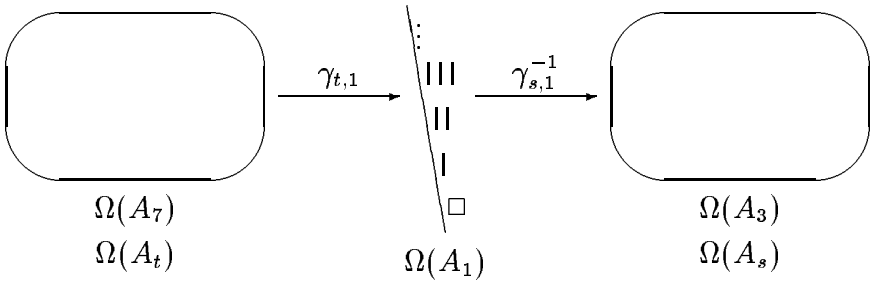
U slučaju $t = 3$, tj. u slučaju γ_3: $\Omega(A_3) \to \Omega(A_1)$ u tabeli su prikazani neki slučajevi $w \in \Omega(A_3)$ i odgovarajućeg $\gamma_3(w) \in \Omega(A_1)$ saglasno (3):

$w =$	$\square$	a_1	a_2	a_3	a_1a_1	a_1a_2	...
$\gamma_3(w) =$	$\square = 0$	$\mid = 1$	$\mid\mid = 2$	$\mid\mid\mid = 3$	$\mid\mid\mid\mid = 4$	$\mid\mid\mid\mid\mid = 5$	...

Slika za $\gamma_{t,s}$:



Slika za redukciju na $s = 1$:



3.10. KONVERZIONA FUNKCIJA σ_n^t

Neka je $n \geq 1$ i $t \geq 1$. Razmotrimo dva skupa: skup $\Omega^n(A_t) = \underbrace{\Omega(A_t) \times \ldots \times \Omega(A_t)}_{n \text{ puta}}$ čiji su članovi uređene n–torke riječi i skup $\Omega(A_t)$ čiji su članovi riječi. Jedan i drugi skup je prebrojiv. Zato postoji funkcija σ_n^t: $\Omega^n(A_t) \to \Omega(A_t)$ koja posjeduje dobra teorijsko–skupovna svojstva: 1) ona je svuda definisana, 2) ona je "1–1" i 3) ona je "na". Naš zadatak u ovom naslovu jeste da definišemo funkciju σ_n^t koja ima još dva dobra svojstva: 4) ona je izračunljiva po Tjuringu i 5) njena inverzna funkcija je izračunljiva po Tjuringu. Umjesto inverzne funkcije $(\sigma_n^t)^{-1}$ posmatraćemo njene komponente u oznaci $\sigma_{n,i}^t$, gdje je $1 \leq i \leq n$. Inverzna funkcija ima n komponenti. Dakle, $\sigma_{n,i}^t$: $\Omega(A_t) \to \Omega(A_t)$ i važe relacije

$$\sigma_n^t(\sigma_{n,1}^t(w), \ldots, \sigma_{n,n}^t(w)) = w \ \text{ i } \ \forall i \ \ \sigma_{n,i}^t(\sigma_n^t(w_1, \ldots, w_n)) = w_i.$$

Kao $f(x,y) = 100x + y$ za $x, y \in \{0, 1, \ldots, 9\}$; $f(3,8) = 308$, $f_1(308) = 3$, $f_2(308) = 8$. Mi zamjenjujemo uslov 5) sljedećim ekvivalentnim uslovom: svaka funkcija $\sigma_{n,i}^t$, gdje je $1 \leq i \leq n$, je izračunljiva po Tjuringu.

U cilju rješavanja postavljenog zadatka, izvršimo prvo jednu redukciju.

Dovoljno je da se riješi slučaj $t = 1$. Zaista, ako je $t > 1$ onda mi stavljamo

$$\sigma_n^t(w_1, \ldots, w_n) = \gamma_t^{-1}(\sigma_n^1(\gamma_t(w_1), \ldots, \gamma_t(w_n))) \tag{1}$$

i tako dobijamo

$$\sigma_{n,i}^t(w) = \gamma_t^{-1}(\sigma_{n,i}^1(\gamma_t(w))),$$

čine se postiže željeno.

Formula (1): svaka riječ prevede se u minimalnu azbuku A_1, zatim se izvrši pakovanje n–torke riječi u jednu riječ i onda se na kraju (pomoću funkcije γ_t^{-1}) vratimo u azbuku A_t.

Umjesto σ_n^1 i $\sigma_{n,i}^1$ odsad ćemo pisati kratko σ_n i $\sigma_{n,i}$.

Izvršimo još jednu redukciju. Dovoljno je da se razmotri slučaj kada je $n = 2$. Zaista, ako je $n > 2$ onda mi stavljamo

$$\sigma_n(w_1, \ldots, w_n) = \sigma_2(\sigma_{n-1}(w_1, \ldots, w_{n-1}), w_n) \tag{2}$$

i tako dobijamo

$$\sigma_{n,i}(w) = \sigma_{n-1,i}(\sigma_{2,1}(w)) \text{ za } 1 \leq i \leq n-1,$$
$$\sigma_{n,n}(w) = \sigma_{2,2}(w),$$

čime se, kao i maloprije, postiže cilj, uz izvođenje indukcije po n.

Formula (2): objekat $w_1, \ldots, w_{n-1}$ i riječ w_n zapakovati u jednu riječ onako kako se dvije riječi pakuju u jednu riječ, s tim da se prethodno objekat pripremi (zapakuje, indukcija).

Prema tome, postavljeni zadatak sveli smo na sljedeće. Treba konstruisati Tjuringovu mašinu za svuda definisanu i uzajamno jednoznačnu funkciju σ_2 iz $\Omega^2(A_1)$ na $\Omega(A_1)$, pri čemu su i funkcije $\sigma_{2,i}$: $\Omega(A_1) \to \Omega(A_1)$ za $i = 1, 2$, gore opisane, izračunljive po Tjuringu.

Na primjer, govoreći opisno, $(||, ||||) \xrightarrow{\sigma_2} ||||||||||$, $||||||||||\xrightarrow{\sigma_{2,1}} ||$, $||||||||||\xrightarrow{\sigma_{2,2}} ||||$.

Kako definisati σ_2: $\Omega(A_1) \times \Omega(A_1) \to \Omega(A_1)$, "1–1" i "na". Prvo ćemo definisati jednu pomoćnu funkciju σ: $\Omega(A_1) \times \Omega(A_1) \to \Omega(A_2)$, "1–1" i "na". Razmotrimo riječ $w \in \Omega(A_2)$ u kojoj se bar jednom pojavljuje slovo a_2. Uočimo prvo pojavljivanje slova a_2 u riječi w. Tako da se riječ w rastavlja na tri dijela, $w = w_1 a_2 w_2$, gdje $w_1 \in \Omega(A_1)$ i $w_2 \in \Omega(A_2)$. Ovim smo definisali funkciju koja paru riječi (w_1, w_2) pridružuje jednu riječ $w_1 a_2 w_2$. Oblast definisanosti te funkcije je skup $\Omega(A_1) \times \Omega(A_2)$, a njen skup vrijednosti je skup $\Omega(A_2)$ bez skupa riječi koje su sastavljene samo od slova a_1. Treba izvršiti dvije popravke te funkcije, da se dobije σ. Stavljamo da je

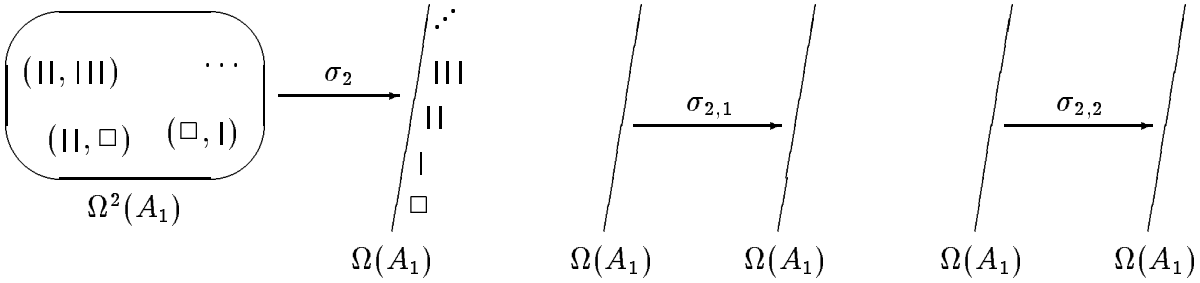
$$\sigma(w_1, w_2) = \begin{cases} w_2, \text{ ako je } w_1 = \square \\ (w_1 - 1) a_2 \gamma_2^{-1}(w_2), \text{ ako je } w_1 \neq \square \end{cases}$$

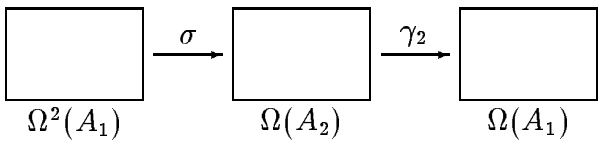
Preostaje da se stavi

$$\sigma_2(w_1, w_2) = \gamma_2(\sigma(w_1, w_2)),$$

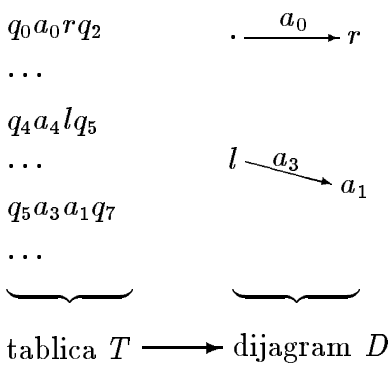
čime je funkcija σ_2 definisana.

Izostavljamo dijagrame za σ_2, $\sigma_{2,1}$ i $\sigma_{2,2}$. Postavljeni zadatak je riješen.





3.11. PREDSTAVLJANJE TJURINGOVE MAŠINE POMOĆU DIJAGRAMA KOJI JE SASTAVLJEN OD ELEMENTARNIH MAŠINA



Znamo da su elementarne mašine r, l i a_i $(0 \leq i \leq t)$, v. naslov 3.1. Neka je t fiksiraan prirodan broj. Razmotrimo T. m. M u azbuci $A = \{a_1, \ldots, a_t\}$. Mašina je definisana svojom tablicom. U ovom naslovu, naš zadatak je da toj tablici pridružimo odgovarajući dijagram, s tim da u dijagramu učestvuju jedino simboli elementarnih mašina. Označimo sa M' mašinu definisanu tim dijagramom, njena azbuka je takođe A. Mašina M' modelira mašinu M. Drugim riječima, mašina M' svojim radom oponaša rad mašine M. Prosto rečeno, M' radi isto ili skoro isto što i M.

Dajemo jednu definiciju.

Kaže se da mašina M' modelira mašinu M (kaže se da mašina M' modelira **u jakom smislu** mašinu M) ako su ispunjena sljedeća dva uslova:

1 Data početna pozicija izaziva mašinsko zaustavljanje (odnosno prelazak iza kraja trake) mašine M poslije konačnog broja koraka ako i samo ako ta ista početna pozicija izaziva mašinsko zaustavljanje (odnosno prelazak iza kraja trake) mašine M' takođe poslije konačnog broja koraka. U oba slučaja (MZ i PT), završne pozicije mašina M i M' moraju da se poklapaju.

2 Niz tekućih pozicija mašine M (za datu početnu poziciju) jeste podniz niza tekućih pozicija mašine M' za tu istu početnu poziciju.

Objašnjenje za 1: M MZ $\Leftrightarrow$ M' MZ i završne pozicije jedne i druge mašine se poklapaju, M PT $\Leftrightarrow$ M' PT i završne pozicije jedne i druge mašine se poklapaju i M vječno radi $\Leftrightarrow$ M' vječno radi, ista je bila početna pozicija jedne i druge mašine. Za 2: označimo niz pozicija mašine M' sa $S'_0, S'_1, S'_2, S'_3, S'_4, \ldots$. Tada niz pozicija mašine M može da bude recimo $S'_0, S'_2, S'_3, S'_6, \ldots$. Kao da mašini M' treba više vremena (više taktova) da dostigne završnu poziciju, da izvrši istu obradu.

Važi sljedeća teorema.

Teorema. Neka je Tjuringova mašina M nad azbukom A definisana tablicom T. Toj tablici može se na efektivan način pridružiti dijagram D sastavljen od simbola r, l, a_0, ..., a_t i tačke, takav da Tjuringova mašina M' definisana tim dijagramom modelira mašinu čija je tablica T.

Objašnjenje: na efektivan način znači algoritamskim sredstvima, nešto može da bude efektivno urađeno znači računar može to da uradi.

Dokaz. Dijagram može da se konstruiše na sljedeći način. Svakom redu $qavq'$ tablice u kome se v ne poklapa sa s, u dijagramu se pridružuje simbol v. U vezi toga, taj simbol v naziva se **odgovarajućim** za red $qavq'$. Osim toga, u dijagram se uključuje i simbol $\cdot$ (tačka).

Sada ćemo izvršiti sljedeća spajanja.

a) Tačku spajamo strelicom sa simbolom koji je odgovarajući za red $q_0a_j\ldots$ (na strelici piše a_j), $\forall j = 0, \ldots, t$.

b) Svaki simbol koji je odgovarajući za red $\ldots q$ spajamo strelicom sa simbolom koji je odgovarajući za red $qa_j\ldots$ (na strelici piše a_j), $\forall q$ i $\forall j$.

Red "$\ldots q$" znači da je četvrti član u redu jednak q. Red "$qa_j\ldots$" znači da je prvi član u redu jednak q, a drugi član a_j. Znamo da svaki red ima četiri člana.

Ova spajanja vrše se samo ako su odgovarajući simboli o kojima se govori – prisutni u dijagramu, tj. ako je odgovarajuća radnja $v \neq s$, kao što je rečeno na početku dokaza.

Ostaje da se provjeri da su ispunjeni uslovi 1 i 2. Nastao je traženi dijagram. **Teorema je dokazana.** Postavljeni zadatak je riješen.

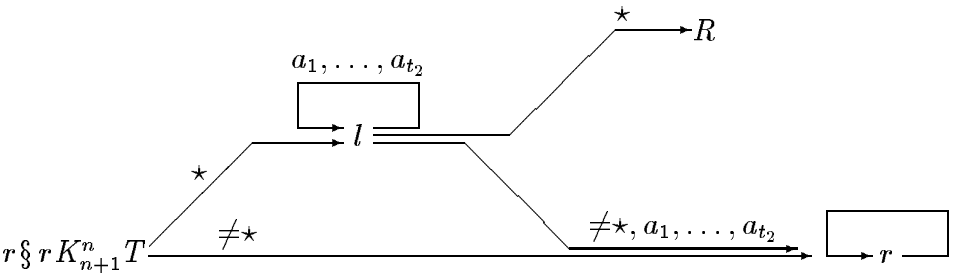
4. NORMALNA IZRAČUNLJIVOST PO TJURINGU

U ovom naslovu 4. biće dokazano da je svaka funkcija izračunljiva po Tjuringu i normalno izračunljiva po Tjuringu, pri čemu odgovarajuća Tjuringova mašina ne koristi dopunska slova. U tom cilju ćemo na osnovu Tjuringove mašine koja računa f konstruisati Tjuringovu mašinu koja normalno računa f i koja ne koristi dopunska slova. Ta konstrukcija biće izvršena u nekoliko etapa.

4.1. MAŠINA $T^{\S}$

Neka je $n \geq 1$, $t_1 \geq 1$ i $t_2 \geq 1$. Razmotrimo djelimičnu funkciju f od n promjenljivih, $f\colon \mathrm{Dom}f \to \Omega(A_{t_2})$, $\mathrm{Dom}f \subset \Omega^n(A_{t_1}) = \Omega(A_{t_1}) \times \ldots \times \Omega(A_{t_1})$ (n puta). Uzmimo da je f izračunljiva (po Tjuringu). To znači da postoji bar jedna T. m. koja računa f. Neka je T jedna T. m. koja računa f. Za $E = A_{t_1}$ kaže se da je ulazna azbuka mašine T, a za $B = A_{t_2}$ kaže se da je izlazna azbuka mašine T. Azbuka (radna azbuka) mašine T je $A_t = \{a_1, \ldots, a_t\}$, gdje je $t_1 \leq t$ i $t_2 \leq t$. Dakle, ako se T primijeni na početnu poziciju $]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$, gdje $w_k \in \Omega(A_{t_1})$ ($1 \leq k \leq n$), onda će se desiti sljedeće. Ako $(w_1, \ldots, w_n) \in \mathrm{Dom}f$ onda će se T zaustaviti i to MZ (mašinsko zaustavljanje) i završna pozicija će imati oblik $]\sim \star w \underset{\uparrow}{\star}$ (T će se zaustaviti poslije riječi w), pri čemu $w \in \Omega(A_{t_2})$ i $f(w_1, \ldots, w_n) = w$. A ako $(w_1, \ldots, w_n) \notin \mathrm{Dom}f$ onda će se T zaustaviti (MZ) ali ne poslije neke riječi ili će se T zaustaviti zbog PT (prelazak preko kraja trake) ili će T raditi vječno.

U ovom naslovu, mašini T pridružuje se druga jedna T. m. u oznaci $T^{\S}$. Azbuka mašine $T^{\S}$ jeste $A_{t+1} = \{a_1, \ldots, a_t, a_{t+1}\} = \{a_1, \ldots, a_t, \S\}$. Mašina $T^{\S}$ definisana je dijagramom, v. dijagram.



Glavnu ulogu u dijagramu ima komponenta T. Vidimo da se mašina T čija je azbuka A_t uključuje u dijagram čija azbuka ima jedno slovo više (mašina $T^{\S}$ koristi više jedno slovo $a_{t+1} = \S$). Kako je ranije definisano, T će se zaustaviti ako naiđe na slovo a_{t+1} koje ona ne prepoznaje. Drugim riječima, smatramo da je tablica mašine T dopunjena redovima $qa_{t+1}sq$ za svako stanje q mašine T. Recimo unaprijed da $T^{\S}$ takođe računa f.

Neka se mašina $T^{\S}$ primjenjuje na poziciju $]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$, gdje $w_k \in \Omega(A_{t_1})$ ($1 \leq k \leq n$). Tvrdimo da će se tada desiti sljedeće:

a) ako $(w_1, \ldots, w_n) \notin \mathrm{Dom}f$ onda $T^{\S}$ vječno radi i

b) ako $(w_1, \ldots, w_n) \in \mathrm{Dom}f$ onda će se $T^{\S}$ zaustviti i to poslije riječi $f(w_1, \ldots, w_n) = w$, tj. završna pozicija će biti $]\sim \star w \underset{\uparrow}{\star}$.

U nastavku, u preostalom dijelu ovog naslova, analizom ćemo se uvjeriti da je iskazano tvrđenje tačno. Tokom analize upravljamo se po dijagramu.

Početna pozicija $\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$ poslije djelovanja komponente $r\S r K^n_{n+1}$ dovodi do tekuće pozicije $\star w_1 \star \ldots \star w_n \star \S \star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$. Sada djeluje komponenta T. Kao i u opštem

slučaju Tjuringove mašine, rad komponente T imaće jedan od sljedeća tri ishoda: T vječno radi ili T PT ili T MZ.

Što se tiče slučaja vječno radi, znači da bi i T sama za sebe primijenjena na $\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star}$ ... vječno radila (n–torka ne pripada domenu). Dakle, $T^{\S}$ u tom slučaju vječno radi (njena komponenta T vječno radi, odnosno ne predaje kontrolu drugoj nekoj komponenti). Ovo je saglasno sa uslovom (a).

Zapaziti sljedeće: komponenta T tokom svog rada nikad neće preći lijevo od polja na traci u kome je upisano slovo § (jer ona to slovo ne prepoznaje). Što se tiče slučaja PT, taj slučaj nikad ne nastupa, u vezi prethodne rečenice. Tako da nema analize koja bi se odnosila na taj slučaj.

Još jedino preostaje da se analizira slučaj MZ. Dakle, neka se rad komponente T okonča za konačno mnogo koraka zbog njenog MZ. Pogledajmo sadržaj radnog polja u tom trenutku. Tekuća pozicija je $] \sim \underset{\uparrow}{x} \sim$. Na osnovu ranije zapaženog, polje koje sadrži x je desno od polja koje sadrži § ili se ta dva polja poklapaju. Za x postoje četiri mogućnosti i)–iv) kako slijedi, pa se svaka mogućnost posebno analizira.

i) $x = \S$. S jedne strane, to znači da bi mašina T sama za sebe primijenjena na običnu poziciju $\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$ prešla preko kraja trake ($\Rightarrow$ n–torka ne pripada domenu). S druge strane, po dijagramu, kontrola se predaje komponenti "r koje se vraća na sebe" ($\Rightarrow$ $T^{\S}$ će raditi vječno). Vidimo da je uslov (a) zadovoljen ako (i).

ii) $x = a_1$ ili ... ili $x = a_t$. Mašina T bi se sama za sebe zaustavila, ali ne poslije neke riječi. U dijagramu se upravljanje predaje komponenti "r koje se vraća na sebe". Imamo slične okolnosti kao maločas pod (i), vidimo da je i (ii) u redu.

Još jedino može da bude $x = a_0$ ($x = \star$).

iii) $x = \star$ s tim da između § i x nema nijednog polja koje bi sadržalo $\star$. Mašina T bi sama za sebe iz obične početne pozicije dostigla završnu poziciju oblika $] w' \underset{\uparrow}{\star} \sim$ za neko $w' \in \Omega(A_t)$ (ne može se reći da se mašina zaustavila poslije neke riječi). U dijagramu se kontrola predaje, posredstvom linije na kojoj piše "$\neq \star, a_1, \ldots, a_{t_2}$", komponenti "$r$ koje se vraća na sebe". Tako da je (a) u redu.

Najzad, iv) $x = \star$ i između polja § i polja x postoji bar jedno polje koje sadrži znak $\star$. Dakle, kada je T uradila svoj dio posla onda pozicija na traci ima oblik $\star w_1 \star \ldots \star w_n \star \S \sim \star w \underset{\uparrow}{\star}$ ili svejedno $\sim \star w \underset{\uparrow}{\star}$ za neko $w \in \Omega(A_t)$. Mašina T bi se sama za sebe zaustavila poslije te iste riječi $w \in \Omega(A_{t_2})$. U dijagramu, uzastopnim radom komponente l, dolazi se do tekuće pozicije $\sim \underset{\uparrow}{\star} w \star$. Još će se kontrola predati komponenti R, čime će se dobiti završna pozicija $\sim \star w \underset{\uparrow}{\star}$. Uslov (b) je u redu. Mogućnost (iv) je u redu. Analiza je završena.

4.2. POSTAVKA ZADATKA O MODELIRANJU NAD AZBUKOM A_1

Tjuringovoj mašini M čija je radna azbuka $A = \{a_1, \ldots, a_t\}$ (gdje je $t \geq 1$) biće pridružena druga jedna mašina $\overline{M}$ koja radi u minimalnoj azbuci $A_1 = \{a_1\}$. Druga mašina $\overline{M}$ svojim radom oponaša rad polazne mašine M. Budući da dvije mašine ne koriste isti skup slova, to odgovarajuće ("iste") pozicije jedne i druge mašine očito ne mogu da se poklapaju. Na početku ćemo razjasniti tu okolnost. Uvedimo jedan novi termin. Kao i dosad, riječ ili svojstvena riječ sastavljena je od slova $a_1, \ldots, a_t$. A novi termin: kaže se **nesvojstvena** riječ ako je ona sastavljena od slova $a_0, a_1, \ldots, a_t$.

Razmotrimo mašinu M čija je radna azbuka A. Proizvoljnom zapisu na traci mašine M može da se pridruži nesvojstvena riječ u azbuci A, tako što se sva slova zapisa napišu redom slijeva udesno, sve do bilo kog polja takvog da desno od tog polja nema svojstvenih slova zapisa. Ovako se jednom zapisu $\mathcal{J}$ pridružuju razne nesvojstvene riječi w, koje se međusobno razlikuju samo po broju znakova $\star$ na desnom kraju. Svakoj nesvojstvenoj riječi w odgovara jedinstven zapis $\mathcal{J}$, tako da w može da se iskoristi za prikazivanje zapisa $\mathcal{J}$. Na primjer, nesvojstvene riječi $w = a_0a_3a_6a_0a_0$ i $w = a_0a_3a_6$ prikazuju jedan te isti zapis $\mathcal{J} =]\star a_3a_6\star\ldots$. Jedno polje na traci je radno polje. Smatramo da nesvojstvena riječ prikazuje neku poziciju jedino ako je prikazan i sadržaj radnog polja (jedino ako je i radno polje obuhvaćeno), čak i ako je to polje prazno, zajedno sa svim poljima desno od njega.

U sljedećem koraku, nesvojstvena riječ w u azbuci A kodira se pomoću nesvojstvene riječi w_1 u azbuci A_1. U tom cilju, umjesto svakog slova riječi w pišemo niz slova |, a dužina tog niza je za jedinicu veća od indeksa odgovarajućeg slova. Tako dobijeni nizovi razdvoje se jedan od drugog znacima $\star$. Još se napiše jedna zvijezda lijevo od nastale nesvojstvene riječi. Zvijezda koja prethodi jednom takvom nizu crta – smatra se da pripada slovu koje taj niz crta kodira. Prosto rečeno, slovo a_k se prikazuje kao $\star||\ldots|$ ($k+1$ crta), gdje je $0 \leq k \leq t$. Recimo, a_4 se prikazuje kao $\star|||||$.

Na primjer, ranije navedene riječi $w = a_0a_3a_6a_0a_0$ i $w = a_0a_3a_6$ kodiraju se redom kao $w_1 = \star|\star||||\star|||||||\star|\star|$ odnosno $w_1 = \star|\star||||\star|||||||$.

Znamo da je $S = (m, \mathcal{J})$, tj. pozicija se sastoji od indeksa radnog polja i zapisa. Neka je data pozicija S sa zapisom iz slova azbuke A. Prvo se tom zapisu pridruži jedna nesvojstvena riječ w u azbuci A, a onda se w kodira nesvojstvenom riječi w_1 u azbuci A_1, kako je jedno i drugo maločas izloženo. Riječ w_1 može da bude napisana na Tjuringovu traku. Još ostaje da se definiše pravilo po kome se označava radno polje. Pođimo od radnog polja pozicije S. U tom polju upisano je izvjesno slovo. To slovo prisutno je i u w. U w_1, tom slovu odgovara $\star$ zajedno sa nekoliko crta poslije nje. Mi uzimamo kao radno polje (proglašavamo za radno polje) na Tjuringovoj traci – ono polje u kome je upisana ta $\star$.

Dakle, poziciji sa zapisom u azbuci A pridružena je pozicija sa zapisom u azbuci A_1. Na primjer, poziciji $S =]\star a_3a_6\underset{\uparrow}{\star}\star\ldots$ odgovara pozicija $\overline{S} =]\star|\star||||\star|||||||\star|\underset{\uparrow}{\star}|\star|\star\ldots$.

Mi ćemo sa $\overline{S}$ označavati svaku poziciju koja je pridružena poziciji S. Po svakoj $\overline{S}$ može se na očit način rekonstruisati S. Pozicija $\overline{S}$ je određena jednoznačno, s tačnošću do količine parova $\star|$ koji slijede iza posljednjeg svojstvenog slova njenog zapisa.

U nastavku ćemo od mašine M konstruisati mašinu $\overline{M}$ koja ima sljedeća dva svojstva:

1 Iz početne pozicije S mašina M se poslije konačnog broja koraka mašinski zaustavlja (odnosno prelazi iza kraja trake) ako i samo ako se mašina $\overline{M}$ iz početne pozicije $\overline{S}$ ($\overline{S}$ je jedna kodirajuća riječ za S) poslije konačnog broja koraka takođe mašinski zaustavlja (odnosno prelazi iza kraja trake). U jednom i u drugom slučaju MZ i PT, završna pozicija mašine $\overline{M}$ jeste neka pozicija $\overline{S_E}$, gdje je S_E – završna pozicija mašine M.

2 Neka iz početne pozicije S_0 mašina M prolazi konačan ili beskonačan niz pozicija S_0, S_1, S_2, ... Tada niz pozicija kroz koje prolazi mašina $\overline{M}$ iz početne pozicije $\overline{S_0}$ sadrži kao svoj podniz niz $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$, ...

Kažemo da mašina $\overline{M}$ modelira **u slabom smislu** mašinu M.

Ponovimo: $\overline{S_E}$ je kod za S_E, $\overline{S_0}$ je kod za S_0, $\overline{S_1}$ je kod za S_1 i slično.

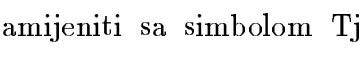
Dakle, neka je mašina M data. Pred nama je zadatak o konstrukciji mašine $\overline{M}$. Postavljeni zadatak biće riješen u nastavku, u nekoliko etapa.

4.3. MODELIRANJE NAD AZBUKOM A_1

Zadatak o modeliranju u minimalnoj azbuci biće riješen u etapama od (1) do (5).

(1) Prvo konstruišemo dijagram mašine M' koja u jakom smislu modelira polaznu mašinu M. Ranije je rađeno kako se sastavlja taj dijagram. Znamo da u dijagramu učestvuju samo simboli elementarnih mašina r, l i a_i $(0 \leq i \leq t)$ i simbol tačka.

U nastavku se u dijagramu vrše prepravljanja. Biće zamijenjeni svi simboli osim simbola tačka, u etapama od (2) do (4). A posljednja etapa (5) odnosi se na strelice. Kada sva prepravljanja budu sprovedena onda će i postavljeni zadatak o modeliranju u slabom smislu biti riješen.

(2) Svaki simbol l u dijagramu zamijeniti sa simbolom Tjuringove mašine 

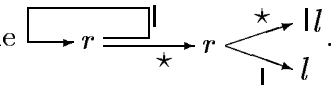
. Ova mašina "l koje se vraća na sebe u slučaju |" u konačno mnogo koraka prevodi poziciju $\sim\star\underbrace{|\ldots|}_{i+1}\overset{\uparrow}{\star}\underbrace{|\ldots|}_{j+1}\star\sim$ tj. $\overline{\sim a_i\, a_j \sim}$ u završnu poziciju $\sim\overset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\underbrace{|\ldots|}_{j+1}\star\sim$ tj. $\overline{\sim a_i\, a_j \sim}$, sa mašinskim zaustavljanjem.

Recimo, u slučaju originalne mašine M, komponenta l vrši sljedeću obradu: $]\, a_0 a_3 \underset{\uparrow}{a_6}\, a_0 \ldots \overset{l}{\Rightarrow}]\, a_0 \underset{\uparrow}{a_3}\, a_6 a_0 \ldots$, čemu odgovara sljedeće u slučaju mašine $\overline{M}$ koja modelira: iz pozicije

$$]\star|\star||||\underset{\uparrow}{\star}|||||||\star\ldots$$

komponenta "l koje se vraća na sebe u slučaju |" će dostići poziciju $]\star|\overset{\uparrow}{\star}||||\star|||||||\star\ldots$. Znamo da je $\overline{a_0 a_3 \underset{\uparrow}{a_6}\, a_0 \ldots} =]\star|\star||||\underset{\uparrow}{\star}|||||||\star\ldots$ i slično.

Pored toga, ova mašina "l koje se vraća na sebe u slučaju |" iz početne pozicije $]\overset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\sim$ tj. $]\overset{\uparrow}{\overline{a_i \sim}}$ prelazi iza kraja trake i pritom ne mijenja poziciju.

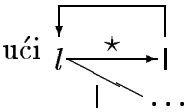
(3) Umjesto (svakog) simbola r pisaćemo simbol Tjuringove mašine 

. Ova mašina iz pozicije $\sim\overset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\underbrace{|\ldots|}_{j+1}\star\sim$ tj. $\overline{\sim a_i\, a_j \sim}$ dovodi do mašinskog zaustavljanja u poziciji $\sim\star\underbrace{|\ldots|}_{i+1}\overset{\uparrow}{\star}\underbrace{|\ldots|}_{j+1}\star\sim$ tj. $\overline{\sim a_i\, a_j \sim}$. Dok iz pozicije $\sim\overset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\ldots$ tj. $\overline{\sim a_i \star\ldots}$ ona dovodi do mašinskog zaustavljanja u poziciji $\sim\star\underbrace{|\ldots|}_{i+1}\overset{\uparrow}{\star}|\star\ldots$ tj. $\overline{\sim a_i \underset{\uparrow}{\star}\ldots}$.

4.4. MODELIRANJE NAD AZBUKOM A_1 (NASTAVAK)

(4) Umjesto simbola a_i pišemo simbol Tjuringove mašine .... Želimo da se uvjerimo da se ova mašina kada se primijeni na početnu poziciju $\sim\underset{\uparrow}{\star}\underbrace{|\ldots|}_{j+1}\star\sim$ tj. $\overline{\underset{\uparrow}{\sim a_j\sim}}$ zaustavlja sa završnom pozicijom $\sim\underset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\sim$ tj. $\overline{\underset{\uparrow}{\sim a_i\sim}}$. Zapaziti da je $0\le i\le t$ i isto tako $0\le j\le t$.

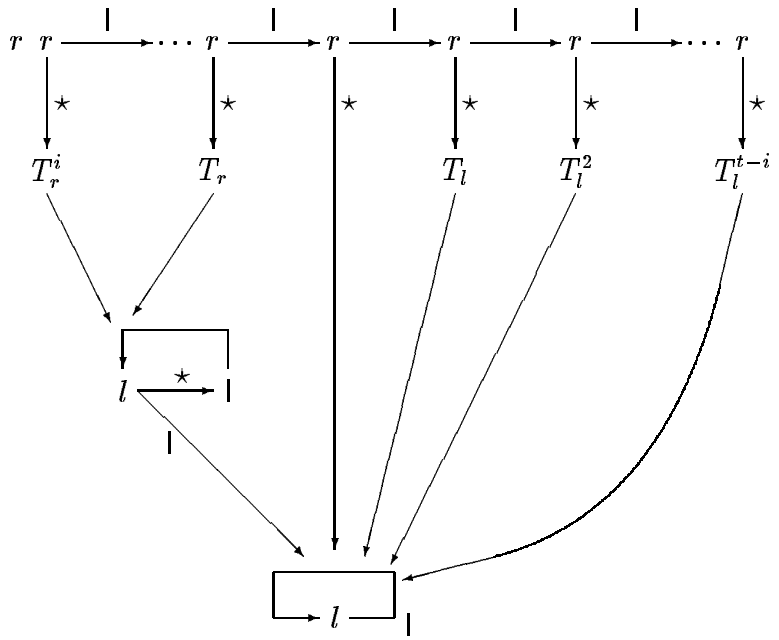
Analizirajmo rad mašine prikazane na slici. Početna pozicija je $\sim\underset{\uparrow}{\star}\underbrace{|\ldots|}_{j+1}\star\sim$. Najprije se $j+2$ puta izvrši pomijeranje udesno, čime se dolazi u polje sa znakom $\star$. Zatim funkcioniše fragment ove mašine prema kome vodi vertikalna strelica koja izlazi iz posljednjeg izvršenog r. Ovdje razdvajamo analizu na tri slučaja a), b) i c).

a) $j<i$: to je fragment T_r^{i-j}. Rezultat rada tog fragmenta jeste da se sav dio zapisa koji stoji desno od tekućeg radnog polja pomjeri udesno za $i-j$ polja. Dakle, dobili smo poziciju $\sim\star\underbrace{|\ldots|}_{j+1}\underbrace{\star\ldots\star}_{i-j}\underset{\uparrow}{\star}\sim$. Primjenjujući  mi $i-j$ zvijezda zamijenimo crtama. Na kraju se nađemo lijevo od nastalog niza od $i+1$ crta jer je djelovao i posljednji dio "l koje se vraća na sebe u slučaju |".

b) $j=i$: taj fragment je (posljednji dio) "l koje se vraća na sebe u slučaju |".

c) $j>i$: taj fragment je T_l^{j-i}. Dejstvovanje tog fragmenta pretvara poziciju $\sim\star\underbrace{|\ldots|}_{j+1}\underset{\uparrow}{\star}\sim$ u poziciju $\sim\star\underbrace{|\ldots|}_{i+1}\underset{\uparrow}{\star}\sim$. Poslije toga djeluje još samo posljednji dio "l koje se vraća na sebe u slučaju |".

Dakle, u sva tri slučaja (a),(b) i (c), završna pozicija jeste $\sim\underset{\uparrow}{\star}\underbrace{|\ldots|}_{i+1}\star\sim$.



4.5. MODELIRANJE NAD AZBUKOM A_1 (NASTAVAK I KRAJ)

Kao što je rađeno, dijagram mašine M' sastoji se od simbola $E=\cdot$ (tačka), $E=r$, $E=l$ i $E=a_i$ i od strelica. U etapama (2), (3) i (4) zamijenjeni su simboli r, l i a_i. Svaki simbol

novog–tekućeg oblika dijagrama, tj. svaki simbol koji je odgovarajući za neki simbol E iz starog dijagrama označimo sa $\overline{E}$. Ako je $E = \cdot$ onda je $\overline{E} = \cdot$.

Sada će u posljednjoj etapi da budu izvršene nove prepravke u dijagramu. Te prepravke tiču se strelica. Kada se te prepravke izvrše onda će i nastati rezultujući dijagram (dijagram mašine $\overline{M}$).

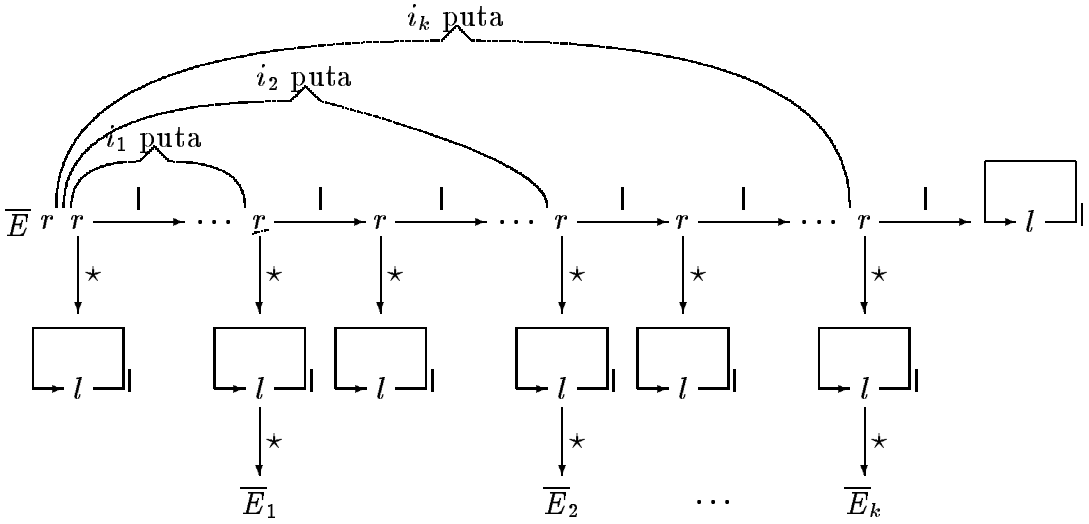
(5) U tekućem obliku dijagrama ukinuti sve strelice. A biće unesene nove strelice i dopunski simboli, kako slijedi.

Posmatrajmo jedan bilo koji simbol E u starom dijagramu. Sagledajmo i sve strelice koje polaze iz E. Neka u starom dijagramu iz E polazi k strelica i neka one vode ka simbolima E_1, ..., E_k, s tim da na njima pišu slova $a_{i_1}, \ldots, a_{i_k}$ redom; od E vodi strelica prema E_j, na toj strelici piše a_{i_j}, za $j = 1, \ldots, k$. Znamo da $i_1, \ldots, i_k \in \{0, 1, \ldots, t\}$. Uzmimo da je numerisano tako da važi $i_1 < i_2 < \ldots < i_k$.

Sada ćemo povezati $\overline{E}$ sa $\overline{E_1}, \ldots, \overline{E_k}$ na sljedeći način: ....

Analizirajmo funkcionisanje mašine prikazane na slici. Ova komponenta će početi da djeluje kada pozicija na traci ima oblik $]\sim\underset{\uparrow}{\star}|\ldots|\star\sim$. Pomoću $\underset{\uparrow}{\star}|\ldots|$ kodirano je neko slovo. Koliko crta ima u $|\ldots|$? Uzmimo da taj niz crta ima tačno $i_1 + 1$ član. Tada će se poslije r (na slici je ovo r malo podvučeno ukoso, radi lakšeg snalaženja) preći na vertikalnu strelicu, tj. tada će se pomoću "l koje se vraća na sebe u slučaju $|$" vratiti do polaznog znaka $\underset{\uparrow}{\star}$. Na kraju, prelazi se na $\overline{E_1}$. Dakle, komponenta $\overline{E}$ predaje upravljanje komponenti $\overline{E_1}$. Jasno je da se slično opisuje funkcionisanje ako niz crta sadrži tačno $i_2 + 1$ ili ... ili $i_k + 1$ član. U slučaju da je broj članova u ovom nizu crta različit od svih ovih k mogućnosti, onda se mašina $\overline{M}$ prosto vrati do polaznog znaka $\star$ i zaustavi se, što je ispravno jer u odgovarajućem slučaju i mašina M treba da se zaustavi.

Time je završena konstrukcija mašine $\overline{M}$. Dovoljno je da radna azbuka mašine $\overline{M}$ sadrži jedno slovo.



4.6. NORMALNO RAČUNANJE PO TJURINGU

U ovom naslovu biće dokazano sljedeće tvrđenje: ako je funkcija f izračunljiva (mašina T) onda je funkcija f i normalno izračunljiva i to bez dopunskih slova (mašina T_2). Šablon oznaka: T računa f, T_1 normalno računa f, T_2 normalno računa f i pritom koristi samo slova koja su stvarno neophodna. Poznata je oznaka: ako je recimo $w = a_3a_4a_1$ onda je $\overline{w} = a_0a_1a_1a_1a_1a_0a_1a_1a_1a_1a_1a_0a_1a_1$ tj. $\overline{w} = \star||||\star|||||\star||$ i slično.

Neka je $t_1 \geq 1$, $t_2 \geq 1$, $n \geq 1$. Razmotrimo funkciju f: Dom$f \to \Omega(A_{t_2})$, gdje je Dom$f \subset \Omega^n(A_{t_1})$ (vidimo da je djelimična). Neka je funkcija f izračunljiva. To znači da postoji T. m. T koja računa f. Neka je azbuka mašine T označena kao $A = \{a_1, \ldots, a_t\}$, gdje je $t_1 \leq t$ i $t_2 \leq t$. Biće dokazano tvrđenje koje je ranije najavljeno: f je i normalno izračunljiva. Treba konstruisati mašinu T_1 koja normalno računa f. Mašina T_1 treba da ispunjava dva uslova više od T: a) ako $(w_1, \ldots, w_n) \in \Omega^n(A_{t_1}) \setminus$ Dom$f$ onda T_1 primijenjena na poziciju $]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$ vječno radi i b) ako $(w_1, \ldots, w_n) \in \Omega^n(A_{t_1}) \cap$ Dom$f$ onda T_1 prevodi tu istu početnu poziciju u završnu poziciju $]\star w_1 \star \ldots \star w_n \star f(w_1, \ldots, w_n) \underset{\uparrow}{\star} \ldots$ a ne više u završnu poziciju

$$]\sim \star f(w_1, \ldots, w_n) \underset{\uparrow}{\star} \tag{1}$$

(kako bi T). Azbuka mašine T_1 sadrži barem slova a_j, gdje je $1 \leq j \leq \max\{t_1, t_2\}$. Biće ustvari dokazano tvrđenje koje je jače od maločas spomenutog tvrđenja: postoji mašina T_2 koja normalno računa f i koja: c) ne koristi pomoćna (dopunska, nova) slova, tj. azbuka mašine T_2 je $A_{\max\{t_1,t_2\}}$. Ako mašina računa neku funkciju onda ta mašina mora da bude u mogućnosti da pročita ulazne podatke i da odštampa rezultat.

Problem (a) prevazilazi se prelaskom sa T na $T^{\S}$. Znamo da mašina $T^{\S}$ ima azbuku $A_{t+1} = \{a_1, \ldots, a_t, a_{t+1}\} = \{a_1, \ldots, a_t, \S\}$.

Problem (c) prevazilazi se prelaskom sa $T^{\S}$ na $\overline{T^{\S}}$. Znamo da $\overline{M}$ vrši istu obradu kao i M, samo što $\overline{M}$ tu obradu vrši u kodiranom obliku, $\overline{M}$ tu obradu vrši u minimalnoj azbuci $A_1 = \{a_1\}$. Za (c) još treba da se $w_k \in \Omega(A_{t_1})$ kodira sa $\overline{w_k} \in \Omega(A_1)$ za $k = 1, \ldots, n$, ovo će uraditi mašina V_n. Za (c) još treba da se $\overline{w} = \overline{f(w_1, \ldots, w_n)} \in \Omega(A_1)$ dekodira u $w = f(w_1, \ldots, w_n) \in \Omega(A_{t_2})$, ovo će uraditi mašina E. Za V_n se kaže da je mašina za kodiranje niza od n riječi. Za E se kaže da je mašina za dekodiranje. Mašina V_n ima azbuku A_{t_1}. Mašina E ima azbuku A_{t_2}.

Ustvari, V_n prvo stvori rezervnu kopiju n–torke riječi $(w_1, \ldots, w_n)$ pa onda kodira tu rezervnu kopiju. Ustvari, E pored dekodiranja još poništi $\sim$ i poništi slova poslije $\underset{\uparrow}{\star}$, v. (1). Tako da je i problem (b) riješen.

Stavimo da je $T_2 = V_n\overline{T^{\S}}E$.

Prelazimo na detalje. Mašina V_n vrši sljedeću radnju:

$$1\ ]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad 2\ ]\star w_1 \star \ldots \star w_n \star \overline{\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots}.$$

Mašina E vrši sljedeću radnju:

$$3\ ]\star w_1 \star \ldots \star w_n \star\star X_1 \overline{\star w \underset{\uparrow}{\star}} \star X_2 \star \ldots \quad \Rightarrow \quad 4\ ]\star w_1 \star \ldots \star w_n \star w \underset{\uparrow}{\star} \ldots.$$

Ovdje $\star X_1$ ima oblik $\star||\ldots|\star||\ldots|\star\ldots\star||\ldots|$, a moguće je i da $\star X_1$ sasvim odsustvuje. Slično za $\star X_2$. Izraz $\star X_1$ predstavlja kodirani oblik od $\sim$, v. (1). Izraz $\star X_2$ je kodirani oblik slova poslije $\underset{\uparrow}{\star}$, v. (1). Treba dokazati da postoje mašine V_n i E koje ispunjavaju navedene

uslove, koje vrše naznačene obrade. Treba konstruisati dijagram za V_n, odnosno za E. Dva dijagrama se izostavljaju.

Mašina V_n prevodi 1 u 2, $\overline{T^{\S}}$ prevodi 2 u 3, E prevodi 3 u 4. Ukupno, mašina $T_2 = V_n\overline{T^{\S}}E$ prevodi 1 u 4.

Na početku analize, ako $(w_1, \ldots, w_n) \in \Omega^n(A_{t_1}) \setminus \mathrm{Dom}f$ onda $T^{\S}$ vječno radi, $\overline{T^{\S}}$ vječno radi, pa i T_2 vječno radi. Razmotrimo i slučaj ulazne n–torke riječi koja pripada domenu funkcije, $(w_1, \ldots, w_n) \in \Omega^n(A_{t_1}) \cap \mathrm{Dom}f$. Sada bi T:

$$]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad]\sim \star w \underset{\uparrow}{\star},$$

gdje je $w = f(w_1, \ldots, w_n)$. Isto tako, tada $T^{\S}$:

$$]\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad]\sim \star w \underset{\uparrow}{\star}.$$

Znamo da je slučaj prelaska iza kraja trake (PT) isključen kada se radi o mašini $T^{\S}$. Zato, ako se na početku trake nešto doda onda to neće uticati na rad mašine $T^{\S}$. Prema tome, važi i sljedeće za $T^{\S}$:

$$]\star w_1 \star \ldots \star w_n \star\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad]\star w_1 \star \ldots \star w_n \star \sim \star w \underset{\uparrow}{\star}.$$

Najzad, za mašinu $\overline{T^{\S}}$ važi sljedeće:

$$2\ ]\star w_1 \star \ldots \star w_n \star \overline{\star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots} \quad \Rightarrow \quad 3\ ]\star w_1 \star \ldots \star w_n \star\star X_1 \overline{\star w \underset{\uparrow}{\star}} \star X_2 \star \ldots.$$

Time je tvrđenje o mašini T_2 dokazano.

(1, 2, 3 i 4 su uslovne oznake, one ne pripadaju "implikacijama", one ne postoje u "$\Rightarrow$". Njih smo mi naknadno dodali, radi boljeg razumijevanja, radi lakšeg snalaženja.)

4.7. SUPERPOZICIJA FUNKCIJA KOJE SU IZRAČUNLJIVE PO TJURINGU

Maločas dokazanu činjenicu da je izračunljiva funkcija i normalno izračunljiva mi ćemo sada iskoristiti da dokažemo iskaz koji se odnosi na superpoziciju funkcija.

Teorema. Neka je $t_1, t_2, t_3, n, m \geq 1$. Neka su $g_1, \ldots, g_m$ djelimične funkcije od n promjenljivih čije promjenljive prolaze kroz skup $\Omega(A_{t_1})$ (to znači da je $\mathrm{Dom}g_k \subset \Omega^n(A_{t_1})$) i čije su vrijednosti u skupu $\Omega(A_{t_2})$. Neka je h djelimična funkcija od m promjenljivih čije promjenljive prolaze kroz skup $\Omega(A_{t_2})$ i čije su vrijednosti u skupu $\Omega(A_{t_3})$. Definišimo funkciju f sljedećom relacijom:

$$f(w_1, \ldots, w_n) = h(g_1(w_1, \ldots, w_n), \ldots, g_m(w_1, \ldots, w_n)). \tag{1}$$

Ako su funkcije $g_1, \ldots, g_m$ i h izračunljive po Tjuringu onda je i funkcija f izračunljiva po Tjuringu.

Objašnjenje. Funkcija f je takođe djelimična, f: $\mathrm{Dom}f \to \Omega(A_{t_3})$, gdje je $\mathrm{Dom}f \subset \Omega^n(A_{t_1})$. Domen funkcije f čine sve n–torke $(w_1, \ldots, w_n) \in \Omega^n(A_{t_1})$ za koje desna strana relacije (1) ima smisla.

Dokaz. Kako su funkcije $g_1, \ldots, g_m$ i h izračunljive po Tjuringu to postoje Tjuringove mašine $T_1, \ldots, T_m$ i T koje **normalno** računaju te funkcije. Sada možemo da definišemo Tjuringovu mašinu M koja će računati f:

$$M = T_1 K^n_{n+1} T_2 K^n_{n+1} T_3 \ldots K^n_{n+1} T_m K_{(m-1)n+m} K_{(m-2)n+m} \ldots K_{(m-m)n+m} T.$$

Analiza je ovdje laka, a osnovne pozicije u toku računanja (u toku rada mašine M) date su u nastavku, gdje g_i služi kao skraćenica za $g_i(w_1, \ldots, w_n)$, dok slovo f služi kao skraćenica za $f(w_1, \ldots, w_n)$: od početne pozicije mašine M do njene završne pozicije:

$\star w_1 \star \ldots \star \underset{\uparrow}{w_n} \star \ldots$ (pomoću T_1 izračunamo $g_1(w_1, \ldots, w_n)$)

$\star w_1 \star \ldots \star w_n \star \underset{\uparrow}{g_1} \star \ldots$ (pomoću K^n_{n+1} prekopiramo n–torku riječi $(w_1, \ldots, w_n)$)

$\star w_1 \star \ldots \star \underset{\uparrow}{w_n} \star \ldots$ (pomoću T_2 izračunamo $g_2(w_1, \ldots, w_n)$)

$\star w_1 \star \ldots \star w_n \star g_1 \star w_1 \star \ldots \star \underset{\uparrow}{g_2} \star \ldots$ itd.

$\star w_1 \star \ldots \star w_n \star g_1 \star w_1 \star \ldots \star w_n \star g_2 \star \ldots \star g_{m-1} \star w_1 \star \ldots \star \underset{\uparrow}{w_n} \star g_m \star \ldots$

(prekopiramo riječi $g_1, \ldots, g_m$)

$\star w_1 \star \ldots \star w_n \star g_1 \star w_1 \star \ldots \star w_n \star g_2 \star \ldots \star g_{m-1} \star w_1 \star \ldots \star w_n \star g_m \star g_1 \star \ldots \star \underset{\uparrow}{g_m} \star \ldots$

(pomoću T izračunamo $h(g_1, \ldots, g_m)$)

$\star w_1 \star \ldots \star w_n \star g_1 \star w_1 \star \ldots \star w_n \star g_2 \star \ldots \star g_{m-1} \star w_1 \star \ldots \star w_n \star g_m \star g_1 \star \ldots \star \underset{\uparrow}{f} \star \ldots$

Možemo pisati $\star w_1 \star \ldots \star \underset{\uparrow}{w_n} \star \ldots \Rightarrow \sim \star \underset{\uparrow}{f} \star \ldots$.

Slično se analizira i slučaj kada $(w_1, \ldots, w_n) \notin \mathrm{Dom} f$. **Dokaz je završen.**

Znamo da u slučaju opšte superpozicije unutrašnje funkcije $g_1, \ldots, g_m$ imaju svaka svoj broj argumenata, a vidimo da u teoremi svaka od njih ima jedan te isti broj argumenata n. Ovim se ne gubi na opštosti, budući da se postupkom uvođenja tzv. fiktivnih promjenljivih te unutrašnje funkcije lako mogu preobraziti u funkcije koje sve imaju jedan te isti broj argumenata n.

5. PRIMJERI NERJEŠIVIH SKUPOVA

Neka je M – neki skup riječi nad azbukom $A = \{a_1, \ldots, a_t\}$ $(t \geq 1)$. Kao što znamo, skup M je rješiv ako i samo ako je njegova karakteristična funkcija izračunljiva po Tjuringu. Klasa algoritama, tj. klasa Tjuringovih mašina je već precizno definisana, jedino ćemo još precizirati koja je radna azbuka Tjuringove mašine. U prethodnom naslovu 4. dokazana je teorema da svaka po Tjuringu izračunljiva funkcija može da bude izračunata na jednoj Tjuringovoj mašini koja ne koristi dopunska slova. Dakle, važi sljedeći iskaz.

Teorema. Neka je $M \subset \Omega(A)$. Skup M je rješiv ako i samo ako postoji Tjuringova mašina čija je radna azbuka A a koja računa karakterističnu funkciju skupa M.

5.1. MAŠINSKE RIJEČI

Tjuringova mašina T, tj. njena tablica (radna azbuka mašine je $A = \{a_1, \ldots, a_t\}$) biće prikazana pomoću jedne riječi $w_T \in \Omega(A)$.

Neka je broj $t \geq 1$ fiksiran. Razmotrimo jednu Tjuringovu mašinu T čija je radna azbuka $A = \{a_1, \ldots, a_t\}$. Pogledajmo tablicu te mašine. Tablica se sastoji od $(t+1)(s+1)$ redova, gdje se broj s odnosi na broj stanja mašine: stanja su $q_0, \ldots, q_s$. Pojedini red tablice sastoji se od četiri znaka, mogući znaci su a_0, ..., a_t, q_0, ..., q_s, r, l i s, tako da očito ima ukupno $t+s+5$ mogućih znakova. Tablica se prikazivala tako što su se redovi pisali jedan ispod drugog. Naravno da redovi mogu da se zapišu jedan za drugim. Ako se tako zapiše onda se dobija jedna riječ w'' u azbuci A_{t+s+5}. Riječ w'' prikazuje ili karakteriše Tjuringovu mašinu T.

Želimo da mašinu T prikažemo pomoću riječi iz njene radne azbuke. Ako na gore konstruisanu riječ w'' primijenimo funkciju $\gamma_{t+s+5,t}$ onda ćemo dobiti upravo jednu riječ w' nad azbukom A (u drugim oznakama: nad azbukom A_t). Ali, mi sada nismo u stanju da na osnovu date riječi w' rekonstruišemo tablicu (rekonstruišemo w''). Nismo u stanju – zato što ne možemo da odredimo s. Jedino ako znamo par riječi (s, w') – mi možemo da opet dobijemo tablicu. Znamo da se za broj s (za njegov unarni prikaz) može reći da je riječ nad azbukom A_1, nad azbukom A. Očito $A_1 \subset A$. Bilo je $w' = \gamma_{t+s+5,t}(w'')$. Prema tome, $w'' = \gamma^{-1}_{t+s+5,t}(w')$. Stavimo $w = \sigma^t_2(s, w')$. Vidimo da $w \in \Omega(A)$. Umjesto w pisaćemo i w_T. Riječ w_T prikazuje Tjuringovu mašinu T (T. m. M čija je radna azbuka A prikazuje se pomoću jedne riječi nad azbukom A, ta se riječ označava kao w_M. I slično. Recimo $N \leftrightarrow w_N$.)

Za w_T se kaže da je jedna mašinska riječ zato što ona prikazuje jednu T. m. Za w_T se kaže da je (mašinska) riječ Tjuringove mašine T zato što ona prikazuje Tjuringovu mašinu T.

Od tablice Tjuringove mašine T dobija se njena riječ w_T u tri koraka: tablica $\rightarrow w'' \rightarrow w' \rightarrow w_T$. Riječ w_T dobija se po jednoj efektivnoj proceduri, tj. dobija se algoritamskim sredstvima. Naime, vidimo da je prvi korak tablica $\rightarrow w''$ jednostavan. Intuitivno je jasno da je prvi korak algoritamski ostvarljiv (algoritamski izvodljiv), tj. da predstavlja jednu izračunljivu proceduru. Što se tiče drugog koraka $w'' \rightarrow w'$ i trećeg koraka $w' \rightarrow w_T$, poznato je da su funkcije $\gamma_{t+s+5,t}$ i σ^t_2 koje ostvaruju obradu o kojoj je riječ – izračunljive. Najzad, superpozicija izračunljivih funkcija je izračunljiva funkcija.

Ne može se desiti da dvije različite Tjuringove mašine (nad A) imaju jednu te istu riječ. Po datoj mašinskoj riječi može da bude efektivno određena polazna riječ w'' (polazna tablica).

Zapaziti da nisu svi članovi skupa $\Omega(A)$ mašinske riječi. Drugim riječima, ne predstavlja svaka riječ nad azbukom A prikaz neke Tjuringove mašine (nad A). Za ovo, recimo broj slova riječi w'' je obavezno djeljiv sa četiri. Dalje, za riječ $w \in \Omega(A)$ efektivno se može ispitati da li je ili nije mašinska. Drugim riječima, postoji **algoritam** koji za ulazni podatak $w \in \Omega(A)$ štampa rezultat "da" ili "ne", tj. "jeste mašinska riječ" odnosno "nije mašinska riječ". Za ovo, pomoću funkcija oblika σ i oblika γ od w se dobije w''. Zatim, w'' je tablica ako: prvo slovo je q_0,

drugo slovo je a_0, treće slovo je neko od $r, l, s, a_0, \ldots, a_t$, četvrto slovo je neko od $q_0, \ldots, q_s$, peto slovo je q_0, šesto slovo je a_1, itd. Možemo reći da smo sastavili algoritam u intuitivnom smislu. Prihvatamo Čerčovu tezu, pa možemo reći da smo sastavili i algoritam. Znamo da Čerčova teza glasi: algoritam u intuitivnom smislu = algoritam. (Ako bi se izvršila konstrukcija u pravom smislu te riječi odgovarajuće Tjuringove mašine (ako bi se sastavila tablica ili dijagram) onda bi upotreba Čerčove teze bila izbjegnuta.)

Postoje razne ideje o tome kako da se tablica T prikaže pomoću jedne riječi, pa pogledajmo još jedno moguće rješenje. Znamo da se tablica sastoji od više redova i da pojedini red sadrži četiri elementa. Pojedini red ima oblik $qavq'$, npr. može da glasi $q_4a_3rq_1$. U prikazivanju, na račun q_i pišemo slovo D i za njim i puta slovo A. Na račun a_j pišemo slovo D i za njim j puta slovo A. Što se tiče trećeg elementa, nakon D treba napisati R ako je $v = r$, treba napisati L ako je $v = l$, slično S ako je $v = s$, dok treba napisati k puta slovo C ako je $v = a_k$. Prikaz pojedinog reda tablice dobija se kada se nadovežu prikazi njegovih elemenata. Recimo, $q_4a_3rq_1$ kodira se kao DAAAADAAADRDA. Drugi primjer: $q_0a_0a_2q_2$ prikazuje se kao DDDCCDAA. Zapis čitave tablice dobija se kada se nadovežu (spoje) prikazi njenih redova, s tim što se na kraju prikaza za pojedini red doda znak ";". Tako nastaje rezultat w. Kompletan primjer: ako tablica glasi $q_0a_0rq_1$ $q_0a_1rq_1$ $q_1a_0a_1q_2$ $q_1a_1a_1q_2$ $q_2a_0sq_2$ $q_2a_1sq_2$ onda će biti

$$w = \text{DDDRDA;DDADRDA;DADDCDAA;DADADCDAA;DAADDSDAA;DAADADSDAA;}$$

Za riječ w (ili w_T) mogli bismo reći da predstavlja mašinsku riječ. Za tu riječ se kaže da vrši deskripciju tablice T (mašine T), da predstavlja njenu standardnu deskripciju, engl. standard description. Po predloženom rješenju, T se prikazuje kao jedna riječ w u azbuci koja ima 7 članova: slova D, A, R, L, S i C i znak tačka–zarez.

5.2. JEDAN NERJEŠIV SKUP (M)

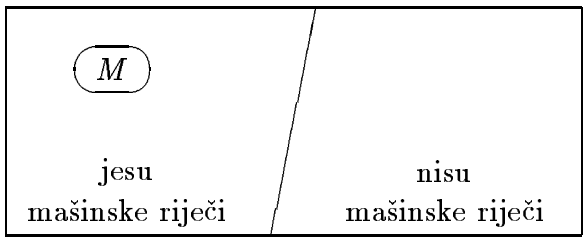
Neka je broj $t \geq 1$ fiksiran. Razmotrimo azbuku $A = \{a_1, \ldots, a_t\}$. Razmotrimo jedan skup riječi M nad azbukom A. Upravo, neka je skup $M \subset \Omega(A)$ definisan sljedećim:

$w \in M \Leftrightarrow$ postoji T. m. T nad A takva da je $w = w_T$ i koja se poslije primjene na w zaustavlja u konačno mnogo koraka iza riječi | (iza riječi a_1).

Lako zapažamo da jedino mašinske riječi mogu pripadati skupu M. Tako da se M može definisati na drugi način, kako slijedi. Neka je T neka T. m. čija je radna azbuka A. Naravno, saglasno oznakama uvedenim u prethodnom naslovu, w_T je njena (mašinska) riječ. Dakle,

$w_T \in M \Leftrightarrow$ mašina T, u slučaju da je primijenjena na zapis poslije riječi w_T, se zaustavlja u konačno mnogo koraka i to iza riječi a_1.

Pomoću formule: $w_T \in M \quad \Leftrightarrow \quad]\star w_T \underset{\uparrow}{\star} \ldots \overset{T}{\Rightarrow}]\sim\star|\underset{\uparrow}{\star}.$



Teorema. Skup M nije rješiv (skup M je nerješiv).

Prije dokaza teoreme, prepričajmo definiciju skupa M. Neka $r \in \Omega(A)$. Da li $r \in M$ ili $r \notin M$. Prvo, ako r nije mašinska riječ onda $r \notin M$. Uzmimo da je r mašinska riječ i označimo

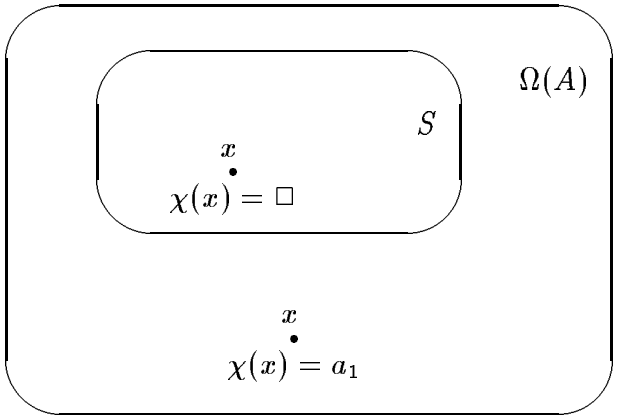
sa X odgovarajuću T. m. ($r = w_X$). Neka početna pozicija bude $]\star r\underset{\uparrow}{\star}\ldots$ i neka sada djeluje X. Ako X ne radi vječno i ako završna pozicija ima oblik $]\sim\star a_1\underset{\uparrow}{\star}$ tada i samo tada $r \in M$.

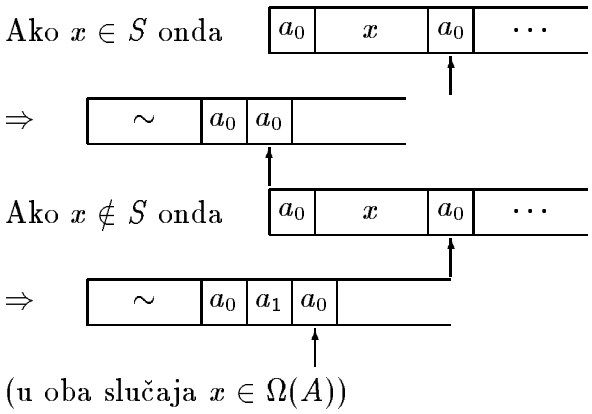
Prije dokaza teoreme, ponovimo: šta je to karakteristična funkcija χ_S skupa $S \subset \Omega(A)$ i kada je skup $S \subset \Omega(A)$ rješiv. Za $w \in \Omega(A)$, po definiciji

$$\chi_S(w) = \begin{cases} \square \text{ ako } w \in S \\ | \text{ ako } w \notin S \end{cases}$$

Napominje se da su u matematici uobičajene sljedeće oznake: $\chi_S(x) = 0$ ako $x \in S$, $\chi_S(x) = 1$ ako $x \notin S$. Također se napominje da su obično 0 i 1 suprotno postavljeni. Zapažamo da je karakteristična funkcija svuda definisana, tj. da je definisana na čitavom skupu $\Omega(A)$. Zapažamo da karakteristična funkcija $\chi_S\colon \Omega(A) \to \Omega(A_1)$ ili svejedno $\chi_S\colon \Omega(A) \to \Omega(A)$ (budući da je očito $A_1 \subset A$, slijedi $\Omega(A_1) \subset \Omega(A)$) uzima samo dvije moguće vrijednosti. Važi tvrđenje: skup S je rješiv ako i samo ako je njegova karakteristična funkcija χ_S izračunljiva.

Ilustracija za karakterističnu funkciju χ skupa S (u opštim oznakama) i ilustracija (ako je skup rješiv) za mašinu koja računa tu funkciju χ:





Dokaz teoreme. Dopustimo da je skup M rješiv. Tada postoji T. m. T_0 nad A koja računa χ_M. Detaljnije, ako se T_0 primijeni na bilo koju riječ $w \in \Omega(A)$ onda će se T_0 zaustaviti poslije konačnog broja koraka i završna pozicija će biti $\star\underset{\uparrow}{\star}$ ili $\star|\underset{\uparrow}{\star}$, čime se saopštava da $w \in M$ odnosno $w \notin M$.

ako $w \in M$ onda važi: $]\star w\underset{\uparrow}{\star}\ldots \overset{T_0}{\Rightarrow}]\sim\star\underset{\uparrow}{\star}$, ako $w \notin M$ onda važi: $]\star w\underset{\uparrow}{\star}\ldots \overset{T_0}{\Rightarrow}]\sim\star|\underset{\uparrow}{\star}$

I T_0 je T. m. nad azbukom A. I ona ima svoju mašinsku riječ w_{T_0}. Budući da i w_{T_0} pripada $\Omega(A)$, možemo se zapitati: ta riječ pripada ili ne pripada skupu M? Da odgovorimo, na početnu poziciju $]\star w_{T_0}\underset{\uparrow}{\star}\ldots$ primijenimo T_0. Postoje dvije moguće završne pozicije $\star\underset{\uparrow}{\star}$ i $\star|\underset{\uparrow}{\star}$, odnosno T_0 će se zaustaviti **poslije riječi** $\square$ **ili poslije riječi** $|$. Ako se T_0 zaustavila poslije riječi $\square$ onda saglasno definiciji karakteristične funkcije χ_M razmatranog skupa M imamo da $w_{T_0} \in M$, a saglasno samoj definiciji skupa M imamo da $w_{T_0} \notin M$ (jer se zaustavila poslije riječi $\square$, tj. ne može se reći da se zaustavila poslije riječi $|$). Dakle, prva moguća završna pozicija je protivrječna. Ako se T_0 zaustavila poslije riječi $|$ onda po jednom osnovu imamo da $w_{T_0} \notin M$, a po drugom osnovu imamo da $w_{T_0} \in M$. Dakle, i druga moguća završna pozicija je protivrječna. Postoje samo dvije moguće završne pozicije. Dobili smo kontradikciju, zato što smo dopustili da je skup M rješiv. **Teorema je dokazana.**

Upravo dokazana teorema može ovako da se prepriča. Data je azbuka A. Za proizvoljnu Tjuringovu mašinu nad tom azbukom razmatra se pitanje: da li će se mašina poslije primjene

na svoju mašinsku riječ zaustaviti poslije konačno mnogo koraka poslije slova |. Ovo pitanje je algoritamski nerješivo.

5.3. DRUGI NERJEŠIV SKUP (M_1)

U ovom naslovu biće naveden još jedan primjer skupa koji nije rješiv (skup M_1) i biće uveden pojam tzv. univerzalne Tjuringove mašine U (U je univerzalni interpreter).

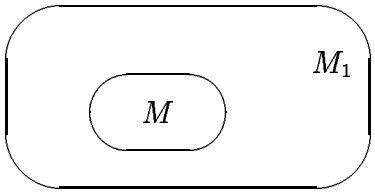
Neka je broj $t \geq 1$ fiksiran. Stavimo $A = \{a_1, \ldots, a_t\}$ (u azbuci ima t članova). Definišimo skup $M_1 \subset \Omega(A)$: $w \in M_1 \Leftrightarrow$ postoji T. m. T nad azbukom A takva da je $w = w_T$ i koja se poslije primjene na w zaustavlja poslije konačnog broja koraka. Razlog zaustavljanja je MZ (mašinsko zaustavljanje) ili PT (prelazak preko kraja trake). Lako vidimo da je $M \subset M_1$.

Dokažimo da je skup M_1 nerješiv.

Pretpostavimo suprotno – da je M_1 rješiv. Tada postoji T. m. T_1 koja računa karakterističnu funkciju skupa M_1. Korišćenjem ove pretpostavke, pokazaćemo da bi onda i skup M bio rješiv, tj. izvešćemo kontradikciju.

Korišćenjem mašine T_1 kao potprograma, u nastavku se pokazuje da je karakteristična funkcija χ_M skupa M izračunljiva. Prelazimo na konstrukciju odgovarajućeg algoritma u intuitivnom smislu.

Uzmimo jednu riječ w nad A. U nastavku se opisuje kako se određuje da li w pripada ili ne pripada skupu M. Prvo odredimo da li je w uopšte nečija mašinska riječ. Ako nije mašinska riječ onda dajemo odgovor "$w \notin M$" i stop. A ako jeste mašinska riječ onda nastavljamo ispitivanje. Po w rekonstruišemo tablicu Tjuringove mašine T koju w prikazuje. Neka sada djeluje komponenta T_1 koja saopštava $w \in M_1$ ili $w \notin M_1$. U slučaju da $w \notin M_1$, saopštavamo da $w \notin M$ i stop, zato što je $M \subset M_1$. U slučaju da $w \in M_1$, nastavljamo ispitivanje. Učinimo da se računanje nastavi kao da mašina T radi polazeći od $]\star w \underset{\uparrow}{\star}\ldots$ kao svoje početne pozicije. Drukčije rečeno, sada se simulira funkcionisanje mašine T za slučaj njene primjene na poziciju $]\star w \underset{\uparrow}{\star}\ldots$! Tako da naša traka u datom trenutku ne glasi $]\star w \underset{\uparrow}{\star}\ldots$, nego će naš algoritam oponašanjem rada mašine T ustanoviti: šta bi T uradila kada bi se primijenila na $]\star w \underset{\uparrow}{\star}\ldots$.



Budući da $w \in M_1$ to će se oponašanje ostvariti u potpunosti za konačno vrijeme, za konačan broj koraka (v. definiciju skupa M_1)! U rezultatu oponašanja, negdje na našoj traci će nekako biti saopšteno: kakva bi bila završna pozicija za slučaj $]\star w \underset{\uparrow}{\star}\ldots \overset{T}{\Rightarrow} S$. Po saopštenom, mi lako odredimo: da li bi ta završna pozicija imala oblik $\star\,|\underset{\uparrow}{\star}$, v. definiciju skupa M (prethodni naslov). Ako bi imala oblik $\star\,|\underset{\uparrow}{\star}$ onda $w \in M$ i to se saopštava kao odgovor. A ako ne bi bila tog oblika onda $w \notin M$ i naš algoritam saopštava "ne". Nakon toga, i u jednom i u drugom slučaju, izvršavanje našeg algoritma se završava. Dakle, ukupno, konstruisali smo algoritam za M. Prema tome, dobili smo kontradikciju. **Dokaz je završen.**

Kada smo maločas govorili o oponašanju, mi smo ustvari implicitno upotrebili sljedeću značajnu teoremu.

Uvedimo potrebne oznake. Razmotrimo n–torku riječi $(w_1, \ldots, w_n)$, gdje $w_k \in \Omega(A)$ (u drugim oznakama: $w_k \in \Omega(A_t)$) za $1 \leq k \leq n$. Ta n–torka može da se prikaže pomoću jedne riječi $\overline{w} \in \Omega(A)$. Riječ $\overline{w} = \overline{w}(w_1, \ldots, w_n)$ definisana je relacijom $\overline{w} = \sigma_2^t(n, \sigma_n^t(w_1, \ldots, w_n))$.

Teorema. Neka je broj $t \geq 1$ fiksiran. Postoji tzv. **univerzalna** Tjuringova mašina U (njena radna azbuka je A) koja je u stanju da oponaša ili modelira rad bilo koje Tjuringove mašine T čija je radna azbuka A. Detaljnije, funkcionisanje mašine U za početnu poziciju $]\star w_T \star \overline{w} \underset{\uparrow}{\star} \ldots$ po svemu odgovara funkcionisanju mašine T za početnu poziciju $] \star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$. Upravo, U vječno radi ako i samo ako bi T vječno radila, U prelazi preko kraja trake ako i samo ako bi T prešla preko kraja trake i U se mašinski zaustavlja ako i samo ako bi se T mašinski zaustavila. Dodatno, u posljednjem–trećem slučaju (MZ), U se zaustavila poslije neke riječi ako i samo ako bi se T zaustavila poslije neke riječi, ako je tako onda se jedna i druga riječ poklapaju.

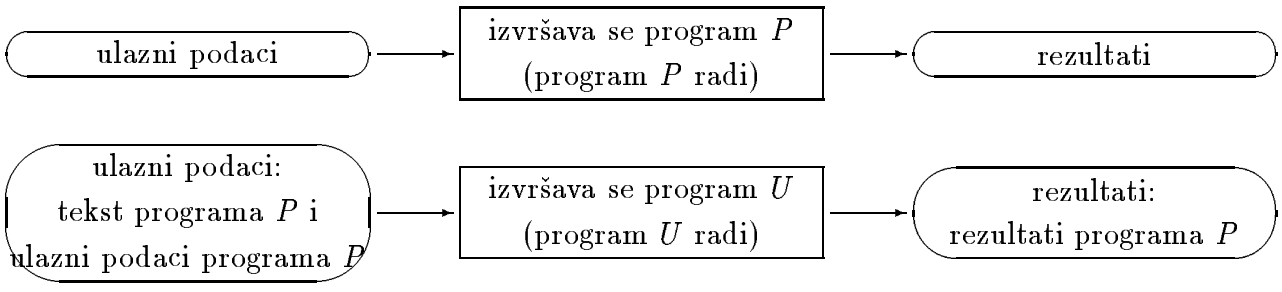
Iz dokaza: interpretacija prati u korak rad originalne mašine T.

Ovu teoremu nećemo dokazivati, već ćemo samo iskazati njen prevod na jezik programera. Postoji program na Paskalu U koji ima sljedeće svojstvo. Neka su ulazni podaci za program U dva teksta w_1 i w_2, s tim da je w_1 – tekst nekog programa na Paskalu T. Ako se U izvršava sa tim ulaznim podacima w_1 i w_2 onda će se dobiti kao rezultat – ono što bi se dobilo kao rezultat da se program T izvršavao sa w_2 kao svojim ulaznim podatkom.

Teorema i prevod su ekvivalentni zato što su mogućnosti Tjuringovih mašina ekvivalentne mogućnostima fizičkog kompjutera. U početnoj poziciji Tjuringove mašine samo konačno mnogo polja sadrži znak različit od $\star$; tokom rada kompjutera njegova memorija može po potrebi da se proširuje.

A. M. Turing je dokazao da ne postoji algoritamsko rješenje za zadatak o zaustavljanju i konstruisao je univerzalni program U u radu "On computable numbers".

Univerzalni program U postoji i u programskom jeziku C, pa izložimo ukratko, makar i uprošćeno. Neka je P bilo koji program na jeziku C. Razmotrimo prvi slučaj, njegovog običnog izvršavanja. Tokom rada programa P, on učitava svoje ulazne podatke a takođe i saopštava rezultate. Pogledajmo drugi slučaj, koji nas interesuje. Tokom rada programa U dešava se ovako. Program U učitava (kao svoje ulazne podatke) – sami tekst programa P, kao i P–ove ulazne podatke. A saopštava iste one rezultate koje bi P saopštio prilikom svog samostalnog izvršavanja. U zaključku, za U se obično kaže da je opšti interpreter zato što je u stanju da izvršava bilo koji program P. Ilustracija za prvi i drugi slučaj po šablonu ulazni podaci $\longrightarrow$ program $\longrightarrow$ rezultat:



Tokom rada U (tokom oponašanja), dešavanja su po svemu slična onima kod prostog izvršavanja P: instrukcije, njihov redosljed, među–rezultati i drugo, samo što je niz među–pozicija širi. Između ostalog, moguće je da program uđe u "ciklus" (da radi beskonačno). Kažimo na kraju da univerzalni program postoji u slučaju bilo kog modela za računanje. Uvijek, programi U i P odnose se na jedan te isti model.

5.4. ZADATAK O ZAUSTAVLJANJU

U ovom naslovu biće dokazana algoritamska nerješivost zadatka o zaustavljanju za Tjuringovu mašinu.

Formulišimo tzv. zadatak o zaustavljanju ili engl. halting problem.

Zadatak. Neka je broj $t \geq 1$ fiksiran. Neka je T bilo koja T. m. nad $A = \{a_1, \ldots, a_t\}$ i neka je w_T njena mašinska riječ. Neka se T primjenjuje na početnu poziciju $] \star w_1 \star \ldots \star w_n \underset{\uparrow}{\star} \ldots$ ($n \geq 1$). Da li će se tada T ikad zaustaviti? Drugim riječima, da li će raditi samo konačno mnogo taktova ili će pak raditi vječno?

Teorema. Zadatak o zaustavljanju je algoritamski nerješiv.

Drugim riječima, ne postoji T. m. T_2 ili H takva da $] \star w_T \star \overline{w} \underset{\uparrow}{\star} \ldots \overset{H}{\Rightarrow}] \sim \star \underset{\uparrow}{\star}$ u slučaju da T sa ulaznim podatkom $(w_1, \ldots, w_n)$ radi samo konačno mnogo taktova, odnosno $] \star w_T \star \overline{w} \underset{\uparrow}{\star} \ldots \overset{H}{\Rightarrow}] \sim \star \, | \underset{\uparrow}{\star}$ u slučaju da radi vječno. Ovdje je $\overline{w} = \overline{w}(w_1, \ldots, w_n) = \sigma_2^t(n, \sigma_n^t(w_1, \ldots, w_n)) \in \Omega(A)$. Vidimo da $\overline{w}$ kodira uređenu n–torku riječi $(w_1, \ldots, w_n)$ pomoću jedne riječi u radnoj azbuci mašine. Zajedno, w_T prikazuje tablicu mašine T, a $\overline{w}$ prikazuje njenu početnu poziciju.

Ili svejedno: skup $M_2 \subset \Omega(A) \times \Omega(A)$ nije rješiv, gdje je skup M_2 definisan uslovom: $(w_T, \overline{w}) \in M_2 \Leftrightarrow T$ sa ulaznim podatkom $(w_1, \ldots, w_n)$ radi samo konačno mnogo taktova.

Prevod zadatka o zaustavljanju na jezik programera: sastaviti program na Paskalu H koji ima sljedeće svojstvo. Neka su njegovi ulazni podaci dva teksta w_1 i w_2, s tim da je w_1 tekst nekog programa T napisanog na Paskalu. Da li bi se T sa w_2 kao svojim ulaznim podatkom ikad zaustavio ili bi pak (suprotno) radio vječno. Neka program H saopštava "da" odnosno "ne".

Program na Paskalu H ne postoji.

Ako su ulazni podaci za H

w_1 = `read(a); read(b); c←a+b; c←c+1; write(c)` i $w_2 = \texttt{10␣20}$

onda H saopštava "da" ili svejedno "zaustaviće se". Ako su ulazni podaci

w_1 = `1: goto 1` i w_2 = prazna riječ

onda H saopštava "ne" ili svejedno "neće se zaustaviti". Jedino šteta što takvog programa H nema.

Dokaz teoreme. Iz prethodnog naslova znamo da je skup M_1 nerješiv, tj. nema programa da se sazna da li bi se data T. m. primijenjena na svoju riječ zaustavila. Dovoljno je dokazati da pretpostavka "postoji H" povlači zaključak M_1 je rješiv (što predstavlja kontradikciju). Uzimajući da H postoji, program za prepoznavanje skupa M_1 glasio bi u glavnim crtama kako slijedi.

Na traci na početku piše $] \star w \underset{\uparrow}{\star} \ldots$, gdje $w \in \Omega(A)$. Pripada li ta riječ skupu? Prvo se ispita da li je w mašinska riječ. Ako nije onda se saopštava "ne" (saopštava se da $w \notin M_1$) i stop. A ako jeste onda se pozove potprogram H da djeluje na poziciju $] \star w \star \overline{w} \underset{\uparrow}{\star} \ldots$, gdje je $\overline{w} = \sigma_2^t(1, w)$. Rezultat koji potprogram saopšti (da li bi se mašina čija je riječ w zaustavila, primijenjena na ulazni podatak $\overline{w}$) preuzimamo da bude naš odgovor (da li $w \in M_1$). Program za M_1 je konstruisan.

Dobili smo kontradikciju, **dokaz je završen**.

U zaključku, algoritamsku nerješivost zadatka o zaustavljanju treba najviše pripisati velikoj opštosti samog zadatka. I sljedeći, njemu donekle slični, zadaci su algoritamski nerješivi:

da li data Tjuringova mašina za datu početnu poziciju na traci tokom svog rada makar nešto odštampa i

da li je funkcija koju data Tjuringova mašina računa konstanta (ili da li je periodična ili da li je ograničena ili da li dati broj pripada njenom skupu vrijednosti).
Nasuprot tome, algoritamski je rješiv recimo sljedeći zadatak:

da li data mašinska riječ prikazuje Tjuringovu mašinu koja ima više od (recimo) 10 stanja.

5.5. RAZNI PRIMJERI NERJEŠIVIH SKUPOVA

U ovom naslovu biće navedeno još nekoliko primjera algoritamski nerješivih zadataka, a koji se odnose na razne oblasti matematike. Dokazi se izostavljaju.

Zadatak 1: Opšta diofantska jednačina. Neka je p polinom ma kog stepena od više promjenljivih sa cijelim koeficijentima. Da li u cijelim brojevima postoji rješenje jednačine $p = 0$?

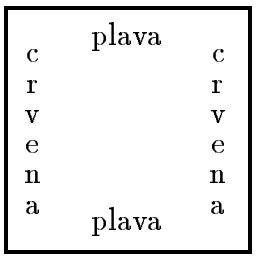
Ne postoji program (nema računara) koji bi rješavao postavljeni zadatak.

Zadatak 2: Igra domina ili zadatak o popločavanju. Kvadratnu ploču čija je veličina 1×1 nazivamo dominom. Svaka od četiri bočne strane domine obojana je jednom bojom. Tako da četiri boje koje su pridružene redom gornjoj, donjoj, lijevoj i desnoj strani definišu jednu vrstu domina. Data je jedna konačna familija vrsta domina. S druge strane, neka su u ravni (x, y) nacrtane sve moguće vertikalne prave $x = k$ $(k \in Z)$ i sve moguće horizontalne prave $y = k$ $(k \in Z)$. Tako da je ravan podijeljena na kvadrate veličine 1×1. U jedan kvadrat očito može da se stavi jedna domina. Želimo da čitavu ravan pokrijemo dominama iz raspoloživih vrsta domina. Za pokrivanje se kaže da je **pravilno** (da je koherentno) ako je sljedeći uslov ispunjen: ako se dvije domine dodiruju onda dvije strane koje se dodiruju imaju istu boju. Nije dozvoljeno da se domina zarotira ili prevrne. Ne dodiruju se kada  . Da li postoji pravilno pokrivanje ravni?

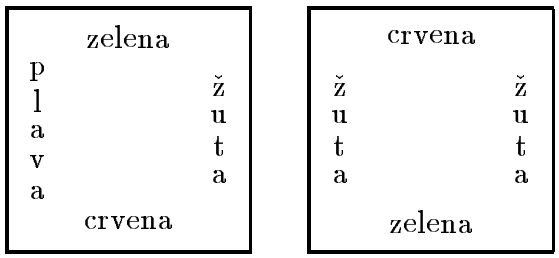
Nema programa koji bi rješavao postavljeni zadatak.

Prevod na jezik programera. Sastaviti program na Paskalu koji učitava spisak vrsta domina (ulazni podaci dati su po nekom sistemu) i saopštava odgovor "da" ili "ne" na pitanje: da li postoji pravilno pokrivanje ravni. Od svake vrste, imamo neograničen broj takvih domina. Takav program na Paskalu ne postoji.

Pogledajmo primjer. Raspoložive vrste domina prikazane su na slici 1. Za taj ulazni podatak odgovor glasi "da" (pravilno pokrivanje postoji). Pogledajmo primjer. Raspoložive vrste domina prikazane su na slici 2. Za taj ulazni podatak odgovor glasi "ne" (pravilno pokrivanje ne postoji).



Slika 1



Slika 2

Zadatak 3: Asocijativni račun u polugrupi. Razmotrimo azbuku $A = \{a, b, c, d, e\}$. Razmotrimo spisak dozvoljenih zamjena:

$$ac \equiv ca, \quad ad \equiv da, \quad bc \equiv cb, \quad bd \equiv db, \quad eca \equiv ce, \quad edb \equiv de, \quad cca \equiv ccae.$$

Imamo pravo da zamijenimo dio riječi. Pojedina zamjena može se koristiti u jednom ili drugom smjeru. Za dvije riječi $w_1 \in \Omega(A)$ i $w_2 \in \Omega(A)$ kaže se da su ekvivalentne ako se uzastopnom primjenom dozvoljenih zamjena može postići da se riječ w_1 pretvori u riječ w_2. Da li su dvije riječi w_1 i w_2 ekvivalentne?

Nema programa koji bi služio za postavljeni zadatak. Ulazni podaci programa bile bi dvije riječi w_1 i w_2, a program bi saopštavao odgovor "da" ili "ne".

Samo jedna vrlo mala ilustracija:

Pođimo od riječi npr. $w_1 = acdcec$. Kada na nju (na ac) primijenimo prvu zamjenu $ac \to ca$ onda se ona pretvara u riječ $cadcec$. Na njen dio ce primijenimo petu zamjenu $ce \to eca$, čime dobijamo $cadecac$. Sada za de upotrebimo šestu zamjenu $de \to edb$, pa tako imamo $caedbcac$. Još samo prva zamjena $ac \to ca$ (na ac), pa $caedbcca = w_2$.

Prema tome, ako je $w_1 = acdcec$ i $w_2 = caedbcca$ onda odgovor glasi "da" (te dvije riječi su ekvivalentne).

U slučaju druge neke azbuke A i drugog nekog spiska dozvoljenih zamjena $a_1 \equiv b_1, \quad \ldots, a_n \equiv b_n$ (ili bi se pisalo $a_1 \leftrightarrow b_1, \quad \ldots, \quad a_n \leftrightarrow b_n$), gdje $a_k, b_k \in \Omega(A)$ za $1 \leq k \leq n$, zadatak o ekvivalentnosti dvije riječi je algoritamski rješiv ili algoritamski nerješiv. Dozvoljeno je da se u riječi njena podriječ a_k zamijeni sa b_k i dozvoljeno je da se u riječi njena podriječ b_k zamijeni sa a_k.

PLAN BUDUĆEG RADA

Iz knjige Aho, Hopcroft, Ullman "The design and analysis of computer algorithms" radićemo ove glave:

u prvom semestru:

1. Models of computation,

u drugom semestru:

2. Design of efficient algorithms
3. Sorting and order statistics
4. Data structures for set manipulation problems
5. Algorithms on graphs

10. NP–complete problems
11. Some provably intractable problems.

\+ Cryptography.

RAZRADA I ANALIZA KOMPJUTERSKIH ALGORITAMA

1. MODELI ZA RAČUNANJE

U ovoj glavi uvode se tri modela za računanje ili tri idealizovana računara: RAM, RASP i Tjuringova mašina.

1.1. Algoritmi i njihova složenost

U ovom naslovu govori se o složenosti algoritma, odnosno o trošku za izvršavanje programa i onda se još govori o ocjeni vrijednosti algoritma, a koja zavisi od njegove složenosti.

Razmotrimo dva uređaja za računanje, od kojih je prvi zamišljen ili teorijski, a drugi je stvaran ili fizički: a) Tjuringova mašina i b) računar ili kompjuter. Za (b), treba konkretizovati vrstu računara, odnosno njegov mašinski jezik ili njegov jezik asemblera. Znamo da su (a) i (b) ekvivalentni po dometu, po oblasti djelovanja, po tome kakva računanja uspijevaju da ostvare. S druge strane razmotrimo jedan zadatak. Opšti oblik postavke zadatka je sljedeći: izračunati vrijednost jedne funkcije za datu vrijednost argumenta (ili argumenata). Ili: sastaviti program za Tjuringovu mašinu ili za računar koji će u rezultatu svog izvršavanja da saopšti kao rezultat vrijednost funkcije za argument (ili argumente) koji su ulazni podaci. Funkcija može da bude od jednog argumenta tj. $f\colon N \to N$ ili $f\colon Z \to Z$ ili od više argumenata tj. $N^n \to N$ ili $Z^n \to Z$, a dopušteno je i da bude djelimična. Recimo, zadatak: sastaviti program za računanje faktorijela prirodnog broja, primjerak tog zadatka: izračunati 4!

Razmotrimo jednu određenu tablicu ili program Tjuringove mašine T. Definišimo složenost tog programa. U trenutku puštanja u rad mašine, na traci je zapisan ulazni podatak. Definišimo najprije veličinu ili dimenzioni broj n tog ulaznog podatka. Smatramo upotrebljenim sva polja koja sadrže znak različit od blanko ($\neq a_0$). Smatramo upotrebljenim i početno radno polje. Polje se smatra upotrebljenim i kada desno od njega ima bar jedno upotrebljeno polje. Ukupan broj upotrebljenih polja i jeste n. Vidimo da ima više početnih pozicija ili više primjeraka zadatka kojima odgovara jedan te isti dimenzioni broj n. Neka mašina odradi svoje. Mi izbrojimo koliko je koraka ili taktova ona izvela. Uočimo maksimum broja koraka po svim ulazima veličine n. Uočenu brojnu vrijednost označimo sa $T(n)$ i nazovimo je vremenskom složenošću. Posmatrano za razne n, imamo funkciju vremenske složenosti $n \mapsto T(n)$ određenog programa T. Recimo, ako je početna pozicija

$$\begin{array}{l}]\ a_1\ a_0\ a_0\ a_0\ a_0\ \ldots \\ \qquad\quad\ \uparrow \end{array}$$

onda je $n = 4$. Postoje i druge početne pozicije kojima odgovara $n = 4$. Neka broj izvršenih koraka za razne početne pozicije sa $n = 4$ iznosi jednom 20, jednom 18, jednom 14 i slično. Tada će biti $T(4) = 20$. Dopunimo definiciju: ako postoji bar jedan ulaz veličine n takav da mašina sa tim ulaznim podatkom radi vječno onda se za to n stavlja da je $T(n) = +\infty$. Neka je $t \geq 1$ i razmotrimo azbuku $A = \{a_1, \ldots, a_t\}$. Ako Tjuringova mašina čija se složenost razmatra služi za računanje funkcije od jedne promjenljive $f\colon \Omega(A) \to \Omega(A)$ onda je njen ulazni podatak neka riječ $w \in \Omega(A)$. U ovom slučaju, početna pozicija u kojoj je ulazni podatak zapisan na običan ili kanonski način glasi

$$\begin{array}{l}]\ a_0\ w\ a_0\ \ldots \\ \qquad\ \uparrow \end{array} \qquad \text{ili} \qquad \begin{array}{l}]\ b_1\ b_2\ \ldots\ b_n\ a_0\ \ldots \\ \quad\ \uparrow \end{array}$$

gdje je $w = b_1 b_2 \ldots b_n$ ($b_i \in A$) i samim tim $|w| = n$ (dužina riječi). Izrazimo formalno definiciju složenosti. Uvedimo potrebne oznake. Neka niz(w) bude niz konfiguracija mašine u slučaju da je u početnoj poziciji upisana na kanonski način riječ w. Neka $\ell(\text{niz}(w))$ bude dužina niza. Imamo:

$$T(n) = \max_{w \in \Omega(A), |w| = n} \ell(\text{niz}(w)).$$

Ovdje očito $n \in N$ i $\ell(\text{niz}(w)) \in N \cup \{+\infty\}$ i također $T(n) \in N \cup \{+\infty\}$. Što se tiče funkcije složenosti $T = T(n)$, najviše nas interesuje njeno ponašanje ili rast kada $n \to \infty$.

Osim vremenske složenosti $T = T(n)$, definiše se i prostorna složenost $S = S(n)$ jednog konkretnog programa. Za početnu poziciju veličine n, mi izbrojimo koliko je polja upotrebljeno tokom rada mašine, odnosno koliko se najviše tokom rada mašine glava trake pomjerila udesno. Maksimalni broj upotrebljenih polja (po svim ulazima veličine n) označimo sa $S(n)$. Ako postoji bar jedan ulaz veličine n takav da se glava neograničeno pomijera udesno onda se za to n stavlja da je $S(n) = +\infty$.

Do sada smo govorili o složenosti u slučaju (a), a sada ćemo nešto reći i o slučaju (b). Dakle, kako se u slučaju računara definišu obim ulaza n i vremenski i prostorni trošak $T(n)$ i $S(n)$ određenog programa P? Obim ulaza n je broj bajta potrebnih da se izraze ili da se zapišu ulazni podaci programa. Za definisanje vremenskog troška, gleda se broj izvršenih taktova tokom rada programa. Dakle, $T(n)$ je maksimum broja izvršenih taktova, po svim ulazima veličine n. Za definisanje prostornih zahtjeva, gleda se broj bajta ili broj memorijskih riječi potrebnih za podatke (potrebnih za ulazne podatke, među–rezultate i rezultate).

O ocjeni vrijednosti algoritma. Ako je funkcija $T = T(n)$ polinom onda se kaže da je algoritam dobar ili efikasan. Na primjer, $T(n) = n^2$. Isto se kaže i u slučaju da se funkcija $T = T(n)$ povinuje nekom polinomu, tj. u slučaju da postoji polinom $p = p(n)$ takav da je $T(n) \leq p(n)$ za svako n. Nasuprot tome, razmotrimo algoritam čija je funkcija složenosti takva da se ne povinuje nikojem polinomu (takva da ne postoji polinom kome bi se ta funkcija povinovala). Za takav algoritam kaže se da je rđav ili neefikasan. Na primjer, $T(n) = 2^n$. Ako su za neki zadatak poznati samo algoritmi koji su svi neefikasni onda se ne može reći da je taj zadatak riješen na zadovoljavajući način, sa stanovišta računara. Vidimo da smo sve algoritme podijelili u dvije velike klase: one čija je složenost polinomska i one čija je složenost nepolinomska ili nadpolinomska ili eksponencijalna.

Slične okolnosti imamo i kod ocjene složenosti jednog zadatka. Za rješavanje jednog zadatka, razni programeri predložiće razne programe čija je složenost različita. Ako je predloženi algoritam neefikasan onda to samo po sebi ne znači da je i sami zadatak težak. Prirodno je za složenost zadatka proglasiti složenost najboljeg (najefikasnijeg) algoritma od svih mogućih algoritama koji služe za njegovo rješavanje.

Definicija. Za zadatak se kaže da je lak ili da je lako rješiv ako za njega postoji bar jedan algoritam čija je složenost polinomska. U suprotnom slučaju za zadatak se kaže da je težak ili da je teško rješiv ili da je engl. intractable.

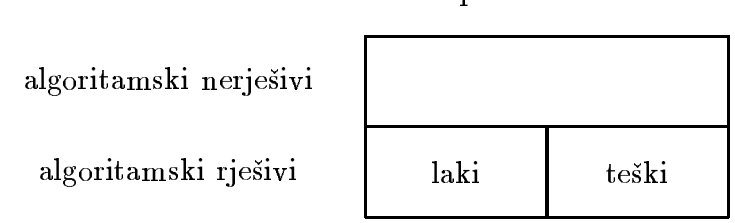
Napominje se da se može govoriti o dvije klase složenosti algoritama, a i zadataka, istim riječima u slučaju bilo vremenske bilo prostorne složenosti. Veći značaj pridaje se proučavanju vremenske nego prostorne. Kada se kaže prosto "složenost" tada se ima u vidu "vremenska složenost". Recimo, "zadatak je lak" znači da za taj zadatak postoji program koji se brzo izvrši, koji zahtijeva ili troši malo vremena.

Napominje se da je uvijek $T(n) \geq S(n)$. Da bi jedno polje trake (ili memorijska lokacija) bila upotrebljena potrebno je izvršiti jedan takt. A ima taktova koji ne traže prostorni trošak. Tako da, recimo: zadatak je težak u smislu prostorne složenosti $\Rightarrow$ zadatak je težak u smislu vremenske složenosti.

Da bi priča o složenosti zadataka bila potpuna, treba učiniti sljedeću dopunu. Očito da o složenosti zadatka ima smisla govoriti samo ako postoji bar jedan algoritam za njegovo rješavanje. Drugim riječima, algoritamski nerješivi zadaci su isključeni iz svega toga. Znamo da su za algoritamski nerješive zadatke sredstva koja pruža teorija algoritama (koja pružaju

računari) nedovoljna ili da ne koriste. Ti zadaci nisu efektivno rješivi.

Sljedeća slika pomaže da se rasporede uvedeni pojmovi:



Zadaci iz donjeg lijevog dijela velikog kvadrata svi zajedno čine važnu klasu $\mathcal{P}$ (zadaci rješivi u polinomskom vremenu).

Navedimo nekoliko primjera zadataka da ilustrujemo uvedene pojmove.

1. Da li diofantska jednačina $p(x_1, \ldots, x_n) = 0$ ima cjelobrojnog rješenja. O ovom zadatku je bilo riječi ranije, kada je bila data i njegova postavka. Znamo da je Matijasevič dokazao da je zadatak algoritamski nerješiv.

2. Zadatak o poluproširenom regularnom izrazu: da li jezik pridružen takvom izrazu ima prazan komplement. Biće formulisan kasnije u nastavku gradiva. Težak tj. $\notin \mathcal{P}$.

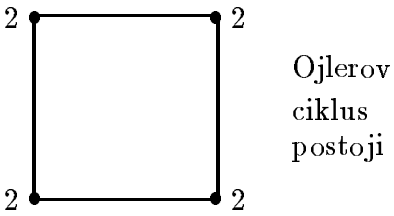
3. Zadatak o podjeli ili o particiji ili o kamenčićima. Ima nekoliko kamenčića čije su težine pozitivni cijeli brojevi. Mogu li se rasporediti na dva tasa vage tako da onda vaga bude u ravnoteži. Ili: mogu li se dati prirodni brojevi $x_1, x_2, \ldots, x_n$ podijeliti u dvije grupe tako da zbir svih članova prve grupe bude jednak zbiru svih članova druge grupe (svaki broj je dospio u jednu od dvije grupe). Poznato je da je zadatak o podjeli algoritamski rješiv. Ne zna se da li je lak ili težak tj. da li $\in \mathcal{P}$ ili $\notin \mathcal{P}$.

Zapaziti da zadatak o podjeli može da bude riješen pregledanjem svih mogućnosti. Upravo, svakoj podjeli odgovara jedan podskup skupa od n članova i obratno. Tako da ima 2^n mogućnosti za prvu grupu brojeva, odnosno za podjelu. Za svaku mogućnost, neka program ispita da li zadovoljava, da li se dva zbira poklapaju. Predloženo rješenje je (kako se to lako vidi) neefikasno, odnosno ono ima eksponencijalnu složenost.

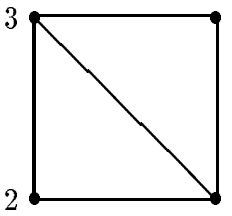
Nasuprot prethodnom, rješenje koje se zasniva na nekom tvrđenju koje se odnosi na zadatak koji treba da bude riješen obično ima polinomsku složenost, kako pokazuje sljedeći primjer.

4. Zadatak o Ojlerovom ciklusu. Dat je neusmjeren graf G. Da li u njemu postoji Ojlerov ciklus, tj. da li G čitav može da bude nacrtan u jednom potezu, odnosno ne podižući olovku sa papira i ne povlačeći neke linije dvaput, s tim da crtanje počinje i završava se u jednom te istom vrhu grafa. Navedeni zadatak je lak tj. $\in \mathcal{P}$.

Naime, poznato je sljedeće tvrđenje: takav ciklus postoji ako i samo ako je stepen svakog vrha grafa paran (svakom vrhu odgovara paran broj ivica), naravno uz uslov da je graf povezan. Ova teorema otvara put ka konstrukciji efikasnog algoritma. Ilustracija:



Ojlerov ciklus postoji



Ojlerov ciklus ne postoji

Još samo da malo ilustrujemo pojam primjerka zadatka i pojam dimenzionog broja ili veličine primjerka n, na primjeru zadatka o podjeli.

Jedan mogući primjerak je: data su tri prirodna broja i to upravo $x_1 = 10$, $x_2 = 14$ i $x_3 = 4$. Za ovaj primjerak, odsgovor na postavljeno pitanje je očito "da".

Mogući primjerak: dati su brojevi $1, 1, 1, 1, 1, 1, 2, 2, 2, 2$. Odgovor je "da".

Mogući primjerak: dati su brojevi $11, 13, 17, 100$. Odgovor je "ne".

Pogledajmo opet prvi primjerak $x_1 = 10$, $x_2 = 14$, $x_3 = 4$. Kako se ovi ulazni podaci prikazuju na traci Tjuringove mašine? Zavisi od azbuke Tjuringove mašine. Ako azbuka obuhvata cifre dekadnog brojnog sistema onda mogu da budu prikazani kao

$$\begin{array}{l}]\ \sqcup\ 1\ 0\ \sqcup\ 1\ 4\ \sqcup\ 4\ \sqcup\ \ldots \\ \qquad\qquad\qquad\qquad\quad \uparrow \end{array} \qquad \text{ili kao} \qquad \begin{array}{l}]\ 1\ 0\ \mathrm{X}\ 1\ 4\ \mathrm{X}\ 4\ \sqcup\ \ldots \\ \uparrow \end{array}$$

(tako da je obim ulaza $n = 7$) ili kao

$$\begin{array}{l}]\ \mathrm{Y}\ 1\ 0\ \mathrm{X}\ 1\ 4\ \mathrm{X}\ 4\ \mathrm{Y}\ \sqcup\ \ldots \\ \uparrow \end{array}$$

ili nešto slično. A ako se koristi binarni brojni sistem onda mogu da budu prikazani kao

$$\begin{array}{l}]\ 1\ 0\ 1\ 0\ \mathrm{X}\ 1\ 1\ 1\ 0\ \mathrm{X}\ 1\ 0\ 0\ \sqcup\ \ldots \\ \uparrow \end{array}$$

(tako da je veličina $n = 13$). Vidimo da svaka nijansa u načinu zapisivanja pomalo utiče na dimenzioni broj primjerka n. Vidimo da izbor osnove brojnog sistema utiče na dimenzioni broj n. Zato izbor osnove brojnog sistema utiče i na $T(n)$. Važno je istaći da izbor osnove ne utiče na pripadnost ili nepripadnost jednog algoritma klasi polinomskih algoritama. Tako da ne utiče ni na pripadnost ili nepripadnost odgovarajućeg zadatka klasi $\mathcal{P}$.

Jedino se zabranjuje da se brojevi prikazuju unarno, ako se proučava složenost. Na primjer, ne bi valjalo da se $\begin{array}{l}]\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \mathrm{X}\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \mathrm{X}\ 1\ 1\ 1\ 1\ \sqcup\ \ldots \\ \uparrow \end{array}$ smatra početnom pozicijom. Vidi se da bi dimenzioni broj bio znatno uvećan.

Ako $d(n)$ i $b(n)$ označavaju redom broj dekadnih odnosno binarnih cifara prirodnog broja n onda važe relacije $d(n) \approx \log n$ i $b(n) \approx \log_2 n$ i zato $\dfrac{d(n)}{b(n)} \approx \log 2 = 0{,}3$.

$*\ *\ *$ Niz primjera Tjuringovih mašina čija je vremenska složenost polinomska. (1) Ranije je rađena mašina za kopiranje jedne riječi K koja rješava zadatak $a_0\, w\, \underset{\uparrow}{a_0}\ \ldots \Rightarrow a_0\, w\, a_0\, w\, \underset{\uparrow}{a_0}\ \ldots$. Njena složenost iznosi $T(n) = 2n^2 + 8n + 3$ za $n \geq 0$, gdje je $n = |w|$ (dužina riječi). Mala je razlika stavili $n = |w|$ ili $n = |w| + 2$ (zbog dva znaka a_0 u početnoj poziciji) – ne utiče na red veličine složenosti, ne utiče na glavni sabirak. (2) Ranije je rađena mašina I koja stvara inverznu riječ. Za njenu složenost važi $T(n) \sim 2n^2$ kad $n \to \infty$. (3) Mašina koja odlučuje o pitanju da li je prirodan broj paran. Broj se prikazuje u binarnom brojnom sistemu (ili dekadnom). Pozitivan ili negativan odgovor saopštava se sa $\sim a_0\, \underset{\uparrow}{a_0}$ odnosno $\sim a_0\, a_1\, \underset{\uparrow}{a_0}$ u završnoj poziciji. (4) Mašina za sabiranje dva prirodna broja prikazana binarno (ili dekadno). Npr. ako je početna pozicija $a_0\, 1\, 1\, 1\, a_0\, 1\, 1\, 1\, \underset{\uparrow}{a_0}\ \ldots$ (tako da je $n = 9$) onda će završna pozicija biti $\sim a_0\, 1\, 1\, 1\, 0\, \underset{\uparrow}{a_0}$, u smislu $7 + 7 = 14$. (5) Mašina za množenje dva prirodna broja. (6) Mašina za računanje NZD dva prirodna broja, na bazi Euklidovog algoritma.

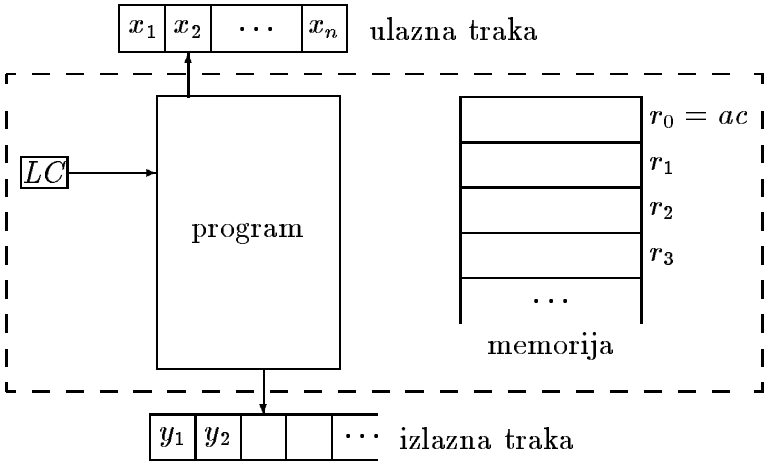
1.2. RAM

RAM je skraćenica za Random access machine, što se prevodi kao mašina sa slučajnim pristupom ili kao mašina sa proizvoljnim pristupom registrima. RAM računa sa cijelim brojevima čija veličina nije ograničena. Program po kome se računa ne mijenja se tokom svog izvršavanja, tako da se može reći da se program nalazi smješten van memorije (tzv. spoljašnji program).

Pogledajmo djelove mašine RAM. Sastavljena je od programa, memorije i dvije trake – ulazne i izlazne. Memorija se sastoji od neograničenog broja registara (lokacija). Pojedini registar može da čuva proizvoljno velik cio broj. Ulazna traka je neograničena i ima svoju glavu, izlazna traka takođe. Ulazna traka jeste niz kvadrata, svaki kvadrat sadrži jedan cio broj. Kada se pročita vrijednost iz jednog kvadrata onda se glava trake pomjeri za jedno mjesto udesno. Izlazna traka je takođe podijeljena na kvadrate koji su u početku svi prazni. Kada se u jedan kvadrat upiše neka vrijednost onda se glava trake pomjeri za jedno mjesto udesno. Mašina RAM modelira kompjuter koji je jedno–akumulatorski i u kome nije dozvoljeno da program mijenja sebe samog (tokom izvršavanja). Kada se kaže jedno–akumulatorski kompjuter onda to znači da postoji jedan poseban registar (akumulator) koji se koristi kada treba da se izvede neka aritmetička operacija. Budući da nije dozvoljeno da naredbe (djelovanjem programa) budu izmujenjene, to se obično kaže da program za RAM nije smješten u memoriji.

Računanja se obavljaju u početnom registru r_0. Za r_0 se kaže da je akumulator, engl. accumulator; skraćeno ac. Ac može da sadrži ma koji cio broj (kao uostalom i svaki drugi memorijski registar).

Slika mašine RAM:



Program je konačan niz naredbi. Jedna naredba sastoji se od dva dijela: kod operacije i adresa. Pored toga, naredba može da ima obilježje (labelu), sa sintaksom `lab: naredba`. Mi ćemo upotrebljavati skup od 12 vrsta naredbi, tj. ima 12 mogućih kodova operacija.

Spisak naredbi:

kod op. adresa

1. LOAD operand	7. READ operand
2. STORE operand	8. WRITE operand
3. ADD operand	9. JUMP label
4. SUB operand	10. JGTZ label
5. MULT operand	11. JZERO label
6. DIV operand	12. HALT —

Kažimo nešto o operandu (o argumentu operacije). Argument operacije može da bude definisan na jedan od sljedeća tri načina. 1) Neposredno. 2) Adresom registra u kome se čuva argument, ovo je običan način. 3) Adresom registra u kome se čuva adresa registra koji sadrži argument, adresa adrese. Za oblik (3) kaže se da predstavlja posredno ili indirektno definisanje adrese. Taj oblik služi prilikom rada sa vektorima (sa nizovima brojeva).

Drugim riječima, operand u naredbi može da bude $=i$ ili i ili $*i$. (1) $=i$ znači prosto i, ovdje je i cio broj. (2) i upućuje na i–ti registar tj. i znači sadržaj i–tog registra, ovdje je $i \geq 0$. (3) $*i$ znači sadržaj j–tog registra, gdje je j cio broj koji se čuva u i–tom registru, ovdje je $i \geq 0$. Ako je $i < 0$ onda se (u rezultatu izvršavanja naredbe) mašina zaustavlja.

Neka se program P izvršava. Želimo da sagledamo trenutno stanje njegovog izvršavanja (stanje mašine u određenom taktu). To stanje određuju dvije veličine c i LC. Za c, želimo da sagledamo trenutni sadržaj svih registara. Neka $c(i)$ označava sadržaj i–tog memorijskog registra (registra r_i, registra i), tako da je $i \geq 0$ i $c(i) \in Z$. Tako da je c preslikavanje, c: $N_0 \to Z$. c od engl. contents (sadržaj). Za c se kaže da je memory map (memorijska karta). LC je skraćenica za location counter (ukazivač položaja, programski brojač). LC određuje koja naredba se sljedeća izvršava.

Na početku izvršavanja je $c(i) = 0$ za svako $i \geq 0$, LC se odnosi na prvu naredbu programa P, a izlazna traka je potpuno prazna. Poslije izvršavanja k–te naredbe, LC se automatski postavlja na $(k+1)$–vu naredbu, tj. na iduću naredbu, osim ako je k–ta naredba bila JUMP, JGTZ, JZERO ili HALT.

Uvedimo matematičku oznaku $v(a)$, gdje je a – operand. $v(a)$ je vrijednost (value) operanda a. Znamo da a može da bude $=i$ ili i ili $*i$. Po definiciji je:

$$v(=i) = i, \qquad v(i) = c(i), \qquad v(*i) = c(c(i)).$$

Pomoću oznake $v(a)$ može precizno da se definiše smisao pojedine naredbe.

U sljedećoj tabeli definisan je smisao svih naredbi iz prethodnog spiska. V. tabelu. Naslov tabele je: Tabela značenja naredbi za RAM. U tabeli, operand a je $=i$ ili i ili $*i$:

	Naredba	**Značenje**
1.	LOAD a	$c(0) \leftarrow v(a)$
2.	STORE i	$c(i) \leftarrow c(0)$
	STORE $*i$	$c(c(i)) \leftarrow c(0)$
3.	ADD a	$c(0) \leftarrow c(0) + v(a)$
4.	SUB a	$c(0) \leftarrow c(0) - v(a)$
5.	MULT a	$c(0) \leftarrow c(0) \cdot v(a)$
6.	DIV a	$c(0) \leftarrow [c(0)/v(a)]$
7.	READ i	$c(i) \leftarrow$ tekući ulazni simbol
	READ $*i$	$c(c(i)) \leftarrow$ tekući ulazni simbol
8.	WRITE a	štampa se $v(a)$
9.	JUMP b	$LC \leftarrow b$
10.	JGTZ b	ako je $c(0) > 0$ onda $LC \leftarrow b$, a inače $LC \leftarrow LC + 1$
11.	JZERO b	ako je $c(0) = 0$ onda $LC \leftarrow b$, a inače $LC \leftarrow LC + 1$
12.	HALT	izvršavanje se prekida

Recimo, [2,6] = 2 i $\lceil$2,6$\rceil$ = 3. Za $x \in R$, $[x]$ je najveći cio broj koji je $\leq x$. Cio dio od x ili pod (floor) ili lo(x) ili $\lfloor x \rfloor$. Za $x \in R$, $\lceil x \rceil$ je najmanji cio broj koji je $\geq x$. Plafon (ceil) ili hi(x). Lo – low, hi – high.

Tabeli su potrebne neke dopune. Kod READ i i READ $*i$. Tekući ulazni simbol jeste sadržaj kvadrata ulazne trake koji je trenutno ispod glave. Poslije učitavanja, glava ulazne

trake pomjeri se za jedan kvadrat udesno. Kod WRITE a. $v(a)$ se štampa u kvadrat izlazne trake koji je trenutno pod glavom. Glava trake onda se pomjeri za jedan kvadrat udesno.

Objašnjenja uz tabelu. JUMP – bezuslovni skok, `goto`, JGTZ – pozitivni skok, JZERO – nula–skok, HALT – zaustavljanje, `stop`.

Naredbe koje nisu definisane (kao STORE $=i$) mogu da se smatraju ekvivalentnim naredbi HALT. Isto tako, izvršavanje naredbe koja želi da podijeli sa nulom – izaziva zaustavljanje mašine. Sada je završena definicija modela RAM.

Vidimo da $c(0)$ ima smisao – sadržaj akumulatora. Umjesto $c(0)$ obično se piše AC. Primjer: naredba LOAD i ima smisao $AC \leftarrow c(i)$, pa se kaže: iz memorije u akumulator.

Neka je P jedan program za mašinu RAM. Pogledajmo ukupan učinak tog programa. Drugim riječima, pogledajmo sadržaj čitave ulazne trake i (kada izvršavanje programa P bude okončano) čitav sadržaj izlazne trake. Program P na prirodan način definiše jedno preslikavanje iz ulaznih traka u izlazne trake. To preslikavanje je (uopšte uzev) djelimično, budući da se u slučaju nekih ulaznih traka mašina nikad neće zaustaviti. Postoje dvije važne interpretacije tog preslikavanja (1) i (2) o kojima se govori u nastavku, (1) – kao funkcija i (2) – kao jezik.

(1) Na mašinu RAM možemo da gledamo kao na uređaj koji služi za računanje vrijednosti funkcije. Pretpostavimo da program P uvijek učitava n cijelih brojeva sa ulazne trake i štampa na izlaznu traku najviše jedan cio broj. Označimo sa $x_1, \ldots, x_n$ tih n brojeva iz prvih n kvadrata ulazne trake. Ako program odštampa y u prvi kvadrat izlazne trake i poslije toga se zaustavi onda kažemo da program računa funkciju $f(x_1, \ldots, x_n) = y$.

Lako se pokazuje da model RAM posjeduje sljedeće svojstvo. RAM je u stanju da izračuna bilo koju izračunljivu funkciju i nikakve druge. Samo se napominje da se umjesto izračunljiva funkcija (po Tjuringu izračunljiva funkcija) svejedno kaže i parcijalno rekurzivna funkcija. Navedeno svojstvo posjeduje uostalom i svaki drugi razuman model kompjutera.

(2) Na mašinu RAM možemo da gledamo i kao na uređaj koji služi za prepoznavanje (prihvatanje) jezika. Neka je $k \geq 1$ i neka je A azbuka sa k slova (neka je A k–točlani skup), $A = \{1, \ldots, k\}$. Jezikom L nazivamo bilo koji podskup skupa svih riječi $\Omega(A)$. Razmotrimo jedan program P za mašinu RAM. Uzmimo jednu riječ s u azbuci A, tj. string $s \in \Omega(A)$. Neka je $|s| = n$ i $s = a_1 a_2 \ldots a_n$, gdje je očito $a_i \in A$ za $1 \leq i \leq n$. Isto $1 \leq a_i \leq k$. Smjestimo u prvi kvadrat ulazne trake slovo a_1, itd. u n–ti kvadrat upišimo simbol a_n. U $(n+1)$–vi kvadrat upišimo broj 0, taj broj služi da označi završetak ulazne riječi (end–marker). Neka se sada primijeni program P. Neka program P pročita sva slova riječi s i neka pročita i oznaku kraja. Neka P odštampa samo jednu vrijednost, tj. uvijek (za svaki ulazni podatak s) odštampa jedino neki cio broj y u prvi kvadrat izlazne trake. Neka se nakon toga program P zaustavi. Tada program P definiše jedan jezik $L \subset \Omega(A)$. Ako je odštampana vrijednost $y = 1$ onda P ptihvata riječ s (riječ s pripada jeziku L). A ako je $y \neq 1$ onda se kaže da P ne prihvata s (to znači da $s \notin L$).

Obično se uzima da P može da odštampa jedino $y = 1$ i $y = 0$, za $s \in L$ odnosno $s \notin L$.

Za dati jezik $L \subset \Omega(A)$, želimo da sastavimo program za RAM koji vrši njegovo prepoznavanje.

Maločas smo izložili kako program za RAM prepoznaje neki jezik L. Svi jezici L za koje takav program postoji čine tzv. klasu rekurzivnih jezika ili svejedno klasu odlučivih jezika. Iste te jezike prepoznaju i drugi modeli kompjutera.

U nastavku se navode dva primjera programa za RAM. U prvom primjeru računa se vrijednost funkcije. Drugi primjer odnosi se na jezik. Pored programa za RAM obično se napiše odgovarajući program na tzv. nestrogom paskalu. Služi da se bolje razumije program na jeziku mašine RAM.

1. zadatak. Neka je funkcija f: $Z \to Z$ definisana relacijom:

$$f(n) = \begin{cases} n^n, \text{ ako je } n \geq 1 \\ 0, \text{ inače} \end{cases}$$

Sastaviti program za RAM koji računa f.

Rješenje se dobija tako što se n množi sa samim sobom $n-1$ puta. Promjenljive R1, R2 i R3 smještene su u registrima 1, 2 i 3 redom. Program na nestrogom paskalu napisan je tako da bude jasna korespondencija sa programom za RAM.

Slijede plan upotrebe memorije, program na nestrogom paskalu i rješenje zadatka (program za RAM):

$r_0 = ac$	
r_1	n
r_2	$n, \ldots, n^n$
r_3	$n-1, \ldots, 0$

```
 read R1;
 if R1 ≤ 0 then write 0 else
       begin R2 ← R1; R3 ← R1 − 1;
       while R3 > 0 do
            begin R2 ← R2 · R1;
            R3 ← R3 − 1 end;
       write R2 end

       READ 1
       LOAD 1
       JGTZ pos
       WRITE= 0
       JUMP endif
  pos: LOAD 1 ⇝
       STORE 2
       LOAD 1 ⇝
       SUB= 1
       STORE 3
while: LOAD 3
       JGTZ continue
       JUMP endwhile
continue: LOAD 2
       MULT 1
       STORE 2
       LOAD 3
       SUB= 1
       STORE 3
       JUMP while
endwhile: WRITE 2
   endif: HALT
```

Ulazni podatak čita se naravno iz prvog kvadrata ulazne trake, a rezultat (vrijednost f) biva odštampan u prvi kvadrat izlazne trake.

Naredbe $\rightsquigarrow$ mogu da budu izostavljene jer one donose u AC broj koji tamo već piše.

2. zadatak. Razmotrimo jezik L u azbuci $A = \{1, 2\}$. Riječ $w \in \Omega(A)$ pripada jeziku ako ona ima svojstvo da se broj jedinica i broj dvojki u njoj poklapaju. Sastaviti program za RAM koji prihvata jezik. Svako slovo ulazne riječi zauzima jedan kvadrat ulazne trake, a sljedeći kvadrat sadrži nulu (oznaka kraja). Ako riječ pripada jeziku, program štampa 1.

Program treba da učita w, izvrši obradu i saopšti $w \in L$ ili $w \notin L$. O rješenju. Svako ulazno slovo učitava se u prvi registar. U drugom registru čuva se privremena vrijednost razlike d između broja jedinica i broja dvojki. Kada se učita oznaka kraja onda program provjerava uslov $d = 0$. Ako je uslov ispunjen onda se štampa 1, a inače se štampa 0. Slijede izgled memorije, odgovarajući program na nestrogom i rješenje zadatka (program za RAM):

$r_0 = ac$	
r_1	x
r_2	d
	$\ldots$

```
 D ← 0;
 read X;
 while X ≠ 0 do
       begin
       if X ≠ 1 then D ← D − 1
            else D ← D + 1;
       read X
       end;
 if D = 0 then write 1
       else write 0
```

```
        LOAD= 0
        STORE 2
        READ 1
while:  LOAD 1
        JZERO endwhile
        LOAD 1 ⇝
        SUB= 1
        JZERO one
        LOAD 2
        SUB= 1
        STORE 2
        JUMP endif
one:    LOAD 2
        ADD= 1
        STORE 2
endif:  READ 1
        JUMP while
endwhile: LOAD 2
        JZERO output
        WRITE= 0
        HALT
output: WRITE= 1
        HALT
```

Tri uzastopne naredbe LOAD 2, SUB= 1 i STORE 2 ostvaruju radnju D $\leftarrow$ D $-$ 1, a naredbe LOAD 2, ADD= 1 i STORE 2 ostvaruju D $\leftarrow$ D $+$ 1.

Pretpostavlja se da u kvadratima ulazne trake pišu jedino brojevi 0, 1 i 2. Naredba $\rightsquigarrow$ može da bude izostavljena. U slučaju kada je krajnja vrijednost razlike $d \neq 0$ program štampa 0, čime se saopštava da $w \notin L$.

Dalje, mašina RAM modelira programiranje na jeziku asemblera.

Navedimo još jedan primjer, da ilustrujemo indirektno adresiranje. Program ostvaruje samo raspoređivanje članova niza u memoriju, tako da može da predstavlja početak nekog zadatka.

3. zadatak. Učitati nekoliko pozitivnih brojeva i smjestiti ih počev od registra r_{11} pa nadalje. Kao oznaka kraja ulaznog niza brojeva neka služi učitavanje broja 0. Sastaviti odgovarajući program za RAM.

Izgled ulazne trake je:

$x_1{>}0$	$x_2{>}0$	$\cdots$	$x_n{>}0$	0	$\cdots$

Neka registar r_1 (svojim sadržajem) ukazuje na tekući registar za upisivanje člana niza x_k. Neka registar r_2 služi da se prihvati član x_k od ulazne trake.

Slijede izgled memorije, program na nestrogom paskalu i rješenje zadatka:

$ac = r_0$	
r_1	$11, 12, \ldots, n+10$
r_2	$x_1, x_2, \ldots, x_n$
	$\cdots$
r_{11}	x_1
	$\cdots$
r_{k+10}	x_k
	$\cdots$
r_{n+10}	x_n
	$\cdots$

```
   R1 ← 11;
1: read R2;
   R0 ← R2;
   if R0 = 0 then stop else
         store* 1;
   R1 ← R1 + 1;
   goto 1
```

```
        LOAD= 11
        STORE 1
read:   READ 2
        LOAD 2
        JZERO stop
        STORE* 1
        LOAD 1
        ADD= 1
        STORE 1
        JUMP read
stop:   HALT
```

Tri uzastopne naredbe LOAD 1, ADD= 1 i STORE 1 ostvaruju radnju R1 $\leftarrow$ R1 $+$ 1.

Vidimo da sadržaj registra r_1 utiče na naredbu s indirektnim adresiranjem STORE* 1. r_1 čuva indeks člana niza sa kojim se vrši neka operacija. Zato se za r_1 kaže da predstavlja indeks–registar. Uopšte kod računara, za procesorov registar koji ima takvu ulogu kaže se da je indeks–registar.

1.3. SLOŽENOST PROGRAMA ZA RAM

Za rješavanje zadatka, napisan je program za RAM i definisana je mjera veličine primjerka zadatka n, a treba da se procijeni trošak za realizaciju (izvršavanje) programa. Za postavku zadatka u kojoj su ulaznim promjenljivim date konkretne vrijednosti kaže se da je primjerak ili instanca zadatka. Na primjer: zadatak: ulazni podatak programa je jedna riječ a treba ispitati da li je ona palindrom; instanca zadatka: da li je riječ potop palindrom; dimenzioni broj ove instance je $n = 5$.

Ako se složenost shvati kao maksimum pojedinačnih složenosti, po svim ulazima čija je veličina n, onda se govori o složenosti najgoreg slučaja. A ako se shvati kao srednja vrijednost složenosti pojedinačnih primjeraka koji su veličine n onda se govori o složenosti očekivanog ili prosječnog slučaja. Prilikom određivanja složenosti očekivanog slučaja treba uzeti izvjesnu pretpostavku o distribuciji (vjerovatnoći nastupanja) svih ulaza veličine n. Teže je odrediti složenost očekivanog nego složenost najgoreg. Algoritam koji je najbolji po kriterijumu najgoreg slučaja obično nije najbolji po kriterijumu očekivanog slučaja. Mi ćemo razmatrati većinom složenost najgoreg slučaja. I podrazumijeva se da se ima u vidu taj kriterijum složenosti, ako nije posebno naglašeno koji od dva moguća kriterijuma se ima u vidu.

Vremenska složenost (vremenska složenost najgoreg slučaja) programa za RAM je jedna funkcija f: $N \to N$. Brojna vrijednost $f(n)$ jednaka je maksimumu, preko svih ulaza veličine n, zbira vremena potrošenih od svake naredbe koja je izvršena. Vremenska složenost očekivanog slučaja jednaka je srednjoj vrijednosti, preko svih ulaza veličine n, tog istog zbira. Ako se umjesto "vrijeme koje se potroši za svaku izvršenu naredbu" kaže "prostor koji se upotrebi za svaki odnosni registar" onda se dobija definicija prostorne složenosti programa za RAM. Imamo u vidu odnosne registre, tj. samo one memorijske registre koji su upotrebljeni, koji su bili uključeni u računanje. Sve takve registre, uključujući i akumulator. Između vremenske i prostorne, ima se u vidu vremenska, osim kada je posebno naglašeno da se ima u vidu prostorna. Da bi ove definicije bile potpune, treba još odrediti šta je to: (1) vrijeme koje je potrebno za izvršavanje jedne naredbe i (2) prostor koji pojedini registar koristi, što će sada biti određeno.

Sada opet postoje dva moguća kriterijuma: kriterijum uniformne cijene i kriterijum logaritamske cijene. Sada će jedan i drugi kriterijum da budu opisani.

U slučaju kriterijuma uniformne cijene imamo sljedeće. (1) Izvršavanje jedne bilo koje naredbe za RAM zahtijeva ili potroši jednu vremensku jedinicu (jednu mikro–sekundu). (2) Svaki registar zahtijeva jednu prostornu jedinicu.

Opišimo i drugi kriterijum (logaritamske cijene). Za pripremu, definišimo tzv. cjelobrojnu logaritamsku funkciju, u oznaci ℓ, gdje je ℓ: $Z \to Z$. Sljedeća relacija definiše tu funkciju:

$$\ell(i) = \begin{cases} [\log_2 |i|] + 1, & i \neq 0 \\ 1, & i = 0 \end{cases}$$

Sada definišemo logaritamsku cijenu $t(a)$ za tri moguća oblika operanda a:

operand a	njegova logaritamska cijena $t(a)$
$=i$	$\ell(i)$
i	$\ell(i) + \ell(c(i))$
$*i$	$\ell(i) + \ell(c(i)) + \ell(c(c(i)))$

Naredna tabela daje opšti pregled potrebnog vremena za svaku naredbu. Naslov tabele je: Logaritamska cijena naredbi za RAM. ac $= c(0)$:

Naredba	**Cijena**
1. LOAD a	$t(a)$
2. STORE i	$\ell(\text{ac}) + \ell(i)$
STORE $*i$	$\ell(\text{ac}) + \ell(i) + \ell(c(i))$
3. ADD a	$\ell(\text{ac}) + t(a)$
4. SUB a	$\ell(\text{ac}) + t(a)$
5. MULT a	$\ell(\text{ac}) + t(a)$
6. DIV a	$\ell(\text{ac}) + t(a)$
7. READ i	$\ell(\text{input}) + \ell(i)$
READ $*i$	$\ell(\text{input}) + \ell(i) + \ell(c(i))$
8. WRITE a	$t(a)$
9. JUMP b	1
10. JGTZ b	$\ell(\text{ac})$
11. JZERO b	$\ell(\text{ac})$
12. HALT	1

Kod kriterijuma logaritamske cijene uzima se u obzir ograničena veličina stvarne memorijske riječi. Ovdje se vodi računa o tome da je potrebno $[\log_2 N] + 1$ bita da bi se predstavio cio broj $N \geq 1$. Posljednja tabela sastavljena je na osnovu sljedeće grube pretpostavke: cijena obrade jedne naredbe srazmjerna je zbirnoj dužini operanada koji se na tu naredbu odnose.

Dosad smo govorili o vremenskoj složenosti (po logaritamskoj cijeni). Sada definišemo logaritamsku prostornu složenost primjerka programa za RAM kao zbir, po svim registrima, uključujući i akumulator, brojeva $\ell(x_i)$, gdje je x_i – najveći po modulu cio broj koji se tokom cijelog računanja našao u registru i. Imamo u vidu registre $i \geq 0$ uključene u računanje.

Kada treba primijeniti uniformnu cijenu, a kada logaritamsku cijenu? Ako je razumno da se pretpostavi da svaki broj koji je prisutan u zadatku može da bude smješten u jednu kompjutersku riječ onda je prihvatljiva funkcija uniformne cijene. A inače više odgovara logaritamska cijena. Mi ćemo dalje koristiti kriterijum uniformne cijene, a ne kriterijum logaritamske cijene, ako nije drukčije rečeno.

Ukupno smo definisali osam mjerila vrijednosti algoritma u obliku $2 \cdot 2 \cdot 2 = 8$ jer n. ili o., v. ili p. i u. ili l. Prednost se daje mjerilu n. v. u.

Mali primjer. Vremenski trošak za naredbu MULT 10 tj. ac $\leftarrow$ ac$\cdot c(10)$ iznosi po uniformnoj cijeni 1, a po logaritamskoj cijeni iznosi grubo računato $\log_2 |\text{ac}| + \log_2 |c(10)|$ tj. broj cifara prvog množitelja + broj cifara drugog množitelja.

Ilustrujmo uvedene pojmove na dva primjera iz prethodnog naslova 1.2.

Zadatak 1. Odrediti vremensku složenost i prostornu složenost, kako po uniformnoj cijeni tako isto i po logaritamskoj cijeni, programa za RAM iz prvog zadatka (izračunati n^n) po kriterijumu najgoreg slučaja. Izaberite kako da definišete n – dimenzioni broj primjerka zadatka, tj. mjeru za obim ulaza.

Rješenje. Neka veličina $n \geq 1$ bude taj pozitivni cio broj koji se učitava. Neka a_{11} označava v. složenost po u. cijeni, a a_{12} po l. cijeni. Neka a_{21} označava p. složenost po u. cijeni, a a_{22} po l. cijeni. Prilikom računanja složenosti, mi obično želimo da složenost ocijenimo sa gornje strane. Takođe, prilikom određivanja složenosti, mi obično zanemarujemo minorne sabirke u izrazu za složenost i još (više od toga) smatramo da imamo dovoljnu informaciju o složenosti i u slučaju da smo odredili jedino red veličine složenosti po n kad $n \to \infty$. U posljednjem slučaju kaže se da je određena asimptotska složenost programa. Imajući u vidu navedene okolnosti, pogledajmo tekst programa od ranije.

Za a_{11} treba da se izbroje taktovi, u smislu: jedan takt – jedna izvršena naredba. Lako vidimo da je $a_{11} = 8 + 9(n-1) + 3 + 2$. Naime, 8 naredbi prije ulaska u petlju, 9 naredbi za jednu iteraciju petlje (uslov petlje + tijelo petlje), 3 naredbe kod iskakanja iz petlje i 2 naredbe nakon iskakanja. Tako da je

$$a_{11} = 9n + 4 \quad \text{ili} \quad a_{11} \sim 9n \text{ kad } n \to \infty \quad \text{ili} \quad a_{11} = O(n) \text{ kad } n \to \infty.$$

Za a_{12}, glavnina troška sačinjena je od tri uzastopne naredbe LOAD 2, MULT 1 i STORE 2. Taj niz dužine tri izvrši se $n-1$ puta. Da odredimo red veličine troška, dovoljno je da posmatramo MULT 1. Vrše se množenja $n \cdot n$, $n^2 \cdot n$, ..., $n^{n-1} \cdot n$. Trošak za operaciju $n^k \cdot n$ jednak je približno $\log_2 n^k + \log_2 n$. Tako

$$a'_{12} = (\log_2 n + \log_2 n) + (\log_2 n^2 + \log_2 n) + \ldots + (\log_2 n^{n-1} + \log_2 n)$$

$$a''_{12} = \log_2 n + \log_2 n^2 + \ldots + \log_2 n^{n-1}$$

$$a''_{12} = \log_2 n + 2\log_2 n + \ldots + (n-1)\log_2 n = \frac{n(n-1)}{2}\log_2 n \sim \frac{1}{2}n^2 \log_2 n$$

$$a'_{12} \sim a''_{12}, \quad a'_{12} \sim \frac{1}{2}n^2 \log_2 n$$

$$a_{12} = O(a'_{12}), \quad a_{12} = O(n^2 \log_2 n)$$

Za a_{21} (jedna promjenljiva – jedna jedinica prostornog troška), jednostavno pogledaj plan upotrebe memorije. Koriste se četiri registra: ac, r_1, r_2 i r_3. Tako da je $a_{21} = 4$.

Za a_{22} treba sagledati za svaku vrijednost i (gdje je $0 \le i \le 3$) čemu je jednako x_i, odnosno koji je najveći po apsolutnoj vrijednosti broj koji se tokom čitavog izvršavanja programa ikad zatekao u registru i (u registru r_i). Lako nalazimo kako slijedi

$$x_0 = n^n, \qquad x_1 = n, \qquad x_2 = n^n, \qquad x_3 = n - 1$$

$$a_{22} = \ell(x_0) + \ell(x_1) + \ell(x_2) + \ell(x_3)$$

$$a_{22} \approx \log_2 x_0 + \log_2 x_1 + \log_2 x_2 + \log_2 x_3 = \log_2 n^n + \log_2 n + \log_2 n^n + \log_2(n-1)$$

$$a_{22} \sim \log_2 n^n + \log_2 n^n = 2\log_2 n^n = 2n \log_2 n$$

Tako dobijamo sljedeći odgovor:

	u. cijena	l. cijena
v. složenost	a_{11}	a_{12}
p. složenost	a_{21}	a_{22}

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} O(n) & O(n^2 \log_2 n) \\ O(1) & O(n \log_2 n) \end{bmatrix} \text{ kad } n \to \infty$$

Bolje da smo za mjeru ulaza uzeli broj binarnih cifara ulaznog podatka.

Zadatak 2. Odrediti vremensku složenost i prostornu složenost, kako po uniformnoj cijeni tako isto i po logaritamskoj cijeni, programa za RAM iz drugog zadatka ($w \in L$ ako i samo ako

se broj slova 1 poklapa sa brojem slova 2) po kriterijumu najgoreg slučaja. Izaberite kako da definišete n – dimenzioni broj primjerka zadatka, tj. mjeru za obim ulaza.

Rješenje. Uzmimo da veličina zadatka n bude jednaka dužini riječi w (broju slova riječi w) koja je ulazni podatak programa, $n = |w|$. Odgovor:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} O(n) & O(n\log_2 n) \\ O(1) & O(\log_2 n) \end{bmatrix} \quad \text{kad} \quad n \to \infty$$

Vrijednosti u drugom stupcu matrice su nešto veće od odgovarajućih u prvom stupcu, tj. logaritamski iznosi su nešto veći od uniformnih iznosa. Zato što (najgori slučaj) vrijednost razlike d može da postane velika, odnosno može da dostigne $d = n$ ili $d = -n$ ako je $w = 11\ldots1$ odnosno $w = 22\ldots2$.

Dopuna o n i t_n.

Razmotrimo zadatak koji se rješava pomoću programa za mašinu RAM. Dimenzioni broj n primjerka definiše se raznoliko, a postoje dva glavna načina za njegovo definisanje. Neka je $k \geq 1$ kvadrata ulazne trake upotrebljeno i neka su cijeli brojevi $x_1, \ldots, x_k$ upisani u tim kvadratima. Prvi način: uzima se da je dimenzioni broj jednak $n = k$. Drugi način: uzima se da je dimenzioni broj jednak

$$n = \ell(x_1) + \ldots + \ell(x_k) = [\log_2 |x_1|] + 1 + \ldots + [\log_2 |x_k|] + 1 \qquad \left(\text{tako da } n \sim \sum_{i=1}^{k} \log_2 |x_i|\right).$$

Kasnije ćemo vidjeti da drugi način predstavlja pravilan način za definisanje dimenzionog broja primjerka n. Dakle, obim pojedinog ulaznog podatka treba da bude jednak broju njegovih binarnih cifara, a obim ulaza n dobija se kada se sabere po svih k brojeva koji predstavljaju ulazne podatke programa. Isto tako, vidjećemo da je pravilan način za definisanje troška izvršavanja programa u oznaci t_n – vremenski trošak po logaritamskoj cijeni, bio je označen sa a_{12} u zadacima. Dakle, trošak za izvođenje aritmetičke operacije nad dva broja koji imaju po $2p$ binarnih cifara treba da bude dvostruko veći od troška u slučaju da imaju po $p \geq 1$ binarnih cifara. Kada smo definisali n i t_n onda gledamo oblik zavisnosti, odnosno pratimo tendenciju kad $n \to \infty$, za razmatrani program za RAM. Za taj program se kaže da je efikasan ako je $t_n \leq 4n^2$ ili $t_n \leq n^3$ ili slično. Finalno, ako za neki zadatak postoji efikasan program (koji služi za njegovo rješavanje) onda se kaže da taj zadatak pripada klasi $\mathcal{P}$.

Slika: Ulazna traka mašine RAM:

x_1	x_2	$\cdots$	x_k	$\cdots$

Na kraju, ponovimo definicije. Najgori slučaj: a_{11} = max preko svih instanci veličine n od $\sum$ (po svim izvršenim naredbama) 1, a_{12} = max preko svih instanci veličine n od $\sum$ (po svim izvršenim naredbama) logaritamska cijena naredbe, a_{21} = max preko svih instanci veličine n od $\sum$ (po svim $i \geq 0$, r_i se koristi) 1, a_{22} = max preko svih instanci veličine n od $\sum$ (po svim $i \geq 0$, r_i se koristi) $\ell(x_i)$. Očekivani slučaj: a_{11} = srednja vrijednost preko svih instanci veličine n od $\sum$ (po svim $\ldots$) $\ldots$, $a_{12} = \ldots$, $a_{21} = \ldots$, $a_{22} = \ldots$.

Pravilan način: obim ulaza $n = \sum_{i=1}^{k} \ell(x_i)$, složenost $t_n = a_{12}$ u najgorem slučaju.

1.4. MAŠINA SA UPISANIM PROGRAMOM RASP

RASP je skraćenica od Random access stored program machine, što se prevodi kao – mašina sa slučajnim pristupom i sa upisanim programom. Model RASP je sličan modelu RAM, a postoje dvije glavne razlike: program mašine RASP smješten je u memoriji (tzv. upisani ili unutrašnji program) i kod modela RASP nema mogućnosti indirektnog adresiranja. Svaka naredba programa za RASP zauzima u memoriji dva uzastopna registra. Prvi od njih sadrži kod operacije, a drugi sadrži adresu. Kod operacije je broj u granicama od 1 do 18. Tabela:

Kodovi naredbi za RASP

Naredba	**Kod operacije**		
LOAD i	1	DIV i	10
LOAD =i	2	DIV =i	11
STORE i	3	READ i	12
ADD i	4	WRITE i	13
ADD =i	5	WRITE =i	14
SUB i	6	JUMP i	15
SUB =i	7	JGTZ i	16
MULT i	8	JZERO i	17
MULT =i	9	HALT	18

Stanje mašine RASP može da se definiše pomoću memorijske karte c i ukazivača položaja LC. LC ukazuje na prvi od dva uzastopna registra koji čuvaju jednu naredbu. S obzirom da je na početku izvršavanja – program smješten u memoriji, to se ne može reći da na početku svi registri treba da sadrže nulu.

Složenost programa za RASP definiše se slično kao u slučaju modela RAM. Takođe su moguće i uniformna cijena i logaritamska cijena. Ako se radi o uniformnom kriterijumu onda je cijena izvršavanja jedne bilo koje naredbe jednaka naravno jednoj vremenskoj jedinici. Ako se radi o logaritamskom kriterijumu onda je na primjer cijena izvršavanja naredbe ADD =i jednaka $l(j) + l(c(0)) + l(i)$, gdje je ta naredba smještena u registrima j–tom i (j + 1)–vom. Dodavanje prvog sabirka l(j) ilustruje razliku u odnosu na RAM jer se ovdje još uračunava i logaritamska cijena ukazivača položaja LC; LC ima vrijednost j.

ADD =i znači $c(0) \leftarrow c(0) + i$ ili svejedno $AC \leftarrow AC + i$.

Želimo da uporedimo dva modela RAM i RASP u pogledu oblasti djelovanja (koje obrade je u stanju da ostvari taj model), a nakon toga i u pogledu složenosti.

Teorema 1.1. Neka je P_1 bilo koji program za RAM. Tada postoji njemu ekvivalentan program P_2 za RASP. Označimo sa $T_1 = T_1(n)$ vremensku složenost programa P_1 po uniformnoj cijeni. Označimo sa $T_2 = T_2(n)$ vremensku složenost programa P_2 po uniformnoj cijeni. Postoji konstanta $k > 0$ takva da je $T_2(n) \leq kT_1(n)$ za svaki prirodan broj n. Posljednja nejednakost $T_2(n) \leq kT_1(n)$ važi i u slučaju da $T_1 = T_1(n)$ i $T_2 = T_2(n)$ označavaju redom vremenske složenosti programa P_1 odnosno P_2 po logaritamskoj cijeni.

Dokaz teoreme. Polazimo od datog programa P_1 za RAM i želimo da konstruišemo program P_2 za RASP koji vrši ekvivalentnu obradu. Želimo da postoji što veća paralelnost u radu jednog i drugog programa. U okviru toga, da sadržaju akumulatora mašine RAM tj. programa P_1 odgovara sadržaj akumulatora mašine RASP tj. programa P_2 (da se jedan i drugi sadržaj poklapaju). Vidjećemo da će paralelnost ponekad morati da bude napuštena.

Da se P_2 konstruiše, prvo isplanirajmo upotrebu memorije mašine RASP. Registar 1 mašine RASP koristiće se za privremeno upisivanje sadržaja akumulatora mašine RAM (prilikom odstu-

panja od paralelnosti). Označimo sa $r-1$ broj registara koje P_2 zauzima. Vrijednost $r-1$ saznaćemo kada konstruišemo program P_2. P_2 će da zauzima registre $2,\ldots,r$ (mašine RASP). Za svako $i \geq 1$, sadržaj registra i mašine RAM poklapaće se sa sadržajem registra $r+i$ mašine RASP. Zato će obraćanja memoriji u slučaju RASP biti za r veća od obraćanja memoriji u slučaju RAM.

Objasnimo kako se konstruiše P_2. Svaka naredba programa P_1 koja ne sadrži indirektno adresiranje biva jednostavno pretvorena u identičnu naredbu programa P_2. Primjera radi, na račun naredbe oblika WRITE i u P_1, imaćemo u P_2 jednu naredbu koja zauzima dva registra, gdje će u prvom pisati (saglasno prethodnoj tabeli kodova) 13, a u drugom će stajati $r+i$.

Ostaje da se objasni kako se pretvaraju naredbe iz P_1 koje sadrže indirektno adresiranje. Da bi se ovo postiglo, koristi se mogućnost modela RASP da naredbe mogu da budu modifikovane tokom izvršavanja programa. Takvoj naredbi programa P_1 biće pridružena grupa od šest naredbi programa P_2. Konstrukcija grupe izvodi se na isti način, bez obzira o kojoj vrsti naredbe se radi (samo da ima indirektnu adresu). Tako da je dovoljno objasniti na jednom primjeru. Uzmimo za primjer naredbu za oduzimanje SUB. Dakle, u P_1 se pojavljuje naredba oblika SUB *i. U P_2 se vrši simulacija ili oponašanje za SUB *i pomoću grupe naredbi. V. sljedeću tabelu koja definiše tu simulaciju. Ona predstavlja dio programa P_2. Za smisao sadržaja parnih registara – v. tabelu kodova. Ponovimo da $\forall i$ važi c(RAM i) = c(r + i), c – contents.

Tabela: Simulacija naredbe SUB *i od strane mašine RASP:

registar	sadržaj	smisao
100	3	STORE 1 (ostavi ac)
101	1	
102	1	LOAD r + i (donesi indirektnu adresu)
103	r + i	
104	5	ADD =r (adaptacija adrese)
105	r	
106	3	STORE 111 (ostavi adresu)
107	111	
108	1	LOAD 1 (vrati ac)
109	1	
110	6	SUB b (oduzimanje)
111	b	

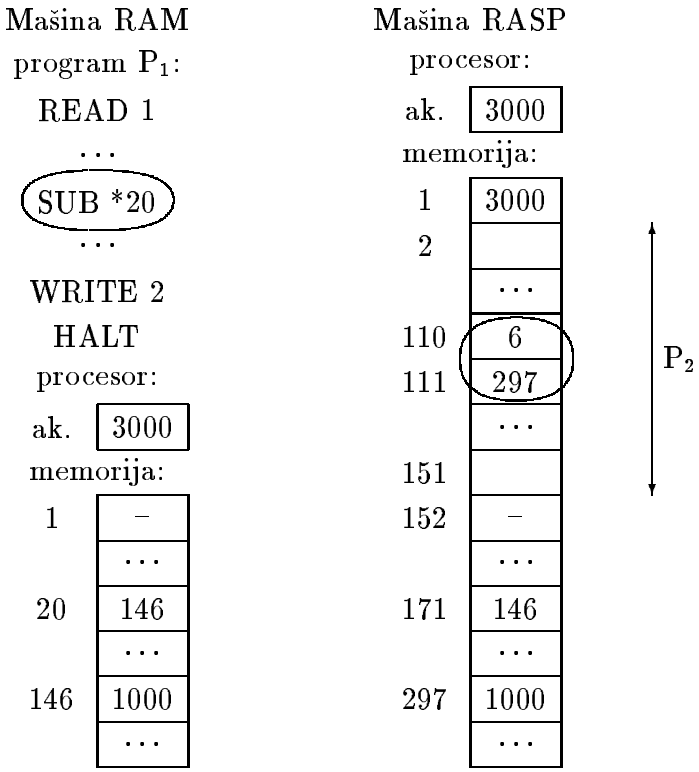
Vidimo da $b = c(111) \leftarrow c(r+i)+r$. Veličina b mijenja se tokom izvršavanja programa. Za naredbu na lokacijama 110—111 kaže se da je tzv. varijabilna naredba.

Objasnimo tabelu. Prvom od šest naredbi, akumulator se pošalje na rezervno mjesto. Zatim se pročita sadržaj i–tog registra (pročita se adresa) i onda se ta adresa plasira na pogodno mjesto. Jedino ne zaboraviti da svaka adresa treba da bude uvećana za r. Pogodnim mjestom smatra se adresno polje naredbe za oduzimanje (u P_2). Znamo da adresno polje u slučaju modela RASP predstavlja drugi od dva uzastopna registra koji pripadaju jednoj naredbi. Petom naredbom (njoj pripadaju 108 i 109) akumulator se vraća sa rezervnog mjesta na tačno mjesto. Vraćamo se paralelnosti. Paralelnost je očito morala da bude napuštena zbog dodatnih računanja u programu P_2. Šestom naredbom (onom sa 110 i 111) izvrši se oduzimanje!

V. u nastavku primjer za SUB *i u kome su sve brojne vrijednosti konkretizovane. Neka bude $r = 151$, tako da P_2 zauzima $2, 3, \ldots, 151$. Neka bude $i = 20$, tako da se u P_1 pojavljuje

SUB *20, čemu odgovara grupa naredbi u P_2. Efektivno, treba izračunati $x - y = 3000 - 1000$. Da ponovimo: na račun naredbe SUB *20 u P_1, imaćemo u P_2 niz od $k = 6$ naredbi.

Konkretan primjer:



Tako da se može reći da je P_2 konstruisan. Odredi se gdje koja naredba programa P_2 ima mjesto. Recimo, u našem primjeru, naredbe koje zamjenjuju SUB *20 imaju mjesto počev od registra 100. Takođe se odredi čemu je jednako r. Slično kao što prevodilac, kada vrši povezivanje, vrši linkovanje nekoliko modula (da bi od njih napravio jednu cjelinu), izračuna fizičke adrese za svaki modul (od – do).

Mi smo pokazali da svaka naredba programa P_1 za RAM može da bude pretvorena u jednu naredbu ili u grupu od šest naredbi u programu P_2 za RASP. Tako da jednom izvršavanju naredbe u slučaju P_1 odgovara najviše šest izvršavanja u slučaju P_2. Tako da smo mi pokazali da je $T_2(n) \leq 6T_1(n)$, u slučaju primjene uniformne cijene.

Slično se pokazuje da važi nejednakost $T_2(n) \leq kT_1(n)$ i kada se primjenjuje logaritamska cijena. Teorema je dokazana.

Tekst programa P_2 mijenja se tokom njegovog izvršavanja uopšte uzev. Prilikom rada sa nizovima, potrebno je da se sadržaj programa mijenja, u slučaju modela RASP. Slično su obradu sa nizovima vršili stari računari, prvi računari sa upisanim programom, računari prve i druge generacije. Znači, program koji se izvršava upisuje razne vrijednosti u razne memorijske lokacije, u tom broju i u memorijske lokacije koje sadrže sami program. Nije pogodno kod ispitivanja ispravnosti programa, tj. kod debagiranja (debug).

Bolje je kada CPU koristi indeks–registre. Primjeri na jeziku asemblera u slučaju Intel: MOV AX, n znači AX $\leftarrow$ n, MOV AX, [n] znači AX $\leftarrow$ c(n), MOV AX, BX znači AX $\leftarrow$ BX, MOV AX, [BX] znači AX $\leftarrow$ c(BX) – ovdje BX ima ulogu indeks–registra,

Teorema 1.2. Svakom programu P_2 za RASP (označimo njegovu vremensku složenost sa $T_2 = T_2(n)$) može da se pridruži program P_1 za RAM koji vrši ekvivalentnu obradu i to takav da postoji konstanta $k > 0$ da je $T_1(n) \leq kT_2(n)$ za svaki prirodan broj n; sa $T_1 = T_1(n)$ označena je vremenska složenost programa P_1. I za RASP i za RAM primijenjena je uniformna cijena. Ili je primijenjena logaritamska cijena za oba programa.

Skica dokaza teoreme. Tokom izvršavanja datog programa P_2 može da dolazi do promjena njegovog sadržaja. Za takve slučajeve, program P_1 koji treba da bude konstruisan koristi mogućnost indirektnog adresiranja kojom model RAM raspolaže. Program P_1 ne zavisi od P_2, tj. P_1 je jedan sasvim određeni program. P_1 je interpreter za ma koji RASP–ov program. P_2 ima ulogu ulaznog podatka za P_1.

Program P_1 pročita jednu naredbu programa za RASP. Zatim se, zavisno od vrste pročitane naredbe, izvrši skok na jedno od 18 mogućih mjesta u P_1, gdje se izvede računanje koje odgovara pročitanoj naredbi. Zatim se pročita iduća naredba programa za RASP, itd. u obliku petlje. Tako da u interpreteru ima jedna velika (kako se u paskalu kaže) "case" naredba sa 18 slučajeva.

Program P_1 upotrebljava memoriju po sljedećem rasporedu: r_1 služi za pripremanje adrese, r_2 služi da čuva RASP–ov LC, a r_3 da čuva RASP–ov akumulator. A ostale registre upotrebljava da preslika RASP–ovu memoriju: RAM–ov registar $i+3$ čuva sadržaj RASP–ovog registra i za $i \geq 1$.

Rad mašine RAM počinje u trenutku kada je u njenu memoriju već upisan (loadovan) program P_2 i to je upisan od registra $i = 4$ pa naprijed. Znači, loadovan je početni sadržaj RASP–ove memorije. Odustajemo od izlaganja punog dokaza teoreme.

U početku je $c(2) = 4$ jer bi P_2 prvo izvršio naredbu iz svog prvog registra (iz svojih prvog i drugog registra). Završena skica dokaza teoreme.

Važe teoreme kao 1.1. i 1.2. gdje umjesto "vremenska" stoji "prostorna".

Pokazali smo da dva modela RAM i RASP imaju jednu te istu oblast djelovanja. Pokazali smo da se složenost dva ekvivalentna programa razlikuje najviše za konstantni faktor, tj. da dvije složenosti imaju jedan te isti red veličine. Takođe: P_1 je polinomske složenosti $\Leftrightarrow$ P_2 je polinomske složenosti.

Ubuduće ćemo više koristiti RAM nego RASP.

1.5. APSTRAKCIJE MAŠINE RAM

U ovom naslovu razmatraju se četiri modela za računanje, od I do IV, koji potiču od modela RAM i koji su jednostavniji od modela RAM. Za pojedini od tih modela, neke karakteristike mašine RAM su sačuvane, a neke su odbačene. Nazivi na engleskom za te modele su redom: I. Straight–line programs, II. Bitwise computations, III. Bit vector operations i IV. Decision trees. Što se prevodi kao: I. Pravolinijski programi, II. Računanja bit–po–bit, III. Operacije sa vektorima bita i IV. Drveta odlučivanja.

I. Pravolinijski programi

U ovom modelu nema naredbi za skokove, pa se kaže da je program slobodan od petlji (loop–free). Takođe je odbačeno indirektno adresiranje. Takođe je isključena naredba READ. Naime, pretpostavlja se da je konačan skup ulaznih veličina (potrebnih za određeni primjerak) već u memoriji kada počinje izvršavanje programa. Slično je i sa naredbom WRITE jer se određeni registri smatraju **izlaznim promjenljivim**, tj. izlazom programa smatraju se vrijednosti tih promjenljivih kada se izvršavanje okonča. Nije potrebna ni naredba HALT jer kraj teksta programa označava zaustavljanje mašine.

Zasad su ostale naredbe LOAD i STORE i četiri aritmetičke naredbe.

Svaki pravolinijski program može da se odnosi samo na konačan broj memorijskih registara. Zato se obično tim registrima daju imena (kao "ime promjenljive"). Dakle, registri se ne numerišu cijelim brojevima, nego se oni pozivaju po svojim tzv. simboličkim adresama.

Sljedeće, nizovi naredbi oblika

LOAD a
ADD b
STORE c

zamjenjuju se naredbom c ← a + b. Tako da je ostalo samo sljedećih pet vrsta naredbi:

x ← y + z
x ← y − z
x ← y · z
x ← y / z
x ← i

Ovdje su x, y i z simboličke adrese (ili promjenljive), a i je konstanta. Pomoću tih pet vrsta naredbi se i pišu programi za ovaj model.

Svakom pravolinijskom programu pridružena su dva skupa promjenljivih: skup ulaznih promjenljivih i skup izlaznih promjenljivih.

Primjer. Sastaviti pravolinijske programe za računanje vrijednosti polinoma n–tog stepena za $n = 1$, $n = 2$ i $n = 3$ (programi P_1, P_2 i P_3). Ulazni podatak programa je vrijednost argumenta polinoma x, a izlazni je nađena vrijednost polinoma p. Još su i koeficijenti $a_0, \ldots, a_n$ ulazni podaci, $p(x) = a_n x^n + \ldots + a_1 x + a_0$. Za računanje vrijednosti polinoma koristiti tzv. Hornerovu šemu:

$n = 3$: $a_3x^3 + a_2x^2 + a_1x + a_0 = ((a_3x + a_2)x + a_1)x + a_0$

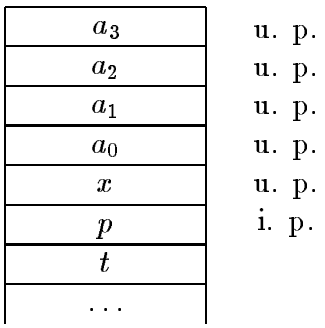
$n = 4$: $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = (((a_4x + a_3)x + a_2)x + a_1)x + a_0$

opšte n: slično

Slijedi rješenje tj. slijede tri programa P_1, P_2 i P_3 koji se odnose redom na slučajeve $n = 1$, $n = 2$ i $n = 3$, gdje je n – stepen polinoma $p = p(x)$:

P_1	$t \leftarrow a_1 \cdot x$	P_2	$t \leftarrow a_2 \cdot x$	P_3	$t \leftarrow a_3 \cdot x$
	$p \leftarrow t + a_0$		$t \leftarrow t + a_1$		$t \leftarrow t + a_2$
			$t \leftarrow t \cdot x$		$t \leftarrow t \cdot x$
			$p \leftarrow t + a_0$		$t \leftarrow t + a_1$
					$t \leftarrow t \cdot x$
					$p \leftarrow t + a_0$

Na slici je prikazan plan upotrebe memorije u slučaju $n = 3$, u. p. – ulazna promjenljiva, i. p. – izlazna promjenljiva.



Time je primjer završen. O složenosti u slučaju modela pravolinijskih programa prvo govorimo služeći se primjerom. Uzmimo da je obim ulaza jednak n, u slučaju da se računa vrijednost polinoma n–tog stepena. Neka je P_n program za računanje vrijednosti polinoma n–tog stepena, gdje je $n \geq 1$. Uvedimo dvije funkcije složenosti izvršavanja $T = T(n)$ i $S = S(n)$, gdje je $T(n)$ vremenska složenost programa P_n, a $S(n)$ je njegova prostorna složenost. Za $T(n)$, bilo koja naredba programa izvrši se tačno jednom. Tako da je $T(n)$ jednako dužini programa. Stavlja se da je $T(n)$ jednako broju izvršenih naredbi kada je obim ulaza jednak n. U našem primjeru je $T(n) = 2n$. Stavlja se da je $S(n)$ jednako broju upotrebljenih memorijskih registara. U našem primjeru je $S(n) = n + 4$.

Slično se definišu vremenska i prostorna složenost bilo kog pravolinijskog programa, tj. niza pravolinijskih programa P_1, P_2, ..., gdje se P_n odnosi na primjerak čiji je obim ulaza jednak n. $T(n)$ je po definiciji jednako broju naredbi programa P_n. A $S(n)$ je po definiciji jednako broju promjenljivih programa P_n.

Ako za recimo vremensku složenost $T(n)$ pravolinijskog programa važi $T(n) = O(f(n))$ kad $n \to \infty$ onda to može da bude zapisano i kao $T(n) = O_A(f(n))$. Slovo A dolazi od riječi engl. arithmetic (aritmetički). Vidjeli smo da su aritmetičke operacije glavna karakteristika razmatranog modela pravolinijskih programa.

Znamo da $T(n) = O(f(n))$ za $n \in N$ znači $(\exists c > 0)\ (\forall n \in N)\ |T(n)| \leq c|f(n)|$. Dok $a_n \sim b_n$ kad $n \to \infty$ znači $\lim_{n\to\infty} a_n/b_n = 1$.

II. Računanja bit–po–bit

Ovaj model sličan je prethodnom modelu, a postoje dvije razlike. (1) Promjenljive više ne uzimaju cijele vrijednosti već sada uzimaju jedino vrijednosti iz skupa $\{0, 1\}$. (2) Umjesto aritmetičkih koriste se logičke operacije. Postoje sljedeće vrste naredbi:

$x \leftarrow y \,\&\, z$, $\quad x \leftarrow y \vee z$, $\quad x \leftarrow y \oplus z$, $\quad x \leftarrow \overline{y}$, $\quad x \leftarrow 0$, $\quad x \leftarrow 1$.

Primjer. Sastaviti program za sabiranje dva binarno prikazana broja, gdje i jedan i drugi broj imaju po dvije binarne cifre (i jedan i drugi sabirak prikazuju se sa po dva bita). Uvedimo potrebne oznake: $(a_1a_0)_2 + (b_1b_0)_2 = (c_2c_1c_0)_2$. Ulazne promjenljive su a_1, a_0, b_1 i b_0, a izlazne promjenljive su c_2, c_1 i c_0.

Poznate su sljedeće formule:

$c_0 = a_0 \oplus b_0$
$c_1 = (a_0 \,\&\, b_0) \oplus a_1 \oplus b_1$
$c_2 = ((a_0 \,\&\, b_0) \,\&\, (a_1 \vee b_1)) \vee (a_1 \,\&\, b_1)$

Slijedi tekst rješenja:

$c_0 \leftarrow a_0 \oplus b_0$
$u \leftarrow a_0 \,\&\, b_0$
$v \leftarrow u \oplus a_1$
$c_1 \leftarrow v \oplus b_1$
$w \leftarrow a_1 \vee b_1$
$x \leftarrow u \,\&\, w$
$y \leftarrow a_1 \,\&\, b_1$
$c_2 \leftarrow x \vee y$

Slijedi izgled memorije:

a_1	a_0	b_1	b_0	c_2	c_1	c_0	u	v	w	x	y	...

Završen primjer.

Vremenska i prostorna složenost $T(n)$ i $S(n)$ definišu se slično prethodnom modelu. Umjesto $T(n) = O(f(n))$ može se pisati $T(n) = O_B(f(n))$ kada se radi o složenosti u slučaju modela računanja bit po bit. Slovo B dolazi od riječi bit.

Vraćajući se na primjer, neka P_n bude program za sabiranje dva binarno prikazana broja koji i jedan i drugi imaju po n binarnih cifara, gdje je $n \geq 1$. Neka upravo broj cifara jednog odnosno drugog sabirka bude dimenzioni broj primjerka n (mi se tako opredjeljujemo). Tako da smo mi maločas napisali program P_2. Prebrojati naredbe i prebrojati registre. Vidimo da je $T(2) = 8$ i da je $S(2) = 12$. Pokazati da je $T(n) = O(n)$ i da je $S(n) = O(n)$ za $n \geq 1$.

A ako se dva binarno prikazana broja više ne sabiraju nego se sada oni množe? Neka se za množenje primjenjuje tzv. obični algoritam koji predviđa uzastopno šiftovanje prvog činioca i paralelno sa tim računanje djelimičnog proizvoda (djelimičnog rezultata), putem uzastopnog sabiranja. Tada je $T(n) = O(n^2)$. Ili se svejedno može pisati da je tada $T(n) = O_B(n^2)$.

Vidimo da se kod predloženog modela vrlo sitničavo mjeri vremenski i prostorni trošak. Model može pogodno da posluži prilikom ispitivanja složenosti algoritama za obavljanje aritmetičkih operacija nad cijelim brojevima i za obavljanje drugih osnovnih operacija (koje se kod drugih modela smatraju polaznim). Druga primjena modela jeste za kombinacione mreže (šeme od funkcionalnih elemenata). Tada je broj koraka $T(n)$ jednak broju logičkih elemenata u kombinacionoj mreži.

III. Operacije sa vektorima bitova

Neka sada promjenljiva bude vektor čije su komponente nule i jedinice. Ne postavlja se ograničenje na broj komponenti jednog vektora, kao što ni u slučaju modela RAM nije bilo postavljeno ograničenje na veličine cijelih brojeva koji se tamo pojavljuju. Nad takvim vektorima mogu da se izvode po–koordinatno operacije $\&$, $\vee$, $\oplus$ i $\neg$, recimo $(a_1 a_2 \ldots a_n) \,\&\, (b_1 b_2 \ldots b_n) = (c_1 c_2 \ldots c_n)$, $c_k = a_k \,\&\, b_k$, $k = 1, 2, \ldots n$. Takođe mogu da se izvode i aritmetičke operacije, smatrajući da takav jedan vektor predstavlja binarni zapis nekog cijelog broja.

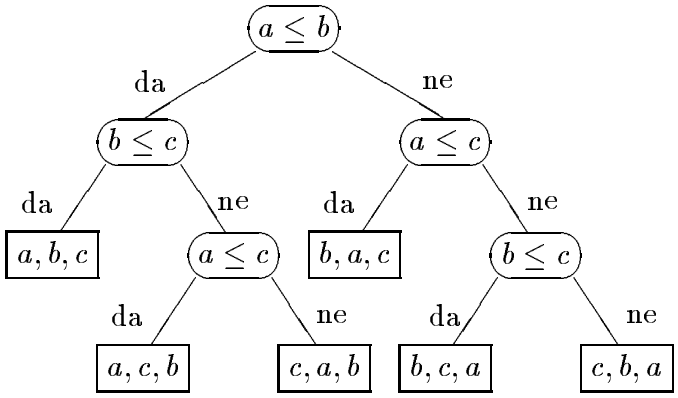
Pišemo $O_{BV}(f(n))$ kada se radi o redu veličine složenosti programa u slučaju ovog modela, BV – bit vector.

Može se dopustiti da se u programima za razmatrani model pojavljuju i obične promjenljive (cijeli brojevi). One mogu da predstavljaju na primjer brojače ili indekse, indeks je recimo promjenljiva `i` u izrazu `niz[i]`.

IV. Drveta odlučivanja

Primjer. Sastaviti program za uređivanje po veličini niza brojeva. Niz se uređuje u rastući oblik (u neopadajući oblik) tj. od najmanjeg člana prema najvećem. Neka dužina niza bude $n = 3$. Ulazni podaci su članovi niza a, b i c. Rezultat je uređeni oblik polaznog niza (učitanog niza).

Rješenje je prikazano na sljedećoj slici (rješenje je prikazano pomoću drveta):



U mnogim programima naredbe za računanje dominiraju po svom broju nad naredbama za grananje (za skokove). U takvim slučajevima pogodna su tri modela od maločas. Međutim, u mnogim programima odnos je suprotan. Tada je pogodno da broj izvršenih naredbi za grananje posluži kao primarna mjera složenosti. Zato je pogodno da se razmotri i model u kome svaki izvršeni korak odražava jedan izbor, jednu odluku, jedno grananje sa dva moguća ishoda. Izbor zavisi od rezultata poređenja dvije vrijednosti.

Obična reprezentacija programa sa grananjem jeste binarno drvo za koje se kaže da je drvo odlučivanja. Svaki unutrašnji vrh drveta odražava jedan izbor. Prvo se izvrši ispitivanje koje se traži u korijenu drveta. Zatim se kontrola predaje jednom od dva nasljednika korijena. Uopšte, kontrola se od strane vrha drveta stalno predaje jednom od dva njegova nasljednika. Pri tome izbor zavisi od odgovora prilikom ispitivanja. Itd. sve dok se drvo grana. Put se završava kada se stigne do nekog spoljašnjeg vrha (do nekog lista). Tamo se nalazi upisan odgovor.

Po definiciji se stavlja da je vremenska složenost $T(n)$ drveta odlučivanja jednaka njegovoj visini izraženoj u zavisnosti od veličine zadatka n. Drugim riječima, zanima nas čemu je jednak najveći mogući broj izvršenih operacija upoređivanja na putu od korijena ka nekom listu. Koristimo oznaku $O_C(f(n))$ kada je riječ o ocjeni složenosti programa u slučaju razmatranog modela. Slovo C dolazi od riječi engl. comparison – poređenje.

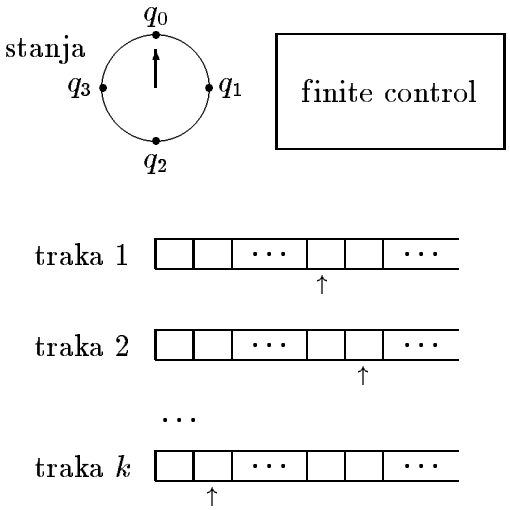
Razmotrimo jedan program za uređivanje (za sortiranje) niza od n brojeva. Neka je upravo $n \ge 1$ veličina zadatka i neka se složenost programa mjeri na razmatrani način. Napominje se da ukupan broj vrhova u drvetu može uveliko da prevazilazi njegovu visinu $T(n)$. Naime, samih listova u drvetu ima $n!$ ili više jer ima tačno $n!$ mogućih odgovora, tj. mogućnosti za uređeni oblik datog niza brojeva $a_1, a_2, \ldots, a_n$. S druge strane, vidjećemo da je dovoljna visina $T(n) \sim n \log_2 n$.

Vraćajući se na primjer, vidimo da je visina drveta $T(3) = 3$.

1.6. PRIMITIVNI MODEL RAČUNANJA – TJURINGOVA MAŠINA

Jednostavnost ili primitivnost osnovnih radnji koje T. m. izvršava treba dovesti u vezu sa sljedećim: teorijski je značajno da se izdvoji jedan najmanji mogući skup radnji za modeliranje algoritama. Sada se pojam T. m. unekoliko generalizuje time što se uvodi u razmatranje tzv. T. m. sa više traka. Kao što i naziv govori, ova mašina se razlikuje od one razmatrane dosad po tome što sada umjesto jedne trake posjeduje izvjesan konačan broj traka k. Broj k je obavezno fiksiran. Ispostaviće se da nema nikakve razlike u pogledu oblasti djelovanja između modela sa jednom trakom i modela sa više traka, a da nema značajnih razlika u pogledu brzine rada.

Definišimo ovaj novi model. T. m. sa nekoliko traka prikazana je na slici. Ona se sastoji od nekoliko traka (od k traka) koje su na desnu stranu beskonačne. Svaka traka je podijeljena na polja od kojih svako sadrži jedan simbol iz jednog konačnog skupa dozvoljenih simbola, taj skup će biti označen kao A. Na svakoj traci se nalazi (iznad jednog njenog polja) glava trake koja je u stanju da čita i piše. Radnje Tjuringove mašine određuje njen (primitivni) program tj. tablica, engl. finite–control. U svakom taktu, program se nalazi u jednom od konačno mnogo mogućih stanja.



Slika: T. m. sa nekoliko traka

U zavisnosti od tekućeg stanja mašine i sadržaja svih k tekućih radnih polja (od onoga što čitaju glave pojedinih traka), mašina će u pojedinom koraku da uradi sljedeće dvije operacije: (1) u tekuće radno polje će upisati novi simbol (preko dotadašnjeg) ili će da pomjeri glavu za jedno mjesto udesno ili ulijevo, nezavisno od trake do trake, po svim trakama i (2) ova mašina promijeni svoje tekuće stanje.

Definicija 1. Tjuringova mašina sa k traka definiše se formalno kao uređena petorka $(Q_s, q_0, A_t, a_0, \varphi)$, gdje je:

$Q = Q_s = \{q_0, q_1, \ldots, q_s\}$ – skup svih mogućih stanja mašine,

$q_0 \in Q_s$ – početno stanje,

$A_t = \{a_1, \ldots, a_t\}$ – skup slova,

$a_0 \notin A_t$ – znak za prazno polje (pa je $A = \{a_0\} \cup A_t$ – skup svih mogućih znakova) i

φ – tablica ili program ili engl. next–move function i to φ: $Q \times A^k \to V^k \times Q$, ovdje je $V = A \cup \{r, l, s\}$ – skup svih mogućih radnji.

Vidimo da smo sa A_t označili skup slova (azbuku mašine), dok A označava tzv. prošireni skup slova.

Poznat je smisao pojedine radnje: a_i – upiši simbol a_i $(0 \leq i \leq t)$, r – pomjeri se udesno, l – pomjeri se ulijevo za jedno mjesto i s – zaustavi se.

Nećemo ponavljati sve detalje koji su u prvoj glavi kompletno izloženi, kao: početna pozicija, mogućnost prelaska iza kraja trake, itd. U slučaju da je $k = 1$ ova mašina se u potpunosti svodi na mašinu koja je bila razmatrana u prvoj glavi.

Ako dođe do zaustavljanja na jednoj traci (bilo zbog s, bilo zbog prelaska iza kraja trake) onda dolazi do zaustavljanja čitave mašine. Drugim riječima, nije moguće da su se neke trake zaustavile a neke druge da još rade.

U slučaju mašine sa jednom trakom, pojedini red tablice je bio oblika $qavq'$, gdje je q – staro stanje, a – sadržaj radnog polja, v – radnja i q' – novo stanje i tablica je imala ukupno $(s+1)(t+1)$ redova. Sada, u slučaju recimo $k = 2$, pojedini red tablice je jedna uređena šestorka $qb_1b_2v_1v_2q'$, gdje je q – sadašnje stanje mašine, b_1 i b_2 – sadržaj radnog polja prve odnosno druge trake, v_1 i v_2 – radnja za prvu odnosno drugu traku i q' – novo stanje mašine. Slično, za opšte k, pojedini red tablice ima oblik $qb_1 \ldots b_k v_1 \ldots v_k q'$.

Znamo da Tjuringova mašina predstavlja najznačajniji model za računanje. Vidimo da se radi o modelu sa spoljašnjim programom. Naime, tablica mašine nije upisana na traci. Zato se tablica ne mijenja tokom izvršavanja.

Na Tjuringovu mašinu sa k traka može se gledati kao na uređaj za računanje vrijednosti funkcije ili kao na uređaj za prepoznavanje jezika. Sljedeća definicija je sasvim slična onoj koja se odnosi na slučaj $k = 1$.

Definicija 2. Neka je T T. m. sa k traka, neka je azbuka $A_t = \{1, \ldots, t\}$ podskup radne azbuke te mašine T i neka je $n \geq 1$. Ova mašina definiše jednu djelimičnu funkciju f: Dom $f \to \Omega(A_t)$, gdje je Dom $f \subset \Omega^n(A_t)$, po sljedećem propisu. Neka je početna pozicija sljedeća: na prvoj traci $]\,0\,w_1\,0\,\ldots\,0\,w_n\underset{\uparrow}{0}\ldots$, na ostalim trakama $]\underset{\uparrow}{0}\ldots$, gdje $w_i \in \Omega(A_t)$ za $1 \leq i \leq k$. Ako se T zaustavlja kroz konačan broj koraka i ako je završna pozicija oblika: na prvoj traci $]\sim 0\,w\underset{\uparrow}{0}\sim$, na ostalim trakama proizvoljno, gdje $w \in \Omega(A_t)$, tada i samo tada n–torka $(w_1, \ldots, w_n) \in$ Dom f i pritom je $f(w_1, \ldots, w_n) = w$.

Za T. m. T sa k traka čija radna azbuka sadrži A_t kaže se da prepoznaje jezik $L \subset \Omega(A_t)$ ako je ispunjeno sljedeće. Kada se ova mašina primijeni na početnu poziciju: na prvoj traci $]\,0\,w\underset{\uparrow}{0}\ldots$, na ostalim trakama $]\underset{\uparrow}{0}\ldots$, gdje $w \in \Omega(A_t)$, onda se ova mašina svakako zaustavlja i to je završna pozicija na prvoj traci oblika $]\sim 0\underset{\uparrow}{0}\sim$ ili $]\sim 0\,1\underset{\uparrow}{0}\sim$ za $w \in L$ odnosno $w \notin L$. Moguć je i drugi dogovor, da dvije završne pozicije budu $]\sim 0\,1\underset{\uparrow}{0}\sim$ i $]\sim 0\underset{\uparrow}{0}\sim$. Ili da budu recimo $]\underset{\uparrow}{1}\sim$ i $]\underset{\uparrow}{0}\sim$. Završna pozicija na ostalim trakama je proizvoljna.

Vremenska i prostorna složenost jednog programa (jedne T. m. T) u slučaju opšteg k definišu se sasvim slično slučaju $k = 1$. Treba definisati dimenzioni broj n i veličine $T(n)$ i $S(n)$.

Definicija 3. Za T. m. T sa k traka, njena veličina ulaza n jednaka je dužini početnog odsječka (prve trake) koji ima svojstvo da su desno od njega sva polja prazna i da obuhvata

radno polje u trenutku puštanja u rad mašine, odnosno jednaka je maksimumu preko svih traka od te dužine.

Za određeno n, vremenskom složenošću $T(n)$ ove mašine naziva se maksimum, po svim ulazima veličine n, broja izvršenih koraka. Ako za neki ulaz čiji je obim n mašina radi vječno onda se stavlja da je $T(n) = +\infty$ za to n.

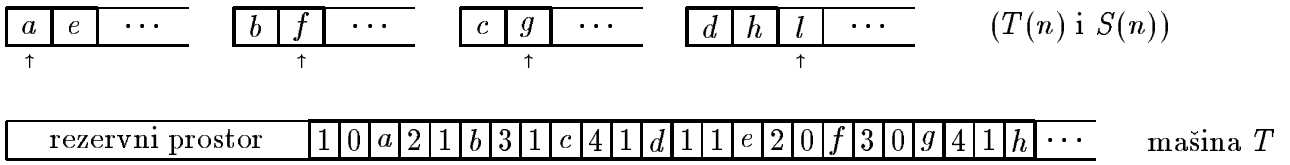
Prostornom složenošću $S(n)$ naziva se maksimum, po svim ulazima dimenzije n i po svim trakama, dužine iskorišćenog odsječka (tj. broja polja koja su bila upotrebljena) na pojedinoj traci, u smislu: koliko se tokom cijelog računanja glava najviše udaljila od lijevog kraja svoje trake. Ako postoji ulaz veličine n takav da se tokom rada mašine na bar jednoj traci glava te trake pomijera neograničeno udesno onda se stavlja da je $S(n) = +\infty$ za to n.

Kakav je odnos između T. m. sa jednom trakom i mašine sa više traka u pogledu oblasti djelovanja i u pogledu brzine obrade?

U jednom smjeru, ovo pitanje je jednostavno. Neka imamo jednu mašinu sa samo jednom trakom koja vrši neku obradu (računa funkciju ili prihvata jezik). Ona ima svoje pokazatelje složenosti $T(n)$ i $S(n)$. Da li postoji mašina sa k traka koja vrši tu istu obradu? Očito je da postoji, ta nova mašina prosto neće koristiti svojih $k-1$ traka. Isto tako, složenost ove nove mašine (vremenska i prostorna) će biti manja ili jednaka od složenosti polazne. Pogledajmo drugi smjer postavljenog pitanja. Sada se polazi od neke mašine sa k traka. Želimo da konstruišemo mašinu T sa jednom trakom koja će izvršiti ekvivalentnu obradu. Dajemo opis mašine T, barem u glavnim crtama.

Plan za memoriju mašine T. Smatrajmo da je njena memorija podijeljena na grupe od po $3k$ polja. U prvoj grupi, prva tri polja odnose se na prvo polje prve trake polazne mašine, sljedeća tri na prvo polje druge trake polazne mašine, itd. Što se tiče pojedine trojke polja, ona sadrže redom: broj i (odgovarajuće polje je sa i–te trake), broj 0 ili 1 (to polje p je tamo radno ili nije radno) i sami sadržaj polja p. Ove grupe od po $3k$ polja ne počinju od sanog lijevog kraja trake već se između početka trake i njihovog početka ostavi jedan rezervni prostor konstantne veličine za dopunske radnje.

> Na primjer, kada je $k = 4$. Prvi red odnosi se na polaznu mašinu sa četiri trake, ona ima svoje pokazatelje $T(n)$ i $S(n)$. Drugi red odnosi se na novu mašinu T sa jednom trakom, ona ima svoje pokazatelje složenosti:



Tablica polazne mašine je svakako data. Kako sastaviti tablicu za mašinu T? Pojedini red polazne tablice se lako pretvara u k redova nove tablice ("osnovni dio" nove tablice). Međutim, tablica koja se formira ovim nije završena nego njoj treba da se doda još redova ("dopunski dio" nove tablice) koji se odnose na pomijeranja glave ulijevo ili udesno. Zapažamo da će tih pomijeranja biti znatno više nego kod polazne mašine, u vezi drukčijeg rasporeda memorije.

Tako smo dobili mašinu T koja simulira (ili modelira) polaznu mašinu. Označimo sa $T(n)$ i $S(n)$ redom vremensku i prostornu složenost polazne mašine. Kolika je složenost računanja

nove mašine T? Prethodno, znamo da je uvijek $S(n) \leq T(n)$ jer se na svako upotrebljeno polje odnosi bar jedna izvršena naredba.

Što se tiče prostora koji je potreban mašini T, lako se vidi da je on $3k$ puta veći od $S(n)$ plus jedan konstantan sabirak, znači $3kS(n) + \text{const}$.

Nešto više posla ima oko ocjenjivanja sa gornje strane vremena koje T zahtijeva. Na račun izvršavanja "osnovnog dijela" svoje tablice ona učini broj koraka koji je $\leq cT(n)$. Između svaka dva takva koraka, ona učini i izvjesan broj pomijeranja glave svoje trake ("dopunski dio" tablice), s tim da je taj broj manji ili jednak od njene prostorne složenosti, eventualno pomnožene sa 2. Zaista, najviše se desi da se glava vrati do početka trake a onda još ode udesno do odgovarajućeg polja. Tako da za vremensku složenost koja se ocjenjuje sa gornje strane važi

$$\leq 2(3kS(n) + \text{const})cT(n) \leq 2(3kT(n) + \text{const})cT(n).$$

U svakom slučaju, ta vremenska složenost jednaka je $O(T^2(n))$.

Slijede formulacije dvije već dokazane teoreme.

Teorema 1. Za svaku Tjuringovu mašinu sa nekoliko traka postoji T. m. T sa samo jednom trakom koja vrši ekvivalentnu obradu.

Teorema 2. Razmotrimo jednu Tjuringovu mašinu sa nekoliko traka i označimo njene funkcije vremenske i prostorne složenosti redom sa $T = T(n)$ i $S = S(n)$. Neka je T njoj odgovarajuća mašina sa jednom trakom (T vrši ekvivalentnu obradu). Tada je vremenska složenost mašine $T \quad \leq cT^2(n)$ (c – konstanta), a njena prostorna složenost je $\leq cS(n)$ (c – konstanta).

Zapazimo sljedeće: ako je $T(n)$ polinom onda je i $cT^2(n)$ polinom. Drugim riječima, ako neki proces obrade može da se modelira pomoću mašine sa k traka čija je složenost polinomska onda postoji i T. m. sa jednom trakom koja vrši tu istu obradu i koja takođe ima polinomsku složenost (takođe je efikasna).

Dajemo jednu definiciju.

Definicija 4. Kažemo da su funkcije $f_1 = f_1(n)$ i $f_2 = f_2(n)$ polinomski povezane ako postoje polinomi $p_1 = p_1(n)$ i $p_2 = p_2(n)$ takvi da važi: $f_1(n) \leq p_1(f_2(n))$ i $f_2(n) \leq p_2(f_1(n))$ za $n \geq n_0$.

Vidimo da su složenosti računanja na Tjuringovim mašinama sa jednom trakom i sa nekoliko traka – polinomski povezane. Možemo reći da su ove dvije mašine ekvivalentne (da su ova dva modela ekvivalentna) ako nas interesuje samo sljedeće: da li se određeni proces obrade (da li se određeni algoritam) na toj mašini modelira sa polinomskom složenošću.

1.7. UZAJAMNI ODNOS MODELA RAM I TJURINGOVE MAŠINE

Teorema 1. Svakom programu za RAM koji ne sadrži množenja i dijeljenja i čija je vremenska složenost po logaritamskoj cijeni označena kao $T(n)$, može da se pridruži ekvivalentan program za Tjuringovu mašinu čija je vremenska složenost najviše $O(T^2(n))$.

Skica dikaza. Prethodno, dimenzioni broj n za jedan i drugi program definisani su ravnopravno. To znači da ulazni podaci zauzimaju prvih n kvadrata ulazne trake mašine RAM i da se njihov sadržaj poklapa sa početnim sadržajem prvih n polja jedne trake Tjuringove mašine. Zato, budući da je azbuka Tjuringove mašine uvijek jedan konačan skup, pojedini kvadrat ulazne trake u ovom slučaju ne može da sadrži proizvoljno velik cio broj. Dalje, program za Tjuringovu mašinu koji treba da bude konstruisan odnosi se na jednu Tjuringovu mašinu sa pet traka. Opišimo ukratko smisao pojedine trake. Prva traka odgovara registrima r_n $(n \geq 1)$ mašine RAM. Predstavljeni su svi registri mašine RAM koji ne sadrže nulu. Brojevi su zapisani u binarnom obliku bez vodećih nula. Redom su prikazani brojevi i_1, c_1, i_2, c_2, itd. razdvojeni pomoću posebnog znaka. Ovdje je c_1 sadržaj RAM–ovog registra i_1, itd. Na sljedećoj slici prikazan je izgled trake (b znači blanko):

X	X	i_1	X	c_1	X	X	i_2	X	c_2	X	X	$\cdots$	X	X	i_m	X	c_m	X	X	b	$\cdots$	$\cdots$

Na drugoj traci upisan je u binarnom obliku sadržaj RAM–ovog akumulatora $r_0 = AC$. Treća traka se koristi kao rezervna memorija. Četvrta i peta traka služe za ulaz i izlaz mašine RAM.

Teorema 2. Vremenska složenost programa za RAM po logaritamskoj cijeni i vremenska složenost ekvivalentnog programa za Tjuringovu mašinu (sa jednom trakom ili sa k traka) su dvije polinomski povezane funkcije.

Iz dokaza. Ovdje je dopušteno da program za RAM sadrži množenja. Tjuringova mašina realizuje množenje kao uzastopno sabiranje. A i dijeljenja su dopuštena. Tjuringova mašina naravno realizuje i dijeljenja. Ako $T(n)$ označava složenost programa za RAM onda će složenost programa za Tjuringovu mašinu biti $O(T^3(n))$.

Slični iskazi važe i za prostornu složenost.

Ako se za RAM umjesto logaritamske cijene primjenjuje uniformna cijena onda analogan iskaz ne važi. Ovo se dokazuje sljedećim primjerom. Pođimo od prirodnog broja $a \neq 1$. Izaberimo $a = 2$ i neka treba sastaviti program čiji je ulazni podatak prirodan broj x a koji računa i štampa $y = a^x$. Označimo sa n broj bita u binarnom zapisu broja x, tako da je $n \approx \log_2 x$. Uzmimo da upravo n predstavlja dimenzioni broj. Ulazna traka mašine RAM:

b_1	$\cdots$	b_n	X	$\cdots$

, gdje je $x = (b_1 \ldots b_n)_2$, $b_i \in \{0, 1\}$ za $1 \leq i \leq n$. Početna pozucija Tjuringove mašine:

b_1	$\cdots$	b_n	X	$\cdots$

(isto što i tamo). Recimo, ako je $x = 101$ onda $x = (1100101)_2$, $n = 7$, $\log_2 x = 6{,}7$. Pogledajmo program za RAM. Vrše se uzastopna kvadriranja $a \cdot a = a^2$, $a^2 \cdot a^2 = a^4$, $a^4 \cdot a^4 = a^8$, itd. Zatim se množenjem odabranih među–rezultata stiže do rezultata y. Npr. ako je $x = 101$ onda je $y = a^{64} \cdot a^{32} \cdot a^4 \cdot a$. Složenost je $O(n)$ po uniformnoj cijeni. Pogledajmo program za Tjuringovu mašinu. Koliko taktova treba za samo štampanje rezultata? Rezultat ima približno 2^n binarnih cifara. Tako da treba približno 2^n taktova kada se prikazuje binarno. Prema tome, složenost programa za Tjuringovu mašinu je $\geq 2^n$ ili je $\geq c \cdot 2^n$. Uporediti dvije funkcije složenosti $T_1(n) = O(n)$ i $T_2(n) \geq c \cdot 2^n$. Lako se vidi da te dvije funkcije nisu polinomski povezane. Ako se brojevi zapisuju dekadno ili slično onda važe slične okolnosti.

1.8. NESTROGI PASKAL – PROGRAMSKI JEZIK VISOKOG NIVOA

Za zapisivanje algoritama koristićemo paskalu blizak jezik – nestrogi paskal, engl. pidgin algol. Recimo unaprijed da je nestrogi paskal nešto višeg nivoa od paskala. Ako se koristi jezik niskog nivoa onda je zapis obiman i teško razumljiv. Zato je bolje da jezik bude što višeg nivoa, samo da smisao zapisa ostane sasvim određen, precizan. Program napisan na nestrogom paskalu se lako, odnosno direktno prevodi u program za RAM. Nećemo imati nikakvih teškoća da razumijemo program napisan na nestrogom paskalu. Recimo samo ukratko o ovom jeziku.

U nestrogom paskalu, dopuštene su konstrukcije kao "neka x i y zamijene mjesta" ili kao "neka je a najmanji član skupa S" ili "označimo sa N_2 niz pozitivnih članova niza N_1" i slično.

O primopredaji parametara potprograma u nestrogom paskalu. Stvarni parametar se predaje potprogramu po referenci (po adresi, engl. by reference) ako je taj stvarni parametar – promjenljiva. A ako je stvarni parametar – izraz (specijalno, ako je – konstanta) onda se predaje naravno po vrijednosti (engl. by value). Drugim riječima, kad god je to moguće, predaja se vrši po referenci. Vidimo da se radi kao u nekadašnjem programskom jeziku fortranu.

Programiranje na jeziku asemblera, model M

Da ilustrujemo kroz primjere sastavljanje programa na nivou jezika asemblera, uvedimo mašinu M koja je slična modelu RASP, jedino što još ima od modela RAM pozajmljenu mogućnost indirektnog adresiranja. Mašina M dobro odgovara mogućnostima jezika asemblera računara, uporediti sa jezikom asemblera za Intelove procesore.

Prvo se definiše mašina M a zatim se navodi niz primjera programa.

Što se tiče definicije, samo je proširen spisak vrsta naredbi a sve ostalo je isto kao u slučaju mašine RASP. Dakle, jedna naredba zauzima dvije uzastopne memorijske lokacije u obliku kod operacije + adresa. Samo će biti nabrojani mogući kodovi operacija, kako slijedi. Trebalo bi svakom opkodu pridružiti brojnu vrijednost, recimo umjesto LOAD=, LOAD, itd. pisati redom 1, 2, itd. Drugim riječima, trebalo bi sa nivoa jezika asemblera (simboličkog jezika) da se svede na niži nivo, upravo na nivo mašinskog jezika. AC znači sadržaj akumulatora tj. c(0), a LC – location counter – znači sadržaj programskog brojača. Sa c(i) označavamo sadržaj adrese i.

Naredba	**Smisao**				
LOAD=i	AC ← i	LOAD i	AC ← c(i)	LOAD∗ i	AC ← c(c(i))
		STORE i	c(i) ← AC	STORE∗ i	c(c(i)) ← AC
ADD=i	AC ← AC + i	ADD i	AC ← AC + c(i)	ADD∗ i	AC ← AC + c(c(i))
SUB=i	AC ← AC − i	SUB i	AC ← AC − c(i)	SUB∗ i	AC ← AC − c(c(i))
MULT=i	AC ← AC · i	MULT i	AC ← AC · c(i)	MULT∗ i	AC ← AC · c(c(i))
DIV=i	AC ← AC / i	DIV i	AC ← AC / c(i)	DIV∗ i	AC ← AC / c(c(i))
		READ i	c(i) ← input	READ∗ i	c(c(i)) ← input
WRITE=i	output ← i	WRITE i	output ← c(i)	WRITE∗ i	output ← c(c(i))

JUMP i	LC ← i	JUMP∗ i	LC ← c(i)
JGTZ i	if AC > 0 then LC ← i	JGTZ∗ i	if AC > 0 then LC ← c(i)
JZERO i	if AC = 0 then LC ← i	JZERO∗ i	if AC = 0 then LC ← c(i)
HALT –	stop		

U programima će biti zapisano na koje dvije lokacije je pojedina naredba upisana. Takođe će biti naznačeno od koje lokacije (od koje adrese) počinje izvršavanje programa ("enter").

Primjer 1. Obični program

Sastaviti program za mašinu M koji učitava dvije vrijednosti a i b i računa i štampa vrijednost izraza $y = a^2 + a \cdot b + 4$.

Rješenje je prikazano u obliku tabele. Dakle, na adresi 1 piše READ (tj. pridružena brojna vrijednost), na adresi 2 piše 23, itd. U produžetku je u istom redu napisan komentar. Vidi se da izvršavanje programa počinje od adrese 1 (od adresa 1, 2) tj. od naredbe READ 23.

Iz rješenja vidimo da adrese od 1 do 22 služe da sadrže naredbe (da sadrže program), a adrese od 23 do 25 služe da sadrže podatke (a, b i a^2).

Ulazna traka treba da bude pripremljena kao | a | b | ... |, a u rezultatu izvršavanja programa dobiće se izlazna traka oblika | y | ... |. Recimo, ulaznoj traci | 10 | 20 | odgovara izlazna traka | 304 |.

enter → 1, 2	READ	23	$c(23) \leftarrow a$	
3, 4	READ	24	$c(24) \leftarrow b$	
5, 6	LOAD	23	$AC \leftarrow a$	
7, 8	MULT	23	$AC \leftarrow a \cdot a$	
9, 10	STORE	25	$c(25) \leftarrow AC$	
11, 12	LOAD	23	$AC \leftarrow a$	
13, 14	MULT	24	$AC \leftarrow a \cdot b$	
15, 16	ADD	25	$AC \leftarrow a \cdot a + a \cdot b$	
17, 18	ADD=	4	$AC \leftarrow a \cdot a + a \cdot b + 4$	
19, 20	WRITE	0	print AC	
21, 22	HALT	–	stop	

23	a
24	b
25	a^2

U prvoj fazi rada mašine, program se loaduje (upiše se sadržaj na adrese od 1 do 22), a onda se poslije toga u drugoj fazi rada mašine program izvršava.

1.9. Rad sa nizom (model M)

Primjer 2. Rad sa nizom

Učitati 10 brojeva sa ulazne trake i štampati zbir prvog i posljednjeg broja $x_1 + x_{10}$.

Rješenje. Neka se ulazni podaci postepeno upisuju na adrese od 101 do 110 redom. Neka adresa 100 posluži kao indeks–registar. Početni sadržaj te adrese treba da bude jednak 101. Sadržaj te adrese postepeno se povećava za po jedan. Nakon pojedinog učitavanja provjerava se sadržaj te adrese i izlazi se iz petlje kada se ispostavi da je sadržaj dostigao vrijednost 110.

Ulaznoj traci primjera radi | 4 | 4 | 7 | 7 | 10 | 10 | 13 | 13 | 16 | 16 | treba da odgovara izlazna traka | 20 |. Druga faza rada mašine tj. faza izvršavanja programa ("enter") treba da počne od adrese 1 (enter = 1). Radi bolje preglednosti, stavljene su strelice ispred naredbi na koje se vrši skok.

enter → 1, 2	LOAD=	101	$AC \leftarrow 101$	
3, 4	STORE	100	$c(100) \leftarrow AC$	
→ 5, 6	READ*	100	$c(c(100)) \leftarrow$ input	
7, 8	LOAD	100	$AC \leftarrow c(100)$	
9, 10	SUB=	110	$AC \leftarrow AC - 110$	
11, 12	JZERO	21	if $AC = 0$ then goto 21	
13, 14	LOAD	100	$AC \leftarrow c(100)$	
15, 16	ADD=	1	$AC \leftarrow AC + 1$	
17, 18	STORE	100	$c(100) \leftarrow AC$	
19, 20	JUMP	5	goto 5	
→ 21, 22	LOAD	101	$AC \leftarrow c(101)$	
23, 24	ADD	110	$AC \leftarrow AC + c(110)$	
25, 26	WRITE	0	output $\leftarrow AC$	
27, 28	HALT	–	stop	

100	101, ..., 110
101	4
102	4
103	7
104	7
105	10
106	10
107	13
108	13
109	16
110	16

1.10. Rad sa potprogramom (model M)

Primjer 3. Rad sa potprogramom

Ovaj primjer služi da se ilustruje običan slučaj upotrebe programa koji koristi potprogram. Da zadatak ne bi bio nepotrebno opterećen, uzeto je da je glavni program M (main

program) jednostavan i da je potprogram S (subroutine) jednostavan, da bi se lakše vidjele glavne okolnosti. Treba predvidjeti mehanizme za: predaju upravljanja S–u od strane M (skok na potprogram), predaju upravljanja u suprotnom smjeru (povratak iz potprograma), predaju parametara potprogramu i predaju rezultata programu. Neka se u primjeru pojavljuje samo jedan potprogram i neka se potprogram poziva na dva mjesta. Neka S računa $y = a^2 + b^2$, tako da ima dva ulazna parametra i jedan izlazni parametar.

Sastaviti program za mašinu M za računanje vrijednosti izraza $y = (a^2 + b^2) - (c^2 + d^2)$, gdje su a, b, c i d ulazni podaci programa i gdje se zbir kvadrata u jednom i drugom slučaju računa u okviru potprograma. Ulaznoj traci | a | b | c | d | treba da odgovara izlazna traka | y |.

Rješenje. Prvo treba napraviti plan upotrebe memorije. Neka za potrebe M služe adrese od 1 do 100, u smislu za potrebe njegovih naredbi i njegovih promjenljivih. Neka za potrebe S služe adrese od 101 do 200 (adrese od 101 pa naprijed).

enter →	1, 2	READ	101	c(101) = a
	3, 4	READ	102	c(102) = b
	5, 6	LOAD=	11	AC ← 11
	7, 8	STORE	104	c(104) = 11
	9, 10	JUMP	105	LC ← 105
→	11, 12	LOAD	103	AC ← $a^2 + b^2$
	13, 14	STORE	80	c(80) ← $a^2 + b^2$
	15, 16	READ	101	c(101) = c
	17, 18	READ	102	c(102) = d
	19, 20	LOAD=	25	AC ← 25
	21, 22	STORE	104	c(104) = 25
	23, 24	JUMP	105	LC ← 105
→	25, 26	LOAD	103	AC ← $c^2 + d^2$
	27, 28	STORE	81	c(81) ← $c^2 + d^2$
	29, 30	LOAD	80	AC ← c(80)
	31, 32	SUB	81	AC ← AC − c(81)
	33, 34	STORE	82	c(82) ← y
	35, 36	WRITE	0	output ← AC
	37, 38	HALT	0	stop

→	105, 106	LOAD	101	AC ← c(101)
	107, 108	MULT	101	AC ← AC · c(101)
	109, 110	STORE	125	c(125) ← AC
	111, 112	LOAD	102	AC ← c(102)
	113, 114	MULT	102	AC ← AC · c(102)
	115, 116	STORE	126	c(126) ← AC
	117, 118	LOAD	125	AC ← c(125)
	119, 120	ADD	126	AC ← AC + c(126)
	121, 122	STORE	103	c(103) ← AC
	123, 124	JUMP*	104	goto c(104)

Memory:

1 – 38	program code M
80 – 82	working space M
100 – 104	working space M and S
105 – 124	program code S
125 – 126	working space S

80	$a^2 + b^2$
81	$c^2 + d^2$
82	y

101	first number
102	second number
103	sum of squares
104	return address

125	squared first
126	squared second

O komunikaciji dva modula M i S. Adresa 101 služi za prvi ulazni parametar potprograma, 102 za drugi, 103 za izlazni parametar i 104 za povratnu adresu. Dakle, M će upisati sadržaj na 101, 102 i 104 prije skoka na potprogram. Isto tako, M očekuje da na 103 ima zapisan rezultat

potprograma, po povratku. Prvoj naredbi potprograma neka pripada adresa 105 (adrese 105–106). Dakle, skok u potprogram ostvaruje se pomoću naredbe JUMP 105.

Posljednja napisana naredba ima opkod JUMP* tj. ona očito predstavlja indirektni bezuslovni skok. Vidi se da lokacije od 80 do 82 čine radni prostor za M. Vidi se da lokacije 125 i 126 čine radni prostor za S. Na slici je prikazan radni prostor (80–82) koji koristi glavni program.

Na slici je prikazan radni prostor (101–104 i 125–126) koji koristi potprogram.

U prvoj fazi rada mašine obavi se loadovanje (loader obavi loadovanje), kako naredbi glavnog programa tako isto naravno i naredbi potprograma. Napominje se da se pomoću loadera može upisati početni sadržaj i u adrese koje se odnose na podatke (na promjenljive). Druga faza rada mašine tj. faza izvršavanja programa treba da počne sa LC = 1 (enter = 1).

Bazna adresa potprograma

Šta može da se popravi? Za prvu adresu u memoriji koja se tiče potprograma može se reći da je bazna adresa potprograma, to je adresa 101. Obično se stavlja da je bazna adresa za jedan niža, tako da je A = 100 jer je tako pogodnije za rad i izražavanje. Zapisati (u glavnom programu) vrijednost A = 100 na neku lokaciju i onda svuda zamijeniti

101, ..., 105 sa A + 1, ..., A + 5,

izvršiti odgovarajuće prepravke u tekstu programa M. Šta se dobija pomoću ove promjene? Vidimo da položaj potprograma u memoriji utiče samo na jednom mjestu na tekst glavnog programa. Dakle, ako se desi da je potprogram zbog nečeg drukčije lociran onda u tekstu glavnog programa samo jedna vrijednost treba da bude prilagođena. Zašto se potprogram premješta?

Tačan položaj potprograma saznaje se tek u fazi povezivanja (link) raznih djelova tj. raznih modula. Naime, položaj zavisi od veličine glavnog programa sa kojim potprogram sarađuje i od eventualnog prisustva i drugih potprograma. Isto tako, moguće je da u memoriji postoje zajedničke zone za podatke ili da postoje stalno prisutni moduli, obično na fiksiranim adresama, moduli koji su "stariji" po hijerarhiji tj. moduli operativnog sistema. Dakle, unutrašnja memorija računara je jedna jedina a ima nekoliko njenih potrošača. Za radnju premještanja potprograma (promjena vrijednosti A) upotrebljava se riječ relokacija.

Relokacija potprograma

U slučaju relokacije našeg potprograma S, adresno polje u svakoj naredbi potprograma bi bilo uvećano odnosno umanjeno za jedan te isti broj ΔA.

Zapaziti da prilikom relokacije u opštem slučaju ne treba dirati adresno polje u naredbama čiji je opkod LOAD= ili ADD= ili slično (slučaj neposrednog ili bukvalnog operanda).

Znamo da je funkcija loadovanja jedna od temeljnih funkcija bilo kog operativnog sistema. Tokom svog rada, loader jednostavno popunjava unutrašnju memoriju (dio unutrašnje memorije), preko ulazne jedinice ili iz spoljašnje memorije, on prepisuje na predviđene adrese. Loader ne razlikuje da li unosi podatke ili naredbe, njega ne interesuje smisao tih naredbi, on ne gleda imaju li one ispravne opkodove i slično. Za jednostavni loader kaže se da je **apsolutni loader**: takvom loaderu saopštavaju se fizičke adrese (od–do) u unutrašnjoj memoriji računara na koje treba da upiše dati modul (M ili S ili slično). Po pravilu, loader vrši i relokaciju. Naravno da mu treba reći za koliko adresa (ΔA) treba da pomjeri pojedini modul.

Primjer 4. Rad sa potprogramom (nastavak)

Navedimo neke tipične mogućnosti. Sami sastavite po jedan primjer za svaku od mogućnosti.

Ako potprogramu treba samo jedan ulazni parametar x i ako potprogram vraća samo jednu izlaznu vrijednost y onda AC (procesorov registar) može da se upotrebi za primopredaju.

Moguće je da se ulazni i izlazni parametri potprograma drže na fiksiranim adresama, tj. da se primopredaja parametara obavlja preko fiksiranih adresa (a da naredbe potprograma i lokalne promjenljive potprograma podliježu relokaciji).

Moguće je da se čitav potprogram drži na fiksiranom mjestu u memoriji. Takav potprogram liči na rutinu za opsluživanje softverskog prekida (softverskog interapta).

1.11. Pojam loadera (model M)

Primjer 5. Loader ili program koji vrši loadovanje

Sastaviti program za loadovanje L. Sastaviti program za mašinu M koji vrši loadovanje sa ulazne trake (u unutrašnju memoriju), u jednostavnom slučaju: loaduje se jedan modul i to se ne vrši relokacija.

Treba se unaprijed dobro razabrati po djelovima memorije. U nastavku je dato rješenje, tj. dat je tekst programa L, s tim da L zauzima od 1 do najviše možda 100 za svoje naredbe i radni prostor (promjenljive). Tako da ono što se tokom rada programa L upisuje u memoriju nikako ne smije da se tiče adresa 1–100. Drugim riječima, izvršni program (aplikativni program) koji predstavlja ulazni podatak programa L smije da počinje od adrese $n+1 = 101$ najniže.

Loaderu svakako treba pripremiti sadržaj kojim se želi popuniti memorija (naredbe koje treba da budu unesene u memoriju). U okviru toga i oznaka kraja: neka 0 služi kao oznaka kraja. Pored toga, loaderu prvo treba saopštiti baznu adresu za unošenje n. Tako da će prva naredba biti unesena na $n+1$, $n+2$. Iduća naredba na $n+3$, $n+4$. Itd. Najzad, na kraju svog rada L treba da preda štafetu. Drugim riječima, mašina treba da nastavi rad od odgovarajućeg mjesta enterp u maločas loadovanom programu. Izvršiće se jedan bezuslovni skok na enterp. Po pravilu je enterp $= n+1$ ali nije obavezno. Vrijednost enterp neka bude upisana u kvadratu ulazne trake koji dolazi neposredno iza kvadrata u kome piše oznaka kraja 0. Tek sada možemo reći da je postavka zadatka zaokružena.

Tako da ulazna traka treba da ima oblik

n	o_1	a_1	o_2	a_2	...	o_k	a_k	0	enterp

,

o od opcode, a od address, o_i i a_i su cijeli brojevi. Posebno važi $o_i \geq 1$, tako da se o_i razlikuje od nule (od oznake kraja). Engleski operation code (opcode), u prevodu kod operacije (opkod).

Pogledajmo na primjeru. Neka treba da bude loadovan program P koji se sastoji od tri naredbe i to prva LOAD= 10, druga LOAD= 20 i treća LOAD 107, tj. prva 1 10, druga 1 20 i treća 2 107 i to naredbe treba da zauzmu adrese od 101 do 106. I da nakon toga izvršavanje programa P počne od njegove prve naredbe. Za taj primjer, ulaznu traku treba pripremiti kao

100	LOAD=	10	LOAD=	20	LOAD	107	0	101

ili bolje reći kao

100	1	10	1	20	2	107	0	101

.

Ponovo, od loadera se u primjeru očekuje da obavi:

$c(101) \leftarrow 1$, $c(102) \leftarrow 10$, $c(103) \leftarrow 1$, $c(104) \leftarrow 20$, $c(105) \leftarrow 2$, $c(106) \leftarrow 107$ i $LC \leftarrow 101$.

Drugim riječima, u rezultatu rada loadera u memoriji treba da bude:

registar	sadržaj	smisao
101	1	LOAD=
102	10	10

103	1	LOAD=
104	20	20
105	2	LOAD
106	107	107

a programskom brojaču nakon toga treba da bude dodijeljena vrijednost 101.

Rješenje. Na jeziku nestrogi paskal rješenje glasi kako slijedi (isto je pisali `<--` ili ←):

```
        read n;
again:  n <-- n+1;
        read opcode;
        if opcode=0 then goto relay;
        c(n) <-- opcode;
        n <-- n+1;
        read address;
        c(n) <-- address;
        goto again;
relay:  read enterp;
        LC <-- enterp;
```

Slijedi tekst rješenja postavljenog zadatka. Ulazna tačka za loader L je enter = 1, prilikom uključivanja mašine važi LC = 1. Raspored promjenljivih i konstanti (raspored radnog prostora) koji L koristi je sljedeći: konstanta 1 na adresi 80, n na 81, tekuće vrijednosti opcode i address na 82 odnosno 83 i ulazna tačka loadovanog programa P (tačka od koje će početi izvršavanje programa P) na 84.

Na slici je prikazan radni prostor koji loader L koristi. Očito je da entry point interveniše u trenutku kada se iz prve faze rada mašine (učitavanje programa P) prelazi u drugu fazu rada mašine (izvršavanje programa P). Na redu je tekst programa L:

enter → 1, 2	LOAD=	1	AC ← 1
3, 4	STORE	80	c(80) ← AC
5, 6	READ	81	c(81) ← n
→ 7, 8	LOAD	81	AC ← c(81)
9, 10	ADD	80	AC ← AC + c(80)
11, 12	STORE	81	c(81) ← AC
13, 14	READ	82	c(82) ← opcode
15, 16	LOAD	82	AC ← c(82)
17, 18	JZERO	35	if 0 then goto 35 (relay)
19, 20	STORE*	81	c(c(81)) ← AC
21, 22	LOAD	81	AC ← c(81)
23, 24	ADD	80	AC ← AC + c(80)
25, 26	STORE	81	c(81) ← AC
27, 28	READ	83	c(83) ← address
29, 30	LOAD	83	AC ← c(83)
31, 32	STORE*	81	c(c(81)) ← AC
33, 34	JUMP	7	goto 7 (again)
→ 35, 36	READ	84	c(84) ← enterp
37, 38	JUMP*	84	goto c(84)

Working space L:

80	1
81	n
82	opcode
83	address
84	entry point

Memory:

1 – 38	**program code L**
80 – 84	**working space L**
101 – 106	program code P
107	working space P

General layout of memory:

1 – 100	**system**
101 – 1024	applications

(initially, LC = 1)

Prema tome, gledano po vremenskoj osi, u memoriji se odvijaju sljedeći događaji:

- u trenutku uključivanja mašine, L je upisan u memoriji;
- tokom rada sistemskog programa L, u memoriju se unose naredbe programa P;
- izvršavanjem naredbe JUMP* 84 tj. goto entry point prelazi se iz prve u drugu fazu;
- tokom rada aplikativnog programa P, dešavanja zavise od samog tog programa P i
- u trenutku kada se P zaustavi (HALT) imamo završnu sliku.

Samo po sebi se zapaža da su granice za radni prostor p = 80 i q = 84 izabrane arbitrarno. Dobro bi bilo da te granice budu zamijenjene sa vrijednostima p = 39 i q = 43. Tada bi se moglo deklarisati da loader zauzima počev od adrese 1 do adrese 43 takođe uključeno.

Kada je program L predao upravljanje (predao štafetu) programu P onda je L postao nepotreban. Tako da P tokom svog rada (tokom svog izvršavanja) može po potrebi da koristi loaderov prostor 1–100. Recimo, da koristi taj prostor za svoje podatke (da ga koristi kao svoj radni prostor). Dakle, sada loader može da bude prebrisan.

Bolje je da loader ne bude prebrisan i bolje je da aplikativni program P vrati upravljanje loaderu, kako bi računar pristupio drugom poslu bez resetovanja računara. Takve i slične radnje pripadaju očito operativnom sistemu računara.

Šta je to boot–strap loader računara?

A ko će L–a da loaduje? Posebnu ulogu ima inicijalni sadržaj memorije. To je sadržaj koji je stalno prisutan u memoriji (čuva se u ROM–u), tj. sadržaj koji je prisutan u trenutku uključivanja računara. Ili je stalno prisutan ili se unosi posebnim postupkom prilikom uključivanja računara (unosi se na mehanički ili polu–automatski način).

U slučaju jednostavne šeme, inicijalni sadržaj memorije jeste upravo loader L. U slučaju bolje razrađene šeme (realne šeme), inicijalni sadržaj memorije jeste inicijalni loader IL ili engl. boot–strap loader čiji je zadatak da loaduje loadera L i da mu preda štafetu. Znači da IL ima ulogu inicijalne kapisle.

U slučaju operativnog sistema DOS/Windows, boot–strap loader (pored ostalog) učita sadržaj prvog sektora na hard–disku ili eventualno na disketi i onda preda upravljanje učitanom modulu, učitanom programskom kodu. Zatim se (pored ostalog) učitaju, takođe sa spoljašnje memorije, DOS–ovi sistemski fajlovi io.sys i msdos.sys. Onda se upravljanje predaje učitanom kodu. Bliže, upravljanje se predaje ulaznoj tački fajla io.sys. Itd. sve dok ne bude sve postavljeno i pripremljeno.

Dakle, pokretanje računara obavlja se u dvije–tri faze tokom kojih se većinom vrši punjenje modula operativnog sistema (moduli se upisuju u unutrašnju memoriju). Radnje punjenja (loadovanja) prepliću se sa drugim neophodnim radnjama, kao što su na primjer: provjera ispravnosti uređaja, upisivanje početnog sadržaja u posebne registre i slično. Tek kasnije će doći na red punjenje aplikativnog programa.

Pokretanje računara u dvije–tri faze (u nekoliko faza) obezbjeđuje da ukupna šema bude elastična, imajući u vidu promjene i dodavanja operativnom sistemu, i da inicijalna kapisla ostane prostorno mala.

Bootstrap loader

From "bootstrap" or "pull oneself up by one's bootstraps". A short program that was read in from cards or paper tape, or toggled in from the front panel switches, which read in a more complex program to which it gave control. On early computers the bootstrap loader was always very short, great efforts were expended on making it short in order to minimise the labour and chance of error involved in toggling it in, but was just smart enough to read in a slightly more complex program, usually from a card or paper tape reader, to which it handed control; this progeram in turn was smart enough to read the application or operating system from a magnetic tape drive or disk drive. Thus, in successive steps, the computer "pulled itself up by its bootstraps" to a useful operating state. Nowadays the bootstrap is usually found in ROM or EPROM, and reads the first stage in from a fixed location on the disk, called the "boot block". When this program gains control, it is powerful enough to load the actual operating system and hand control over to it.

Primjer 6. Loader koji vrši zajedno loadovanje i relokaciju

Na ulaznoj traci zapisan je kao ulazni podatak još i broj ΔA koji govori za koliko pozicija u memoriji treba da bude pomjerena rutina (modul) koji se unosi, za koliko pozicija u odnosu na svoj originalni tekst koji je dat na ulaznoj traci. Sastaviti program za mašinu M, tj. sastaviti tekst odgovarajućeg loadera. Sami za vježbu.

Sastaviti tekst (sastaviti kod) loadera za mašinu M koji popunjava nekoliko segmenata memorije. Dakle, treba da bude uneseno nekoliko rutina, gdje svaka rutina ima svoju vrijednost load point n, gdje počinje popunjavanje. Recimo, vidimo da u ranijem primjeru 3 dvije rutine treba da budu upisane u memoriju (M i S). Sami za vježbu. Precizno definisati koje sve ulazne podatke treba pripremiti, redosljed ulaznih podataka i slično.

1.12. Univerzalna Tjuringova mašina

Biće napisan univerzalni program u slučaju modela M i biće definisan pojam univerzalnog programa, odnosno biće opisan značaj tog programa U. Od svih programa koji su napisani za računare, koji je najvažniji? Bez sumnje, to je univerzalni program. Kako se definiše univerzalni program? To je program koji je u stanju da izvršava bilo koji program P. Znamo da postoje razni modeli za računanje i da su svi uzajamno ekvivalentni (Tjuringova mašina, RAM, model M, Intel Pentium, itd.). Naravno da se P i U odnose na jedan te isti model. Mi ćemo u nastavku napisati U u slučaju modela M. Kada je U napisan u slučaju modela Tjuringova mašina onda se za U kaže da je to univerzalna Tjuringova mašina. Taj program sastavio je Tjuring 1936. godine, da bi dokazao nerješivost nekih zadataka (kakav je halting problem).

Razmotrimo dvije šeme, od kojih prva služi za pripremu. Prva šema odnosi se na slučaj običnog (samostalnog) izvršavanja programa P. Tada, u trenutku kada počinje rad računara, pretpostavlja se da se u memoriji već nalaze upisane naredbe (i podaci) programa P. Ulazna tačka enterp programa P unosi se ručno preko svičeva, tako da je u trenutku kada počinje rad računara LC = enterp. Obično je enterp = 1. Druga šema. U trenutku kada počinje rad programa U u memoriji se već nalaze (već su loadovani) program P – zauzima adrese od 1 do 900, kao i program U – zauzima adrese od 901 do 2000. U okviru toga, $c(902)$ = enterp, tj. ulazna tačka programa P već je loadovana i to na mjesto 902. Još, ulazna tačka programa U unosi se ručno preko svičeva. Dakle, u trenutku kada počinje rad računara važi LC = enter tj. $LC = 1001$ ("enteru"). Dalje, $c(901)$ = bound tj. $c(901) = 900$. Biće provjeravano da P ne djeluje van područja $1 \leq a \leq 900$, tj. izašlo bi se iz programa ako se desi $a > 900$. Granica bound je svojstvena interpreterima za stvarni računar (Intel Pentium), a ona može da se izbjegne u slučaju teorijskih modela (Tjuringova mašina, RAM, model M).

U slučaju naredbe ADD i treba da bude $a = i$, a u slučaju naredbe ADD$*$ i treba da bude $a = c(i)$. To su naredbe iz programa P, očito. Slično, u slučaju naredbe JUMP i treba da bude $a = i$, a u slučaju naredbe JUMP$*$ i treba da bude $a = c(i)$. Vidimo da je a – fizička adresa. Dalje, u slučaju naredbe oblika ADD$=$ i, veličina a je nedefinisana ili tada možemo staviti da je $a = 900$.

Zadatak. Sastaviti program U za model M koji će da interpretira (oponaša) rad bilo kog programa P (program P je takođe za model M). Pored oponašanja, program U još treba i da redom štampa tekuće naredbe programa P.

Rješenje. Djelimični plan za upotrebu memorije programa U:

Adresa	Sadržaj	Smisao promjenljive
901	bound	granica, bound $\leftarrow$ 900
902	enterp	ulazna tačka programa P
903	lcp	kopija programskog brojača programa P
904	acp	kopija akumulatora programa P
905	opcodep	opkod tekuće naredbe programa P
906	addressp	adresni dio tekuće naredbe programa P
907	a	fizička adresa kojoj se obraća tekuća naredba programa P
908	indikator	(moguće vrste tekućih naredbi podijeljene su u tri grupe)

Tekst programa U (izvršavanje počinje od 1001): prikazane su adrese (labele), naredbe i iza znaka $\diamondsuit$ komentari:

```
        995–996   LOAD 904 ♢ AC ← c(904) (AC ← acp)
        997–998   (bilo šta) ♢ biće napakovano
       999–1000   JUMP dva ♢ goto dva, vraćam se
→ 1001–1002       c(901) ← 900 ♢ na mjesto za granicu upiši odgovarajuću vrijednost
                  c(903) ← c(902) ♢ tekuća vrijednost brojačaP ← početni brojačP
          iduća:  c(905) ← c(c(903)), c(906) ← c(c(903) + 1) ♢ prepiši tekuću naredbuP
                        na mjesto 905–906
                  c(907) ← a ♢ izračunaj a i upiši ga na 907 (ovo je ustvari mali potp.)
                  ako je tekuća naredbaP HALT onda indikator ← 2, a ako je skok onda
                        indikator ← 1, a inače indikator ← 0
                  WRITE lcp, acp, opcodep, addressp, a, indikator ♢ štampa tekuće
                  if lcp > bound or a > bound then { WRITE "prekoračenje"; HALT }
                  if indikator = 2 then HALT
                  if indikator = 1 then goto ind1 else goto ind0
          ind1:   uslov ← (opcodep = JUMP ili JUMP∗) or
                        (opcodep = JGTZ ili JGTZ∗) and (acp > 0) or
                        (opcodep = JZERO ili JZERO∗) and (acp = 0)
                  if uslov then c(903) ← c(907) else c(903) ← c(903) + 2 ♢ if uslov
                        then lcp ← a else lcp ← lcp + 2
                  JUMP iduća ♢ goto iduća
          ind0:   c(997) ← c(905), c(998) ← c(906) ♢ tamo, napakovati naredbu
                  JUMP 995 ♢ goto 995 (goto jedan)
           dva:   STORE 904 ♢ c(904) ← AC (acp ← AC)
                  c(903) ← c(903) + 2 ♢ lcp ← lcp + 2 [ jedna naredba – dvije adrese ]
                  JUMP iduća ♢ goto iduća
```

Sada treba da pojasnimo tekst predloženog rješenja, da pojasnimo program U.

Opišimo radnje koje program U izvodi redom. On prvo donosi na određeno mjesto 905–906 tekuću naredbu programa P. Prilikom donošenja (odakle da se donese, šta da se donese) on se upravlja po promjenljivoj c(903) = lcp, location counter P. Zatim on prosto odštampa c(905) i c(906). Zatim analizira tekuću naredbu (analizira ono što je donio), gdje postoje dva slučaja. Prvi slučaj: opcodep = JUMP ili opcodep = JGTZ ili opcodep = JZERO. Postoji opasnost da U ispusti kontrolu situacije, što se njemu uopšte ne bi dopadalo. Zato U preuzima na sebe izvršavanje takve naredbe, on će prilagoditi vrijednost lcp na odgovarajući način. Međutim, u drugom slučaju kada nije skok ili stop postupa se kako slijedi. Na mjestu 997–998 formira se naredba (opcodep,addressp) koja treba da bude izvedena i onda se izvrši skok na to mjesto (labela "jedan"). Možemo reći da će naredba (opcodep, addressp) biti "hardverski" izvedena. Još, obezbijeđeno je (pomoću 999–1000) da se nakon toga nastavi sa interpretacijom (skok na labelu "dva"). Nakon prvog ili drugog slučaja, krug se zatvara, odnosno donosi se nova naredba programa P, ona se analizira, itd.

Pogledajmo naredbu na mjestu 995. To je naredba LOAD 904 (ima smisao: $AC \leftarrow$ acp). Kopija akumulatora upisuje se na sistemsko mjesto (vraća se). Pogledajmo i naredbu STORE 904 (ima smisao: acp $\leftarrow AC$). Akumulator se spašava. Lako se vidi da su ove dvije naredbe neophodne. Naime, s jedne strane, mi želimo da neke naredbe interpretiranog programa (P) budu izvedene hardverski. A s druge strane, program koji vrši interpretaciju (U) izvodi i puno nekih svojih obrada. Recimo, računa a ili štampa lcp.

Treba još ponešto da pojasnimo. Vidimo da smo program U napisali u najvećoj mjeri na neformalnom jeziku asemblera modela M, iz očitih tehničkih razloga. Drugo, na izlaznoj traci biće pomiješano odštampano ono što bi P samostalno odštampao i ono što U po svojoj volji štampa.

Univerzalni program dosta se razlikuje po izgledu, zavisno za koji model je napisan. Mi smo se opredijelili za model M da bi rješenje bilo razumljivije.

Na kraju, dio programa "štampaj lcp" i slično svojstven je dibagerima koji štampaju razna obavještenja i svako malo zastanu u toku svog rada i slično. Međutim, univerzalna Tjuringova mašina U ne vrši nikakve dodatne radnje. Ona samo oponaša rad neke druge Tjuringove mašine (ona samo interpretira neku Tjuringovu mašinu). U zaključku, kada govorimo o računarima i programima uopšte, onda možemo kazati: dovoljan je samo jedan program (U) jer on može da izvršava sve programe.

U nastavku – diskusija.

(1) U napisanom programu za model M koristi se mogućnost kojom taj model raspolaže da se program mijenja, da se naredbe mijenjaju tokom rada programa (naredba na adresi 997). Naglasimo da korišćenje te mogućnosti nije neophodno. Dakle, može se sastaviti rješenje u slučaju modela M gdje se tekst univerzalnog programa ne mijenja tokom njegovog rada, izvršavanja (tekst je "konstantan"). Po prilici, u takvoj varijanti univerzalnog programa, U preuzima na sebe izvršavanje jedne po jedne naredbe programa P, slično onome što ima u izloženom rješenju kada su u pitanju naredbe za skokove programa P (tamo je bilo nazvano "prvim slučajem").

(2) Izložimo šemu UTM. Neka je $t \geq 1$ i neka je $A_t = \{a_1, \ldots, a_t\}$ azbuka sa t članova. Postoji univerzalna Tjuringova mašina u oznaci U. S druge strane, neka je T bilo koja T. m. čija je azbuka A_t. Označimo sa w_T njenu mašinsku riječ, gdje $w_T \in \Omega(A_t)$ (prikaz njene tablice). Neka je $n \geq 1$ i neka su $w_1, \ldots, w_n$ riječi u azbuci A_t. Označimo $\overline{w} = \sigma_2^t(n, \sigma_n^t(w_1, \ldots, w_n))$, gdje $\overline{w} \in \Omega(A_t)$ (prikaz njene početne pozicije). Dalje, neka se mašina U primijeni na početnu poziciju $a_0\, w_T\, a_0\, \overline{w}\, a_0\, \ldots$. Kao ishod rada, dobiće se rezultat (vrijednost funkcije) u oznaci $w \in \Omega(A_t)$ ili se neće dobiti rezultat. Tako je definisana jedna djelimična funkcija f: Dom $f \to \Omega(A_t)$ (Dom $f \subset \Omega^n(A_t)$). Ta funkcija se poklapa sa funkcijom pridruženom mašini T, tj. ishod rada je isti sa ishodom koji bi se dobio prilikom primjene mašine T na početnu poziciju $a_0\, w_1\, a_0\, \ldots\, a_0\, w_n\, a_0\, \ldots$.

(3) Ako posmatramo program UTM, onda je jasno da je taj program "konstantan" (ne mijenja se tokom izvršavanja). Zaista, TM je model za računanje sa tzv. spoljašnjim programom, tako da je svaki program P za TM (u tom broju i program U) stalnog sadržaja. Drugim riječima, tehnički je neizvodljivo da se u njemu bilo šta promijeni (da se neka naredba promijeni) tokom njegovog rada.

Programiranje I
Sadržaj predavanja:

1.1 Intuitivni pojam algoritma drugaaj.tex
1.2 Pojam izračunljive funkcije i pojam rješivog skupa drugaaj.tex
2.1 Definicija Tjuringove mašine drugaaj.tex
2.2 Računar naspram Tjuringove mašine drugaaj.tex
2.3 Pojmovi konfiguracija mašine i pozicija mašine drugaaj.tex
2.4 Pojam funkcije izračunljive po Tjuringu drugaaj.tex
3.1 Elementarne mašine drugaak.tex
3.2 Dva primjera Tjuringovih mašina drugaak.tex
3.3 Definicija Tjuringovog dijagrama drugaak.tex
3.4 Konstrukcija tablice po zadatom dijagranu drugaak.tex
3.5 Dalji primjeri Tjuringovih mašina drugaak.tex
3.6 Dalji primjeri Tjuringovih mašina (nastavak) drugaal.tex
3.7 Dalji primjeri Tjuringovih mašina (nastavak i kraj) drugaal.tex
3.8 Mašina za množenje dva broja drugaal.tex
3.9 Konverziona funkcija $\gamma_{t,s}$ drugaam.tex
3.10 Konverziona funkcija σ_n^t drugaam.tex
3.11 Predstavljanje Tjuringove mašine pomoću dijagrama koji je sastavljen od elementarnih mašina drugaam.tex
4.1 Mašina $T^{\S}$ drugaan.tex
4.2 Postavka zadatka o modeliranju nad azbukom A_1 drugaao.tex
4.3 Modeliranje nad azbukom A_1 drugaao.tex
4.4 Modeliranje nad azbukom A_1 (nastavak) drugaap.tex
4.5 Modeliranje nad azbukom A_1 (nastavak i kraj) drugaap.tex
4.6 Normalno računanje po Tjuringu drugaaq.tex
4.7 Superpozicija funkcija koje su izračunljive po Tjuringu drugaaq.tex
5.1 Mašinske riječi drugaar.tex
5.2 Jedan nerješiv skup (M) drugaar.tex
5.3 Drugi nerješiv skup (M_1) drugaar.tex
5.4 Zadatak o zaustavljanju drugaas.tex
5.5 Razni primjeri nerješivih skupova drugaas.tex
1.1 Algoritmi i njihova složenost drugaa.tex
1.2 RAM drugaau.tex & drugaav.tex
1.3 Složenost programa za RAM drugaax.tex
1.4 Mašina sa upisanim programom RASP drugaa1.tex
1.5 Apstrakcije mašine RAM drugab.tex
1.6 Primitivni model računanja – Tjuringova mašina drugac.tex
1.7 Uzajamni odnos modela RAM i Tjuringove mašine drugad.tex
1.8 Nestrogi paskal – programski jezik visokog nivoa drugad.tex
1.9 Rad sa nizom (model M) drugae.tex
1.10 Rad sa potprogramom (model M) drugae.tex
1.11 Pojam loadera (model M) drugae.tex
1.12 Univerzalna Tjuringova mašina drugae1.tex

Fajlovi (gsview): drugaaj.tex 10 pages drugaak.tex 6 pages drugaal.tex 3 pages drugaam.tex 4 pages drugaan.tex 2 pages drugaao.tex 2 pages drugaap.tex 2 pages drugaaq.tex 3 pages drugaar.tex 5 pages drugaas.tex 3 pages drugaa.tex 4 pages drugaau.tex 3 pages drugaav.tex 2 pages drugaax.tex 4 pages drugaa1.tex 4 pages drugab.tex 4 pages drugac.tex 4 pages drugad.tex 2 pages drugae.tex 8 pages drugae1.tex 3 pages. $\sum = 20$ files & 78 pages