Teorija izračunljivosti

Damir Hasić, Adis Alihodžić, Elmedin Selmanović

Prazna stranica.

UNIVERZITET U SARAJEVU
PRIRODNO-MATEMATIČKI FAKULTET

DAMIR HASIĆ
ADIS ALIHODŽIĆ
ELMEDIN SELMANOVIĆ

TEORIJA IZRAČUNLJIVOSTI

Prvo izdanje

PMF
SARAJEVO, 2021.

Autori:
dr. Damir Hasić
dr. Adis Alihodžić
dr. Elmedin Selmanović

Naziv udžbenika:
Teorija izračunljivosti
Prvo izdanje

Izdavač:
Prirodno-matematički fakultet Univerziteta u Sarajevu

Recenzenti:
prof. dr. Pilav Esmir
prof. dr. Željko Jurić

Korekturu izvršili autori.

Kompjuterska obrada teksta: Damir Hasić

Naslovna stranica i LaTeX predložak:
The Legrand Orange Book (Mathias Legrand)

CIP - katalogizacija u publikaciji
Nacionalna i univerzitetska biblioteka Bosne i Hercegovine, Sarajevo

51-3(075.8)

HASIĆ, Damir
Teorija izračunljivosti [Elektronski izvor] / Damir Hasić, Adis Alihodžić, Elmedin Selmanović. - 1. izd. - Sarajevo : Prirodno-matematički fakultet, 2021

Način pristupa (URL): http://math.pmf.unsa.ba/kategorija/elektronska-izdanja/. - El. knjiga. - Nasl. sa nasl. ekrana. - Opis izvora dana 16. 9. 2021. - Bibliografija: str. [189]-193.

ISBN 978-9926-453-36-7

1. Alihodžić, Adis 2. Selmanović, Elmedin

COBISS.BH-ID 45389830

Sadržaj

Predgovor i

1 Uvodni pojmovi 1

1.1 Skupovi 1

1.2 Alfabet, riječi i jezici 10

2 Konačni automati 15

2.1 Regularni jezici 15

2.2 Deterministički konačni automati 19

2.3 Nedeterministički konačni automati 29

2.4 Neregularni jezici 49

2.5 Neke primjene konačnih automata 53

2.6 Zadaci za samostalan rad 54

3 Potisni automati 57

3.1 Kontekstno nezavisne gramatike 57
3.1.1 Stablo raščlanjivanja 65
3.1.2 Dvosmislenost gramatika i jezika 67
3.1.3 Chomsky normalna forma 68
3.2 Nedeterministički potisni automati 71
3.2.1 Pumpajuća lema za kontekstno nezavisne jezike 83
3.2.2 k-potisni automati 88
3.3 Primjena gramatika 90

4 Turingove mašine 95

4.1 Definicija Turingove mašine 95
4.2 Church-Turingova teza 101
4.3 Varijante Turingove mašine 102
4.3.1 Turingova mašina sa nepomičnom glavom 102
4.3.2 Turingova mašina sa više traka 104
4.3.3 Turingova mašina sa dvostruko beskonačnom trakom 104
4.3.4 Nedeterministička Turingova mašina 105
4.3.5 k-PDA 107
4.3.6 Primjeri 109
4.4 Osobine Turing-prepoznatljivih i odlučivih jezika 110
4.5 Primjeri odlučivih jezika 119
4.6 Problem zaustavljanja 124
4.7 Primjeri neodlučivih i Turing-neprepoznatljivih jezika 128
4.8 Riceov teorem 135
4.9 Enumerator 137
4.10 Neograničene gramatike 141
4.11 Chomskyeva hijerarhija formalnih jezika 143
4.12 Registarska mašina 147

5 NP-teški problemi 149

5.1 Asimptotska notacija 149

5.2 Vremenska kompleksnost i klasa P 155
5.3 Polinomna redukcija 162
5.4 Problemi zadovoljivosti logičkih formula 164
5.5 Klasa NP 167
5.6 Cook-Levinova teorema 172
5.7 Osobine i primjeri NP-kompletnih problema 176
5.8 Zadaci za samostalan rad 183

Bibliografija 189

Prazna stranica.

Predgovor

Ne postoji oblast nauke, tehnologije ili struke u kojoj se ne koriste kompjuteri za rješavanje problema. Šta znači riješiti problem? Kakve vrste rješenja kompjuter može dati? Na koji način problem može biti prikazan da bi ga kompjuter razumio? Da li se dati problem može uopšte riješiti (u konačno mnogo koraka)? Ovo su pitanja kojim se bavi *teorija izračunljivosti*.

Iako probleme rješavamo intuicijom, jedini način da pravilno izložimo njihova rješenja je formalizam. Prije toga i sam problem mora biti izložen na formalan način. Tek nakon što smo na formalno zadovoljavajući način izložili problem i rješenje, možemo biti sigurni (koliko je to moguće) da je rješenje tačno.

Većina studenata kompjuterskih nauka vide budućnost u IT sektoru, te formalan način zasnivanja problema izaziva podozrenje. Upravo je ovo razlog zašto većina studija kompjuterskih nauka pokušava balansirati sa stepenom formalizma materije. Međutim, formalizam se ne može izbaciti, jer bez njega nema preciznosti, a bez preciznosti nema ni ispravnog rješenja. Upravo programski kodovi zahtijevaju veliki stepen preciznosti - veći nego kod teoretskih nauka kao što su teoretska matematika ili teoretska fizika. Razlog za ovo je

da kompjuter ne "podrazumijeva" stvari kao ljudski mozak, već i najmanja sitnica se mora specificirati.

Postojeća literatura iz teorije izračunljivosti i kompleksnosti je često izložena na veoma formalan način, jako težak za razumijevanje, pogotovo za studente nižih godina koji se tek susreću sa ovom materijom.

U ovom udžbeniku smo se trudili da nivo apstraktnosti odgovara studentima koji slušaju predmet *Teorija izračunljivosti*. U tu svrhu su ubačeni riješeni primjeri, koji pobliže objašnjavaju teoriju, kao i dokaze tvrdnji. Sami dokazi su pojednostavljeni, koliko je to moguće. Obrađeni su pojmovi sa kojima bi studenti (teoretskih) kompjuterskih nauka trebali biti upoznati.

Centralno pitanje, koje se obrađuje u ovom udžbeniku, je: "Koji problemi se mogu riješiti pomoću algoritma". Slijedeće pitanje bi bilo: "Šta je algoritam?". Intuitivno, "algoritam" je konačan niz, dobro definisanih, koraka. Formalnije, algoritme predstavljamo pomoću raznih tipova mašina. Ograničenja koja mašine imaju se prenose i na algoritme, pa govorimo o *algoritmima sa ograničenjima*. Različiti tipovi algoritama/automata mogu riješiti različite tipove problema/jezika.

U poglavlju 1, *Uvodni pojmovi*, je na kratak način izložena teorija skupova. Metode konstruisanja prebrojivih skupova dijele zajedničku osobinu sa metodama konstruisanja algoritama: svaki element mora biti dobro definisan, što podrazumijeva i da se nalazi na konačnoj poziciji unutar definisane strukture. Pojam skupa se koristi da se uvede pojam *jezika*, pomoću kojeg na formalan način zapisujemo probleme.

Poglavlje 2, *Konačni automati*, uvodi pojmove (ne)determinističkih konačnih automata, kojima su ekvivalentni pojmovi *regularnih jezika* i *izraza*. Regularni jezici predstavljaju jednu od najjednostavnijih familija jezika, kojima odgovaraju najjednostavnije mašine/algoritmi.

Poglavlje 3, *Potisni automati*, uvodi napredniju klasu automata/algoritama, kao i ekvivalentnu klasu problema/jezika: *kontekstno nezavisni jezici*. Ovi jezici se mogu predstaviti i pomoću struktura koje nazivamo *kontekstno nezavisne gramatike*.

Poglavlje 4, *Turingove mašine*, navodi najgeneralniju klasu mašina, kojima odgovara pojam algoritma u punom smislu te riječi. U ovom poglavlju navodimo i probleme koji se ne mogu riješiti u konačno mnogo koraka. Tu razlikujemo dvije vrste problema: neodlučivi i Turing-neprepoznatljivi problemi.

Poglavlje 5, *NP-teški problemi*, posmatra klasu problema koji se mogu

riješiti u konačno mnogo koraka, ali nije poznato da li postoji brzi (polinomni) algoritam za njihovo rješavanje. Mnogi praktični problemi su NP-kompletni, što navodi na zaključak da vjerovatno ne postoji brzo i tačno rješenje - upravo je ovo razlog zašto se većina problema iz prakse rješava pomoću heuristika.

Autori se zahvaljuju recenzentima prof.dr. Esmiru Pilavu i prof. dr. Željku Juriću na trudu i korisnim sugestijama.

Sarajevo, 2021. godine.

Autori.

1. Uvodni pojmovi

U ovom poglavlju dajemo pregled nekih osnovih pojmova i osobina iz matematike. Ovo nam je potrebno jer se formalna teorija zasniva na matematici; jezici se definišu pomoću skupova; neke ideje iz teorije skupova ćemo koristiti za konstruisanje algoritama; odnosi između objekata se mogu prikazivati pomoću relacija ili dijagrama.

1.1 Skupovi

Skup i *element* su osnovni pojmovi koje ne definišemo. Ako element x pripada skupu A, tada to pišemo sa $x \in A$. U suprotnom pišemo $x \notin A$.

■ **Primjer 1.1** Dat je skup $A = \{1,2,3,4,5\}$. Imamo da $2 \in A$ i $7 \notin A$. ■

Sa $\mathbb{N} = \{1,2,3,\ldots\}$ označavamo skup prirodnih brojeva, sa $\mathbb{N}_0 = \{0,1,2,3,\ldots\}$ označavamo skup prirodnih brojeva sa nulom, sa $\mathbb{Z} = \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$ označavamo skup cijelih brojeva, sa $\mathbb{Q} = \{\frac{m}{n} \mid m \in \mathbb{Z} \text{ i } n \in \mathbb{N}\}$ označavamo skup racionalih brojeva, te sa $\mathbb{R}$ označavamo skup realnih brojeva. Realnih brojevi su svi konačno i beskonačno-decimalni brojevi.

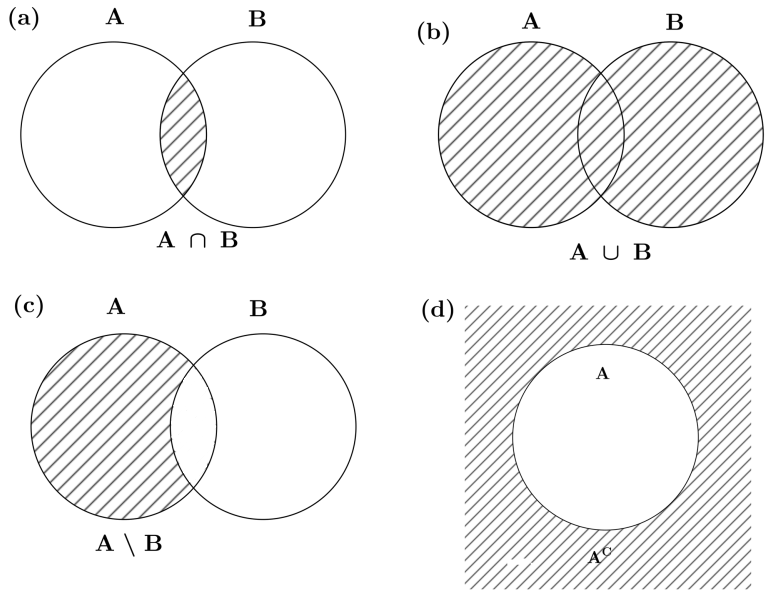
Definicija 1.1 — Podskup skupa. Neka su A i B skupovi. Ako iz $x \in A$ slijedi $x \in B$, tada kažemo da je A *podskup skupa* B i pišemo $A \subseteq B$. Ako je dodatno $A \neq B$, tada kažemo da je A *pravi podskup skupa* B i pišemo $A \subset B$. Ako A nije podskup skupa B, onda pišemo $A \not\subseteq B$.

■ **Primjer 1.2** Neka je $A = \{1,2,3\}$, $B = \{1,2,3,4,5,6\}$, $C = \{4,5,6,7,8\}$. Imamo da važi $A \subseteq B$, $A \subset B$, $A \not\subseteq C$. ■

Važi $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$.

Sa $|A|$ označavamo broj elemenata skupa. Ako su A, B, C skupovi iz prethodnog primjera, imamo $|A| = 3$, $|B| = 6$, $|C| = 5$. Prazan skup je skup bez elemenata i označavamo ga sa $\varnothing$. Imamo $|\varnothing| = 0$. Podrazumijevamo da je skup $\{1, \ldots, n\}$ za $n < 1$ prazan skup.

Definicija 1.2 — Unija, presjek, razlika skupova. Neka su A i B skupovi. Skup $A \cap B = \{x \mid x \in A \text{ i } x \in B\}$ nazivamo *presjek skupova* A i B. Skup $A \cup B = \{x \mid x \in A \text{ ili } x \in B\}$ nazivamo *unija skupova* A i B. Skup $A \setminus B = \{x \mid x \in A \text{ i } x \notin B\}$ nazivamo *razlika skupova* A i B.



Slika 1.1: Operacije između skupova A i B. **(a)** Presjek skupova. **(b)** Unija skupova. **(c)** Razlika skupova. **(d)** Komplement skupa.

■ **Primjer 1.3** Neka su A,B,C skupovi iz primjera 1.2. Tada imamo $A\cup B=\{1,2,3,4,5,6\}$, $A\cup C=\{1,2,3,4,5,6,7,8\}$, $A\cap B=\{1,2,3\}$, $A\cap C=\varnothing$, $B\cap C=\{4,5,6\}$, $A\setminus B=\varnothing$, $B\setminus C=\{1,2,3\}$. ■

Da bi definisali pojam *komplementa skupa*, potreban nam je pojam *univerzalnog skupa*. Možemo reći da je univerzalni skup proizvoljan skup koji sadrži sve skupove koje posmatramo u određenoj tvrdnji ili primjeru. U većini slučajeva se ne navodi, već se iz konteksta podrazumijeva.

Definicija 1.3 — Komplement skupa. Neka je A proizvoljan skup i $\mathscr{U}$ neki univerzalni skup. *Komplement skupa* A označavamo sa $\overline{A}$ i definišemo kao $\overline{A}=\{x\mid x\in\mathscr{U}\text{ i }x\notin A\}=\mathscr{U}\setminus A$. Ako se univerzalni skup podrazumijeva, tada pišemo $\overline{A}=\{x\mid x\notin A\}$.

■ **Primjer 1.4** Odredićemo komplement slijedećih skupova: A je skup svih prirodnih parnih brojeva; B je skup svih iracionalnih brojeva.

Kod skupa A podrazumijevamo da je univerzalni skup jednak skupu prirodnih brojeva, pa je $\overline{A}$ skup svih prirodnih neparnih brojeva.

Kod skupa B podrazumijevamo da je univerzalni skup jednak skupu realnih brojeva, pa je $\overline{B}=\mathbb{Q}$.

Napomenimo da se skup svih parnih prirodnih brojeva može označiti i kao $2\mathbb{N}$, a skup svih neparnih prirodnih brojeva se može označiti kao $2\mathbb{N}-1$, ali ne i kao $2\mathbb{N}+1$ (objasniti). ■

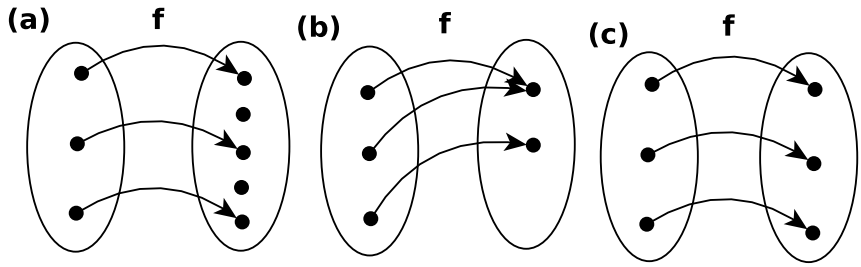
Definicija 1.4 — Injekcija, sirjekcija, bijekcija. Za preslikavanje $f{:}A{\to}B$ kažemo da je *injekcija* ako $x_1\neq x_2$ implicira $f(x_1)\neq f(x_2)$. Za preslikavanje f kažemo da je *sirjekcija* ako za svako $y\in B$ postoji $x\in A$ tako da je $y=f(x)$. Za preslikavanje kažemo da je *bijekcija* ako je injekcija i sirjekcija.

■ **Primjer 1.5** Neka su $f,g,h:\mathbb{N}\to\mathbb{N}$ funkcije date sa: $f(n)=5n+1$, $g(n)=\lfloor\frac{n+1}{2}\rfloor$, $h(n)=n+(-1)^{n+1}$.

Dokazaćemo da je f injekcija i nije sirjekcija, g je sirjekcija i nije injekcija, te h je bijekcija.

Funkcija f je injekcija, jer $f(n_1)=f(n_2)\implies 5n_1+1=5n_2+1\implies n_1=n_2$. Funkcija f nije injekcija, jer $y=f(n)\implies y=5n+1$, što znači da y prilikom dijeljenja sa 5 daje ostatak 1, a to nije osobina svih prirodnih brojeva.

Oznaku $\lfloor x\rfloor$ koristimo za funkciju najveće cijelo, koja se definiše kao najveći



Slika 1.2: Preslikavanje $f : A \to B$. **(a)** Preslikavanje f je injekcija. **(b)** Preslikavanje f je sirjekcija. **(c)** Preslikavanje f je bijekcija.

cio broj koji nije veći od x. Tako imamo $\lfloor 2.7 \rfloor = 2$, $\lfloor 5 \rfloor = 5$, $\lfloor -2.7 \rfloor = -3$. Funkcija g nije injekcija, jer $g(2) = g(3) = 1$. Sa druge strane imamo $y = \lfloor \frac{2y}{2} \rfloor = g(2y-1)$ za proizvoljno $y \in \mathbb{N}$, što znači da je g sirjekcija.

Dokažimo da je h bijekcija. To ćemo uraditi tako što ćemo dokazati da je sirjekcija i injekcija. Uočimo da je $h(n) = n-1$ za n parno, te $h(n) = n+1$ za n neparno. Znači, parni brojevi se preslikavaju u neparne i neparni brojevi se preslikavaju u parne.

Neka je $y \in \mathbb{N}$ proizvoljno. Ako je y neparno, tada je $y+1$ parno, te imamo $h(y+1) = (y+1)-1 = y$. Ako je y parno, tada je $y-1$ neparno, te imamo $h(y-1) = (y-1)+1 = y$. Znači, sa proizvoljno $y \in \mathbb{N}$ postoji elemenat iz $\mathbb{N}$ koji se, preko h, slika u y, pa je h sirjekcija.

Neka je $h(n_1) = h(n_2)$. To znači da su n_1 i n_2 iste parnosti. Ako su n_1 i n_2 neparni, tada imamo $h(n_1) = h(n_2) \implies n_1 + 1 = n_2 + 1 \implies n_1 = n_2$. Ako su n_1 i n_2 parni, tada imamo $h(n_1) = h(n_2) \implies n_1 - 1 = n_2 - 1 \implies n_1 = n_2$. Znači, h je injektivno preslikavanje. Pošto je h injekcija i sirjekcija, tada je h bijekcija. ∎

U slijedećoj tvrdnji navodimo osobine operacija sa skupovima, koje će nam biti korisne.

Tvrdnja 1.1 Neka su A, B i C proizvoljni skupovi. Tada važi:

1. $\overline{\overline{A}} = A$;
2. $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$;
3. $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$;
4. $\overline{A \cup B} = \overline{A} \cap \overline{B}$;

5. $\overline{A \cap B} = \overline{A} \cup \overline{B}$;

Definicija 1.5 — Partitivni skup. Neka je A skup. Skup

$$\mathscr{P}(A) = \{X \mid X \subseteq A\}$$

nazivamo *partitivni skup skupa A*.

Znači, partitivni skup skupa A je skup svih podskupova od A.

■ **Primjer 1.6** Neka je $A = \{1,2,3\}$. Odredićemo $\mathscr{P}(A)$.

Imamo da važi $\mathscr{P}(A) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$.

Uočimo da je $|A| = 3$ i $|\mathscr{P}(A)| = 8 = 2^3$. Za svaki skup A važi $|\mathscr{P}(A)| = 2^{|A|}$. ■

Intuitivno je jasno šta je konačan skup. Slijedećom definicijom ovaj pojam i formalno uvodimo.

Definicija 1.6 — Konačni i beskonačni skupovi. Za skup A kažemo da je konačan, ako postoji $n \in \mathbb{N}_0$ i bijekcija sa A na $\{1, \ldots, n\}$. Ako skup nije konačan, onda je beskonačan.

■ **Primjer 1.7** Skup $\{a,b,c\}$ je konačan, jer postoji bijekcija sa $\{a,b,c\}$ na $\{1,2,3\}$. ■

Primjeri beskonačnih skupova su $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$.

Konačni skupovi A i B imaju isti broj elemenata akko postoji bijekcija između njih, te pišemo $|A| = |B|$. Ovu činjenicu možemo intuitivno proširiti i na beskonačne skupove. Da bi izbjegli rasprave kako jedan beskonačan skup može imati više elemenata od drugog beskonačnog skupa, umjesto pojma *broj elemenata* koristimo pojam *moć skupa*.

Definicija 1.7 — Moć skupa. Za dva skupa A i B kažemo da *imaju istu moć*, ako postoji bijekcija sa A na B i pišemo $|A| = |B|$. Ako ne postoji bijekcija sa A na B, pišemo $|A| \neq |B|$. Oznaku $|A|$ nazivamo *moć skupa A*.

Kod konačnih skupova, moć skupa predstavlja broj elemenata skupa. Ovu predstavu možemo usvojiti i za beskonačne skupove. Ako dva skupa imaju istu moć, tada, intuitivno, imaju isti broj elemenata.

Slijedeća definicija uvodi mogućnost da jedan skup ima više elemenata od drugog skupa. Ovaj koncept je očigledan za konačne skupove, ali ne i za beskonačne.

Definicija 1.8 — Uspoređivanje moći skupova. Ako postoji injekcija sa skupa A u skup B, tada pišemo $|A| \leq |B|$. Ako je $|A| \leq |B|$ i $|A| \neq |B|$, tada pišemo $|A| < |B|$.

Drugim riječima, ako postoji injekcija sa A u B i ne postoji bijekcija sa A na B, pišemo $|A| < |B|$.

Slijedeća tvrdnja je očigledna za konačne skupove, ali ne i za beskonačne.

Teorema 1.1 — Cantor–Schröder–Bernsteinova teorema. Ako je $|A| \leq |B|$ i $|B| \leq |A|$, tada je $|A| = |B|$.

Prethodna teorema tvrdi da ako postoji injekcija sa A u B i sa B u A, tada postoji bijekcija sa A na B.

Definicija 1.9 — Niz. Neka je A skup. Ako postoji bijekcija $a : \mathbb{N} \to A$, tada a nazivamo *niz skupa* A i kažemo da smo elemente skupa A *poredali u niz* a.

Znači, elemente skupa A možemo prikazati kao $A=\{a(1),a(2),a(3),\ldots\}=$ $=\{a_1,a_2,a_3,\ldots\}$.

Definicija 1.10 — Prebrojiv skup. Za skup A kažemo da je prebrojiv ako postoji bijekcija sa $\mathbb{N}$ na A.

Znači, skup A je prebrojiv ako je beskonačan i ako mu se elementi mogu poredati u niz, tj. $A = \{a_1,a_2,\ldots\}$. Napomenimo da ne smeta ako se neki elementi ponavljaju, jer brisanjem ponavljanja možemo dobiti niz bez ponavljanja.

■ **Primjer 1.8** Skupovi $\mathbb{Z}$ i $\mathbb{Q}$ su prebrojivi.

Ovo ćemo dokazati tako što ćemo elemente skupova poredati u niz. Za prvi skup imamo $\mathbb{Z} = \{0,1,-1,2,-2,3,-3,\ldots\}$.

Elemente drugog skupa možemo poredati na slijedeći način $\mathbb{Q} = \{\frac{0}{1}, \frac{1}{1}, \frac{-1}{1}, \frac{1}{2}, \frac{-1}{2}, \frac{2}{1}, \frac{-2}{1}, \frac{1}{3}, \frac{-1}{3}, \frac{3}{1}, \frac{-3}{1}, \ldots\}$. Pojasnimo kako smo ih poredali. Za razlomak $\frac{m}{n}$ posmatramo zbir $|m|+n$. Prvo postavimo razlomke kod kojih je dati zbir jednak 1. Zatim postavimo razlomke kod kojih je dati zbir jednak 2, pa razlomke sa zbirom jednakim 3 itd. Pri tome brojeve, koji su se javljali ranije, ne pišemo ponovo, iako ne bi smetalo da isti broj pišemo više puta. ■

K Princip korišten u prethodnom primjeru je veoma bitan u nekim kasnijim primje-

rima. Zato ćemo još malo obratiti pažnju na njega. Gdje je greška u slijedećim "rješenjima"?

$\mathbb{Z} = \{0,1,2,3,\ldots,-1,-2,-3,\ldots\}$ - prvo poredamo u niz nulu i pozitivne brojeve, a zatim negativne brojeve.

$\mathbb{Q} = \{\frac{0}{1}, \frac{1}{1}, \frac{-1}{1}, \frac{2}{1}, \frac{-2}{1}, \ldots, \frac{0}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{2}{2}, \frac{-2}{2}, \ldots, \frac{0}{3}, \frac{1}{3}, \frac{-1}{3}, \frac{2}{3}, \frac{-2}{3}, \ldots\}$ - prvo poredamo razlomke sa nazivnikom 1, zatim razlomke sa nazivnikom 2, pa sa nazivnikom 3, itd.

Greška je u slijedećem. Koji god element niza da uzmemo, on se mora nalaziti na konačnoj poziciji, tj. prije njega smijemo imati samo konačno mnogo elemenata - u suprotnom ne radi se o nizu.

Posmatrajmo "rješenje" za skup $\mathbb{Z}$ i odaberimo npr. broj -1. Prije njega se nalaze svi pozitivni brojevi, tj. prije njega dolazi beskonačno mnogo brojeva, što je nemoguće za element niza.

Kod skupa $\mathbb{Q}$ odaberimo npr. $\frac{1}{3}$. Prije ovog elementa u "nizu" se nalazi beskonačno mnogo članova, što je nemoguće.

Znači, ako želimo elemente beskonačnog skupa poredati u niz, moramo se pobrinuti da svaki element bude zastupljen (barem) jednom i da se svaki element niza nalazi na konačnoj poziciji, tj. da prije njega smijemo imati konačno mnogo elemenata.

Za prebrojivi skup kažemo da ima moć $\aleph_0$. Prema tome:

$$\aleph_0 = |\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}|.$$

Slijedeća dva primjera se odnose na reprezentaciju beskonačnih decimalnih brojeva.

■ **Primjer 1.9** Dokažimo da je $1 = 0.999\ldots$, pri čemu na desnoj strani imamo beskonačan broj devetki. Ovaj broj možemo drugačije napisati i kao $0.999\ldots = 0.\dot{9}$.

Neka je $0.\dot{9} = x$. Nakon množenja sa 10, imamo $9.\dot{9} = 10x = 9x + x = 9x + 0.\dot{9}$, a odavde $9.\dot{9} - 0.\dot{9} = 9x$, tj. $9 = 9x$ i na kraju $x = 1$. Znači, $0.\dot{9} = 1$.

Ovo smo mogli dokazati koristeći i zbir geometrijskog reda: $0.\dot{9} = \sum_{i=1}^{+\infty} \frac{9}{10^i}$ $= 9\sum_{i=1}^{+\infty} \frac{1}{10^i} = 9(\sum_{i=0}^{+\infty} \frac{1}{10^i} - 1) = 9(\frac{1}{1-\frac{1}{10}} - 1) = 1$.

Posljedni red smo izračunali koristeći formulu za zbir geometrijskog reda: $\sum_{i=0}^{+\infty} a^n = \frac{1}{1-a}$, za $|a| < 1$. ■

■ **Primjer 1.10** Na sličan način kao u prethodnom primjeru, imamo da važi npr: $0.123 = 0.122\dot{9}$, $14.81 = 14.80\dot{9}$, itd. Znači, svaki decimalan broj sa konačno

mnogo decimala se može, na jedinstven način, predstaviti kao beskonačno-decimalni broj. ∎

Slijedeći primjer daje primjer skupa koji nije prebrojiv. Koristićemo tzv. *dijagonalni princip*.

∎ **Primjer 1.11** Dokažimo da važi $|\mathbb{N}| < |(0,1)|$.

Prvo dokažimo da važi $|\mathbb{N}| \leq |(0,1)|$. To ćemo dokazati tako što ćemo konstruisati injekciju sa $\mathbb{N}$ u $(0,1)$. Neka je $f : \mathbb{N} \to (0,1)$ tako da važi npr: $f(1) = 0.11$, $f(15) = 0.151$, $f(1762) = 0.17621$, itd. Znači, ako je $n = \overline{a_k a_{k-1} \dots a_1 a_0}$, tada je $f(n) = 0.a_k a_{k-1} \dots a_1 a_0 1$. Ovako preslikavanje je očigledno injekcija. Formalnije, ovo preslikavanje možemo prikazati kao: $f(n) = \frac{n}{10^{\lfloor log(n) \rfloor + 1}} + \frac{1}{10^{\lfloor log(n) \rfloor + 2}}$.

Dokažimo da ne postoji bijekcija sa $\mathbb{N}$ na $(0,1)$. Pretpostavimo suprotno, da postoji. Tada je skup $(0,1)$ prebrojiv, pa se svi njegovi elementi mogu poredati u niz. Neka je $(0,1) = \{a_1, a_2, \dots\}$, pri čemu $a_i = 0.a_{1i}a_{2i}a_{3i}\dots$ je beskonačno-decimalni broj između 0 i 1. Date brojeve stavimo u tabelu:

$$\begin{array}{c} 0.\underline{a_{11}}a_{21}a_{31}\dots a_{i1}\dots \\ 0.a_{12}\underline{a_{22}}a_{32}\dots a_{i2}\dots \\ 0.a_{13}a_{23}\underline{a_{33}}\dots a_{i3}\dots \\ \dots \\ 0.a_{1i}a_{2i}a_{3i}\dots \underline{a_{ii}}\dots \\ \dots \\ \hline 0.b_1b_2b_3\dots b_i\dots \end{array}$$

Sada ćemo konstruisati novi broj između 0 i 1. Neka je $b_i \in \{1,2,3\dots,8\}$ proizvoljno odabrano tako da je $b_i \neq a_{ii}$ $(i = 1,2,3,\dots)$. Dalje, uzmimo $b = 0.b_1b_2b_3\dots b_i\dots$. Zbog $b_i \neq a_{ii}$ imamo $b \neq a_i$ $(i = 1,2,\dots)$.

Zaključujemo da $b \notin \{a_1, a_2, \dots\}$, što je u kontradikciji sa pretpostavkom da smo sve brojeve iz $(0,1)$ poredali u niz. Prema tome, ne postoji bijekcija sa $\mathbb{N}$ na $(0,1)$, pa skup $(0,1)$ nije prebrojiv. ∎

Prethodni primjer demonstrira da postoje beskonačni skupovi koji nisu prebrojivi. Njih nazivamo *neprebrojivim* skupovima. Intuitivno, možemo reći da interval $(0,1)$ "ima više" elemenata od skupa $\mathbb{N}$.

Iz slijdećeg primjera imamo da interval $(0,1)$ i skup $\mathbb{R}$ imaju "isti broj" elemenata, što je neočekivano.

∎ **Primjer 1.12** Dokaži da važi $|(0,1)| = |\mathbb{R}|$.

Konstruisaćemo bijekciju sa $(0,1)$ na $\mathbb{R}$.

Preslikavanje $f:(0,1)\to(-\pi/2,\pi/2)$ dato sa $f(x)=\pi x-\pi/2$ je bijekcija. Pošto je funkcija tangens $\operatorname{tg}:(-\pi/2,\pi/2)\to\mathbb{R}$ takođe bijekcija, to je i njihova kompozicija $\operatorname{tg}\circ f:(0,1)\to\mathbb{R}$ bijekcija.

Znači, postoji bijekcija između $(0,1)$ i $\mathbb{R}$, pa je $|(0,1)|=|\mathbb{R}|$. ∎

Koristimo oznaku $c=|R|$ i imamo $\aleph_0<c$. Znači i kod beskonačnih skupova ima smisla, barem na intuitivnom nivou, upoređivati broj elemenata. Tako možemo reći da skupovi $\mathbb{N}$, $\mathbb{Z}$ i $\mathbb{Q}$ imaju "isti broj elemenata", dok $\mathbb{R}$ ima "više elemenata" od njih. Oznaka $|A|$ se još naziva i *kardinalni broj* skupa A.

Koliko imamo kardinalnih brojeva, da li su $\aleph_0$ i c jedini kardinalni brojevi? Slijedeća tvrdnja govori da imamo beskonačno mnogo kardinalnih brojeva.

Tvrdnja 1.2 — Cantorova teorema. Neka je A skup, tada je $|A|<|\mathscr{P}(A)|$.

Dokaz. Pošto je $f:A\to\mathscr{P}(A)$, dato sa $f(x)=\{x\}$ za svako $x\in A$, injekcija, imamo da je $|A|\leq|\mathscr{P}(A)|$.

Neka je $g:A\to\mathscr{P}$ proizvoljna funkcija. Dokažimo da g nije sirjekcija. Neka je $S=\{a\in A\mid a\notin g(a)\}$.

Dokažimo $g(x)\neq S$ za svako $x\in A$. Pretpostavimo suprotno, postoji $x_0\in A$ tako da je $g(x_0)=S$. Imamo dvije situacije: $x_0\in S$ ili $x_0\notin S$.

Ako $x_0\in S$, tada iz definicije skupa S imamo $x_0\notin g(x_0)=S$, što je kontradikcija.

Ako $x_0\notin S$, tada imamo $x_0\in g(x_0)=S$, što je opet kontradikcija.

Znači ne postoji sirjekcija sa A na $\mathscr{P}$, pa ne postoji ni bijekcija. ■

K Razmislite koja je sličnost između prethodnog dokaza i dijagonalnog principa.

Iz prethodne tvrdnje imamo da ne postoji najveći skup, tj. ne postoji skup koji sadrži sve. Koliko god veliki skup X da uzmemo, važi $|X|<|\mathscr{P}(X)|$, pa je $\mathscr{P}(X)$ veći od X.

Takođe, dobijamo da postoji beskonačno mnogo beskonačnih kardinalnih brojeva. Neka je skup A beskonačan. Tada važi: $|A|<\mathscr{P}(A)<\mathscr{P}(\mathscr{P}(A))<\ldots$.

Zanimljivo je pitanje da li imamo kardinalnih brojeva između $\aleph_0$ i c, tj. da li postoji skup X takav da je $\aleph_0<|X|<c$. *Hipoteza kontinuuma* govori da takav skup ne postoji. Kurt Gödel i Paul Cohen (1940. i 1963.) su dokazali da je ova hipoteza nezavisna od aksioma Zermelo–Fraenkel teorije skupova, čak

ni kada se uključi aksiom izbora (koji govori da iz svake familije disjunktnih skupova možemo uzeti po tačno jedan element). Drugim riječima, hipoteza kontinuuma se ne može dokazati ni opovrgnuti koristeći spomenutu teoriju.

Slijedeću tvrdnju navodimo bez dokaza.

Tvrdnja 1.3 (a) Unija prebrojivih skupova je prebrojiv skup, tj.

$$|A| = |B| = \aleph_0 \implies |A \cup B| = \aleph_0.$$

(b) Dekartov proizvod prebrojivih skupova je prebrojiv skup, tj.

$$|A| = |B| = \aleph_0 \implies |A \times B| = \aleph_0.$$

(c) Prebrojiva unija prebrojivih skupova je prebrojiv skup, tj.

$$|A_i| = \aleph_0, (\forall i \in \mathbb{N}) \implies \Big|\bigcup_{i=1}^{+\infty} A_i\Big| = \aleph_0.$$

1.2 Alfabet, riječi i jezici

Alfabet je konačan skup simbola. Označavaćemo ga sa Σ i podrazumijevamo $\Sigma = \{0,1\}$, ako drugačije ne kažemo.

String (riječ) je konačan niz simbola iz alfabeta. Možemo ga zapisati kao $\omega = \alpha_1\alpha_2\ldots\alpha_n$, pri čemu $\alpha_i \in \Sigma$ $(i = 1,\ldots,n)$. Za ω kažemo da je dužine n i pišemo $|\omega| = n$.

Sa ε označavamo prazan string i važi $|\varepsilon| = 0$. Sa Σ^* označavamo skup svih stringova. Sa Σ_ε označavamo skup $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Za string ω' kažemo da je *podstring* stringa $\omega = \alpha_1\alpha_2\ldots\alpha_n$, ako je $\omega' = \alpha_i\alpha_{i+1}\ldots\alpha_j$ za $1 \le i \le j \le n$ ili $\omega' = \varepsilon$. Za string ω' kažemo da je *prefix* stringa $\omega = \alpha_1\alpha_2\ldots\alpha_n$, ako je $\omega' = \alpha_1\alpha_2\ldots\alpha_j$ za $1 \le j \le n$ ili $\omega' = \varepsilon$. Za string ω' kažemo da je *sufix* stringa $\omega = \alpha_1\alpha_2\ldots\alpha_n$, ako je $\omega' = \alpha_i\alpha_{i+1}\ldots\alpha_n$ za $1 \le i \le n$ ili $\omega' = \varepsilon$.

Za dva stringa $\omega_1 = \alpha_1\alpha_2\ldots\alpha_m$ i $\omega_2 = \beta_1\beta_2\ldots\beta_n$, pri čemu $\alpha_i, \beta_j \in \Sigma$ $(i = 1,\ldots,m,\ j = 1,\ldots,n)$, kažemo da su *jednaki* i pišemo $\omega_1 = \omega_2$, ako $m = n$ i $\alpha_i = \beta_i$ $(i = 1,\ldots,n)$.

■ **Primjer 1.13** Primjeri stringova.

(*i*) Neka je $\omega_1 = 01011$. Dužina ovog stringa je 5 i pišemo $|\omega_1| = 5$. String

101 je podstring od ω_1, dok 111 nije podstring od ω_1. Svi prefiksi stringa ω_1 su: $\varepsilon, 0, 01, 010, 0101, 01011$, dok su svi sufiksi: $\varepsilon, 1, 11, 011, 1011, 01011$.

(*ii*) Niz znakova 01010101... nije string. Zašto?
Zato što se radi o beskonačnom nizu znakova. String (riječ) mora biti konačan niz.

(*iii*) Da li su stringovi $\omega_2 = 001$ i $\omega_3 = 010$ jednaki?
Nisu jednaki, jer ω_2 na drugoj poziciji ima 0, a ω_3 na drugoj poziciji ima 1.

■

Definicija 1.11 — Leksikografski poredak. Neka je $x = x_1 \dots x_k$, $y = y_1 \dots y_m$, pri čemu $x_i, y_j \in \Sigma$, $(i = 1, \dots, k, j = 1, \dots, m)$. Ako je $|x| < |y|$ ili $x_{i_0} < y_{i_0}$, pri čemu je i_0 najmanji broj takav da je $x_{i_0} \neq y_{i_0}$, tada pišemo $x < y$.

Da bi imali leksikografski poredak stringova, trebamo imati deinisan poredak na alfabetu Σ. U većini slučajeva, taj poredak je prirodan, tj. ako je $\Sigma = \{0, 1\}$, tada je $0 < 1$.

■ **Primjer 1.14** Leksikografski poredak.

(*i*) Poredaj u leksikografksom poretku 010, 11, 001.
Prednost dajemo kraćim stringovima, pa na prvo mjesto ide 11. Sada trebamo rasporediti 010 i 001. Prvo poredimo znakove na prvoj poziciji: 010 i 001. Pošto su isti, poredimo znakove na slijedećoj poziciji: 010 i 001. Pošto 0 ide prije 1 (tj. $0 < 1$), to string 001 ide prije 010.
Konačan poredak je: 11, 001, 010.

(*ii*) Poredaj u leksikografskom poretku sve stringove dužine ne veće od 4.
Koristeći slično razmišljanje kao u prethodnom slučaju, imamo poredak: ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

■

Uočite da leksikografski poredak podsjeća na poredak riječi u riječniku, ali mu nije identičan.

Tvrdnja 1.4 $|\Sigma^*| = \aleph_0$.

Dokaz. Sve stringove (tj. elemente skupa Σ^*) možemo poredati u niz. Prvo

stavimo prazan string, pa sve stringove dužine 1, pa sve stringove dužine 2, pa sve stringove dužine 3, itd: ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ... Ovaj poredak ne mora biti nužno leksikografski.

Pošto smo sve elemente iz Σ^* poredali u niz, imamo da je Σ^* prebrojiv, tj. $|\Sigma^*| = \aleph_0$. ■

Definicija 1.12 — Jezik. Neka je $A \subseteq \Sigma^*$. Tada se A naziva *jezik*.

Tvrdnja 1.5 Jezika ima neprebrojivo mnogo.

Dokaz. Skup svih jezika je dat sa $\mathscr{P}(\Sigma^*)$. Iz Cantorove teoreme i tvrdnje 1.4 imamo $|\mathscr{P}(\Sigma^*)| > |\Sigma^*| = \aleph_0$, odakle slijedi tvrdnja. ■

Jezik će obično odgovarati problemu, odnosno formulaciji problema, tj. probleme ćemo obično na formalan način iskazivati preko jezika. U većini slučajeva će jezici biti beskonačni, tj. sadržavaće beskonačno mnogo stringova. Pošto nije moguće ispisati sve stringove beskonačnog jezika, potrebno je jezik opisati na drugačiji način, tj. potrebno je dati konačnu reprezentaciju (opis) jezika, tj. potrebno je opisati jezik pomoću konačno mnogo simbola.

■ **Primjer 1.15** Konačna reprezentacija jezika.

(*i*) Jezik $A = \{0, 00, 000, 0000, \ldots\}$ možemo opisati kao $A = \{\omega \in \Sigma^* \mid \omega \text{ sadrži samo } 0\}$.

(*ii*) Jezik $B = \{\varepsilon, 00, 01, 10, 11, 0000, 0001, 0010, 0011, 0100, \ldots\}$ možemo napisati kao $B = \{\omega \in \Sigma^* \mid \omega \text{ je parne dužine}\}$.

■

Kasnije ćemo dati i neke druge vidove konačne reprezentacije jezika, kao npr. regularni izrazi, gramatike, konačni automati.

Da bi olakšali reprezentaciju jezika, uvodimo neke operacije. Ovdje ćemo uvesti operacije nadovezivanja i Kleeneove zvijezde.

Definicija 1.13 — Operacija nadovezivanja. Neka su A i B jezici. *Nadovezivanjem* jezika A i B dobijamo jezik $A \circ B = AB = \{ab \mid a \in A, b \in B\}$.

■ **Primjer 1.16** Neka je $A = \{101, 00\}$ i $B = \{01, 111, 010\}$. Tada je $AB = \{10101, 101111, 101010, 0001, 00111, 00010\}$. ■

■ **Primjer 1.17** Neka je $A = \{1, 11, 111, 1111, \ldots\}$, $B = \{0, 00, 000, 0000, \ldots\}$, tj. jezik A se sastoji od svih stringova koji sadrže samo 1, a jezik B od stringova

koji sadrže samo 0. Tada je $AB = \{10, 100, 110, 1000, 1100, 1110, \ldots\}$, tj AB se sastoji od stringova koji počinju sa konačnim nizom jedinica koji prati konačan niz nula. ∎

Kada nadovezujemo iste jezike, onda možemo koristit kraći zapis. Tako imamo $A^2 = AA$, $A^3 = AAA$, itd. Možemo dati i rekurzivnu definiciju:

$$A^0 := \{\varepsilon\},$$
$$A^n := A^{n-1}A.$$

Sličan zapis možemo koristiti i za stringove. Tako imamo $111 = 1^3$; $00000 = 0^5$; $01010101 = (01)^4$. Obratite pažnju da je $0^4 1^4 = 00001111$. Znači $(01)^4 \neq 0^4 1^4$.

∎ **Primjer 1.18** Neka je $A = \{0, 11\}$. Tada je $A^2 = \{00, 011, 110, 1111\}$, $A^3 = \{000, 0011, 0110, 01111, 1100, 11011, 11110, 111111\}$. ∎

Slijedeću operaciju možemo shvatiti kao neku vrstu beskonačnog nadovezivanja.

Definicija 1.14 — Kleeneova zvijezda. Neka je A jezik. Jezik A^* definišemo kao $A^* = \bigcup_{i=0}^{+\infty} A^i$.

Važi $A^* = \{a_1 a_2 \ldots a_n \mid n \geq 0, a_i \in A, i = 1, \ldots, n\}$.

Neka je $A^+ := AA^* = \bigcup_{k=1}^{+\infty} A^k$. Ako $\varepsilon \notin A$, tada $A^+ = A^* \backslash \{\varepsilon\}$. Sa druge strane, za $\varepsilon \in A$ imamo da važi $A^+ = A^*$. Slično kao i za A^*, vrijedi $A^+ = \{a_1 a_2 \ldots a_n \mid n \geq 1, a_i \in A, i = 1, \ldots, n\}$.

∎ **Primjer 1.19** Neka je $A = \{1\}$. Tada je $A^* = \{\varepsilon, 1, 11, 111, 1111, \ldots\} = \{1^n \mid n \geq 0\}$.

Uočite da smo uzeli $1^0 := \varepsilon$. ∎

Iz slijedećeg primjera vidimo zašto sa Σ^* označavamo skup svih stringova.

∎ **Primjer 1.20** Neka je $\Sigma = \{0, 1\}$. Tada je $\Sigma^* = \{a_1 a_2, \ldots a_n \mid n \geq 0, a_i \in \{0, 1\}, i = 1, \ldots, n\}$, a ovo je upravo skup svih stringova nad Σ. ∎

K Prioritet do sada uvedenih operacija i oznaka je slijedeći:

(*i*) $(,)$
(*ii*) $^*, ^+, ^n$;
(*iii*) $\circ$;
(*iv*) $\cap$;

(v) $\cup$.

Znači, zagrade imaju najveći prioritet, a unija najmanji. Tako, npr, važi:

$$(A \cup (B \cap (C \circ (D^*)))) = A \cup B \cap CD^*.$$

Tvrdnja 1.6 — Osobine operacije nadovezivanja. Operacija $\circ$ ima slijedeće osobine:

(a) asocijativnost: $(AB)C = A(BC)$, $\forall A,B,C \subseteq \Sigma^*$;

(b) distributivnost nadovezivanja u odnosu na uniju:
$(A \cup B)C = AC \cup BC$,
$C(A \cup B) = CA \cup CB$, $\forall A,B,C \subseteq \Sigma^*$.

Za operaciju $\circ$ važi da:

(a) nije komutativno, tj. postoje jezici A i B za koje važi $AB \neq BA$;

(b) nije distributivno u odnosu na presjek, tj. postoje jezici A,B,C tako da je
$(A \cap B)C \neq AC \cap BC$,
$C(A \cap B) \neq CA \cap CB$.

Zadatak 1.1 — Za samostalan rad. Pronađite netrivijalne primjere skupova koji demonstriraju da operacija nadovezivanja nije komutativna i nije distrubutivna u odnosu na presjek. ■

■ **Primjer 1.21 — Inverzni string.** Neka je $\omega = \alpha_1\alpha_2...\alpha_n$, pri čemu $\alpha_i \in \Sigma$. Sa $\omega^R = \alpha_n\alpha_{n-1}...\alpha_1$ označavamo *inverzni string* stringa ω. Tako, npr. ako je $\omega = 1101$, tada je $\omega^R = 1011$. ■

■ **Primjer 1.22 — Palindrom.** Za string $\omega = x_1 \ldots x_n$ kažemo da je palindrom ako vrijedi $x_i = x_{n+1-i}$, $\forall i \in \{1,...,n\}$. Ovo još možemo pisati i kao $x_i = x_{n+1-i}$, $\forall i \in \{1,...,\lfloor n/2 \rfloor\}$.

String je palindrom ako ga čitajući sa lijeve i sa desne strane strane dobijamo isti string. Tako npr stringovi $\omega_1 = 110011$, $\omega_2 = 10101$ su palindromi, dok $\omega_3 = 11010$ nije palindrom.

Uočite da je string ω palindrom akko je $\omega = \omega^R$. ■

2. Konačni automati

U ovom poglavlju ćemo se baviti jednostavnim problemima, koji imaju jednostavan opis i koji se mogu riješiti pomoću ograničenih automata (algoritama).

Probleme ćemo opisivati pomoću *regularnih jezika*. To su jezici koji se mogu dobiti kombinacijom operacija $\cup$, $\circ$, *, te simbola iz Σ.

Zatim ćemo uvesti ograničene automate (algoritme), koje nazivamo *(ne)deterministički konačni automati*. Pomoću ovih automata ćemo rješavati probleme koji se mogu opisati pomoću regularnih jezika.

2.1 Regularni jezici

Vidjeli smo da se jezici mogu predstaviti opisno. Ovdje uvodimo klasu jezika koja se može predstaviti pomoću struktura koji se zovu *regularni izrazi*. Takav jezik se naziva *regularan jezik*.

Regularan jezik se gradi pomoću regularnog izraza tako što se započne od simbola alfabeta (tj. od 0 i 1), te primjenom operacija unije, nadovezivanja i zvijezde dobije se izraz koji odgovara datom jeziku.

Slijedeća definicija, na rekurzivan način, uvodi pojam regularnog izraza.

Definicija 2.1 — Regularan izraz. Kažemo da je R *regularan izraz* ako je R jednako:
(a) $\varnothing$;
(b) ε;
(c) simbolu iz alfabeta Σ;
(d) $(R_1 \cup R_2)$;
(e) $(R_1 \circ R_2) = (R_1 R_2)$;
(f) (R_1^*);
pri čemu su R_1, R_2 regularni izrazi.

Uočite da je i $R^+ := RR^*$ regularan izraz, ako je R regularan izraz.

Koja je veza između regularnih izraza i regularnih jezika, tj. koji jezik odgovara datom regularnom izrazu? Da bi dali pojašnjenje, samo ćemo pratiti definiciju regularnog izraza. Regularnim izrazima $\varnothing$, ε, 0, 1 odgovaraju jezici $\varnothing$, $\{\varepsilon\}$, $\{0\}$, $\{1\}$. Ako regularnim izrazima R_1 i R_2 odgovaraju jezici A_1 i A_2, tada regularnim izrazima $R_1 \cup R_2$, $R_1 R_2$, R_1^* odgovaraju jezici $A_1 \cup A_2$, $A_1 A_2$, A_1^*.

■ **Primjer 2.1** Regularnim izrazima ćemo pridruživati jezike.
(i) Regularnom izrazu 1^* odgovara jezik $\{1\}^* = \{\varepsilon, 1, 11, 111, \ldots\}$.
(ii) Regularnom izrazu (01) odgovara jezik $\{01\}$.
(iii) Regularnom izrazu $(01)(110)$ odgovara jezik $\{01\}\{110\} = \{01110\}$.
(iv) Regularnom izrazu $(0^*)(111)$ odgovara jezik $\{0\}^*\{111\} = \{111, 0111, 00111, 000111, 0000111, \ldots\}$.
(v) Regularnom izrazu $(0 \cup 1)^*$ odgovara jezik $\Sigma^* = \{0,1\}^* = \{x_1 \ldots x_k \mid k \geq 0, x_i \in \{0,1\}, i = 1, \ldots, k\}$, skup svih stringova nad alfabetom Σ.
(vi) $0^*1^* = \{\varepsilon, 0, 00, 000, \ldots\}\{\varepsilon, 1, 11, 111, \ldots\} = \{0^n 1^m \mid m, n \geq 0\}$.

■

Od sada poistovjećujemo regularne izraze sa odgovarajućim jezicima, tj. između njih stavljamo znak jednakosti. Ako regularnom izrazu R odgovara jezik A, pored *odgovara* koristimo i termine *opisuje*, *generiše*.

■ **Primjer 2.2** Opišimo regularan izraz koji generiše jezik $A = \{\omega \mid \omega$ sadrži 110 kao podstring$\}$.

Ako $\omega \in A$, tada ω ima oblik $\omega = \omega_1 110 \omega_2$, pri čemu $\omega_1, \omega_2 \in \Sigma^*$. Znači, $A = \Sigma^* 110 \Sigma^*$ ■

Slijedeći jezik je komplement prethodnog.

■ **Primjer 2.3** Opišimo regularan izraz koji generiše jezik $B = \{\omega \mid \omega$ ne sadrži

110 kao podstring$\}$.

Komplementiranje regularnih izraza nije jednostavno. Zasada ne dajemo standardni postupak komplementiranja, već ćemo analizirati B, te ga prikazati preko jednostavnijih jezika (koristeći uniju, nadovezivanje i zvijezdu).

Posmatrajmo sve jezike koji ne sadrže 110. Očigledno, među njima su 1^*, 0^*, 0^*1^*. Uočite da su prva dva podskup od trećeg, tako da je dovoljno navesti samo 0^*1^*.

Koji još jezici ne sadrže 110? Kako dati generalan opis stringa ω koji ne sadrži 110 ? Ono što je bitno za string ω je da, ako sadrži dvije uzastopne jedinice, iza ne smije dolaziti nula, tj. ω ne smije biti oblika $\omega = \ldots 11 \ldots 0 \ldots$, dok iza tačno jedne jedinice smije biti proizvoljan broj 0, tj. string ω može biti oblika $\omega = 0\ldots 0\underbrace{100\ldots 0}_{10^+}\underbrace{100\ldots 0}_{10^+}\ldots\underbrace{100\ldots 0}_{10^+}11\ldots 1 =$
$0\ldots 0\underbrace{10^+10^+\ldots 10^+}_{(10^+)^*}11\ldots 1 = \underbrace{0\ldots 0}_{0^*}(10^+)^*\underbrace{11\ldots 1}_{1^*} = 0^*(10^+)^*1^*.$

Znači, $B = 0^*(10^+)^*1^*$. ∎

Regularan jezik definišemo na analogan način kao i regularne izraze.

Definicija 2.2 — Regularan jezik. Za jezik A kažemo da je *regularan* ako je A jednako:
(a) $\varnothing$;
(b) $\{\varepsilon\}$;
(c) $\{a\}$, za neko $a \in \Sigma$;
(d) $A_1 \cup A_2$;
(e) A_1A_2;
(f) A_1^*,
pri čemu su A_1 i A_2 regularni jezici.

Znači, jezik je regularan akko ga neki regularan izraz generiše. U nastavku dajemo primjere regularnih jezika.

Uočimo da je i jezik $A^+ := AA^* = \bigcup_{k=1}^{+\infty} A^k$ regularan jezik. Ako $\varepsilon \notin A$, tada $A^+ = A^* \setminus \{\varepsilon\}$. Sa druge strane, za $\varepsilon \in A$ imamo da važi $A^+ = A^*$.

■ **Primjer 2.4** Jezik $A = \{\omega \mid \omega$ je neparne dužine$\}$ je regularan.

Posmatrajmo neki string neparne dužine, npr. $\omega_1 = 101101001$. Možemo ga podijeliti na slijedeći način: $\omega_1 = 10|11|01|00|1$. Znači, string neparne dužine se sastoji od određenog brojas stringova dužine 2 i na kraju se nalazi string

dužine 1. Svi stringovi dužine dva su: $00, 01, 10, 11$, a svi stringovi dužine 1 su: $0, 1$.

Prema tome, jezik A možemo napisati u obliku $A = \{00, 01, 10, 11\}^*\{0, 1\}$, pa je odgovarajući regularan izraz $(00 \cup 01 \cup 10 \cup 11)^*(0 \cup 1)$. ∎

∎ **Primjer 2.5** Jezik $A = \{\omega \mid \omega$ ima neparan broj nula$\}$ je regularan.

Posmatrajmo neki string sa neparnim brojem nula, npr. $\omega_1 = 11010111001101$ 1. Ovaj string možemo podijeliti na slijedeći način $\omega_1 = 11010|11100|11011$. U svakom segmentu, osim posljednjem, imamo po dvije nule. Pored nula, imamo i određeni broj jedinica.

Regularan izraz, koji generiše jezik A, je dat sa $(1^*01^*0)^*1^*01^*$.

∎

∎ **Primjer 2.6** Regularan izraz koji generiše jezik $A = \{\omega \mid \omega$ sadrži 01 kao podstring$\}$ je dat sa $\Sigma^*01\Sigma^*$. Znači, A je regularan jezik. ∎

∎ **Primjer 2.7** Za jezik $A = \{\omega \mid |\omega| \leq 5\}$, odgovarajući regularan izraz je $(\varepsilon \cup 0 \cup 1) \cup (\varepsilon \cup 0 \cup 1) \cup (\varepsilon \cup 0 \cup 1) \cup (\varepsilon \cup 0 \cup 1) \cup (\varepsilon \cup 0 \cup 1)$, pa je A regularan jezik. ∎

∎ **Primjer 2.8** Jezik $A = \{\varepsilon, 0\}$ je regularan, jer ga generiše regularan izraz $\varepsilon \cup 0$. ∎

∎ **Primjer 2.9** Jezik $A = \varnothing$ je, po definiciji, regularan. ∎

Kada posmatramo neku klasu objekata, koje se operacije mogu izvoditi nad objektima, a da ostanemo u istoj klasi? Preciznije, koje operacije možemo izvoditi nad regularnim jezicima, a opet dobijemo regularan jezik? O tome govori slijedeća tvrdnja.

Tvrdnja 2.1 Neka su A i B regularni jezici. Tada su i slijedeći jezici regularni:
- (a) $A \cup B$ - zatvorenost u odnosu na uniju;
- (b) AB - zatvorenost u odnosu na nadovezivanje;
- (c) A^* - zatvorenost u odnosu na zvjezdicu;
- (d) $A \cap B$ - zatvorenost u odnosu na presjek;
- (e) $\overline{A}$ - zatvorenost u odnosu komplement.

Dokaz. Dokaz za (a), (b) i (c) slijedi direktno iz Definicije 2.2(d)-(f).

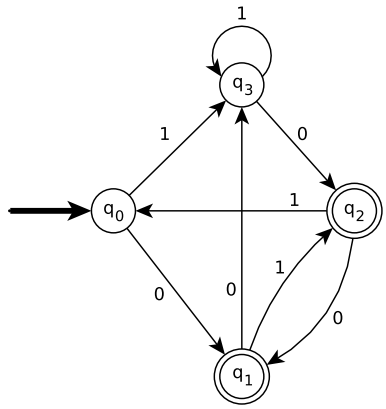
Da bi dokazali (d) i (e) koristićemo tzv. *konačne automate*. ■

K Ranije smo rekli da ćemo probleme na formalan način predstavljati pomoću jezika. Koji je problem pridružen jeziku $A = \{\omega \mid \omega \text{ ima neparan broj nula}\}$? Odgovarajući problem je *P:"Da li dati string ima neparan broj nula?"*.

Odgovor na ovo pitanje može biti "Da, ima neparan broj nula" ili "Ne, nema neparan broj nula". Inače, sve ćemo probleme postavljati u obliku na koji se može odgovoriti sa "Da/Ne".

2.2 Deterministički konačni automati

Započećemo sa primjerom determinističkog konačnog automata, koje ćemo još nazivati i DFA (skraćenica od *Deterministic Finite Automaton*).



Slika 2.1: Primjer determinističkog konačnog automata (DFA).

■ **Primjer 2.10** Na slici 2.1 je dat primjer dijagrama stanja DFA. Princip rada DFA je slijedeći. Uvijek počinje u istom početnom stanju i prima ulazni string. String se čita slijeva nadesno i prilikom svakog čitanja simbola automat prelazi na slijedeće stanje. DFA prestaje sa radom nakon što pročita string.

(*i*) Početno stanje je q_0 i označavamo ga sa ulaznom strelicom. Prilikom izvršavanja, automat uvijek počinje iz početnog stanja.

(*ii*) Automat može imati nula ili više završnih (prihvatnih) stanja. Završna stanja označavamo sa dvostrukim kružnicama. U ovom primjeru, završna stanja su q_1 i q_2.

(*iii*) Uzećemo da je $\omega = 1101$ ulazni string i ispisaćemo niz stanja kroz koje prolazi automat. Daćemo dva zapisa.
Prvi zapis je u obliku: $q_0 \xrightarrow{1} q_3 \xrightarrow{1} q_3 \xrightarrow{0} q_2 \xrightarrow{1} q_0$.
Niz stanja možemo zapisati i ovako: $q_0 1101$, $1q_3 101$, $11q_3 01$, $110q_2 1$, $1101q_0$. U ovom zapisu pročitani dio stringa ide ispred, a nepročitani iza oznake za stanje.

(*iv*) Ako, nakon čitanja stringa, automat završi u završnom stanju, tada kažemo da *automat prihvata string*. String $\omega = 1101$ dati automat ne prihvata, jer njegovim čitanjem završava u stanju q_0, koje nije prihvatno stanje.
Primjer stringa koji dati automat prihvata je $\omega_1 = 10$, jer nakon što ga pročita završi u stanju q_2, koje je prihvatno stanje.

∎

Da bi formalno definisali DFA, potrebno je da uočimo njegove elemente. Svaki DFA ima skup stanja i njega označavamo sa Q. U prethodnom primjeru imamo $Q = \{q_0, q_1, q_2, q_3\}$. Alfabet koji koristimo da formiramo stringove označavamo sa Σ. Ako ne naglasimo, podrazumijevamo da je $\Sigma = \{0,1\}$, što je slučaj i u prethodnom primjeru. Svaki DFA ima tačno jedno početno stanje i njega smo označili sa q_0. Skup završnih stanja označavamo sa F i imamo $F = \{q_1, q_2\}$. Možda najbitniji dio DFA je *funkcija tranzicije*. Nju označavamo sa δ i na dijagramu je predstavljena pomoću usmjerenih grana sa oznakama 0 i 1. Ovu funkciju možemo prikazati i tabelarno:

δ	q_0	q_1	q_2	q_3
0	q_1	q_3	q_1	q_2
1	q_3	q_2	q_0	q_3

Vidimo da je $\delta(q_0, 0) = q_1$ i $\delta(q_0, 1) = q_3$, te $\delta : Q \times \Sigma \to Q$.

Funkcija tranzicije se naziva *petlja* ako počinje i završava u istom stanju. U prethodnom primjeru imamo petlju kod stanja q_3 prilikom čitanja simbola 1.

Sada dajemo i formalnu definiciju DFA.

Definicija 2.3 — Deterministički konačni automat. Deterministički konačni automat je uređena petorka $(Q, \Sigma, \delta, q_0, F)$, pri čemu

1. Q je konačan skup stanja;
2. Σ je alfabet;

3. $\delta : Q \times \Sigma \to Q$ je funkcija tranzicije;
4. $q_0 \in Q$ je početno stanje;
5. $F \subseteq Q$ je skup završnih stanja.

Sama definicija DFA nam ne daje na koji način DFA funkcioniše. Za to nam je potrebna slijedeća definicija.

Posmatrajmo opet primjer 2.10 i ulazni string $\omega = 1101$. Sa ovim ulazom, automat je prolazio kroz niz stanja: $(q_0, q_3, q_3, q_2, q_0)$. Ovaj niz nazivamo *računanje DFA nad ulaznim stringom* ω.

Definicija 2.4 — Računanje DFA nad ulaznim stringom. Neka je $D = (Q, \Sigma, \delta, q_0, F)$ DFA, $\omega = \alpha_1 \dots \alpha_k$, $\alpha_i \in \Sigma$ $(i = 1, \dots, k)$. Za niz stanja $(q_0, \dots, q_k)$ iz Q kažemo da je *računanje DFA D nad ulaznim stringom* ω, ako je q_0 početno stanje od D i $q_{i+1} = \delta(q_i, \alpha_{i+1})$ $(i = 0, \dots, k-1)$.

Ako $q_k \in F$, kažemo da D *prihvata* ω. U suprotnom kažemo da D *ne prihvata* ili da *odbija* ω.

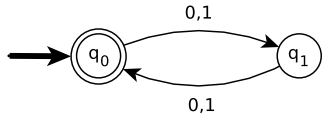
Postavlja se pitanje kakve probleme može rješavati DFA? U formalnom smislu, trebamo uspostaviti vezu između DFA i određene klase jezika. Zato je zanimljiv skup svih stringova koje određeni DFA prihvata.

Definicija 2.5 — Prepoznavanje jezika. Neka je D DFA i $A = \{\omega \mid D \text{ prihvata } \omega\}$. Tada kažemo da D prepoznaje jezik A i pišemo $A = L(D)$.

Neka je D DFA iz primjera 2.10. Tada imamo $1101 \notin L(D)$ i $10 \in L(D)$.

U slijedećim primjerima ćemo za date jezike konstruisati DFA koji ih prepoznaju.

■ **Primjer 2.11** DFA koji prepoznaje jezik $A = \{\omega \mid \omega \text{ je parne dužine}\}$ je dat na slici 2.2.



Slika 2.2: DFA koji prihvata string akko je parne dužine.

Objasnimo princip rada ovog automata. Automat se na početku nalazi u stanju q_0. Koji god simbol da pročita, preći će u stanje q_1. Zatim, nakon čitanja slijedećeg simbola prelazi u stanje q_0, itd. Znači, nakon pročitanog 1-og, 3-eg,

5-og,... simbola preći će u stanje q_1, dok nakon pročitanog 2-og, 4-og, 6-og,... simbola preći će u stanje q_0. Prema tome, automat će završiti u prihvatnom stanju, tj. u q_0, akko je string parne dužine.

U ovom primjeru smo uveli dva stanja: q_0 i q_1 - jedno za parna mjesta, a drugo za neparna mjesta u stringu. Stanja q_0 i q_1 smo iskoristili kao neku vrstu memorije - pomoću ovih stanja smo pamtili da li smo pročitali paran ili neparan broj simbola. Ovo je ideja koja se često koristi u konstruisanju DFA; stanja koristimo kao neku vrstu ograničene memorije, pomoću koje, na neki način, memorišemo osobinu pročitanog stringa. ∎

K Jezik predstavlja formalan način zapisivanja problema. Ako automat prepoznaje neki jezik, tada dati automat rješava odgovarajući problem.

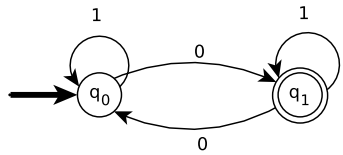
Posmatrajmo prethodni primjer. Imali smo jezik $A = \{\omega \mid \omega \text{ je parne dužine}\}$. Problem koji odgovara jeziku A je: "Da li je dati string parne dužine?".

Konstruisan je DFA (označimo ga sa D) koji prepoznaje jezik A. Znači D prihvata string akko je parne dužine. Ako automat završi u prihvatnom stanju, možemo reći da vraća *accept*, a ako završi u neprihvatnom stanju, vraća *reject*. Prema tome, $D(\omega) = accept$ ako je ω parne dužine, te $D(\omega) = reject$ ako je ω neparne dužine.

Znači, DFA D za proizvoljan string može odrediti da li je parne dužine ili ne. Na ovaj način, automat D rješava problem: "Da li je dati string parne dužine?".

∎ **Primjer 2.12** Konstruisati DFA koji prepoznaje jezik $A = \{\omega \mid \omega$ ima neparan broj nula$\}$.

Rješenje je dato na slici 2.3.



Slika 2.3: DFA koji prihvata string akko ima neparan broj nula.

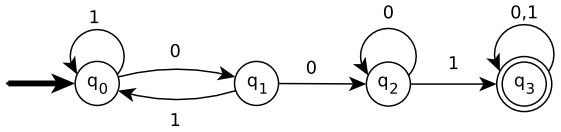
Pojasnimo rad ovog automata. Šta je nama bitno kod stringova koje posmatramo? Bitno nam je samo koliko nula ima dati string, tj. da li ih ima paran ili neparan broj. Znači, jedinice nam nisu bitne. Prema tome, automat

ne mora reagovati kada pročita jedinicu, tj. ne mora promijeniti stanje. Zato imamo petlju kod svih stanja u slučaju da je pročitani simbol 1.

Automat reaguje, tj. mijenja stanje, samo ako pročita 0. Da li je pročitao paran ili neparan broj nula, memoriše pomoću stanja q_0 i q_1 - slično kao u primjeru 2.11. ∎

∎ **Primjer 2.13** Za jezik $A = \{\omega \mid \omega \text{ sadrži } 001 \text{ kao podstring}\}$ konstruiši DFA koji ga prepoznaje.

Odgovarajući DFA je dat na slici 2.4.



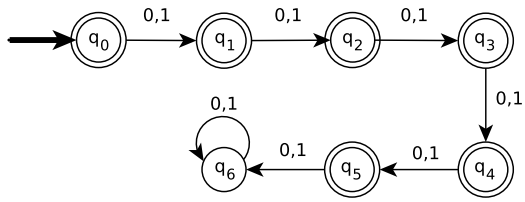
Slika 2.4: DFA koji prihvata string akko ima 001 kao podstring.

Objasnimo rad ovog automata. Na početku, automat ignoriše jedinice, jer mu ništa ne znače. Zato imamo petlju na stanju q_0.

Prva značajna informacija mu je ako naiđe na 0. U tom slučaju, automat reaguje, tj. mijenja stanje i prelazi na q_1. U slučaju da nakon prve nule pročita jedinicu, ništa od pročitanog mu ne vrijedi i mora krenuti od početka. Zato se vraća na početno stanje.

Ukoliko nakon prve nule, pročita i drugu nulu, dobija još jednu značajnu informaciju, pa zato mijenja stanje, tj. prelazi u q_2. Ukoliko ima nula i nakon druge nule, one mu ništa ne vrijede, te zato ne reaguje, tj. ne mijenja stanje. Automat će reagovati samo ako nakon nekog vremena naiđe na jedincu. To će značiti da ulazni string ima 001 kao podstring i automat će prihvatiti ulazni string. Ostatak stringa automat će da pročita, ali će ostati u prihvatnom stanju - zato imamo petlju na q_3. ∎

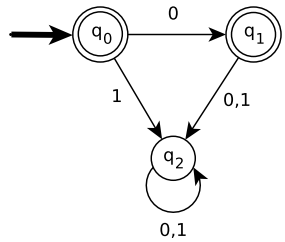
∎ **Primjer 2.14** Dat je jezik $A = \{\omega \mid |\omega| \leq 5\}$. Traženi DFA je dat na slici 2.5.



Slika 2.5: DFA koji prihvata string akko je dužine manje od 6.

■

■ **Primjer 2.15** DFA koji prepoznaje jezik $A = \{\varepsilon, 0\}$ je dat na slici 2.6.:



Slika 2.6: DFA koji prihvata string akko je jednak 0 ili ε.

■

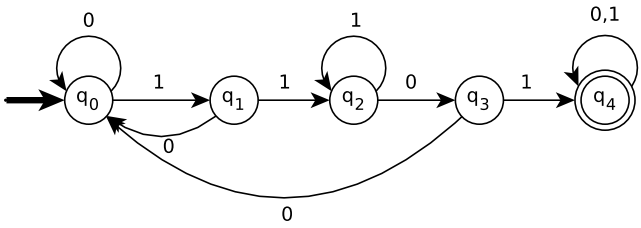
■ **Primjer 2.16** Posmatrajmo jezik $A = \varnothing$.

Koji DFA prepoznaje ovaj jezik? To je automat koji ne prihvata nijedan string. Da bi to bilo ispunjeno, dovoljno je da automat nema prihvatnih stanja. Znači, bilo koji automat bez prihvatnih stanja prepoznaje jezik A.

Možemo reći i generalnije; bilo koji DFA, kod kojeg ne postoji usmjereni put od početnog stanja do nekog od završnih stanja, prepoznaje jezik A. ■

■ **Primjer 2.17** Konstruiši DFA koji prepoznaje jezik $A = \{\omega \mid \omega$ sadrži 1101 kao podstring$\}$.

Rješenje je dato na slici 2.7.



Slika 2.7: DFA koji prihvata string akko sadrži 1101 kao podstring.

■

K Iz primjera i definicije smo vidjeli da DFA čita string sa lijeva nadesno. Nije nepravilno ako se string čita i sa desna nalijevo. Razlog za to leži u činjenici da jezik A prepoznaje neki DFA, akko jezik A^R prepoznaje neki DFA (vidi zadatak 2.10).

U slijedećem primjeru konstruišemo DFA koji čita string sa desna nalijevo.

■ **Primjer 2.18** Posmatrajmo problem sabiranja binarnih brojeva. Tako npr. važi

$$\begin{array}{cccc} & 1 & 1 & 0 \\ + & & 1 & 0 \\ \hline = & 1 & 0 & 0 \end{array}.$$ Ovo možemo zapisati i kao: $$\begin{array}{cccc} & 0 & 1 & 1 & 0 \\ + & 0 & 0 & 1 & 0 \\ \hline = & 1 & 0 & 0 & 0 \end{array},$$

odnosno $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}$.

Ovo je motivacija za slijedeći alfabet i jezik. Neka je

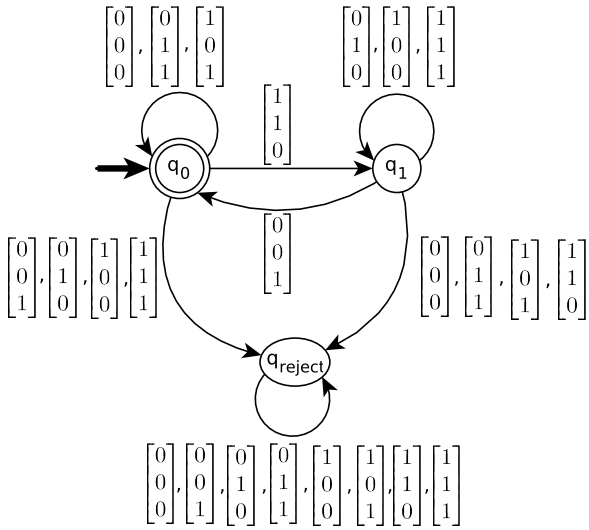
$$\Sigma = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix} \right\}$$

i $X = \{\omega \mid \omega \in \Sigma^*,$ pričemu je binarni zbir prve dvije vrste od ω jednak trećoj$\}$.

Tako npr. $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix} \in X$, dok $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix} \notin X$.

Znači, simboli alfabeta su vektori-kolone, a string iz Σ^* je niz vektora-kolona.

Zadatak je da se konstruiše DFA koji prepoznaje jezik X. Rješenje je dato na slici 2.8. U ovom primjeru podrazumijevamo da automat čita string sa desna nalijevo.



Slika 2.8: DFA koji rješava problem zbira binarnih brojeva.

Prilikom sabiranja binarnih brojeva po kolonama, u narednu kolonu možemo prenijeti 0 ili 1. Ako prenosimo 0, prelazimo u stanje q_0, a ako prenosimo 1, prelazimo u stanje q_1. Znači, pomoću stanja q_0 i q_1 pamtimo koji smo broj prenijeli iz prethodne kololone.

Objasnimo konstrukciju DFA na nekoliko primjera.

Ako se nalazimo u stanju q_0 i čitamo simbol $\begin{bmatrix}0\\1\\1\end{bmatrix}$, to znači da smo iz prethodne kolone prenijeli 0 i imamo $0+0+1=1$. Prema tome u slijedeću kolonu prenosimo 0, pa ostajemo u stanju q_0.

Ako se nalazimo u stanju q_0 i čitamo simbol $\begin{bmatrix}1\\1\\0\end{bmatrix}$, to znači da smo iz prethodne kolone prenijeli 0 i imamo $0+1+1+0=10$. Prema tome u slijedeću kolonu prenosimo 1, pa prelazimo u stanje q_1.

Ako se nalazimo u stanju q_0 i čitamo simbol $\begin{bmatrix}0\\1\\0\end{bmatrix}$, to znači da smo iz prethodne kolone prenijeli 0 i imamo $0+1+0=1$. Ovdje imamo nemoguću situaciju, jer simbol u trećoj vrsti (tj. 0) ne odgovara simbolu koji smo dobili sabiranjem (tj. 1). Zaključujemo da zbir prve dvije vrste ulaznog stringa, nije jednak trećoj vrsti, pa automat ovaj string odbija. Zato prelazimo u stanje q_{reject}.

Ako se nalazimo u stanju q_1 i čitamo simbol $\begin{bmatrix}1\\0\\0\end{bmatrix}$, to znači da smo iz prethodne kolone prenijeli 1 i imamo: $1+1+0+0=10$. Prema tome u slijedeću kolonu prenosimo 1, pa ostajemo u stanju q_1.

Slično važi i za sve ostale situacije.

Pošto na kraju sabiranja, ne prenosimo 1 u slijedeću kolonu, stanje q_0 je završno stanje. ∎

Zadatak 2.1 — Za samostalan rad. Stringove $\omega_1=\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}\in X$ i $\omega_2=\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\notin X$ uzeti kao ulazne stringove za automat iz prethodnog primjera, te ispisati odgovarajući niz stanja. ∎

U slijedeća tri primjera ćemo konstruisati presjek, uniju i komplement DFA na datim primjerima. Postupci su generalni i mogu se primijeniti na bilo koji DFA.

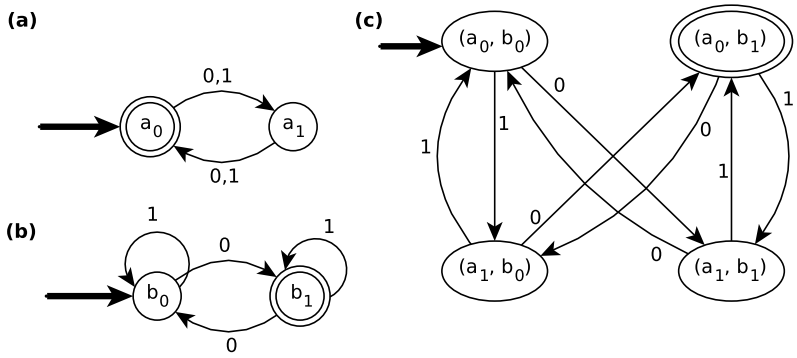
∎ **Primjer 2.19 — Presjek DFA.** Neka su D_1 i D_2 DFA. Na primjeru ćemo demonstrirati kako konstruisati njihov presjek, tj. kako konstruisati DFA D tako da je $L(D)=L(D_1)\cap L(D_2)$.

Neka su $D_1=(Q_1,\Sigma,\delta_1,a_0,F_1)$ i $D_2=(Q_2,\Sigma,\delta_2,b_0,F_2)$ DFA koji prepoznaju jezike $A=\{\omega \mid \omega \text{ je parne dužine}\}$ i $B=\{\omega \mid \omega \text{ ima neparan broj nula}\}$ (primjeri 2.11 i 2.12). Na koji način ćemo konstruisati $D=(Q,\Sigma,\delta,q_0,F)$?

Da bi D prihvatio neki string ω, moraju ga prihvatiti automati D_1 i D_2. Automate D_1 i D_2 ćemo pokrenuti paralelno i posmatrati stanja kroz koja prolaze. Ako oba automata završe na prihvatnom stanju, D će prihvatiti ω.

Na koji način ćemo posmatrati paralelno stanja kroz koja prolaze D_1 i D_2? To ćemo uraditi tako što će svako stanje automata D biti uređen par stanja iz D_1 i D_2. Prelazeći kroz stanja automata D zapravo ćemo prolaziti kroz stanja automata D_1 i D_2 paralelno (slika 2.9).

Znači, $Q = Q_1 \times Q_2 = \{(a_0,b_0),(a_0,b_1),(a_1,b_0),(a_1,b_1)\}$.



Slika 2.9: **(a)** DFA koji prihvata string akko je parne dužine. **(b)** DFA koji prihvata string akko ima neparan broj 0. **(c)** DFA koji je presjek automata pod (a) i (b). Prema tome, prihvata string akko je parne dužine i ima neparan broj nula.

Da bi D prihvatio string, oba automata D_1 i D_2 moraju završiti u prihvatnim stanjima. Znači, $F = F_1 \times F_2 = \{(a,b) \mid a \in F_1, b \in F_2\} = \{(a_0,b_1)\}$.

Za funkciju tranzicije važi:

$$\delta((a,b),\alpha) = (\delta_1(a,\alpha),\delta_2(b,\alpha)).$$

Tako, npr. imamo: $\delta((a_0,b_0),0) = (\delta_1(a_0,0),\delta_2(b_0,0)) = (a_1,b_1)$. Ponavljajući prethodni postupak za svako stanje i svaki simbol alfabeta, dobijamo tabelu:

δ	(a_0,b_0)	(a_0,b_1)	(a_1,b_0)	(a_1,b_1)
0	(a_1,b_1)	(a_1,b_0)	(a_0,b_1)	(a_0,b_0)
1	(a_1,b_0)	(a_1,b_1)	(a_0,b_0)	(a_0,b_1)

.

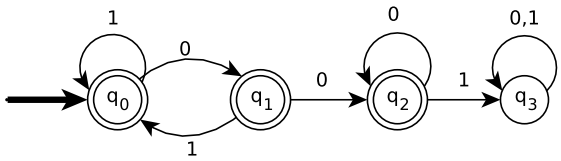
■

■ **Primjer 2.20 — Unija DFA.** Neka su A i B jezici iz primjera 2.19. Konstrukcija unije automata D_1 i D_2 je skoro identična konstrukciji presjeka automata, pa se ostavlja čitaocu.

Jedina razlika je kod završnih stanja. Naime, D prihvata string akko ga prihvata D_1 ili D_2. Prema tome $F = \{(a,b) \mid a \in F_1 \text{ ili } b \in F_2\} = \{(a_0,b_0),(a_0,b_1),(a_1,b_1)\}$. Stanja, kao i funkcija tranzicije su ista kao u primjeru 2.19. ∎

∎ **Primjer 2.21 — Komplement DFA.** Neka su A i D_1 jezik i DFA iz primjera iz primjera 2.13 i slike 2.4. Konstruišimo komplement od D_1, tj. konstruišimo automat $D' = (Q', \Sigma, \delta', a', F')$ tako da važi $L(D') = \overline{L(D_1)} = \overline{A} = \{\omega \mid \omega \text{ ne sadrži 001 kao podstring}\}$.

Konstrukcija je jednostavna. Stanja i funkcije tranzicije su nepromijenjene, tj. važi: $Q' = Q_1, \delta' = \delta_1, a' = a_0$. Završna stanja mijenjaju uloge, tj. stanja koja nisu bila završna postaju završna, a ona koja su bila završna to više nisu. Formalnije, $F' = Q \backslash F$ (slika 2.10). ∎



Slika 2.10: DFA koji je komplement DFA iz primjera 2.13 i slike 2.4.

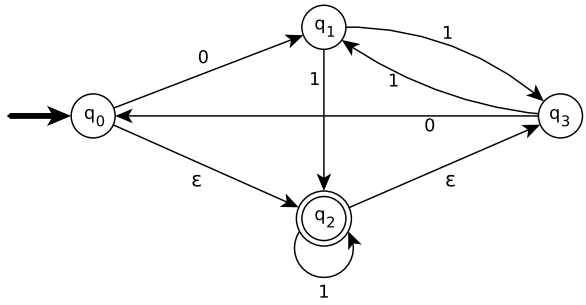
Zadatak 2.2 — Za samostalan rad. Konstruisati razliku dva DFA. Ako su dati automati D_1 i D_2, potrebno je konstruisati DFA D takav da važi $L(D) = L(D_1) \backslash L(D_2)$. ∎

Da bi povezali konačne automate i regularne jezike, te na taj način dokazali neke osobine regularnih jezika, potrebno je uvesti malo fleksibilniju verziju DFA. Zato definišemo *nedeterminističke konačne automate* ili skraćeno NFA (eng. *Nondeterministic Finite Automaton*).

2.3 Nedeterministički konačni automati

U ovom odjeljku uvodimo automate koji su naizgled moćniji od DFA, ali kasnije ćemo dokazati da su ekvivalentni sa DFA.

∎ **Primjer 2.22** Dat je nedeterministički konačni automat (*NFA - Nondeterministic Finite Automaton*) (slika 2.11).



Slika 2.11: Primjer nedeterminističkog konačnog automata (NFA).

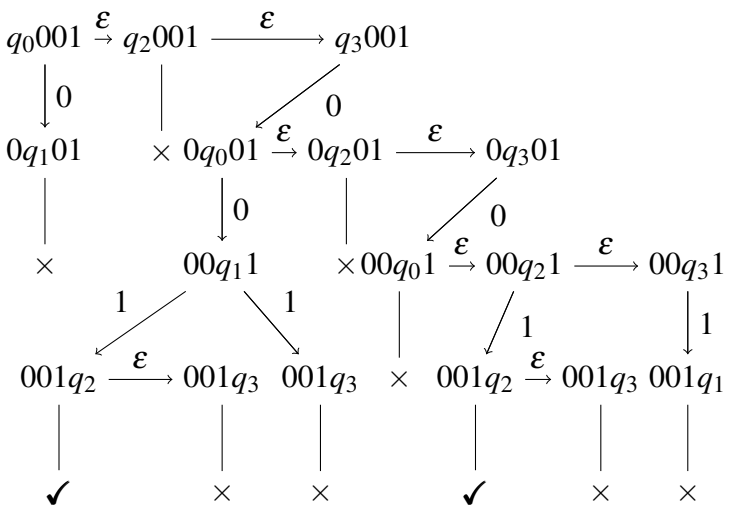
1. Uočimo neke razlike u onosu na DFA. NFA može imati više strelica za jedan znak - vidi stanje q_1 koje ima dvije strelice za znak 1. Strelicama može biti pridružen simbol ε, koji predstavlja prazan string. To znači da sa jednog stanja možemo preći na drugo stanje bez čitanja simbola sa ulaznog stringa. Ukoliko, za pročitani simbol, dato stanje nema definisanu izlaznu strelicu, tada NFA odbija string.

2. Za ulazni string $\omega_1 = 001$ daćemo niz stanja.

Ukoliko stanju NFA odgovara više izlaznih strelica za jedan simbol, tada se NFA dijeli na više NFA (za svaku strelicu po jedan) i svaka kopija NFA nastavlja sa izvršavanjem, nezavisno od drugih kopija. Slična je situacija i kada dođemo u stanje iz kojeg izlazi strelica sa ε. Nastaje onoliko kopija automata koliko imamo stanja u koje možemo doći idući samo preko ε strelica.

Niz stanja, datog NFA za ulazni string ω_1, je dat na slici 2.12. Vidimo da struktura koju smo dobili zapravo nije niz, već stablo. Ovako konstruisano stablo ćemo nazivati *računanje NFA nad ulaznim stringom*. Listove dobijenog stabla označavamo sa $\times$ ili $\checkmark$, zavisno od toga da li je ulazni string odbijen ili prihvaćen. Put od korjena stabla do jednog lista nazivamo *grana računanja*. Ukoliko barem jedna grana računanja prihvata string, kažemo da dati NFA prihvata dati ulazni string. Da bi NFA odbio ulazni string, sve grane računanja moraju odbiti dati string.

Znači, dati NFA prihvata string ω_1.



Slika 2.12: Primjer računanja NFA nad ulaznim stringom.

3. δ je dato kao:

	0	1	ε
q_0	$\{q_1\}$	$\varnothing$	$\{q_2\}$
q_1	$\varnothing$	$\{q_2, q_3\}$	$\varnothing$
q_2	$\varnothing$	$\{q_2\}$	$\{q_3\}$
q_3	$\{q_0\}$	$\{q_1\}$	$\varnothing$

Uočimo da sada važi $\delta : Q \times \Sigma_\varepsilon \to \mathscr{P}(Q)$, pri čemu je $\mathscr{P}(Q)$ partitivni skup skupa Q.

■

U slijedeće dvije definicije formalizujemo prethodna razmatranja vezana za NFA.

Definicija 2.6 — Nedeterministički konačni automat. Nedeterministički konačni automat je uređena petorka $(Q, \Sigma, \delta, q_0, F)$, pri čemu

1. Q je konačan skup stanja;
2. Σ je konačan alfabet;
3. $\delta : Q \times \Sigma_\varepsilon \to \mathscr{P}(Q)$ je funkcija tranzicije i $\mathscr{P}(Q)$ je partitivni skup skupa Q;
4. $q_0 \in Q$ je početno stanje;
5. $F \subseteq Q$ je skup završnih stanja.

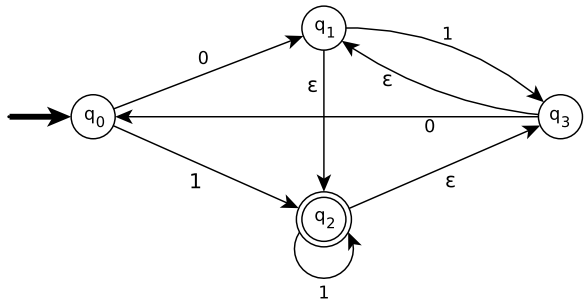
NFA možemo shvatiti kao mašinu koja je u stanju da vrši paralelno raču-

nanje. Stablo na slici 2.12 prikazuje na koji način NFA vrši računanje. Radi jednostavnije definicije, umjesto stabla, posmatraćemo skup nizova: $\Re = \{(q_0,q_1),(q_0,q_2),\ (q_0,q_2,q_3,q_0,q_1,q_2),\ (q_0,q_2,q_3,q_0,q_1,q_2,q_3),\ (q_0,q_2,q_3,q_0,q_1,q_3),\ (q_0,q_2,q_3,q_0,q_2),\ (q_0,q_2,q_3,q_0,q_2,q_3,q_0),\ (q_0,q_2,q_3,q_0,q_2,q_3,q_0,q_2,q_2),\ (q_0,q_2,q_3,q_0,q_2,q_3,q_0,q_2,q_2,q_3),\ (q_0,q_2,q_3,q_0,q_2,q_3,q_0,q_2,q_3,q_1)\}$.

Zadatak 2.3 — Za samostalan rad. Posmatrajmo NFA sa slike 2.13. Dati jedan ulazni string za koji NFA nikada ne staje sa radom. Kažemo da dati NFA *ulazi u beskonačnu petlju*.

Možemo zaključiti da NFA ulazi u beskonačnu petlju, za neki ulazni string, akko ima ciklus sastavljen od ε-strelica.

Na koji način biste izbjegli beskonačnu petlju kod NFA? ■



Slika 2.13: Primjer NFA koji ulazi u beskonačnu petlju za određeni ulazni string.

Definicija 2.7 — Računanje NFA nad ulaznim stringom. Neka je $N = (Q,\Sigma,\delta,q_0,F)$ NFA i $\omega = \alpha_1 \ldots \alpha_n$ ulazni string. Sa $\Re_1$ označimo skup svih konačnih nizova $(q_0,\ldots,q_k)$ stanja od N takvih da važi:

1. q_0 je početno stanje od N;
2. $q_{i+1} \in \delta(q_i,\beta_{i+1})$, $\beta_{i+1} \in \Sigma_\varepsilon$ $(i = 0,\ldots,k-1)$;
3. $\beta_1 \ldots \beta_k = \omega$ ili $(\beta_1 \ldots \beta_k = \alpha_1 \ldots \alpha_s, s < n$ i $\delta(q_k,\alpha_{s+1}) = \varnothing)$.

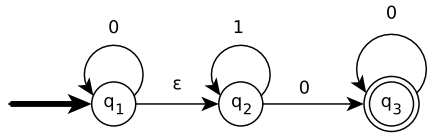
Sa $\Re_2$ označimo skup svih beskonačnih nizova $(q_0,\ldots,q_k,\ldots)$ stanja od N takvih da važi:

1. q_0 je početno stanje od N;
2. $q_{i+1} \in \delta(q_i,\beta_{i+1})$, $\beta_{i+1} \in \Sigma_\varepsilon$ $(i = 0,1,\ldots)$;
3. $\beta_1 \ldots \beta_k = \alpha_1 \ldots \alpha_s, s \leq n$ i $\beta_j = \varepsilon, \forall j > k$.

Tada $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2$ nazivamo *računanje automata N nad ulaznim stringom* ω. Ako postoji niz $(q_0, \ldots, q_k) \in \mathfrak{R}$ takav da je q_k završno stanje i $\beta_1 \ldots \beta_k = \omega$, tada kažemo da *automat N prihvata string* ω. U suprotnom kažemo da *automat N odbija (ili ne prihvata) string* ω. Element (tj. niz) iz $\mathfrak{R}$ nazivamo *grana računanja* automata N nad ulaznim stringom ω.

■ **Primjer 2.23** Dati NFA sa tri stanja koji prepoznaje jezik $0^*1^*0^+$.

Automat je dat na slici 2.14.



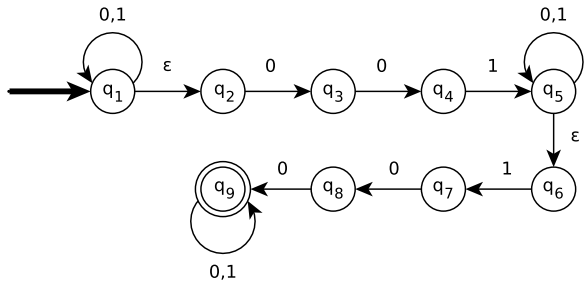
Slika 2.14: NFA sa tri stanja koji prepoznaje $0^*1^*0^+$.

Upotrebom petlje na stanju q_1 možemo konstruisati proizvoljan niz nula, tj. element iz 0^*. Prelaskom preko ε-strelice ne dodajemo dodatne simbole. Upotrebom petlje na stanju q_2 konstruišemo proizvoljan niz jedinica, tj. element iz 1^*. Prelaskom preko slijedeće strelice dodajemo jednu nulu, što zajedno sa petljom na stanju q_3 daje proizvoljan niz nula sa barem jednom nulom, tj. element iz 0^+. ■

■ **Primjer 2.24** Dati NFA koji prepoznaje jezik $A = \Sigma^*001\Sigma^*100\Sigma^* = \{\omega \mid \omega$ ima $001\omega'100$ kao podstring, za neko $\omega' \in \Sigma^*\}$.

Sličnu funkcionalnost imate u MS Word-u kada tražite tekst sa specijalnim simbolima. Ovom primjeru bi odgovaralo traženje riječi "$001*100$", pri čemu $*$ mijenja proizvoljan tekst.

Dijagram stanja traženog NFA je dat na slici 2.15.



Slika 2.15: NFA koji prepoznaje jezik $A = \{\omega \mid \omega$ ima $001\omega'100$ kao podstring, za neko $\omega' \in \Sigma^*\}$.

Ovu mašinu označimo sa N. Važi da N prihvata neki string ω akko $\omega = \omega_1 001 \omega' 100 \omega_2$, za neke $\omega_1, \omega', \omega_2 \in \Sigma^*$. Stanje q_1 služi da pročita ω_1. Slično, u stanju q_5 čitamo ω', te u q_9 čitamo ω_2. Uloga ostalih stanja je jasna - pomoću njih čitamo 001 i 100.

Kako N određuje na koje mjestu prestaje sa čitanjem prefiksa ω_1, te započinje sa čitanjem 001? Ova mašina to ne može odrediti. Kao što smo vidjeli u primjeru 2.22, ako iz trenutnog stanja izlazi ε-strelica, tada mašina grana na dva slučaja: kada uzimamo ε-strelicu i kada ne uzimamo. Na ovaj način dobijamo više slučajeva, tj. grana računanja.

Dovoljno je da samo jedna grana računanja prihvati string. Uzmimo npr. string $\omega = 1100010010011$, pri čemu je $\omega_1 = 110$, $\omega' = 00$ i $\omega_2 = 11$. Jedna od grana računanja, koja prihvata dati string, je data sa: $q_1 1100010010011$, $1q_1 100010010011$, $11q_1 00010010011$, $110q_1 0010010011$, (sada idemo preko ε-strelice) $110q_2 0010010011$, $1100q_3 010010011$, $11000q_4 10010011$, $110001 q_5 0010011$, $1100010q_5 010011$, $11000100q_5 10011$, (opet prelazimo preko ε-strelice) $11000100q_6 10011$, $110001001q_7 0011$, $1100010010q_8 011$, $11000100 100q_9 11$, $110001001001q_9 1$, $1100010010011q_9$. String je pročitan do kraja i završili smo u prihvatnom stanju. Na ovaj način smo odredili jednu granu računanja koja prihvata dati string, pa i N prihvata dati string.

Ako želimo demonstrirati da neka mašina ne prihvata dati string, onda to možemo uraditi tako što konstruišemo cijelo stablo računanja (kao u primjeru 2.22). Drugi način je da dokažemo da mašina neće prihvatiti string, posmatranjem raznih slučajeva, ali ovo nećemo raditi zbog složenosti.

Na ovom primjeru vidimo da su NFA fleksibilnije mašine od DFA. ∎

Slijedeći pojam objašnjava kada su dvije mašine suštinski jednake.

Definicija 2.8 — Ekvivalentnost mašina. Za dvije mašine M_1 i M_2 kažemo da su *ekvivalentne* ako važi $L(M_1) = L(M_2)$.

Očigledno je svaki DFA ujedno i NFA bez ε i višestrukih strelica. To znači da je skup svih DFA sadržan u skupu svih NFA. Postavlja se pitanje da li postoji i koji je to jezik kojeg neki NFA prepoznaje, a nijedan DFA ne može prepoznati? Uskoro ćemo dokazati da takav jezik ne postoji. Za svaki jezik kojeg neki NFA prepoznaje možemo konstruisati neki DFA koji će ga prepoznati. Intuitivno, ovo znači da NFA nisu "moćniji" automati; prepoznaju istu klasu jezika kao i DFA.

U slijedećem primjeru ilustrujemo kako proizvoljni NFA konvertovati u ekvivalentan DFA.

■ **Primjer 2.25** Za NFA iz primjera 2.22 konstruisati njemu ekvivalentan DFA.

NFA je neka vrsta paralelnog računanja, tj. u jednom trenutku, automat se može nalaziti u više stanja. Konstruisanje ekvivalentnog DFA predstavlja pretvaranje paralelnog računanja u "obično", sekvencijalno računanje. To radimo tako što skup svih stanja u kojima se DFA nalazi shvatimo kao jedno stanje.

Sa D označimo DFA ekvivalentan sa N, pri čemu je N NFA iz primjera 2.22. Skup stanja automata D je $\mathscr{P}(Q(N))$. Početno stanje od D je skup koji se sastoji od početnog stanja N i svih stanja do kojih možemo doći prelazeći samo preko ε-strelica (slika 2.16). Završna stanja od D su stanja kojima odgovaraju skupovi koji sadrže barem jedno završno stanje automata N.

Iz automata N imamo:

- $q_0 \xrightarrow{0} \{q_1\}$, $q_0 \xrightarrow{1} \varnothing$;
- $q_1 \xrightarrow{0} \varnothing$, $q_1 \xrightarrow{1} \{q_2, q_3\}$;
- $q_2 \xrightarrow{0} \varnothing$, $q_2 \xrightarrow{1} \{q_2, q_3\}$;
- $q_3 \xrightarrow{0} \{q_0, q_2, q_3\}$, $q_3 \xrightarrow{1} \{q_1\}$.

Na osnovu prethodne liste formiramo funkciju tranzicije automata D - vidi tabelu 2.1.

Vrijednost npr. od $\delta_D(\{q_1, q_3\}, 0)$ odredimo kao uniju vrijednosti $\delta(q_1, 0)$ i $\delta(q_3, 0)$, a što je jednako $\varnothing \cup \{q_0, q_2, q_3\} = \{q_0, q_2, q_3\}$. Dijagram stanja traženog DFA je dat na slici 2.16.

Sa $E(q)$ označimo skup svih stanja automata N do kojih možemo doći polazeći od q i prelazeći samo preko ε-strelica, zajedno sa stanjem q. Tako imamo $E(q_0) = \{q_0, q_2, q_3\}$, $E(q_1) = \{q_1\}$, $E(q_2) = \{q_2, q_3\}$, $E(q_3) = \{q_3\}$.

δ_D	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\{q_0\}$	$\{q_1\}$	$\varnothing$
$\{q_1\}$	$\varnothing$	$\{q_2,q_3\}$
$\{q_2\}$	$\varnothing$	$\{q_2,q_3\}$
$\{q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_1\}$
$\{q_0,q_1\}$	$\{q_1\}$	$\{q_2,q_3\}$
$\{q_0,q_2\}$	$\{q_1\}$	$\{q_2,q_3\}$
$\{q_0,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_1\}$
$\{q_1,q_2\}$	$\varnothing$	$\{q_2,q_3\}$
$\{q_1,q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_2,q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_0,q_1,q_2\}$	$\{q_1\}$	$\{q_2,q_3\}$
$\{q_0,q_1,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_0,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_1,q_2,q_3\}$	$\{q_0,q_2,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_0,q_1,q_2,q_3\}$	$\{q_0,q_1,q_2,q_3\}$	$\{q_1,q_2,q_3\}$

Tablica 2.1: Funkcija tranzicije za DFA iz primjera 2.25.

Kada iz automata sa slike 2.16 izbacimo beskorisna stanja dobijemo automat na slici 2.17. Za stanje q' kažemo da je *beskorisno stanje* ako ne postoji usmjeren put od početnog stanja do q'.

■

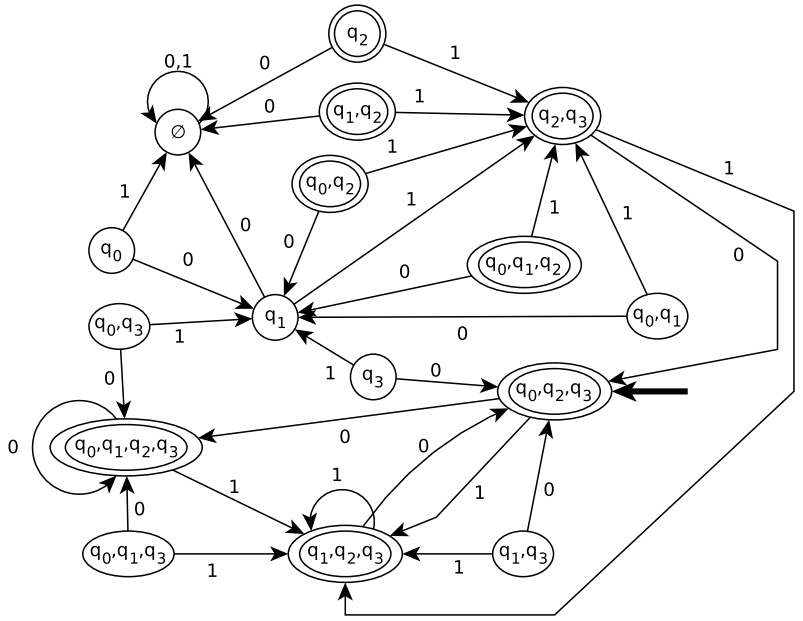
K Uočite da, u prethodnom primjeru, nismo morali određivati funkciju tranzicije za sva stanja sa slike 2.16, već samo za stanja koja se nalaze na slici 2.17.

Podsjetimo se da je svaki DFA ujedno i NFA. Slijedeća teorema govori da, u određenom smislu, važi i obratno.

Teorema 2.1 — Ekvivalentnost NFA i DFA. Za svaki NFA postoji njemu ekvivalentan DFA.

Dokaz. Ideja dokaza je opisana u primjeru 2.25, pa predlažemo da ga pogledate prije čitanja ovog dokaza.

Neka je $N=(Q,\Sigma,\delta,q_0,F)$ NFA. Konstruišimo DFA $D=(Q',\Sigma,\delta',q_0',F')$ na slijedeći način.



Slika 2.16: DFA ekvivalentan sa NFA iz primjera 2.22.

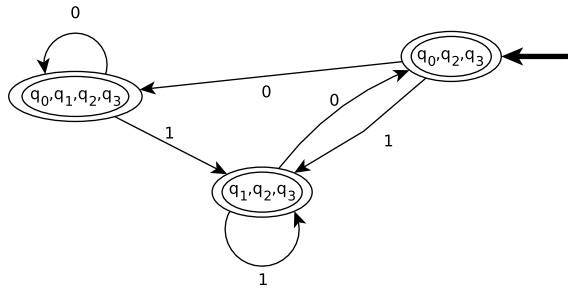
Sa $E(q)$ ($q \in Q$) označimo skup stanja automata N do kojih možemo doći počinjući u q i prelazeći samo preko ε-strelica zajedno sa stanjem q. Očigledno, $q \in E(q)$ za svako $q \in Q$.

Uzmimo $Q' = \mathscr{P}(Q)$, tj. stanja automata D su podskupovi od Q.

Početno stanje definišemo kao $q'_0 = E(q_0)$, tj. skup koji se sastoji od stanja automata N do kojih možemo doći počinjući od q_0 i prelazeći samo preko ε-strelica.

Skup završnih stanja automata D definišemo kao $F' = \{X \mid X \subseteq Q, X \cap F \neq \varnothing\}$. Drugim riječima, stanje $X \in \mathscr{P}(Q)$ automata D je završno samo ako sadrži barem jedno završno stanje automata N.

Sada definišimo i najbitniji dio svakog automata, funkciju tranzicije. Neka je $X \in \mathscr{P}(Q)$ (tj. $X \subseteq Q$). Uzmimo $\delta'(X,\alpha) = \bigcup_{q \in X} E(\delta(q,\alpha))$. Pojasnimo posljednju relaciju. Ako imamo skup stanja X u kojima se automat N nalazi u određenom trenutku, tada skup stanja u kojima se automat N nalazi u slijedećem potezu, nakon čitanja simbola α, označimo sa $\delta'(X,\alpha)$. Kako odrediti $\delta'(X,\alpha)$? Svako stanje q automata N se preslikava u $\delta(q,\alpha)$, te u sva stanja do kojih možemo doći počinjući u $\delta(q,\alpha)$ i prelazeći samo preko ε-



Slika 2.17: DFA ekvivalentan sa DFA sa slike 2.16, dobijen izbacivanjem beskorisnih stanja.

strelica. Ukratko, iz stanja q prelazimo u skup stanja $E(\delta(q,\alpha))$. Ponavljajući ovaj postupak za svako $q \in X$, dobijamo $\delta'(X,\alpha) = \bigcup_{q\in X} E(\delta(q,\alpha))$.

Dokažimo da su N i D ekvivalentni, tj. da važi $L(N) = L(D)$. Jednakost skupova se dokazuje u dva smjera: $\omega \in L(N) \implies \omega \in L(D)$, te $\omega \in L(D) \implies \omega \in L(N)$.

Smjer 1.

Neka $\omega = \alpha_1 \dots \alpha_n \in L(N)$, $(\alpha_i \in \Sigma,\ i = 1,\dots,n)$. To znači da automat N prihvata string ω. Dokažimo da $\omega \in L(D)$. Pošto N prihvata string ω, postoji niz stanja $(q_0,\dots,q_k)$ tako da važi:

1. q_0 je početno stanje of N;
2. $q_{i+1} \in \delta(q_i,\beta_{i+1})$, $\beta_{i+1} \in \Sigma_\varepsilon$, $(i = 0,\dots,k-1)$;
3. $\beta_1 \dots \beta_k = \alpha_1 \dots \alpha_n$;
4. $q_k \in F$.

Dokažimo da automat D prihvata ω, tj. da postoji niz stanja u D, koji string ω prevode od početnog stanja automata D, do stanja automata D koji sadrži q_k. Dokaz ćemo sprovesti indukcijom po n.

Neka je $n = 0$. Tada je $\omega = \varepsilon$ i $\beta_1 = \dots = \beta_k = \varepsilon$. Tada do stanja $q_1,\dots,q_k$ možemo doći počinjući od q_0 i prelazeći samo preko ε-strelica. Tj. $q_1,\dots,q_k \in E(q_0)$. Dalje imamo $q_0,q_1,\dots,q_k \in q_0' = E(q_0)$ - početno stanje od D. To znači da automat D počinje u q_0' i, pošto je ulazni string $\omega = \varepsilon$, završava u q_0', tj. završava u stanju koji sadrži q_k, što je i trebalo dokazati.

Neka tvrdnja važi za $n = 0,\dots,m-1$, $(m \geq 1)$.

Dokažimo da tvrdnja važi i za $n = m$. Neka je $\beta_1 \dots \beta_k = \alpha_1 \dots, \alpha_n$, te $\beta_1,\dots \beta_s = \alpha_1 \dots \alpha_{n-1}$ i $\beta_{s+1} = \alpha_n$. Odavde imamo $\beta_{s+2} = \dots = \beta_k =$

ε. Iz posljednje konstrukcije imamo da stringu $\alpha_1 \dots \alpha_{n-1}$ odgovara niz stanja $q_0 \dots q_s$ iz N. Po induktivnoj pretpostavci, D prevodi string $\beta_1 \dots \beta_s$ iz početnog stanja do stanja $X_s \subseteq X$, koji sadrži q_s.

Imamo da $q_{s+1} \in \delta(q_s, \beta_{s+1})$ i $\delta'(X_s, \beta_{s+1}) = \bigcup_{q \in X_s} E(\delta(q, \beta_{s+1}))$. Pošto $\beta_{s+2} = \dots = \beta_k = \varepsilon$, to iz q_{s+1} možemo doći do q_k prelazeći samo preko ε-strelica. Prema tome, $q_k \in E(q_{s+1}) = E(\delta(q_s, \beta_{s+1})) \subseteq \bigcup_{q \in X_s} E(\delta(q, \beta_{s+1})) = \delta'(X_s, \beta_{s+1})$, što je i trebalo dokazati.

Smjer 2.

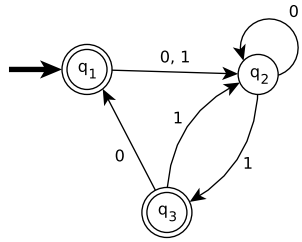
Neka $\omega = \alpha_1 \dots \alpha_n \in L(D)$ ($\alpha_i \in \Sigma$, $i = 1, \dots, n$). Dokažimo da $\omega \in L(N)$. Neka je $X_1, \dots, X_n$ niz stanja iz D koji odgovara stringu ω, te $X_1 = q_0'$, $X_n \in F'$. Iz definicije δ' imamo $X_{i+1} = \delta'(X_i, \alpha_i) = \bigcup_{q \in X_i} E(\delta(q, \alpha_i))$ ($i = 0, \dots, n-1$). Neka je q_n proizvoljno završno stanje automata N koje pripada X_n. Tada $q_n \in X_n = \delta'(X_{n-1}, \alpha_n) = \bigcup_{q \in X_{n-1}} E(\delta(q, \alpha_n))$. Odavde imamo $q_n \in E(\delta(q_{n-1}, \alpha_n))$, za neko $q_{n-1} \in X_{n-1}$. To znači da iz q_{n-1} možemo stići u q_n prelazeći preko strelice sa oznakom α_n i određenog broja strelica sa oznakom ε, tj. postoje stanja $q_{n-1}^1, \dots, q_{n-1}^{k_{n-1}} \in Q$ tako da važi $q_{n-1} \xrightarrow{\alpha_n} q_{n-1}^1 \xrightarrow{\varepsilon} q_{n-1}^2 \dots \xrightarrow{\varepsilon} q_{n-1}^{k_{n-1}} \xrightarrow{\varepsilon} q_n$.

Analognim razmatranjem, postoji sličan put od q_{n-2} do q_{n-1}, tj. od q_{n-2} do q_n. Nastavljajući ovaj postupak, dobijamo da postoji put od q_0 do q_n stanja iz Q pridružen stringu koji je sastavljen od $\alpha_1, \dots, \alpha_n$ i određenog broja simbola ε. Prema tome automat N prihvata string ω. ■

K Ranije smo konstatovali da NFA može uću u beskonačnu petlju (zadatak 2.3). Ovo možemo izbjeći tako što ćemo NFA konvertovati u ekvivalentan DFA i računanje vršiti na DFA. Ovo je formalizovano u tvrdnji 4.10.

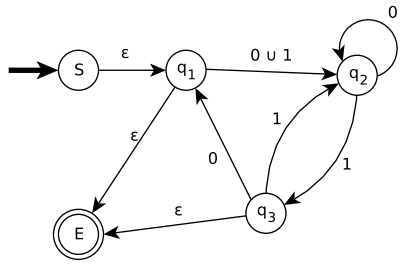
Sada dajemo tvrdnje o ekvivalentnosti DFA i regularnih izraza. Prije svake tvrdnje dajemo primjer koji je ilustruje.

■ **Primjer 2.26** DFA dat na slici 2.18 konvertovati u ekvivalentan regularan izraz.



Slika 2.18: DFA koji konvertujemo u regularan izraz (primjer 2.26).

Dati DFA označimo sa D. Prvi korak je da ubacimo novo početno i novo završno stanje, koje ćemo označiti sa S i E. Stanje S sa prethodnim početnim stanjem je povezano ε-strelicom. Prethodna završna stanja su sa novim završnim stanjem povezana ε-strelicama. Ako je neka strelica imala više simbola, sada te simbole povežemo unijom. Prema tome, imamo novi DFA dat na slici 2.19.



Slika 2.19: Dati DFA nakon početnih transformacija (primjer 2.26).

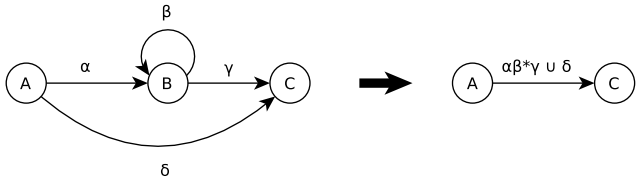
U svakom koraku ćemo brisati po jedno stranje, te vršiti odgovarajuće transformacije nad DFA. Postupak se završava kada ostanu samo stanja S i E. Princip je demonstriran na slici 2.20.

Potpuni postupak je prikazan na slici 2.21.

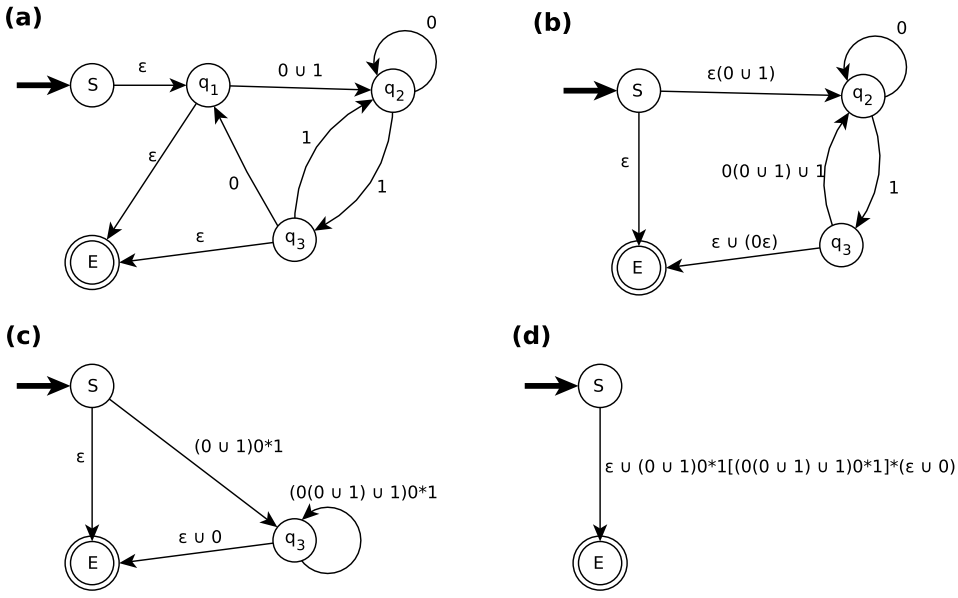
Datom DFA je ekvivalentan regularan izraz

$$R = \varepsilon \cup (0 \cup 1)0^*1\left[(0(0 \cup 1) \cup 1)0^*1\right]^*(\varepsilon \cup 0).$$

Drugim riječima, $L(D) = L(R)$, tj. D prihvata neki string ω akko R generiše ω. ∎



Slika 2.20: Princip izbacivanja čvora iz grafa. Pretpostavljamo $A \neq B$ i $B \neq C$, ali može biti $A = C$ (primjer 2.26).



Slika 2.21: Postupak transformacije DFA u ekvivalentan regularan izraz (primjer 2.26). **(a)** DFA sa ubačenim stanjima S i E. **(b)** Novi automat dobijen izbacivanjem čvora q_1. **(c)** Automat dobijen izbacivanjem q_2. **(d)** Automat dobijen izbacivanjem posljednjeg stanja q_3. Izraz pridružen tranziciji (S,E) predstavlja regularan izraz ekvivalentan datom DFA.

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 2.2 Za svaki DFA postoji njemu ekvivalentan regularan izraz.

Dokaz. Preporučujemo da proučite primjer 2.26 prije čitanja ovog dokaza.

Početni automat označimo sa D_0. Sa D_1 označimo automat dobijen iz D_0 tako što su simboli na strelicama zamijenjeni regularnim izrazima na slijedeći način. Ubacimo novo početno stanje S, koje je sa prethodnim početnim stanjem povezano ε-strelicom. Dalje, ubacimo novo završno stanje, označeno sa E, koje je sa prethodnim završnim stanjima povezano ε-strelicama.

Ako strelica (funkcija tranzicije) ima samo simbol $a \in \{0,1,\varepsilon\}$, tada a smatramo regularnim izrazom. Ako ima dva simbola $0,1$, tada ih mijenjamo sa izrazom $0 \cup 1$. Ovim završavam konstrukciju automata D_1.

Neka je D_i automat ($i \geq 1$). Sa D_{i+1} označimo automat dobijen iz D_i izbacivanjem jednog stanja, različitog od S i E, na slijedeći način. Neka je B (slika 2.20) čvor kojeg izbacujemo. Neka su A i C proizvoljni čvorovi tako da važi $A \neq B$, $C \neq B$; tranzicijama $(A,B),(B,B),(B,C),(A,C)$ su pridruženi redom izrazi $\alpha,\beta,\gamma,\delta$. Nakon uklanjanja čvora B, tranziciji (A,C) pridružimo izraz $\alpha\beta^*\gamma \cup \delta$. Prethodni postupak ponavljamo za sve čvorove A i C koji zadovoljavaju navedene uslove. Uočite da može biti $A = C$.

Prelazeći preko stanja automata D_i ($i \geq 1$), od regularnih izraza pridruženim funkciji tranzicije mi formiramo regularan izraz. Posmatrajmo skup svih regularnih izraza koje možemo dobiti počinjući od S (početno stanje automata D_i) i završavajući na E (završno stanje automata D_i). Taj skup označimo sa R. Trebamo dokazati da skup stringova koje generišu izrazi iz R je jednak $L(D_0)$. Za dokaz ćemo koristiti matematičku indukciju.

Tvrdnja je očigledna za D_1, a slijedi iz načina konstrukcije automata D_1.

Pretpostavimo da tvrdnja važi za D_i ($i \geq 1$) i dokažimo da važi za D_{i+1}.

Neka je R regularan izraz dobijen iz automata D_i. Ako je dobijen iz usmjerenog puta P koji ne sadrži izbačeni čvor B, tada je taj put sadržan u D_{i+1}, pa se i regularan izraz može dobiti iz D_{i+1}.

Sada pretpostavimo da put sadrži izbačeni čvor B. Posmatrajmo jedno pojavljivanje čvora B u putu. Prepostavimo da se B javlja tačno k puta. Dalje, neka su čvorovi A i C susjedi čvora B u putu P, tj. put $P_1 = A \to B \to B \ldots \to B \to C$ je sadržan u putu P. Putu P_1 je pridružen izraz $\alpha\beta^k\gamma$, koji je sadržan u izrazu R. Uklanjanjem čvora B dobijamo put $P_2 = A \to C$, put P' unutar D_{i+1}, koji sadrži P_2. Izraz R' pridružen P' u D_{i+1} sadrži $\alpha\beta^*\gamma \cup \delta$, koji opet sadrži $\alpha\beta^k\gamma$. Prema tome R' sadrži R. Na sličan način dokažemo tvrdnju ako se niz čvorova B javlja na više mjesta unutar puta P.

Dokazali smo da svaki string kojeg sadrži neki izraz generisan iz D_i je sadržan u nekom izrazu generisanom od strane D_{i+1}, tj. važi $L(D_i) \subseteq L(D_{i+1})$. Dokažimo da važi obratno.

Neka je R' regularan izraz pridružen putu P', koji se nalazi unutar D_{i+1}.

Ako se R' može dobiti iz D_i, onda nemamo šta dokazivati.

Pretpostavimo sada da ne možemo dobiti R' iz D_i. Tada je R' pridružen putu P' iz D_{i+1} koji je dobijen izbacivanje čvora B, koji se javlja tačno jednom. Neka je P put iz D_i iz kojeg je dobijen P' i neka je $P_1 = A \to B \to C$. Neka je ω proizvoljan string kojeg generiše izraz R'. Dokažimo da postoji izraz u D_i koji generiše string ω. Umjesto puta $P_1 = A \to B \to C$ u D_{i+1} imamo put $P_2 = A \to C$, kojem je pridružen izraz $\alpha\beta^*\gamma \cup \delta$. Ako smo string ω dobili koristeći δ, tada se ω može dobiti iz D_i koristeći tranziciju (A,C). Sa druge strane, ako smo ω dobili iz $\alpha\beta^*\gamma$, tada ω sadrži podstring $\alpha\beta^k\gamma$. Ovaj se podstring može dobiti ako u put P', umjesto tranzicije $A \to C$ ubacimo niz tranzicija $P_1 = A \to B \to B \ldots \to B \to C$, pri čemu se B javlja tačno k puta. Posljednji niz tranzicija je upravo dio puta u D_i. na ovaj način iz P' možemo konstruisati put u D_i koji generiše izraz koji sadrži string ω. Znači, $\omega \in L(D_i)$. Na sličan način možemo dokazati tvrdnju ako se niz stanja B nalazi na više mjesta u putu.

Iz svega imamo da je $L(D_i) = L(D_{i+1})$. Prema tome važi $L(D_0) = L(D_m)$, pri čemu je D_m posljednji dobijeni automat, tj. automat koji sadrži samo stanja S i E. izraz pridružen tranziciji (S,E) unutar D_m generiše sve stringove iz $L(D_m) = L(D_0)$, tj. taj izraz je ekvivalentan automatu D. ■

Slijedećim primjerima ilustrujemo operacije unije, presjeka, komplementiranja, nadovezivanja i zvijezde sa NFA.

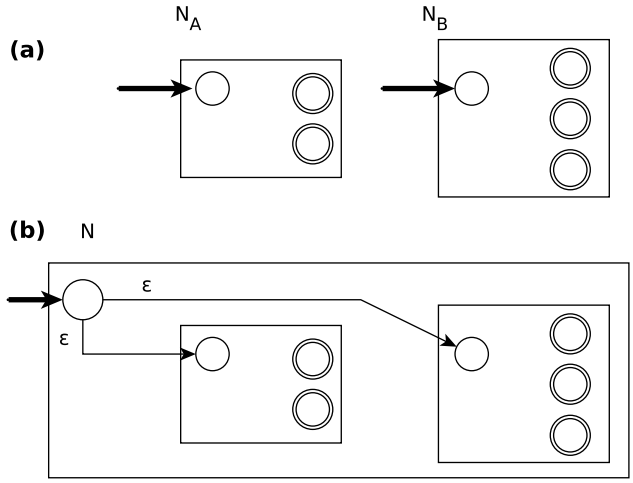
■ **Primjer 2.27 — Unija NFA.** Uniju NFA možemo naći na isti način kao i uniju DFA (primjer 2.20), pri čemu ε tretiramo na isti način kao i simbole 0 i 1. Pored ovog načina, uniju NFA možemo naći na slijedeći način.

Neka su N_A i N_B nedeterministički konačni automati. Automat N, koji prepoznaje $L(N_A) \cup L(N_B)$ ćemo konstruisati tako što ubacimo novo početno stanje i povežemo da ε-strelicama sa početnim stanjima od N_A i N_B (slika 2.22).

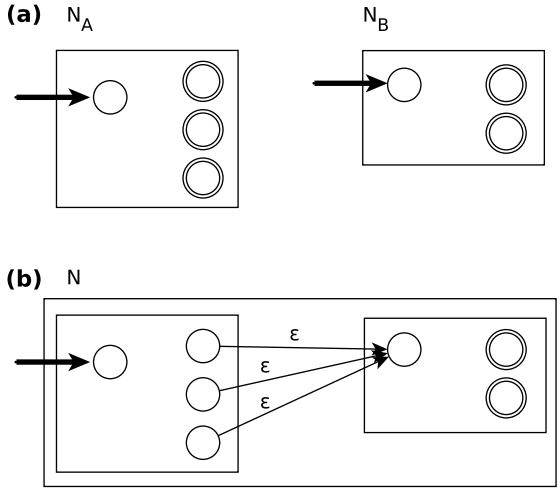
■

■ **Primjer 2.28 — Presjek NFA.** Presjek NFA možemo naći na isti način kao i presjek DFA (primjer 2.19), pri čemu ε tretiramo na isti način kao i simbole 0 i 1. ■

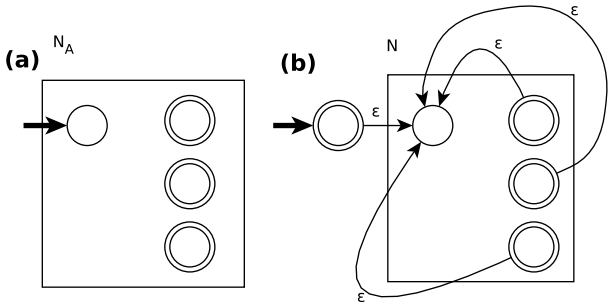
■ **Primjer 2.29 — Nadovezivanje NFA.** Neka su N_A i N_B nedeterministički konačni automati. Automat N, koji prepoznaje $L(N_A)L(N_B)$, ćemo konstruisati



Slika 2.22: Unija NFA. **(a)** Dati su automati N_A i N_B. **(b)** Automat N je unija automata N_A i N_B, tj. prepoznaje jezik $L(N_A) \cup L(N_B)$.



Slika 2.23: Nadovezivanje NFA. **(a)** Dati su automati N_A i N_B. **(b)** Automat N koji prepoznaje jezik $L(N_A)L(N_B)$.



Slika 2.24: Kleeneova zvijezda nad NFA. Automat N prepoznaje jezik $L(N_A)^*$.

tako što završna stanja automata A spojimo ε-strelicama sa početnim stanjem automata B. Početno stanje of N_A je početno stanje od N, te završna stanja od N_B su završna stanja od N (slika 2.23). ■

■ **Primjer 2.30 — Operacija * na NFA.** Neka je N_A nedeterministički konačni automat. Automat N, koji prepoznaje $L(N_A)^*$, ćemo konstruisati tako što ubacimo novo početno stanje, koje je sta prethodnim početnim stanjem povezano ε-strelicom. Završna stanja povežemo ε-strelicama sa prethodnim početnim stanjem (slika 2.24).

■

Zadatak 2.4 — Za samostalan rad. Ako je dat NFA N_A, kako biste konstruisali NFA koji prepoznaje $L(N_A)^+$? ■

■ **Primjer 2.31** Regularan izraz $(0\cup 1)^*000(0\cup 1)^*$ konvertovati u ekvivalentan NFA.

Drugima riječima, ako dati regularni izraz shvatimo kao jezik, mi tražimo NFA koji prepoznaje jezik $(0\cup 1)^*000(0\cup 1)^*$. Na slici 2.25 je ilustrovan postupak konvertovanja regularnog izraza u NFA.

Slika 2.25a opisuje DFA pridruženih izrazima 0 i 1.

Na slici 2.25b je dat NFA pridružen izrazu $0\cup 1$. Pravilo koje smo koristili je za uniju NFA (primjer 2.27 i slika 2.22).

Slika 2.25c opisuje NFA pridružen $(0\cup 1)^*$. Pravilo koje smo koristili je za operator zvijezda nad NFA (primjer 2.30 i slika 2.24).

Za formiranje NFA pridruženog izrazu $(0\cup 1)^*000$ (slika 2.25d) smo koristili pravilo nadovezivanja NFA (primjer 2.29 i slika 2.23).

Isto pravilo nadovezivanja smo koristili i za konstrukciju automata na slici 2.25e, koji je ekvivalentan izrazu $(0\cup 1)^*000(0\cup 1)^*$.

■

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 2.3 Za svaki regularan izraz postoji njemu ekvivalentan NFA.

Dokaz. Prije čitanja ovog dokaza, predlažemo da proučite primjer 2.31. Neka je R regularan izraz koji želimo konvertovati u NFA. Za dokaz ćemo koristiti matematičku indukciju po n, pri čemu je n dužina regularnog izraza R.

Za $n = 1$ imamo $R = 0$ ili $R = 1$, a ekvivalentan NFA je dat na slici 2.25a.

Neka tvrdnja važi za sve regularne izraze dužine $n = 1,\ldots,k$ $(k \geq 1)$. Dokažimo da tvrdnja važi i za regularne izraze dužine $k+1$. Iz definicije 2.1 imamo slijedeće mogućnosti za R: $R = R_1 \cup R_2$, $R = R_1R_2$, $R = R_1^*$, pri čemu su R_1 i R_2 regularni izrazi. Pošto su dužine od R_1 i R_2 manje od $k+1$, na ove izraze možemo primijeniti matematičku indukciju, tj. postoje NFA $N_1 = (Q_1,\Sigma,\delta_1,q_1',F_1)$ i $N_2 = (Q_2,\Sigma,\delta_2,q_2',F_2)$ koji su ekvivalentni sa R_1 i R_2. Posmatrajmo slučajeve.

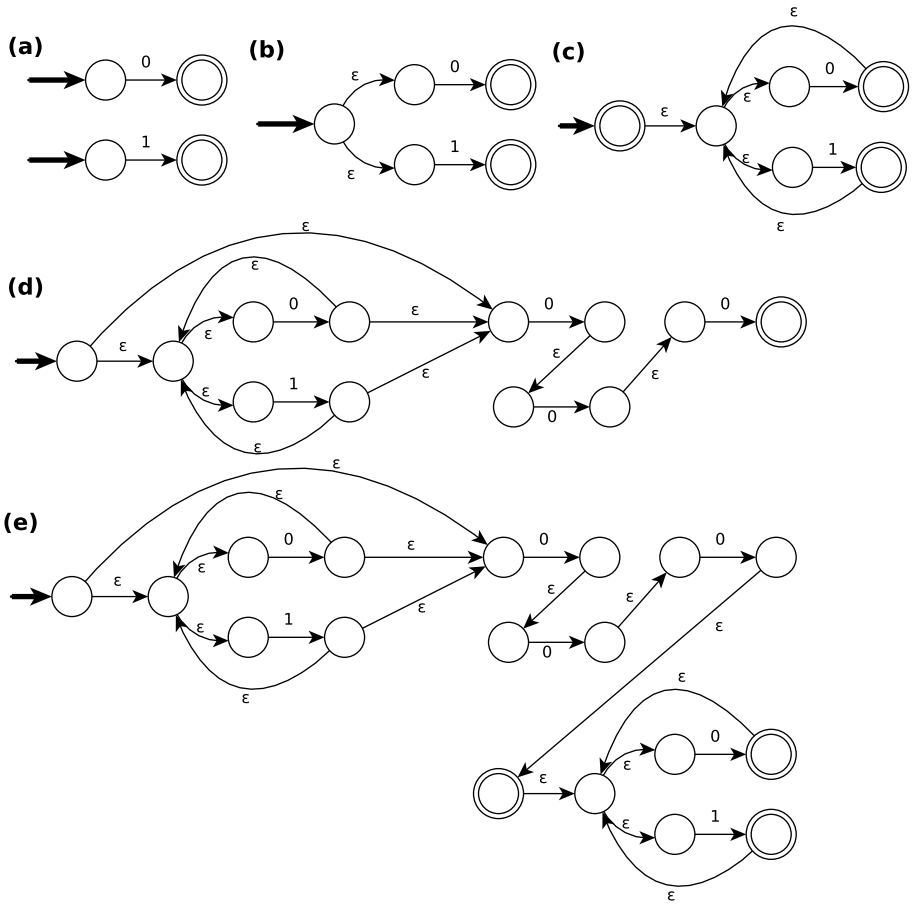
Slučaj 1. Neka je $R = R_1 \cup R_2$. Konstruišimo NFA $N = (Q,\Sigma,\delta,q',F)$ za koji važi $L(N) = L(N_1)\cup L(N_2)$ (slika 2.22). Uzmimo $Q = Q_1 \cup Q_2 \cup \{q'\}$, $F = F_1 \cup F_2$. Za funkciju tranzicije uzimamo $\delta : Q\times\Sigma_\varepsilon \to \mathscr{P}(Q)$, $\delta(x,a) = \delta_1(x,a)$, $\delta(y,a) = \delta_1(y,a)$ za sve $x \in Q_1$, $y \in Q_2$ i $a \in \Sigma_\varepsilon$, te $\delta(q',\varepsilon) = \{q_1',q_2'\}$. Za ovakav NFA očigledno imamo $L(N) = L(N_1)\cup L(N_2)$.

Slučaj 2. Neka je $R = R_1R_2$. Sada za N (slika 2.23) želimo da važi $L(N) = L(N_1)L(N_2)$. Uzmimo $Q = Q_1 \cup Q_2$, $q' = q_1'$, $F = F_2$. Za funkciju tranzicije uzimamo $\delta : Q\times\Sigma_\varepsilon \to \mathscr{P}(Q)$, $\delta(x,a) = \delta_1(x,a)$, $\delta(y,a) = \delta_1(y,a)$ za sve $x \in Q_1\backslash F_1$, $y \in Q_2$ i $a \in \Sigma_\varepsilon$, $\delta(z,b) = \delta_1(z,b)$, $\delta(z,\varepsilon) = \delta_1(z,\varepsilon)\cup q_2'$ za sve $z \in F_1$ i $b \in \Sigma$. Za ovakav automat očigledno važi $L(N) = L(N_1)L(N_2) = L(R_1)L(R_2)$.

Slučaj 3. Za $R = R_1^*$ automat je dat na slici 2.24. Imamo $Q = Q_1 \cup \{a'\}$, $F = F_1 \cup \{q'\}$. Za funkciju tranzicije imamo: $\delta(x,a) = \delta_1(x,y)$ za sve $x \in Q_1\backslash F_1$ i $a \in \Sigma_\varepsilon$, te $\delta(z,\varepsilon) = \delta_1(z,\varepsilon)\cup q_1'$ za sve $z \in F_1$ i $\delta(q',\varepsilon) = \{q_1'\}$.

Znači, tvrdnja važi i za regularan izraz dužine $k+1$. ■

Kao posljedicu prethodnih tvrdnji slijedeću teoremu.



Slika 2.25: Postupak transformacije regularnog izraza u NFA (primjer 2.31).

Teorema 2.2 Jezik je regularan akko ga neki DFA prepoznaje, odnosno akko ga neki NFA prepoznaje.

Dokaz. Neka je A regularan jezik. Iz definicije imamo da postoji regularan izraz R koji ga generiše, tj. važi $L(R) = A$. Iz tvrdnje 2.3, postoji NFA N, koji je ekvivalentan sa R, tj. $L(N) = L(R)$, pa je $L(N) = A$. Dobijamo da postoji NFA koji prepoznaje A.

Iz teoreme 2.1 postoji DFA D koji je ekvivalentan sa N, tj. $L(D) = L(N)$, pa $L(D) = A$. Dobijamo da postoji DFA koji prepoznaje A.

Neka za A postoji NFA N koji ga prepoznaje, tj. važi $L(N) = A$. Iz teoreme 2.1 postoji DFA D koji je ekvivalentan sa N, tj. $L(D) = L(N)$, pa $L(D) = A$. Iz tvrdnje 2.2 postoji regularan izraz R ekvivalentna sa D, tj. $L(R) = L(D)$, pa $L(R) = A$. Dobijamo da je A regularan jezik, što je i trebalo dokazati. ■

Da bi dokazali da je neki jezik regularan, dovoljno je konstruisati DFA ili NFA koji ga prepoznaje.

Ranije smo dokazali da su regularni jezici zatvoreni pod operacijama unije, nadovezivanja, zvijezde (tvrdnja 2.1). Sada ćemo dokazati da je klasa regularnih jezika zatvorena pod operacijama presjeka i komplementiranja.

Tvrdnja 2.4 Klasa regularnih jezika je zatvorena pod operacijama:
(a) presjeka;
(b) komplementiranja.

Dokaz. Prije čitanja ovog dokaza pogledajte primjere 2.19 i 2.21.

(a) Dokažimo da presjek dva regularna jezika je regularan jezik. Neka su A_1 i A_2 regularni jezici, te $D_1 = (Q_1, \Sigma, \delta_1, q_1', F_1)$ i $D_2 = (Q_2, \Sigma, \delta_2, q_2', F_2)$ DFA koji ih prepoznaju. Konstruišimo DFA $D = (Q, \Sigma, \delta, q', F)$, koji prepoznaje jezik $A = A_1 \cap A_2$.

Uzmimo $Q = Q_1 \times Q_2$, $q' = (q_1', q_2')$, $F = \{(a,b) \mid a \in F_1 \text{ i } b \in F_2\}$. Funkciju tranzicije definišemo sa $\delta : Q \times \Sigma \to Q$, $\delta((q_1,q_2),\alpha) = (\delta_1(q_1,\alpha), \delta_2(q_2,\alpha))$.

Dokažimo da automat Q prihvata string akko ga prihvataju automati Q_1 i Q_2. Neka je $\omega = \alpha_1 \ldots \alpha_n$, $\alpha_i \in \Sigma$ $(i = 1, \ldots, n)$, i neka je $(a_0, b_0), (a_1, b_1), \ldots, (a_n, b_n)$ niz stanja automata Q koji odgovara stringu ω. Tada je $a_0 = q_1'$, $b_0 = q_2'$. Važi $\delta((a_i, b_i), \alpha_{i+1}) = (a_{i+1}, b_{i+1})$ $(i = 0, \ldots, n-1)$. Dalje imamo $(\delta_1(a_i, \alpha_{i+1}), \delta_2(b_i, \alpha_{i+1})) = (a_{i+1}, b_{i+1})$, pa $\delta_1(a_i, \alpha_{i+1}) = a_{i+1}$ i $\delta_2(b_i, \alpha_{i+1}) = b_{i+1}$. To znači da stringu ω u D_1 i D_2 odgovara niz stanja $a_0, \ldots, a_n$ i $b_0, \ldots, b_n$.

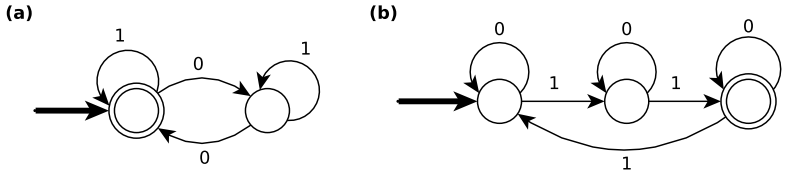
Automat Q prihvata ω akko je $(a_n, b_n) \in F$, a to važi akko je $a_n \in F_1$ i $b_n \in F_2$, a to opet važi akko automati D_1 i D_2 prihvataju string ω. Prema tome $L(D) = A$.

(b) Konstruišimo automat $\overline{D} = (\overline{Q}, \Sigma, \overline{\delta}, \overline{q'}, \overline{F})$ koji je komplement automata $D = (Q, \Sigma, \delta, q', F)$. Uzmimo $\overline{Q} = Q$, $\overline{\delta} = \delta$, $\overline{q'} = q'$, $\overline{F} = Q \backslash F$. Automat D prihvata string akko čitanjem stringa završi na prihvatnom stanju, odnosno akko $\overline{D}$ završi na neprihvatnom stanju. Posljednje važi akko $\overline{D}$ odbije string. Prema tome $L(\overline{D}) = \overline{L(D)}$. ■

■ **Primjer 2.32** Dokaži da je jezik $A = \{bc \mid b, c \in \Sigma^*, b$ sadrži paran broj nula i c sadrži broj jedinica koji pri dijeljenju sa 3 daje ostatak $2\}$ regularan.

Imamo $A = BC$, pri čemu je $B = \{b \mid b \in \Sigma^*, b$ sadrži paran broj nula $\}$ i $C = \{c \mid c \in \Sigma^*, c$ sadrži broj jedinica koji pri dijeljenju sa 3 daje ostatak $2\}$.

Automati koji prepoznaju jezike B i C su je dati na slici 2.26. Prema tome, B i C su regularni jezici. Iz tvrdnje 2.1(b) imamo da je i $A = BC$ regularan jezik.



Slika 2.26: **(a)** Automat koji prepoznaje jezik sastavljen od stringova koji sadrže paran broj nula. **(b)** Automat koji prepoznaje jezik sastavljen od stringova koji imaju $3k+2$ jedinica ($k \in \mathbb{N}_0$).

■

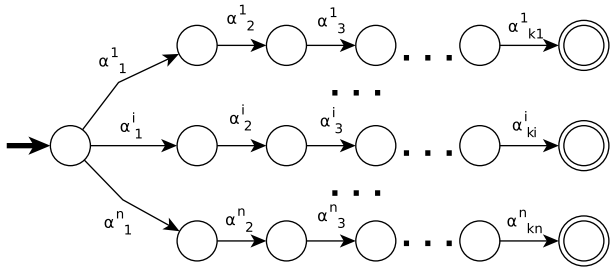
■ **Primjer 2.33** Dokaži da je svaki konačan jezik regularan.

Neka je $A = \{a_1, \ldots, a_n\}$ konačan jezik i $a_i = \alpha_1^i \ldots \alpha_{k_i}^i$ $(i = 1, \ldots, n)$. NFA koji prepoznaje jezik A je dat na slici 2.27. Za svaki string iz A imamo po jedan put koji vodi od početnog do prihvatnog stanja.

■

2.4 Neregularni jezici

Jezika ima neprebrojivo mnogo (tvrdnja 1.5). Sa druge strane, regularni jezici se opisuju pomoću DFA, koji su uređene petorke, pa njih ima prebrojivo



Slika 2.27: NFA koji prepoznaje konačan jezik $A = \{\alpha^i_j \mid i = 1,\ldots,n, j = 1,\ldots,k_i\}$ (primjer 2.33).

mnogo. Kao posljedicu imamo da postoje jezici koji nisu regularni. Vidimo da je NFA ograničen automat: nema memoriju, ne može da čita string u oba smjera, nema mogućnost zapisivanja. Izvlačimo intuitivni zaključak da NFA ne može riješiti široku klasu problema.

Kako možemo dokazati da neki problem nije rješiv pomoću NFA, odnosno da odgovarajući jezik nije regularan? Jedan od načina da se to uradi je Pumpajuća lema.

Teorema 2.3 — Pumpajuća lema. Ako je A regularan jezik, tada postoji broj $p > 0$ takav da svaki string $s \in A$, dužine veće ili jednake p, se može podijeliti na tri dijela $s = xyz$, tako da važi:

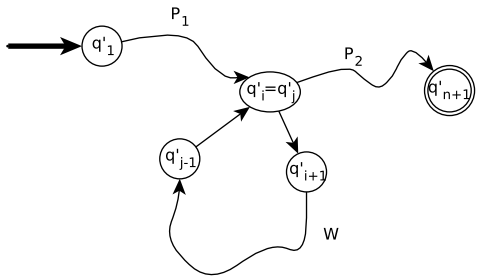
1. $xy^kz \in A$, za svako $k \geq 0$;
2. $|y| > 0$;
3. $|xy| \leq p$.

Broj p nazivamo *pumpajuća dužina*.

Dokaz. Neka je D DFA koji prepoznaje A i $Q = q_1,\ldots,q_n$ skup stanja automata D. Uzmimo da je $p = n+1$. Ako je $s \in A$ takav da je $|s| \geq p$. Tada postoji niz stanja $q'_1,\ldots,q'_{n+1}$ koji odgovara računanju automata D nad stringom s, pri čemu je q'_1 početno, a q'_{n+1} završno stanje,.

Pošto D ima tačno n stanja, u nizu $q'_1,\ldots,q'_{n+1}$ neka dva stanja su jednaka. Neka je $q'_i = q'_j$ $(i < j)$, pri čemu je indeks j minimalan. Sa x označimo string koji je pridružen nizu stanja $P_1 = q'_1,\ldots,q'_i$, sa y string pridružen nizu stanja $W = q'_i,\ldots,q'_j$, a sa z označimo string pridružen nizu $P_2 = q'_j\ldots,q'_{n+1}$ (slika 2.28). Tada je $s = xyz$.

Uočimo da je niz W zatvorena šetnja u usmjerenom grafu koji odgovara dijagramu stanja automata D. Stringu y^k odgovara k obilazaka zatvorene šetnje W ($k \geq 0$) pri čemu obilasci počinju i završavaju u q'_i. Ako obilazimo usmjereni graf prelazeći preko usmjerenog puta P_1, zatim W obiđemo k puta ($k \geq 0$), te na kraju pređemo preko P_2. Dati obilazak označimo sa $P_1W^kP_2$. String koji odgovara ovom obilasku je xy^kz. Pošto $P_1W^kP_2$ započinje u početnom stanju q'_1, završava u završnom stanju q'_{n+1}, automat D prihvata string xy^kz, tj. $xy^kz \in A$.



Slika 2.28: Ako automat prihvata string s, kojem odgovara šetnja P_1WP_2, tada će automat prihvatiti i svaki string oblika $s = xy^kz$ ($k \geq 0$), pri čemu string x odgovara šetnji P_1, string y odgovara zatvorenoj šetnji W, te z odgovara P_2.

Pošto je $i < j$ imamo da je $|y| > 0$.

Dokažimo da je $|xy| \leq p = n+1$. Pretpostavimo suprotno, da važi $|xy| \geq n+2$. Uklanjanjem stanja q'_j iz P_1W dobijamo niz od barem $n+1$ stanja, pa neka dva stanja moraju biti jednaka, tj. $q'_{i_1} = q'_{j_1}$ i $i_1 < j_1 < j$, što je u kontradikciji sa minimalnošću indeksa j. Zaključujemo da je $|xy| \leq p$. ■

Pumpajuću lemu ćemo koristiti da dokažemo da neki jezik A nije regularan. Princip koji ćemo koristiti je da pretpostavimo suprotno, A jeste regularan, pa koristeći xy^kz konstruišemo string koji nije u A, što je u kontradikciji sa uslovom 1 Pumpajuće leme.

■ **Primjer 2.34** Dokaži da jezik $A = \{\omega\omega \mid \omega \in \Sigma^*\}$ nije regularan.

Pretpostavimo suprotno, da jeste regularan. Tada postoji pumpajuća dužina $p > 0$ koja zadovoljava Pumpajuću lemu. Neka je $s = 0^p1^p0^p1^p \in A$. Neka su x, y, z parametri iz Pumpajuće leme. Iz $|xy| \leq p$ imamo da je xy sadržano u 0^p, tj. $x = 0^a, y = 0^b$, $b > 0$, $a + b \leq p$, $z = 0^{p-a-b}1^p0^p1^p$. Iz Pumpajuće leme imamo $s_2 = xy^2z = 0^a0^{2b}0^{p-a-b}1^p0^p1^p = 0^{p+b}1^p0^p1^p \in A \implies$

$0^{p+b}1^p0^p1^p = \omega'\omega'$. Iz posljednje relacije imamo da ω' počinje sa 0 i završava sa 1, pa je $\omega' = 0^{p+b}1^p$ i $\omega' = 0^p1^p$, što je nemoguće, jer $b > 0$. Dobijamo kontradikciju sa prepostavkom da je A regularan jezik. Prema tome, A nije regularan jezik. ∎

Vidi primjer 1.22 za definiciju palindroma.

■ **Primjer 2.35** Dokaži da jezik $A = \{\omega \mid \omega \text{ je palindrom}\}$ nije regularan.

Pretpostavimo suprotno, da A jeste regularan. Tada postoji pumpajuća dužina $p > 0$. Neka je $s = 0^p10^p$. Pošto je s palindrom, imamo $s \in A$. Neka su x, y, z veličine iz Pumpajuće leme. Iz $|xy| \leq p$ imamo da je xy sadržano u 0^p, pa je $x = 0^a$, $y = 0^b$, $b > 0$, $a + b \leq p$, $z = 0^{p-a-b}10^p$.

Neka je $s_2 = xy^2z = 0^a0^{2b}0^{p-a-b}10^p = 0^{p+b}10^p$. Iz Pumpajuće leme imamo da $s_2 \in A$.

Sa druge strane, pošto je $b > 0$, s_2 i s_2^R na poziciji $p+1$ nemaju isti simbol. Naime, s_2 na toj poziciji ima 0, dok s_2^R ima 1. Znači, $s_2^R \neq s_2$, pa s_2 nije palindrom - kontradikcija. Znači, A nije regularan jezik. ∎

■ **Primjer 2.36** Dokaži da jezik $A = \{0^n1^n \mid n \geq 0\}$ nije regularan.

Pretpostavimo suprotno, A jeste regularan jezik. Neka je p pumpajuća dužina.

Posmatrajmo string $s = 0^p1^p \in A$. Neka je $s = xyz$ podjela iz Pumpajuće leme. Iz $|xy| \leq p$ imamo da je xy sadržano u 0^p, tj. važi $x = 0^a$, $y = 0^b$, $a + b \leq p$ i $z = 0^{p-a-b}1^p$. Iz $|y| > 0$ imamo $b > 0$. Neka je $s_2 = xy^2z$. Imamo $s_2 = 0^a0^{2b}0^{p-a-b}1^p = 0^{p+b}1^p \notin A$, što je u kontradikciji sa Pumpajućom lemom. Prema tome, A nije regularan jezik. ∎

■ **Primjer 2.37** Neka je $\Sigma = \{a, b, c\}$. Dokaži da jezik $A = \{a^nb^mc^m \mid m, n \geq 0\}$ nije regularan.

Pretpostavimo suprotno, da A jeste regularan jezik. Neka je p pumpajuća dužina.

Posmatrajmo string $s = a^0b^pc^p = b^pc^p$. Ostatak rješenja je sličan kao u primjeru 2.36, pa je ostavljeno čitaocu za samostalan rad da dovrši. ∎

■ **Primjer 2.38** Neka je $\Sigma = \{0, \#\}$ i $A = \{x_1\#x_2\#\ldots\#x_k \mid k \geq 0, x_i \in 0^*, x_i \neq x_j \text{ za } i \neq j, (i, j = 1, \ldots, k)\}$. Dokaži da A nije regularan.

Pretpostavimo suprotno, A je regularan. Neka je $p > 0$ pumpajuća dužina i $s = 0^p\#0^{p+1}\#0^{p+2}\#\ldots\#0^{2p} \in A$. Neka su x, y, z veličine iz Pumpajuće leme. Iz $|xy| \leq p$ imamo da je xy sadržano u 0^p, pa je $x = 0^a$, $y = 0^b$, $b > 0$, $a + b \leq p$, $z = 0^{p-a-b}\#0^{p+1}\#0^{p+2}\#\ldots\#0^{2p}$. Iz Pumpajuće leme imamo da $s_2 = xy^2z =$

$0^a0^{2b}0^{p-a-b}\#0^{p+1}\#0^{p+2}\#\ldots\#0^{2p} = 0^{p+b}\#0^{p+1}\#0^{p+2}\#\ldots\#0^{2p} \in A$. Neka je $x_1 = 0^{p+b}$, $x_2 = 0^{p+1}$, ..., $x_{p+1} = 0^{2p}$. Pošto je $0 < b \leq p$ imamo da je $x_1 = x_{b+1}$ i $1 \neq b+1$. Iz definicije skupa A imamo $s_2 \notin A$ - kontradikcija. Znači, A nije regularan jezik. ∎

∎ **Primjer 2.39** Neka je $\Sigma = \{0,1,=,+\}$. Dokaži da jezik $A = \{x = y+z \mid x,y,z \in \{0,1\}^*,\ x$ je binaran zbir od y i $z\ \}$ nije regularan jezik.

Pretpostavimo suprotno, A je regularan jezik. Neka je p pumpajuća dužina, $s = "1^p = 0+1^p"$, i x,y,z su veličine iz Pumpajuće leme. Iz $|xy| \leq p$ imamo da je xy sadržano u 1^p, pa je $x = 1^a$, $y = 1^b$, $b > 0$, $a+b \leq p$, $z = "1^{p-a-b} = 0+1^p"$. Iz $s_2 = xy^2z = "1^a1^{2b}1^{p-a-b} = 0+1^p"$ imamo $s_2 = "1^{p+b} = 0+1^p"$, te iz Pumpajuće leme slijedi $s_2 \in A$. Sa druge strane, iz $b > 0$ imamo da relacija $1^{p+b} = 0+1^p$ nije tačna, pa iz definicije jezika A imamo $s_2 \notin A$ - kontradikcija. Znači, jezik A nije regularan. ∎

Iz prethodnog primjera imamo problem sabiranja binarnih brojeva (u datom obliku) nije regularan jezik/problem. Primjer 2.18 demonstrira da sabiranje binarnih brojeva može biti regularan jezik, ako je problem dat u drugom obliku.

2.5 Neke primjene konačnih automata

DFA/NFA je standardni alat u kompjuterskoj nauci za proučavanje šablona.

Traženje riječi u tekstu

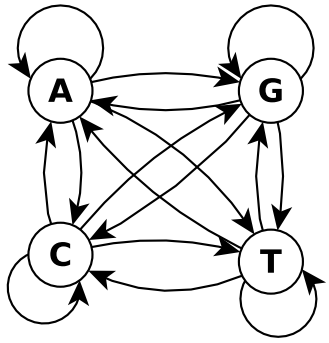
Ovaj problem se još naziva i *uparivanje stringova* (eng. *string matching*). Rješenje se zasniva na principu datom u primjerima 2.17 i 2.24. Ovaj princip se zaista koristi i u praksi. Npr. u .NET dokumentaciji potražite naslov *"Details of regular expression behavior"* - dovoljno je da proguglate ovaj tekst.

Markovljevi lanci i vjerovatnoća gena

Pojednostavljeno govoreći, gen je određeni niz nukleobaza: A (adenin), G (guanin), C (citozin) i T (timin). Nukleobaze možemo predstaviti pomoću stanja DFA (slika 2.29).

Funkciju tranzicije možemo predstaviti pomoću:

$$a_{st} = P(x_i = t | x_{i-1} = s),$$



Slika 2.29: Upotreba DFA u procjeni nastanka gena. Svaka tranzicija predstavlja vjerovatnoću da ćemo dobiti nukleobazu, ako mu je poznat prethodnik.

što predstavlja Markovljev lanac.

Vjerovatnoća gena $x = x_n x_{n-1} \ldots x_1$ je data sa

$$P(x) = P(x_n, x_{n-1}, \ldots, x_1) = P(x_n|x_{n-1}, \ldots, x_1)P(x_{n-1}|x_{n-2} \ldots x_1) \ldots P(x_1),$$

primjenjujući pravilo $P(X,Y) = P(X|Y)P(Y)$ više puta.

Osobina Markovljevog lanca je da ovisi samo od posljednjeg stanja, pa:

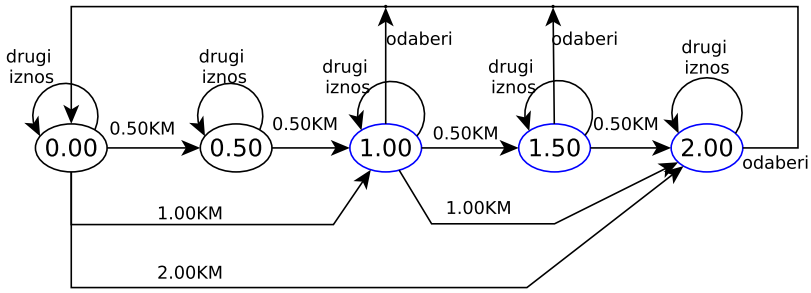
$$P(x) = P(x_n|x_{n-1})P(x_{n-1}|x_{n-2}) \ldots P(x_1) = P(x_1)a_{x_{n-1}x_n}a_{x_{n-2}x_{n-1}} \ldots a_{x_1x_2}.$$

Šema rada mašine

Generalno, DFA se nekada može koristiti i kao dijagram toka algoritma, što sa nekog višeg nivoa apstrakcije on to u suštini i jeste.

Na slici 2.30 je dat šematski prikaz rada mašine sa grickalicama. Možemo ubaciti kovanice od 0.50KM, 1KM, 2KM. Ostale kovanice mašina ne prihvata ("drugi iznos"). Grickalice mogu koštati 1KM, 1.50KM, 2KM. Ako, npr. ubacimo: 0.50KM, 0.50KM, 0.10KM, završićemo na stanju "1.00". Ako uzmemo "odaberi", mašina će nam izbaciti grickalicu i vratiti se na početno stanje ("0.00").

2.6 Zadaci za samostalan rad



Slika 2.30: Šematski prikaz rada mašine sa grickalicama.

Zadatak 2.5 Za jezike, date u ovom poglavlju, formulisati odgovarajuće probleme. ■

Zadatak 2.6 Konstruiši NFA koji prepoznaje slijedeće jezike:
(*a*) $(01)^* \cup (11)^* 0^+$;
(*b*) $((01 \cup 1^*)10)^*$.
■

Zadatak 2.7 Riješi prethodni zadatak koristeći postupak konverzije regularnog izraza u NFA. ■

Zadatak 2.8 Konstruisati NFA za jezik $A_n = \{\omega \mid \omega$ je binaran zapis prirodnog broja djeljivog sa $n\}$, pri čemu je $n \in \mathbb{N}$ dati prirodni broj. ■

Zadatak 2.9 Neka je $B \subseteq \Sigma^*$. Dokaži da je $B = B^+$ akko je $BB \subseteq B$. ■

Zadatak 2.10 Neka je A regularan jezik. Dokaži da je svaki od slijedećih jezika regularan:
(*a*) $A^R = \{\omega^R \mid \omega \in A\}$;
(*b*) $\text{Pref}(L) = \{\omega \mid \omega y \in A$, za neko $y \in \Sigma^*\}$;
(*c*) $\text{Suf}(L) = \{\omega \mid y\omega \in A$, za neko $y \in \Sigma^*\}$.
■

Zadatak 2.11 Neka je $A \subseteq 0^*$ i neka skup $\{n \mid 0^n \in A\}$ formira aritmetičku progresiju. Dokaži da je A regularan jezik. ■

Zadatak 2.12 Dokaži da slijedeći jezici nisu regularni:
(*a*) $\{0^n 10^m 10^{m+n} \mid m,n \geq 1\}$;
(*b*) $\{\omega\omega^R \mid \omega \in \Sigma^*\}$;
(*c*) $\{\omega\omega \mid \omega \in \Sigma^*\}$;
(*d*) $\{0^{2^k} \mid k \in \mathbb{N}\}$;
(*e*) $\{0^p \mid p \text{ je prost broj}\}$.
■

Zadatak 2.13 Dokaži ili opovrgni slijedeće tvrdnje:
(*a*) svaki podskup regularnog jezika je regularan jezik;
(*b*) svaki regularan jezik ima pravi neprazni podskup koji je regularan;
(*c*) ako je A regularan jezik, tada je i $B = \{xy \mid x \in A, y \notin A\}$ regularan.
■

3. Potisni automati

Koristeći Pumpajuću lemu smo vidjeli da postoje jezici koji nisu regularni, tj. postoje jednostavni problemi koje ne mogu riješiti (ne)deterministički konačni automati. Zato u ovom poglavlju uvodimo klasu jezika širu od klase regularnih jezika: *kontekstno nezavisni jezici*. Svaki regularan jezik je ujedno i kontekstno nezavisan, obratno ne mora važiti.

Strukture koje generišu kontekstno nezavisne jezike su *kontekstno nezavisne gramatike*, tj. jezik je kontekstno nezavisan ako ga neka kontekstno nezavisna gramatika generiše.

Automati koje prepoznaju/prihvataju kontekstno nezavisne jezike su *potisni automati*. Važi da je jezik kontekstno nezavisan akko ga neki potisni automat prepoznaje. Potisni automati, za razliku od NFA, imaju memoriju.

3.1 Kontekstno nezavisne gramatike

Kontekstno nezavisna gramatika je skup pravila pomoću kojih se generiše neki jezik, odnosno string iz datog jezika. Na koji način se generiše određeni string? Počne se od varijable koja se naziva *početna varijabla*, koja se zamijeni

sa nekim drugim varijablama i stringovima. U svakom koraku, varijabla se mijenja sa nekim drugim varijablama i stringovima. Postupak se završava kada nemamo više varijabli, tj. kada nam je ostao samo string iz Σ^*.

Demonstrirajmo postupak na primjeru.

■ **Primjer 3.1** Data je gramatika G and alfabetom $\Sigma = \{0, +, (,), \times\}$:

$$\begin{array}{lcl} E & \to & E + T \mid T \\ T & \to & T \times F \mid F \\ F & \to & (E) \mid 0 \end{array}$$

Pokažimo kako gramatika G *generiše string* $\omega = (0+0\times 0)\times(0+0)\times(0+0+0)$.

$E \to$
$T \to$
$T \times F \to$
$T \times F \times F \to$
$F \times F \times F \to$
$(E) \times F \times F \to$
$(E) \times (E) \times F \to$
$(E) \times (E) \times (E) \to$
$(E+T) \times (E) \times (E) \to$
$(E+T) \times (E+T) \times (E) \to$
$(E+T) \times (E+T) \times (E+T) \to$
$(E+T) \times (E+T) \times (E+T+T) \to$
$(E+T\times F) \times (E+T) \times (E+T+T) \to$ (svi T u F - ukupno 4 operacije)
$(E+F\times F) \times (E+F) \times (E+F+F) \to$ (svi E u T - ukupno 3 operacije)
$(T+F\times F) \times (T+F) \times (T+F+F) \to$ (svi T u F - ukupno 3 operacije)
$(F+F\times F) \times (F+F) \times (F+F+F) \to$ (svi F u 0 - ukupno 8 operacija)
$(0+0\times 0) \times (0+0) \times (0+0+0)$ ■

Objekti F, T, E se nazivaju *varijable*, pri čemu je E *početna varijabla*. Članovi alfabeta Σ (u ovom slučaju je $\Sigma = \{+, 0, (,), \times\}$) se nazivaju *terminali*. Na koji način generišemo string alfabeta Σ ? Generišemo tako što počnemo od početne varijable i zamijenimo je sa stringom sa desne strane od $\to$, a sastoji se od drugih varijabli i terminala. Postupak ponavljamo za svaku varijablu u trenutnom stringu, a završava kada se u stringu nalaze samo članovi alfabeta Σ. Oznaka $|$ je skraćeni prikaz više pravila sa istom varijablom na lijevoj strani. Tj. $E \to E+T$ i $E \to T$ možemo zamijeniti sa kraćim zapisom $E \to E+T|T$.

Skup varijabli označavamo sa V, te u prethodnom primjeru imamo $V = \{E, F, T\}$. Stringove nad alfabetom $\Sigma \cup V$ označavamo sa Ω_i. U datom primjeru je $\Omega_1 = E, \Omega_2 = T, \Omega_3 = T \times F, \Omega_4 = T \times F \times F, \ldots, \Omega_{29} = (0+0\times 0)\times(0+0)\times(0+0+0)$. Uočimo da u prethodnom primjeru, string je izveden u 29 koraka.

Prethodne pojmove ćemo sada i formalno definisati. Sa $(V \cup \Sigma)^*$ označavamo skup svih stringova iz alfabeta $V \cup \Sigma$.

Definicija 3.1 — Kontekstno nezavisna gramatika. Kontekstno nezavisna gramatika je uređena četvorka (V, Σ, R, S) pri čemu:

1. V je skup varijabli;
2. Σ je alfabet i $\Sigma \cap V = \varnothing$;
3. R je skup pravila kod kojih se jedna varijabla mijenja sa stringom iz $(V \cup \Sigma)^*$;
4. S je početna varijabla.

Od sada ćemo kontekstno nezavisnu gramatiku skraćeno zvati CFG (*Context - Free Grammar*) ili samo *gramatika*.

Definicija kontekstno nezavisne gramatike ne daje na koji način gramatika generiše određeni string. Zato sada formalno uvodimo pojam generisanja stringa.

Definicija 3.2 — String kojeg generiše gramatika. Kažemo da gramatika $G = (V, \Sigma, R, S)$ generiše string $\omega \in \Sigma^*$, ako postoji niz $\Omega_0, \Omega_1, \ldots, \Omega_k$ iz $(V \cup \Sigma)^*$, tako da važi:

1. $\Omega_0 = S$;
2. $\Omega_k = \omega$;
3. Ω_i i Ω_{i+1} se mogu napisati u obliku $\Omega_i = \alpha A \beta$, $\Omega_{i+1} = \alpha \rho \beta$ $(i = 0, \ldots, k-1)$, pri čemu $\alpha, \beta \in (V \cup \Sigma)^*$ i pravilo $A \to \rho$ pripada R.

Sa $L(G)$ označavamo skup svih stringova iz Σ^* koje generiše gramatika G. Niz $\Omega_0, \Omega_1, \ldots, \Omega_k$ nazivamo *generatorni niz* stringa ω nad gramatikom G.

CFG nam omogućava da uvedemo klasu jezika širu od klase regularnih jezika.

Definicija 3.3 — Kontekstno nezavisni jezici. Za jezik A kažemo da je *kontekstno nezavisan*, ako ga generiše neka gramatika G, tj. ako važi $A = L(G)$.

Kontekstno nezavisne jezike ćemo skraćeno zvati CFL (*Context-Free Language*).

■ **Primjer 3.2** Dati CFG koja generiše slijedeći jezik $A=\{\omega \mid \omega$ je parne dužine$\}$.

Na koji način možemo generisati sve stringove parne dužine? Možemo započeti sa praznim stringom, te dodavati po proizvoljna dva simbola iz Σ. Dodavanjem po tačno dva simbola uvijek dobijamo stringove parne dužine. Ovo je ideja koja nam služi za konstruisanje gramatike:

$$S \rightarrow 00S \mid 01S \mid 10S \mid 11S \mid \varepsilon.$$

■

Jezik iz prethodnog primjera je regularan i CFL. Naime, svaki regularan jezik je ujedno i CFL. Da bi dokazali ovu tvrdnju, trebamo pokazati na koji način konstruisati gramatiku za regularan jezik. Preciznije, ako je dat regularan izraz, kako konstruisati gramatiku koja mu je ekvivalentna?

■ **Primjer 3.3** Za slijedeće regularne izraze konstruiši ekvivalentne gramatike.

(i) 1^* - dati dvije gramatike;
(ii) 01^*011;
(iii) 0^*1^*;
(iv) $(10)^*11 \cup 10^*$;
(v) $(10^*11)^*$.

(i) Potrebno je generisati skup stringova koji sadrže samo jedinicu. Te stringove možemo generisati tako što počnemo sa praznim stringom, te u svakom koraku dodajemo jedinicu. Jedna gramatika je data sa $S_1 \rightarrow 1S_1 \mid \varepsilon$, a druga sa $S_2 \rightarrow S_2S_2 \mid 1 \mid \varepsilon$.

(ii) Jezik 01^*011 možemo konstruisati u dva koraka. U prvom koraku dodamo 0 kao prefiks i 011 kao sufiks, a u drugom koraku konstruišemo 1^*. Prema tome, imamo gramatiku:

$$S \rightarrow 0A011$$
$$A \rightarrow 1A \mid \varepsilon.$$

(iii) Jezik 0^*1^* možemo konstruisati u dva koraka. U prvom koraku konstruišemo 0^* i 1^*, a u drugom koraku ih nadovežemo. Prema tome, imamo gramatiku:

$$S \rightarrow AB$$
$$A \rightarrow 0A \mid \varepsilon$$

$B \rightarrow 1B \mid \varepsilon$.

(iv) Prioritet operacija je slijedeći. Operacija * im viši prioritet od operacije nadovezivanja, dok nadovezivanje ima viši prioritet od operacije $\cup$. Jezik $(10)^*$ možemo konstruisati pomoću gramatike $A \rightarrow 10A|\varepsilon$. Jezik $(10)^*11$ generišemo pomoću $B \rightarrow A11$. Jezik 0^* ćemo generisati sa $C \rightarrow 0C \mid \varepsilon$, te 10^* sa $D \rightarrow 1C$. Operaciji $\cup$ odgovara simbol $|$. Na kraju, imamo gramatiku:

$S \rightarrow B \mid D$
$B \rightarrow A11$
$A \rightarrow 10A \mid \varepsilon$
$D \rightarrow 1C$
$C \rightarrow 0C \mid \varepsilon$.

(v) Jezik 10^*11 ćemo generisati sa $A \rightarrow 0A \mid \varepsilon$, $B \rightarrow 1A11$. Na kraju, $(10^*11)^*$ ćemo generisati sa $S \rightarrow SB \mid \varepsilon$. Gramatika je data sa:

$S \rightarrow SB \mid \varepsilon$
$B \rightarrow 1A11$
$A \rightarrow 0A \mid \varepsilon$.

■

■ **Primjer 3.4** Regularan izraz $((1^+0)^* \cup (001)^*)^* \cup (10^*)^+$ konvertuj u ekvivalentnu gramatiku.

Postupak je rekurzivan. Pretpostavimo da su regularnim izrazima R_1 i R_2 pridružene gramatike G_1 i G_2 sa početnim varijablama A_1 i A_2 i da ove dvije gramatike nemaju zajedničkih varijabli. Tada imamo:

1. Izrazu $R_1 \cup R_2$ pridružujemo gramatiku $A \rightarrow A_1|A_2$ i dodajemo gramatike G_1 and G_2;
2. Izrazu R_1R_2 pridružujemo $A \rightarrow A_1A_2$ i dodajemo gramatike G_1 and G_2;
3. Izrazu R_1^* pridružujemo $A_1 \rightarrow A_1A_1|\varepsilon$ i dodajemo gramatiku G_1;
4. Literalima $0, 1, \varepsilon$ odgovara gramatika $A \rightarrow 0|1|\varepsilon$.

Koristeći $A^+ = AA^*$, dati izraz transformišemo u $((11^*0)^* \cup (001)^*)^* \cup (10^*)(10^*)^*$. Koristeći prethodna pravila imamo:

(i) Prethodnom izrazu pridružimo gramatiku $S \rightarrow A_1|A_2$, pri čemu su A_1 i A_2 početne varijable gramatika pridružene $((11^*0)^* \cup (001)^*)^*$ i $(10^*)(10^*)^*$.

(ii) Dalje imamo: $A_1 \rightarrow A_1A_1|\varepsilon$ i $A_2 \rightarrow A_{2,1}A_{2,2}$, pri čemu su A_1, $A_{2,1}$ i $A_{2,2}$ početne varijable gramatika pridružene $(11^*0)^* \cup (001)^*$, 10^* i $(10^*)^*$.

(*iii*) Izraz $(11^*0)^* \cup (001)^*$ generišemo pomoću $A_1 \to A_{1,1}|A_{1,2}$, pri čemu su $A_{1,1}$ i $A_{1,2}$ početne varijable pridružene izrazima $(11^*0)^*$ i $(001)^*$.

(*iv*) Izrazima $(11^*0)^*$, $(001)^*$, 10^* i $(10^*)^*$ redom pridružujemo slijedeće četiri gramatike:

$A_{1,1} \to A_{1,1}A_{1,1}|\varepsilon \quad A_{1,2} \to 001A_{1,2}|\varepsilon \quad A_{2,1} \to 1C \quad A_{2,2} \to A_{2,2}A_{2,1}|\varepsilon$
$A_{1,1} \to 1B0 \qquad\qquad\qquad\qquad C \to 0C|\varepsilon$
$B \to 1B|\varepsilon$

(*v*) Iz prethodnih razmatranja imamo konačnu gramatiku:

$S \to A_1|A_2$
$A_1 \to A_1A_1|\varepsilon$
$A_2 \to A_{2,1}A_{2,2}$
$A_1 \to A_{1,1}|A_{1,2}$
$A_{1,1} \to A_{1,1}A_{1,1}|\varepsilon$
$A_{2,2} \to A_{2,2}A_{2,1}|\varepsilon$
$A_{1,1} \to 1B0$
$B \to 1B|\varepsilon$
$A_{1,2} \to 001A_{1,2}|\varepsilon$
$A_{2,1} \to 1C$
$C \to 1C|\varepsilon$

■

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 3.1 Svaki regularan jezik je kontekstno nezavisan.

Tvrdnju ćemo dokazati nakon što uvedemo potisne automate (vidi tvrdnju 3.8).

Sada dajemo primjere jezika koji nisu regularni, ali jesu kontekstno nezavisni.

■ **Primjer 3.5** Dokaži da je jezik $A = \{\omega \mid \omega \in \Sigma^*, \omega = \omega^R\}$ kontekstno nezavisan.

Jezik A je skup svih palindroma. U primjeru 2.35 smo dokazali da ovaj jezik nije regularan. Neka je $\omega = \alpha_1\alpha_2\ldots\alpha_{n-1}\alpha_n$ string. Tada je ω palindrom akko važi $\alpha_1 = \alpha_n$, $\alpha_2 = \alpha_{n-1}, \ldots, \alpha_{\lceil n/2 \rceil} = \alpha_{\lfloor (n+1)/2 \rfloor}$. Koristeći ovu činjenicu, imamo gramatiku:

$$S \;\to\; 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

■

■ **Primjer 3.6** Jezik $A = \{0^n 1^n \mid n \geq 0\}$ je kontekstno nezavisan.

Jednostavno se dokaže, pomoću Pumpajuće leme, da ovaj jezik nije regularan (vidi primjer 2.36). Imamo da $\omega = \alpha_1 \alpha_2 \ldots \alpha_{n-1} \alpha_n$ pripada A akko je $n = 2k$ ($k \in \mathbb{N}_0$), $\alpha_1 = \ldots = \alpha_k = 0$ i $\alpha_{k+1} = \ldots = \alpha_n = 1$. Znači, ω možemo konstruisati tako što k puta na početak stringa dodamo 0, a na kraj stringa dodamo 1. Tu činjenicu možemo zapisati i ovako:

$$S \rightarrow 0S1 \mid \varepsilon$$

■

■ **Primjer 3.7** Jezik $L = \{a^n b^m c^m \mid n \geq 0, m \geq 0\}$ je kontekstno nezavisan. Ovdje je $\Sigma = \{a, b, c\}$.

Ranije smo dokazali da jezik nije regularan (vidi primjer 2.37). String iz L možemo podijeliti na dva nezavisna dijela: a^n i $b^m c^m$. Prvi dio (tj. a^n) ćemo generisati sa gramatikom čija je početna varijabla A, a drugi dio (tj. $b^m c^m$) sa gramatikom čija je početna varijabla B. na kraju ćemo ih spojiti sa AB. Gramatika koja generiše L je data ispod.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid \varepsilon \\ B &\rightarrow bBc \mid \varepsilon \end{aligned}$$

■

Slijedećim primjerima ilustrujemo činjenicu da je klasa kontekstno nezavisnih jezika zatvorena pod unijom, nadovezivanjem i zvjezdicom.

■ **Primjer 3.8** Neka je $A = \{a^n b^n \mid n \geq 0\}$ i $B = \{a^n b^m c^m \mid m, n \geq 0\}$. Konstruiši gramatiku za slijedeće jezike:

(a) $A \cup B$;
(b) AB;
(c) A^*.

Za jezike A i B imamo slijedeće gramatike. Gramatika koja generiše A počinje sa X, dok gramatika koja generiše B počinje sa Y.

$$X \rightarrow aXb|\varepsilon \qquad \begin{aligned} Y &\rightarrow Y_1 Y_2 \\ Y_1 &\rightarrow aY_1|\varepsilon \\ Y_2 &\rightarrow bY_2c|\varepsilon. \end{aligned}$$

Radi jednostavnosti, uzimamo da gramatike, koje ulaze u operacije, nemaju zajedničke varijable.

(a) Operaciji $\cup$ odgovara logičko *ili*, a u gramatičkom smislu to odgovara simbolu $|$. Gramatika koja generiše $A \cup B$ je data sa:

$S \to X|Y$
$X \to aXb|\varepsilon$
$Y \to Y_1Y_2$
$Y_1 \to aY_1|\varepsilon$
$Y_2 \to bY_2c|\varepsilon$.

(b) Nadovezivanju jezika dogovara nadovezivanje početnih varijabli koje gramatika koje generišu date jezike. Gramatika koja generiše AB je data sa:

$S \to XY$
$X \to aXb|\varepsilon$
$Y \to Y_1Y_2$
$Y_1 \to aY_1|\varepsilon$
$Y_2 \to bY_2c|\varepsilon$.

(c) Gramatika koja generiše A^* je data sa:

$S \to SX|\varepsilon$
$X \to aXb|\varepsilon$.

■

Ovim dolazimo do slijedeće tvrdnje.

Tvrdnja 3.2 Klasa kontekstno nezavisnih jezika je zatvorena pod operacijama unije, nadovezivanja i zvjezdice.

Dokaz. Dokaz imitira postupak u primjeru 3.8. Neka su A i B kontekstno nezavisni jezici, G_1 i G_2 gramatike koje generišu A i B, te S_1 i S_2 početne varijable gramatika G_1 i G_2. Pretpostavimo da ove dvije gramatike nemaju zajedničke varijable.

Gramatiku za jezik $A \cup B$ dobijamo tako što ubacimo novu početnu varijablu $S \to S_1|S_2$, te prepišemo gramatike G_1 i G_2. Počinjući od varijable S, možemo nastaviti preko varijable S_1 ili S_2. U prvom slučaju ćemo dobiti string iz A, a u drugom slučaju string iz B. Znači, dobićemo string iz $A \cup B$. Pošto se varijable iz G_1 i G_2 ne ponavljaju, nećemo dobiti nijedan drugi string. Prema tome, konstruisali smo gramatiku za $A \cup B$, pa je $A \cup B$ kontekstno nezavisan jezik.

Gramatiku za jezik AB dobijamo tako što ubacimo novu početnu varijablu $S \to S_1S_2$, te prepišemo gramatike G_1 i G_2. Počinjući od varijable S, dobijamo S_1S_2. Nastavljajući preko S_1 i S_2 dobićemo string iz A nadovezano stringom iz B. Znači, dobićemo string iz AB. Pošto se varijable iz G_1 i G_2 ne ponavljaju,

nećemo dobiti nijedan drugi string. Prema tome, konstruisali smo gramatiku za AB, pa je AB kontekstno nezavisan jezik.

Gramatiku za jezik A^* dobijamo tako što ubacimo novu početnu varijablu $S \to S_1 S|\varepsilon$, te prepišemo gramatiku G_1. Počinjući od varijable S, dobijamo $S_1 S$. Nastavljajući preko S više puta, dobijamo $SS_1 S_1 \ldots S_2$. Mijenjajući S sa ε, te nastavljajući preko svakog S_1, dobićemo string $\omega_1 \omega_2 \ldots \omega_k$, pri čemu svako ω_i $(i = 1, \ldots, k)$ generiše gramatika G_1. Znači, dobićemo string iz A^*. Prema tome, konstruisali smo gramatiku za A^*, pa je A^* kontekstno nezavisan jezik. ■

Kasnije ćemo pokazati da klasa kontekstno nezavisnih jezika nije zatvorena pod operacijama presjeka i komplementiranja.

Slijedeći primjer bi nas mogao navesti na pogrešan zaključak da je klasa kontekstno nezavisnih jezika zatvorena pod operacijom komplementiranja.

▪ **Primjer 3.9** Konstruiši gramatiku za $\overline{A}$, pri čemu je $A = \{0^n 1^n \mid n \geq 0\}$.

Prvo trebamo opisati dati jezik. Imamo da važi $\overline{A} = \{0^n 1 \omega 1^n \mid \omega \in \Sigma^*, n \geq 0\} \cup \{0^n \omega 01^n \mid \omega \in \Sigma^*, n \geq 0\}$. Sada je gramatiku jednostavno konstruisati.

$S \to X|Y$ -varijabla X generiše prvi skup, a Y drugi skup unije;
$X \to 0X1|X_1$
$X_1 \to 1X_2$
$X_2 \to 0X_2|1X_2|\varepsilon$
$Y \to 0Y1|Y_1$
$Y_1 \to Y_2 0$
$Y_2 \to 0Y_2|1Y_2|\varepsilon$.

▪

Tvrdnja 3.3 Klasa kontekstno nezavisnih jezika nije zatvorena u odnosu na presjek i komplement.

Dokaz. Vidi primjere 3.21 i 3.22 ■

3.1.1 Stablo raščlanjivanja

Pomoću stabla raščlanjivanja (eng. *parse tree*) grafički prikazujemo na koji način neka gramatika generiše određeni string.

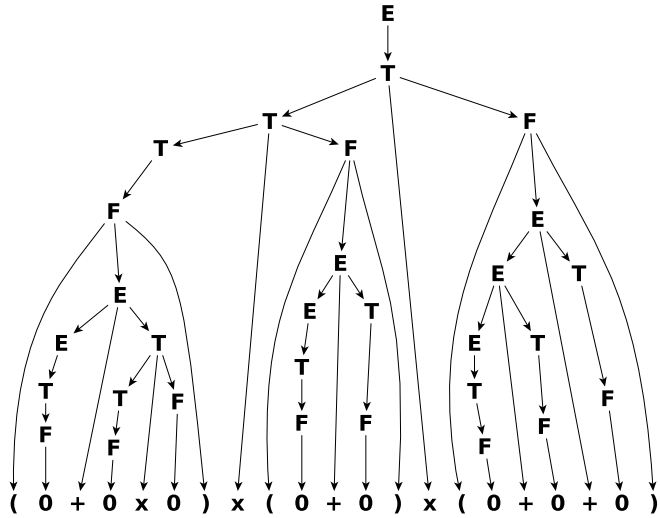
▪ **Primjer 3.10** Za string i gramatiku iz primjera 3.1 dati stablo raščlanjivanja.

Stablo raščlanjivanja je stablo sa korijenom. Korijen stabla je pridružen početnoj varijabli (u ovom slučaju je to E), dok su listovi stabla simboli stringa ω (slika 3.1).

U generisanju stringa ω, prvo pravilo je $E \to T$. Zbog toga ubacujemo čvor T i usmjerenu granu od E prema T.

Slijedeće pravilo je $T \to T \times F$. Ubacujemo varijable T i F, te spajamo postojeći T sa novim T i F i odgovarajućim simbolom $\times$.

Gornji postupak ponavljamo za svako pravilo iz generatornog niza stringa ω nad gramatikom G. Konačni rezultat je dat na slici 3.1.



Slika 3.1: Stablo raščlanjivanja za string $\omega = (0+0\times 0)\times(0+0)\times(0+0+0)$ (primjer 3.10).

■

Sada i formalno definišemo stablo raščlanjivanja.

Definicija 3.4 — Stablo raščlanjivanja. Neka je G kontekstno nezavisna gramatika i $\omega = \alpha_1 \ldots \alpha_n$ string. Za uređeno stablo T kažemo da je *stablo raščlanjivanja* stringa ω za gramatiku G ako važi:

1. korijenu stabla je pridruženo početna varijabla gramatike G;
2. listovima stabla su pridruženi terminali;
3. unutrašnjim čvorovima stabla su pridružene varijable gramatike G;

4. ako je x unutrašnji čvor stabla T, X varijabla pridružena x i $Y_1, \ldots Y_k$ su varijable/terminali pridruženi djeci od x, tada je $X \to Y_1 \ldots Y_k$ pravilo gramatike G;
5. literali pridruženi listovima od T, u datom poretku, formiraju string ω.

3.1.2 Dvosmislenost gramatika i jezika

Koristeći pravila gramatike, može se dogoditi da neki string možemo dobiti na dva različita načina.

Definicija 3.5 Za gramatiku G kažemo da je *nedvosmislena* ako svaki string iz $L(G)$ se može izvesti na tačno jedan način. U suprotnom kažemo da je *dvosmislena*.

■ **Primjer 3.11** Dati gramatiku za $A = \{a^i b^j c^k \mid i = j \vee j = k, i, j, k \geq 0\}$. Da li je data gramatika dvosmislena?

Imamo da važi $A = \{a^i b^i c^k \mid i, k \geq 0\} \cup \{a^i b^j c^j \mid i, j \geq 0\}$. Odgovarajuća gramatika je

$$S \to X|Y$$
$$X \to X_1X_2$$
$$X_1 \to aX_1b|\varepsilon$$
$$X_2 \to cX_2|\varepsilon$$
$$Y \to Y_1Y_2$$
$$Y_1 \to aY_1|\varepsilon$$
$$Y_2 \to bY_2c|\varepsilon.$$

String $\omega = abc$ možemo izvesti na dva načina: $S \to X \to X_1X_2 \to aX_1bcX_2 \to abc$ ili $S \to Y \to Y_1Y_2 \to aY_1bY_2c \to abc$. Prema tome data gramatika je dvosmislena. ■

Za prethodni jezik ne postoji nedvosmislena gramatika (nećemo dokazivati tu tvrdnju). Za takve jezike kažemo da su *dvosmisleni*.

Za neke jezike postoje i dvosmislene i nedvosmislene gramatike.

■ **Primjer 3.12** Za jezik 1^* možemo dati dvije gramatike; jednu nedvosmislenu, a drugu dvosmislenu:

(a) $S \to 1S|\varepsilon$

(b) $S \to XY$
$X \to X1|\varepsilon$
$Y \to Y1|\varepsilon.$

■

Definicija 3.6 — Dvosmisleni jezici. Za kontekstno nezavisan jezik kažemo da je *inherentno dvosmislen* ili samo *dvosmislen* ako ne postoji nedvosmislena gramatika koja ga generiše.

Generalni problem određivanja da li je neki jezik dvosmislen je *neodlučiv* (Poglavlje 4, [18]), tj. ne može se riješiti u konačno mnogo koraka.

3.1.3 Chomsky normalna forma

Chomsky normalna forma nam omogućava da kontekstno nezavisnu gramatiku zapišemo u jednostavnijem obliku.

Definicija 3.7 — Chomsky normalna forma. Chomsky normalna forma je oblik kontekstno nezavisne gramatike dat sa:

$$\begin{aligned} S &\to \varepsilon \quad (\textit{opcionalno}) \\ A &\to BC \\ A &\to a \end{aligned}$$

pri čemu su A, B, C varijable, B i C ne mogu biti početne varijable, $a \in \Sigma$ je terminal.

Slijedećim primjerom demonstriramo kako proizvoljnu gramatiku prikazati u Chomsky normalnoj formi.

■ **Primjer 3.13** Slijedeću gramatiku ćemo konvertovati u Chomsky normalnu formu.

$$\begin{aligned} A &\to BAB \mid B \mid \varepsilon \\ B &\to 00 \mid \varepsilon \end{aligned}$$

Postupak se sastoji od četiri koraka.

(1) Ubacimo novu početnu varijablu:
$S \to A$
$A \to BAB|B|\varepsilon$
$B \to 00|\varepsilon.$

(2) Uklonimo pravila oblika $X \to \varepsilon$. Ako imamo pravilo oblika $Y \to \Omega_1 X \Omega_2$, tada ubacimo novo pravilo $Y \to \Omega_1 \Omega_2$.
Prvo ćemo ukloniti $A \to \varepsilon$:
$S \to A|\varepsilon$
$A \to BAB|B|BB$
$B \to 00|\varepsilon.$

Zatim uklonimo $B \to \varepsilon$:
$S \to A|\varepsilon$
$A \to B|BB|BAB|BA|AB$
$B \to 00.$

(3) Uklonimo pravila oblika $X \to Y$. Za svako pravilo $Y \to \Omega$, ubacimo novo pravilo $X \to \Omega$.

Prvo ćemo ukloniti $A \to B$:
$S \to A|\varepsilon$
$A \to BB|BAB|BA|AB|00$
$B \to 00.$

Zatim ćemo ukloniti $S \to A$.
$S \to BB|BAB|BA|AB|00|\varepsilon$
$A \to BB|BAB|BA|AB|00$
$B \to 00.$

(4) Uvodimo pravila oblika $A \to a$ za svako $a \in \Sigma$. Pravila oblika $X \to \Omega_1 a \Omega_2$ mijenjamo sa $X \to \Omega_1 A \Omega_2$.

U ovom slučaju to je pravilo $X_2 \to 0$.
$S \to BB|BAB|BA|AB|X_2X_2|\varepsilon$
$A \to BB|BAB|BA|AB|X_2X_2$
$B \to X_2X_2$
$X_2 \to 0.$

(5) Sva ostala pravila pretvaramo u oblik $X \to YZ$. Umjesto pravila $X \to Y_1Y_2Y_3$, ubacujemo pravila $X \to X_1Y_3$ i $X_1 \to Y_1Y_2$.
$S \to BB|X_1B|BA|AB|X_2X_2|\varepsilon$
$A \to BB|X_1B|BA|AB|X_2X_2$
$B \to X_2X_2$
$X_1 \to BA$
$X_2 \to 0$

■

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 3.4 Svaka kontekstno nezavisna gramatika se može prikazati u Chomsky normalnoj formi.

Dokaz. Prvo proučite primjer 3.13.

Neka je $G = (V, \Sigma, R, S)$ kontekstno nezavisna gramatika. Primjenom niza transfromacija, gramatiku G ćemo prikazati u Chomsky normalnoj formi.

(1) Neka je G_1 gramatika dobijena iz G dodavanjem nove početne varijable S' i novog pravila $S' \to S$. Ovo je trivijalna transformacija i očigledno važi $L(G) = L(G_1)$.

(2) Gramatiku G_2 ćemo dobiti tako što uklonimo pravila oblika $A \to \varepsilon$. To radimo tako što za svako pravila oblika $B \to \Omega_1 A \Omega_2$ dodamo pravilo $B \to \Omega_1 \Omega_2$. Postupak ponavljamo dok ne dodamo sva pravila koja je moguće dobiti na ovaj način. Na kraju, uklonimo pravilo $A \to \varepsilon$.

Dokažimo da je $L(G_1) = L(G_2)$. Neka je $\omega \in L(G_1)$ i $(\Omega_0, \Omega_1, \ldots, \Omega_n)$ generatorni niz stringa ω u gramatici G_1. Ako se u nizu nije iskorišteno pravilo $A \to \varepsilon$, tada očigledno $\omega \in L(G_2)$. Pretpostavimo da je pravilo $A \to \varepsilon$ iskorišteno u nizu. Tada, u nizu, također je iskorišteno pravilo oblika $B \to \Omega' A \Omega''$. Iz konstrukcije, gramatika G_2 ima pravilo oblika $B \to \Omega' \Omega''$, a upravo ovo pravilo možemo iskoristiti umjesto kombinacije pravila $B \to \Omega' A \Omega''$ i $A \to \varepsilon$. Prema tome, $\omega \in L(G_2)$.

Neka $\omega \in L(G_2)$. Dokažimo da $\omega \in L(G_1)$. Ako generatorni niz stringa ω u G_2 sadrži pravilo koje se ne nalazi u G_1, tada se to pravilo može zamijeniti sa pravilima oblika $B \to \Omega' A \Omega''$ i $A \to \varepsilon$. Prema tome, $\omega \in L(G_1)$.

(3) Gramatiku G_3 dobijamo tako što umjesto pravila oblika $X \to Y$ $(X, Y \in V)$ stavljamo pravilo $X \to \Omega$ $(\Omega \in (V \cup \Sigma)^*)$, pri čemu je pravilo $Y \to \Omega$ već prisutno u G_2. Dokaz da je $L(G_3) = L(G_2)$ je analogan prethodnom slučaju.

(4) Za svaki literal $a \in \Sigma$ ubacimo pravilo oblika $A \to a$, te svako pravilo oblika $B \to \Omega' a \Omega''$ zamijenimo sa $B \to \Omega' A \Omega''$. Ovako dobijenu gramatiku označimo sa G_4 i očigledno je $L(G_3) = L(G_4)$.

(5) Svako pravilo oblika $A \to A_1 A_2 \ldots A_k$ $(k \geq 3)$ zamijenimo sa $A \to B A_k$ i $B \to A_1 \ldots A_{k-1}$. Postupak ponavljamo sve dok ne ostanu samo pravila sa tačno dvije varijable sa desne strane. Ovakvu gramatiku označimo sa G_5.

Gramatika G_5 ima Chomsky normalnu formu i važi $L(G) = L(G_5)$.

■

Ako je gramatika prikazana u Chomsky normalnoj formi, tada svaki string, generisan datom gramatikom, možemo izvesti u linearnom vremenu.

Tvrdnja 3.5 Neka je G gramatika u Chomsky normalnoj formi, $\omega \in L(G)$ i $|\omega| = n > 0$. Tada string ω možemo izvesti u $2n-1$ koraka.

Dokaz. Neka je $\omega = \alpha_1 \ldots \alpha_n$, $(\alpha_i \in \Sigma, i = 1, \ldots, n)$. Za svako α_i imamo po jedno pravilo oblika $X_i \to \alpha_i$. To je n koraka.

Da bi dobili n varijabli X_i $(i = 1, \ldots, n)$, počinjući od početne varijable S, primjenjujemo pravila oblika $X \to YZ$. Svako ovakvo pravilo povećava broj varijabli za 1. Pošto smo počeli sa jednom varijablom S, a želimo dobiti n varijabli X_i $(i = 1, \ldots, n)$, potrebno je $n-1$ koraka.

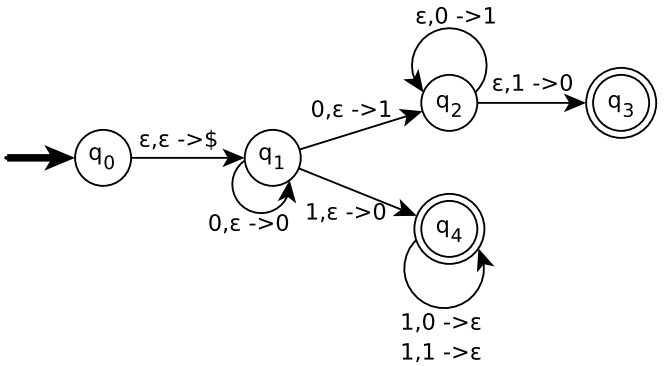
Znači, ukupno imamo $n + (n-1) = 2n-1$ koraka. ■

3.2 Nedeterministički potisni automati

Za razliku od (ne)determinističkih konačnih automata, *potisni automati* (eng. *PDA - Pushdown Automaton*) ima vrstu memorije koja se naziva stek/kamara/stog/gomila (eng. *stack*). To je memorija kod koje možemo pristupiti samo zadnjem pohranjenom podatku. Princip rada ove memorije se naziva *LIFO* (eng. *Last In First Out*).

Kod regularnih jezika smo imali ekvivalentnost regularnih izraza i (ne)determinističkih konačnih automata. U ovom poglavlju ćemo demonstrirati ekvivalentnost kontekstno nezavisnih gramatika i potisnih automata. Dokazaćemo da je jezik kontekstno nezavisan akko ga neki potisni automat prepoznaje.

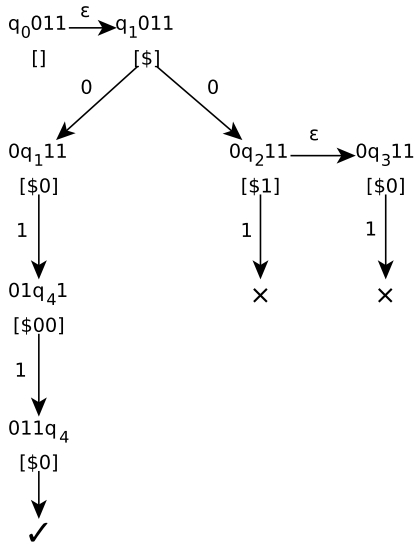
■ **Primjer 3.14** Za potisni automat sa slike 3.2 i ulazni string $\omega = 011$, konstruiši niz stanja kroz koje dati automat prolazi.



Slika 3.2: Primjer potisnog automata (PDA). U oznaci $a, b \to c$, a označava simbol koji se čita sa stringa, b je simbol koji se skida sa steka, dok je c simbol koji se stavlja na stek.

Računanje je dato na slici 3.3, te se odvija slično kao kod NFA, sa razlikom da

ovdje imamo i memoriju u vidu steka. Stek je predstavljen pomoću zagrada $[,]$. Prvi simbol na steku se nalazi skroz desno, npr. ako imamo $[\$01]$ zadnji simbol koji je dodan na stek je 1 i njemu prvom pristupamo. Pošto DFA ne može odrediti da li je stek prazan, na početku stavljamo specijalan simbol \$ da označimo prazan stek.



Slika 3.3: Primjer računanja PDA iz primjera 3.14 i slike 3.2 sa ulaznim stringom $\omega = 011$.

Skup stanja je $Q = \{q_0, q_1, q_2, q_3, q_4\}$, alfabet ulaznog stringa je $\Sigma = \{0, 1\}$, alfabet steka je $\Gamma = \{0, 1, \$\}$. Usmjerene grane predstavljaju funkciju tranzicije i imamo $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$. Tako, npr imamo: $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$, $\delta(q_1, 0, \varepsilon) = \{(q_1, 0), (q_2, 1)\}$.

Sa $\mathfrak{R}$ označimo skup svih nizova stanja kroz koje prolazi dati PDA sa datim ulaznim stringom. U ovom primjeru imamo $\mathfrak{R} = \{(q_0, q_1, q_1, q_4, q_4), (q_0, q_1, q_2), (q_0, q_1, q_2, q_3)\}$. ∎

Potisni automat izgleda kao nedeterministički konačni automat sa stekom. Sada dajemo formalnu definiciju potisnog automata.

Definicija 3.8 — (Nedeterministički) potisni automat. Nedeterministički potisni automat, ili samo potisni automat, je uređena šestorka $(Q,\Sigma,\Gamma,\delta,q_0,F)$, pri čemu

1. Q je konačan skup stanja automata;
2. Σ je konačan alfabet ulaznog stringa;
3. Γ je konačan alfabet potisnog automata i $\Sigma \subseteq \Gamma$;
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathscr{P}(Q \times \Gamma_\varepsilon)$ je funkcija tranzicije;
5. q_0 je početno stanje;
6. $F \subseteq Q$ je skup završnih stanja.

Za nedeterministički potisni automat, ćemo koristiti kraći naziv: potisni automat. Koristićemo i engleske skraćenice: NPDA (*Nondeterministic Pushdown Automaton*) ili samo PDA (*Pushdown Automaton*).

Zadatak 3.1 — Za samostalan rad. Dati primjer PDA koji, za određeni ulazni string, ulazi u beskonačnu petlju, tj. nikada ne staje sa radom. ■

Definicija 3.9 — Računanje potisnog automata nad ulaznim stringom. Neka je $P = (Q,\Sigma,\Gamma,\delta,q_0,F)$ potisni automat i $\omega = \alpha_1 \ldots \alpha_n$ ($\alpha_j \in \Sigma$, $j = 1,\ldots,n$) ulazni string. Sa $\mathfrak{R}_1$ označimo skup konačnih nizova $(q_0,\ldots,q_k)$ takvih da postoje $\beta_i \in \Sigma_\varepsilon$, $\gamma_i, \delta_i \in \Gamma_\varepsilon$, $\Omega_0, \Omega_i, \Omega_i' \in \Gamma^*$ ($i = 1,\ldots,k$) tako da važi:

1. q_0 je početno stanje od P;
2. $(q_{i+1},\delta_i) \in \delta(q_i,\beta_{i+1},\gamma_i)$, za $i = 0,\ldots,k-1$;
3. $\Omega_0 = \varepsilon$, $\Omega_i = \Omega_i'\gamma_i$, $\Omega_{i+1} = \Omega_i'\delta_i$, $(i = 0,\ldots,k-1)$;
4. $\beta_1 \ldots \beta_k = \alpha_1 \ldots \alpha_n$ ili $(\beta_1 \ldots \beta_k = \alpha_1 \ldots \alpha_s, s < n$ i $\delta(q_k,\alpha_{s+1},\gamma_k) = \varnothing)$.

Sa $\mathfrak{R}_2$ označimo skup beskonačnih nizova $(q_0,q_1,\ldots)$ takvih da postoje $\beta_i \in \Sigma_\varepsilon$, $\gamma_i, \delta_i \in \Gamma_\varepsilon$, $\Omega_0, \Omega_i, \Omega_i' \in \Gamma^*$ ($i = 1,2,\ldots$) tako da važi:

1. q_0 je početno stanje od P;
2. $(q_{i+1},\delta_i) \in \delta(q_i,\beta_{i+1},\gamma_i)$, za $i = 0,1,\ldots$;
3. $\Omega_0 = \varepsilon$, $\Omega_i = \Omega_i'\gamma_i$, $\Omega_{i+1} = \Omega_i'\delta_i$, $(i = 0,1,\ldots)$;
4. $\beta_1 \ldots \beta_j = \alpha_1 \ldots \alpha_{s_j}, s_j \leq n, \forall j \geq 0$.

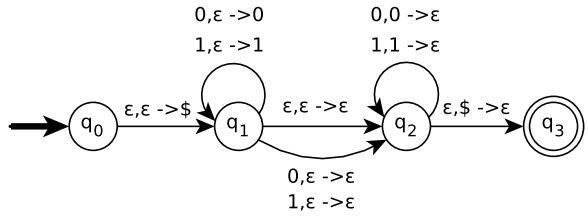
Tada $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2$ nazivamo *računanje automata N nad ulaznim stringom ω*. Ako postoji niz $(q_0,\ldots,q_k) \in \mathfrak{R}$ tako da je q_k završno stanje i $\beta_1 \ldots \beta_k = \omega$, tada kažemo da *automat N prihvata string ω*. U suprotnom kažemo da *automat N odbija (ili ne prihvata) string ω*.

U slijedećim primjerima konstruišemo potisne automate za neke konte-

kstno nezavisne jezike.

■ **Primjer 3.15** Konstruiši potisni automat koji prepoznaje jezik $A = \{\omega \mid \omega = \omega^R\}$.

Traženi automat je dat na slici 3.4.



Slika 3.4: PDA koji prihvata string akko je palindrom.

Jezik A je skup svih palindroma. Za string $\omega = x_1 \ldots x_n$ kažemo da je palindrom ako vrijedi $x_i = x_{n+1-i}$, $\forall i \in \{1,...,n\}$. Ovo još možemo pisati i kao $x_i = x_{n+1-i}$, $\forall i \in \{1,...,\lfloor n/2 \rfloor\}$.

Princip rada je slijedeći. Na početku, na stek stavljamo \$ - oznaku za prazan stek. Ideja je da string čitamo do njegove sredine, te da na stek stavljamo simbole koje smo pročitali. Nakon što prođemo sredinu stringa, sa steka skidamo simbole, te ih poredimo sa simbolima koji se nalaze u drugoj polovini stringa. Ako na svakom mjestu imamo podudaranje, string je palindrom. Ako na barem jednom mjestu imamo različite simbole, string nije palindrom.

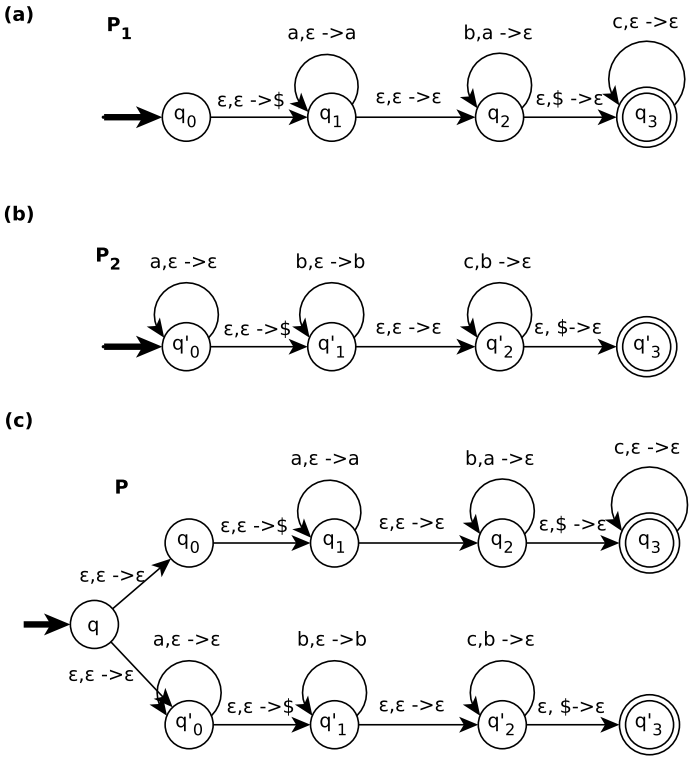
Problem je što, prilikom čitanja stringa, ne znamo kada smo stigli do sredine. Zato, u svakom koraku, nedeterministički granamo na dva slučaja: ako smo stigli do sredine stringa i ako nismo. U prvom slučaju počinjemo skidati simbole sa steka, te ih porediti sa sa simbolima koje čitamo u nastavku. U drugom slučaju, nastavljamo sa stavljanjem pročitanih simbola na stek.

Ako pročitamo cijeli ulazni string i stek ostane prazan, ulazni string je palindrom. U suprotnom, nije palindrom. ■

■ **Primjer 3.16 — Unija potisnih automata.** Konstruisati potisni automat koji prepoznaje jezik $A = \{a^i b^j c^k \mid i = j \vee j = k, i, j, k \geq 0\}$.

Uočimo da je $\Sigma = \{a, b, c\}$. Neka je $A_1 = \{a^i b^i c^k \mid i, k \geq 0\}$, $A_2 = \{a^i b^j c^j \mid i, j \geq 0\}$. Tada je $A = A_1 \cup A_2$. Sa P_1 i P_2 označimo PDA koji prepoznaju A_1 i

za A_2 (slika 3.5). Na slici 3.5c je data unija P automata P_1 i P_2.



Slika 3.5: **(a)** PDA koji prepoznaje jezik $\{a^i b^i c^k \mid i,k \geq 0\}$. **(b)** PDA koji prepoznaje jezik $\{a^i b^k c^k \mid i,k \geq 0\}$. **(c)** PDA koji prepoznaje jezik $A = \{a^i b^j c^k \mid i = j \vee j = k, i,j,k \geq 0\}$, dobijen kao unija PDA pod *(a)* i *(b)*.

Automat P_1 konstruišemo tako što čitamo simbole a i stavljamo ih na stek. Zatim, nastavljamo sa čitanjem simbola b, te za svaki pročitani b, skinemo po jedan a sa steka. Nakon što pročitamo sve b-ove, stek treba ostati prazan. Preostale c-ove samo pročitamo, bez ikakvij operacija na steku.

Automat P_2 konstruišemo na sličan način.

Uniju automata P_1 i P_2 konstruišemo slično kao uniju *NFA*: dodamo novo početno stanje, koje sa starim početnim stanjim povežemo ε-strelicama. Kod PDA, ε-strelica je oblika $\varepsilon, \varepsilon \to \varepsilon$. ∎

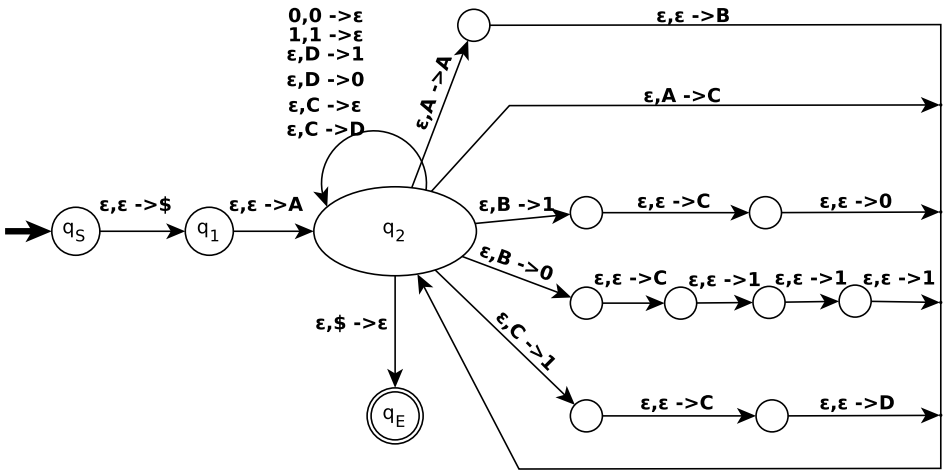
Zadatak 3.2 — Za samostalan rad. Konstruisati automat koji predstavlja nadovezivanje i automat koji predstavlja zvjezdicu datih automata. Konstrukcija je slična kao kod NFA, jedino ovdje trebate paziti na stek. ■

Slijedeći primjer demonstrira kako kontekstno nezavisnu gramatiku pretvoriti u (ekvivalentni) potisni automat.

■ **Primjer 3.17** Datu gramatiku konvertovati u ekvivalentni potisni automat.

$$\begin{aligned} A &\rightarrow BA \mid C \\ B &\rightarrow 0C1 \mid 1\,1\,1\,C0 \\ C &\rightarrow DC1 \mid D \mid \varepsilon \\ D &\rightarrow 0 \mid 1 \end{aligned}$$

Rješenje je dato na slici 3.6. Na stek stavimo oznaku za prazan stek (\$), te početno stanje gramatike. Za svako pravilo gramatike oblika $X \rightarrow Y_1 \ldots Y_k$, konstruišemo niz stanja automata, koji rade slijedeće: za steka skinemo X, a na stek stavimo $Y_k, Y_{k-1}, \ldots, Y_1$, te sa ulaznog stringa ne čitamo simbol. Na kraju, sa stringa čitamo a, sa steka skidamo a ($a \in \Sigma$) i na stek ne stavljamo ništa. Ako dođemo u situaciju da je string prazan, tj. da sa steka skidamo \$, idemo u završno stanje. Ako smo, na ovaj način, pročitali cijeli string, automat ga prihvata.



Slika 3.6: PDA ekvivalentan sa CFG iz primjera 3.17.

Za samostalan rad, datom gramatikom generisati neki string, te ga iskoristiti kao ulaz za potisni automat, koji ste konstruisali. Uočiti sličnost između koraka gramatike i automata. Možete uzeti primjer $A \to C \to DC1 \to 1C1 \to 101$. Uvijek se prvo mijenja prva varijabla sa lijeve strane. ∎

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 3.6 Za svaku kontekstno nezavisnu gramatiku G postoji ekvivalentni potisni automat P, tj. važi $L(G) = L(P)$.

Dokaz. Prvo proučite primjer 3.17.

Opišimo na koji način konstruišemo PDA P za datu CFG $G = (V, \Sigma, R, S)$. Na stek stavimo oznaku za prazan stek (\$), te početno stanje gramatike. Za svako pravilo gramatike oblika $X \to Y_1 \dots Y_k$, konstruišemo niz stanja automata, koji rade slijedeće: za steka skinemo X, a na stek stavimo $Y_k, Y_{k-1}, \dots, Y_1$, te sa ulaznog stringa ne čitamo simbol. Na kraju, sa stringa čitamo a, sa steka skidamo a ($a \in \Sigma$) i na stek ne stavljamo ništa. Ako dođemo u situaciju da je string prazan, tj. da sa steka skidamo \$, idemo u završno stanje. Ako smo, na ovaj način, pročitali cijeli string, automat ga prihvata.

Dokažimo da automat prihvata string ω akko ga generiše gramatika G. Dokaz ćemo sprovesti indukcijom po dužini generatornog niza, u oznaci k.

Neka je $k = 1$. Tada je $\Omega_0 = S$ i $\Omega_1 = a \in \Sigma$. To znači da je $\omega = a$ i da ga generišemo sa $S \to a$, pri čemu je S početna varijabla gramatike. Automat P će prihvatiti string $\omega = a$ na slijedeći način: $q_S \overset{\varepsilon,\varepsilon\to\$}{\to} q_1 \overset{\varepsilon,\varepsilon\to S}{\to} q_2 \overset{\varepsilon,S\to a}{\to} q_2 \overset{a,a\to\varepsilon}{\to} q_2 \overset{\varepsilon,\$\to\varepsilon}{\to} q_E$. Pošto smo pročitali cijeli string i završili u prihvatnom stanju, automat će prihvatiti string $\omega = a$.

Pretpostavimo da tvrdnja važi za sve generatorne nizove dužine $1, \dots, k$ ($k \geq 1$). Dokažimo da važi i za generatorne nizove dužine $k+1$. Neka je $S \to S_1 \dots S_t$. Na početku se na steku nalazi $S_1, \dots, S_t$, pri čemu je S_t na vrhu steka. Da bi automat P pristupio $S_1, \dots, S_{t-1}$, prvo se S_t mora konvertovati u string ω_t, te se taj string pomoću pravila $a, a \to \varepsilon$ ($a \in \Sigma$) skida sa steka.

Umjesto $S_1, \dots, S_{t-1}$ postavimo simbol \$, te posmatrajmo podgramatiku G_t sa početnom varijablom S_t. Pošto je dužina generatornog niza za string ω_t manja od k, možemo primjeniti induktivnu prepostavku. Gramatika G_t generiše string w_t, pa će mašina P prihvatiti ω_t. Tj, imaćemo konfiguraciju sa stanjem q_2 i simbolom \$ na steku. Sada, umjesto \$ vratimo $S_1, \dots, S_{t-1}$, te opet, kao maloprije, primjenimo induktivnu pretpostavku i dobijemo da će mašina P prihvatiti $\omega_1, \dots, \omega_{t-1}$, odnosno da će prihvatiti string ω.

Tvrdnju u drugom smjeru možemo dokazati analogno, pomoću matematičke indukcije. ■

Slijedeći primjer demonstrira kako za potisni automat konstruisati ekvivalentnu kontekstno nezavisnu gramatiku.

■ **Primjer 3.18** Potisni automat dat na slici 3.7a konvertuj u ekvivalentnu kontekstno nezavisnu gramatiku.

Prvi korak je da PDA normalizujemo, tj. pretvorimo u pogodniji oblik. Pravila normalizacije su data na slici 3.7c-e. Ako imamo više završnih stanja, ubacimo novo završno stanje koje je sa starim završnim stanjima povezano ε-strelicama. Opet ubacimo novo završno stanje, a na starom završnom stanju ispraznimo stek. Na kraju, obezbjedimo se da svaka tranzicija stavlja string na stek ili ga skida sa steka, ali ne oboje.

Slika 3.7b prikazuje normalizovani automat.

Sada možemo pristupiti konstruisanju odgovarajuće gramatike. Za svaki par (p,q) stanja automata ćemo konstruisati varijablu A_{pq} (slika 3.8).

Za svako stanje q dodamo jedno pravilo $A_{qq} \to \varepsilon$.

Za svaka tri stanja p,q,r dodamo pravilo $A_{pr} \to A_{pq}A_{qr}$. Ovo možemo (a i ne moramo) optimizirati tako što ćemo postaviti uslov da mora postojati put od p do q, te od q od r.

Za svaka četiri stanja p,q,r,s, za koja važi: $(q,x) \in \delta(p,a,\varepsilon)$ i $(s,\varepsilon) \in \delta(r,b,x)$, za neke $a,b,x \in \Sigma$, dodajemo pravilo $A_{ps} \to aA_{qr}b$.

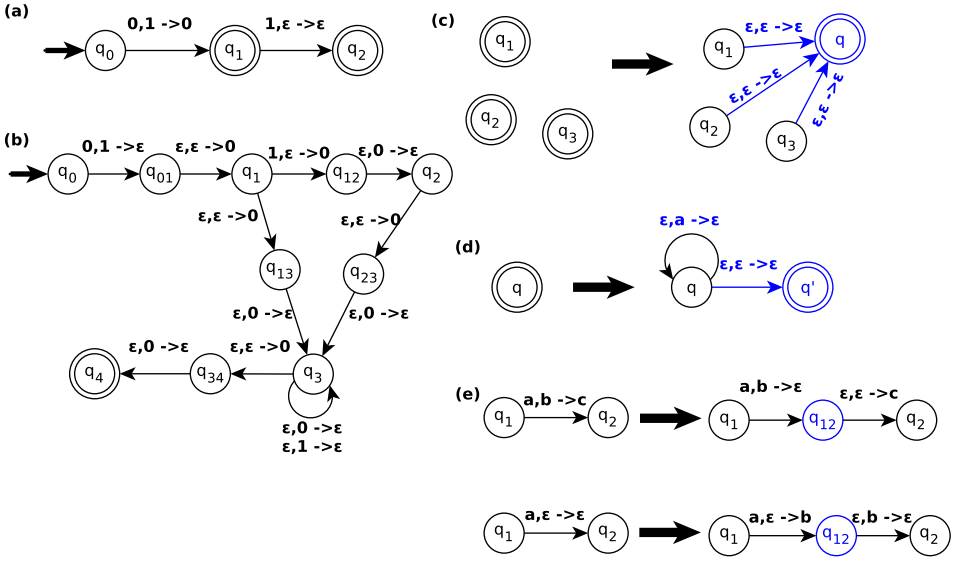
Ako su S i E početno i završno stanje automata, tada je A_{SE} početna varijabla gramatike. U našem primjeru, početna varijabla je $A_{q_0q_4}$.

Na osnovu prethodnih pravila generišemo slijedeću gramatiku. Zbog velikog broja varijabli, nećemo napisati sva pravila gramatike.

$A_{q_0q_4} \to A_{q_0q_{01}}A_{q_{01}q_4} \mid A_{q_0q_1}A_{q_1q_4} \mid \ldots$
$A_{q_0q_0} \to \varepsilon$
$A_{q_1q_1} \to \varepsilon$
$A_{q_2q_2} \to \varepsilon$
$A_{q_3q_4} \to \varepsilon$
$A_{q_4q_4} \to \varepsilon$
$A_{q_{12}q_{12}} \to \varepsilon$
$A_{q_{13}q_{23}} \to \varepsilon$
$A_{q_{23}q_{23}} \to \varepsilon$
$A_{q_{34}q_{34}} \to \varepsilon$

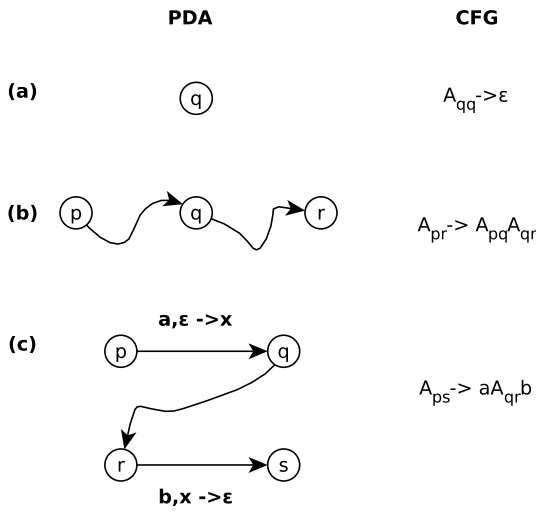
$$
\begin{aligned}
&A_{q_0q_2} \to A_{q_0q_1}A_{q_1q_2}\\
&A_{q_0q_2} \to A_{q_0q_{12}}A_{q_{12}q_2}\\
&A_{q_0q_{13}} \to A_{q_0q_{01}}A_{q_{01}q_{03}}\\
&A_{q_0q_{13}} \to A_{q_0q_1}A_{q_1q_{03}}\\
&\dots\\
&A_{q_3q_4} \to A_{q_3q_{34}}A_{q_{34}q_4}
\end{aligned}
$$

$$
\begin{aligned}
&A_{q_0q_3} \to \varepsilon A_{q_1q_{13}}\varepsilon\\
&A_{q_0q_3} \to \varepsilon A_{q_1q_{23}}\varepsilon\\
&\dots\\
&A_{q_0q_4} \to \varepsilon A_{q_1q_{34}}\varepsilon
\end{aligned}
$$



Slika 3.7: PDA iz (a) transformišeno u PDA iz (b) koristeći pravila (c-e). **(c)** Ako imamo više završnih stanja, ubacimo novo završno stanje koje je sa starim završnim stanjima povezano ε-strelicama. **(d)** Nakon što dobijemo samo jedno završno stanje, opet ubacimo novo završno stanje, a na starom završnom stanju ispraznimo stek. **(e)** Obezbjedimo se da svaka tranzicija stavlja string na stek ili ga skida sa steka, ali ne oboje. Važi $b \neq \varepsilon, c \neq \varepsilon$, ali može biti $a = \varepsilon$.

■



Slika 3.8: Generalna pravila za konvertovanje PDA u CFG. **(a)** Svakom stanju q pridružujemo pravilo gramatike oblika $A_{qq} \to \varepsilon$. **(b)** Ako postoji put od p do q i od q do r, tada ubacujemo pravilo oblika $A_{pr} \to A_{pq}A_{qr}$. **(c)** Ako imamo luk od p do q, sa steka ne skidamo ništa i stavljamo x; put od q do r; te luk od r do s, pri čemu skidamo x sa steka i ne stavljamo ništa, tada ubacujemo pravilo oblika $A_{ps} \to aA_{qr}b$. Dozvoljeno je $a = \varepsilon$ ili $b = \varepsilon$.

Prethodni primjer ilustruje slijedeću tvrdnju.

Tvrdnja 3.7 Za svaki potisni automat postoji ekvivalentna kontekstno nezavisna gramatika.

Dokaz. Neka je P potisni automat. Koristeći pravila opisana na slici 3.7(c-e) možemo dobiti PDA koji: ima tačno jedno završno stanje, isprazni stek prije prihvatanja stringa (ako ga prihvata) i svaka tranzicija stavlja simbol na stek ili ga skida sa steka. Ovako dobijeni PDA označimo sa P'.

Gramatiku G iz P' možemo koristiti koristeći pravila opisana na slici 3.8.

*N*eka je ω string kojeg prihvata automat P'. Dokažimo da ga generiše gramatika G.

Prvo ćemo dokazati neke leme.

Lema 3.1 Pretpostavimo da sa stringom α možemo početi iz stanja p sa praznim stekom i stići u stanje q sa praznim stekom. Tada, koristeći pravila gramatike G i počinjući sa varijablom A_{pg}, možemo generisati string α.

Dokaz. Koristićemo indukciju po dužini puta od p do q. Dokažimo da tvrdnja važi ako je dužina puta jednaka 2. Tada imamo situaciju: $p \overset{a,\varepsilon\to x}{\to} s \overset{b,x\to\varepsilon}{\to} q$. Tada smo sa stringom $\omega = ab$ stigli od p do q. Sa druge strane, $A_{pq} \to aA_{ss}b$ i $A_{ss} \to \varepsilon$ generišu isti string.

Neka tvdrnja važi za sve puteve dužine $1,2\ldots,k$ ($k \geq 2$). Dokažimo da važi za put dužine $k+1$. Ako počnemo u p sa praznim stekom, tada je prvi korak stavljanje simbola, recimo x, na stek. Neka je s prvo stanje u koje ulazimo sa praznim stekom.

Ako je $s \neq q$, tj. ako s dolazi prije q, tada imamo $A_{pq} = A_{ps}A_{sq}$. Neka je $\omega = \omega_1\omega_2$, pri čemu ω_1 odgovara putu od p do s, a ω_2 odgovara putu od s do q. Prema induktivnoj pretpostavci, A_{ps} generiše ω_1 i A_{sq} generiše ω_2. Iz $A_{pq} = A_{ps}A_{sq}$ imamo da A_{pq} generiše $\omega = \omega_1\omega_2$.

Neka je sada $s = q$. Uzmimo da put od p do q, koji generiše ω, ima oblik: $p \overset{a,\varepsilon\to x}{\to} p_1 \to \ldots \to q_1 \overset{b,x\to\varepsilon}{\to} q$. Pošto od p_1 do q_1 simbol x nije skidan sa steka, to znači da sa praznim stekom možemo krenuti od p_1 i stići do q_1 sa praznim stekom. Neka je $\omega = a\omega'b$. Iz induktivne pretpostavke imamo da sa $A_{p_1q_1}$ možemo generisati ω'. Iz $A_{pq} \to aA_{p_1q_1}b$ imamo da, koristeći pravila gramatike G i koristeći A_{pq} kao početnu varijablu, možemo generisati ω. ■

Neka je q_0 početno stanje i q_f završno stanje automata P'. Svaki string ω koji automat P' prihvata počinje iz q_0 i završava u q_f sa praznim stekom. Iz leme 3.1 imamo da $A_{q_0q_f}$ generiše string ω. Pošto je $A_{q_0q_f}$ početna varijabla gramatike G, imamo da G generiše ω, što je i trebalo dokazati.

*N*eka je sada ω string kojeg generiše gramatika G. Dokažimo da automat P' prihvata ω. Dokaz je sličan kao u suprotnom smjeru.

Lema 3.2 Neka gramatika G, sa A_{pq} kao početnom varijablom, generiše string ω. Tada u automatu P' možemo krenuti od p sa praznim stekom i stići do q sa praznim stekom sa stringom ω.

Dokaz. Dokaz je sličan dokazu leme 3.1 i nećemo ga ponavljati. ■

Pošto G generiše ω, i $A_{q_0q_f}$ je početna varijabla od G, to sa stringom ω možemo krenuti od q_0 sa praznim stekom i stići do q_f sa praznim stekom. Drugim riječima, automat P' prihvata string ω, što je i trebalo dokazati. ■

Iz prethodne dvije tvrdnje direktno slijedi slijedeća tvrdnja.

Teorema 3.1 Jezik je kontekstno nezavisan akko ga neki potisni automat prepoznaje.

Dokaz. Neka je A kontekstno nezavisan jezik. Tada postoji kontekstno nezavisna gramatika G koja generiše A, tj. važi $L(G) = A$. Iz tvrdnje 3.6 postoji potisni automat P tako da je $L(P) = L(G)$. Pošto je $L(G) = A$, imamo $L(P) = A$. Znači, potisni automat P prepoznaje jezik A.

Neka je P potisni automat, koji prepoznaje A, tj. važi $L(P) = A$. Tada postoji kontekstno nezavisna gramatika G ekvivalentna sa P (tvrdnja 3.7), tj. važi $L(G) = L(P)$. Znači, $L(G) = A$, tj. gramatika G generiše jezik A, pa je A kontekstno nezavisan jezik. ■

K Prethodna teorema govori da ako se neki problem može predstaviti pomoću kontekstno nezavisnog jezika, tada se taj problem može riješiti pomoću potisnog automata.

Važi i obratno, ako se neki problem može riješiti pomoću potisnog automata, onda se može prikazati u obliku kontekstno nezavisnog jezika.

K Ranije smo konstatovali da PDA može ući u beskonačnu petlju (zadatak 3.1).

Ovo možemo izbjeći tako što ćemo PDA P konvertovati u ekvivalentnu gramatiku G, a gramatiku prikazati u Chomsky normalnoj formi G'.

Ako P prihvata string ω, tada G' generiše ω u $2|\omega| - 1$ koraka. Rješenje je da, pomoću G_1, generišemo sve stringove dužine $2|\omega| - 1$, te provjerimo da li se među njima nalazi ω. Ako da, tada P prihvata ω. U suprotnom ga odbija.

Ovo razmatranje je formalizovano zadatkom 4.10.

Slijedeća tvrdnja je očigledna.

Tvrdnja 3.8 Svaki regularan jezik je i kontekstno nezavisan.

Dokaz. Neka je A regularan jezik. Tada postoji NFA N koji prepoznaje A, tj. važi $L(N) = A$. Pošto je N ujedno i PDA koji ne koristi stek, tada postoji PDA koji prepoznaje A, pa je A kontekstno nezavisan jezik.

Znači, proizvoljan regularan jezik je kontekstno nezavisan. Ovim je tvrdnja dokazana. ■

3.2.1 Pumpajuća lema za kontekstno nezavisne jezike

Slično kao kod regularnih jezika, koristićemo Pumpajuću lemu za kontekstno nezavisne jezike da bi dokazali da neki jezik nije kontekstno nezavisan.

Prvo uvedimo pojam koji će nam trebati u dokazu slijedeće teoreme.

Definicija 3.10 — Rekurzivna varijabla. Za varijablu R gramatike G kažemo da je *rekurzivna* ako postoji generatorni niz $R \to \Omega_1 \to \Omega_2 \ldots \to \Omega_k$ tako da $\Omega_i = \alpha R \beta$, za neko $i \in \{1,2\ldots,k\}$, $\alpha, \beta \in (V \cup \Sigma)^*$, $\alpha \neq \varepsilon$ ili $\beta \neq \varepsilon$.

■ **Primjer 3.19** Gramatika G data sa

$S \to 0AB1$
$A \to 01 \mid CD$
$B \to 00A \mid 1$
$C \to B1 \mid 0$
$D \to D0 \mid 1$

ima rekurzivnu varijablu A. Primjer za to je $A \to CD \to B1D \to 00A1D$. ■

Lema 3.3 Gramatika G generiše beskonačan jezik akko ima rekurzivnu varijablu.

Dokaz. Pretpostavimo da gramatika G ima rekurzivnu varijablu. Dokažimo da generiše beskonačan jezik. Neka je A rekurzivna varijabla. Tada važi $A \to^* \alpha A \beta$, pri čemu sa $\to^*$ označavamo neki generatorni niz. Ponavljajući prethodni postupak k puta imamo: $A \to^* \alpha^k A \beta^k$. Pošto je $\alpha \neq \varepsilon$ ili $\beta \neq \varepsilon$ za $k = 1,2\ldots$ dobijamo beskonačno mnogo stringova.

*S*ada pretpostavimo da gramatika G nema rekurzivnu varijablu. Znači da proizvoljan put od početne varijable S do nekog terminala u proizvoljnom

stablu rašlanjivanja ne ponavlja varijable. Prema tome, dužina svakog takvog puta je ograničena brojem varijabli, pa je i broj mogućih stabala raščlanjivanja konačan, odnosno broj mogućih stringova je konačan. ■

Teorema 3.2 — Pumpajuća lema za kontekstno nezavisne jezike. Neka je A kontekstno nezavisan jezik. Tada postoji $p > 0$ (pumpajuća dužina), tako da za svako $s \in A$ za koje je $|s| \geq p$ postoji podjela $s = uvxyz$ tako da važi:

1. $uv^k xy^k z \in A$ za svako $k \geq 0$;
2. $|vy| > 0$;
3. $|vxy| \leq p$.

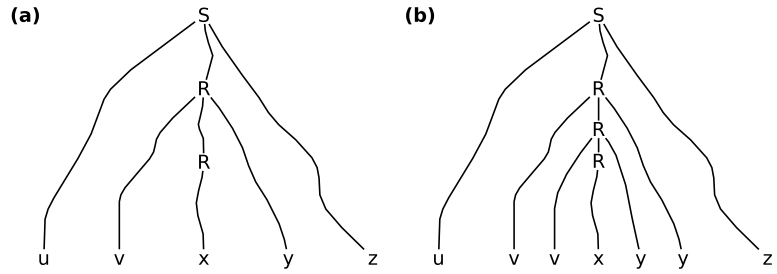
Dokaz. Neka je G gramatika koja generiše jezik A. Posmatrajmo dva slučaja: jezik A je konačan ili beskonačan.

Neka je jezik A konačan. Tada postoji $p > 0$ tako da je $|s| < p$ za svako $s \in A$. Za ovakvo p važi tvrdnja leme.

Neka je sada jezik A beskonačan i neka je G gramatika koja generiše A. Posmatrajmo sve stringove iz A koji se mogu generisati bez rekurzivnog ponavljanja varijable. Takvih stringova ima konačno mnogo (lema 3.3), pa su ograničeni po dužini. Neka je p_1 gornja granica za dužine takvih stringova. To znači da, ako $s \in A$ i $|s| > p_1$, tada postoji generatorni niz za s u kojem se neka varijabla (recimo R) rekurzivno ponavlja. Neka je $S \to^* uRz \to^* uvRyz \to^* uvxyz = s$. Znači, imamo $R \to^* vRy$, tj. umjesto R možemo staviti vRy proizvoljan broj puta (slika 3.9). Na taj način imamo: $S \to^* uRz \to^* uvRyz \to^* uvvRyyz = uv^2Ry^2z \to^* uv^kRy^kz \to^* uv^kxy^kz$. Koristeći pravila gramatike G, generisali smo string uv^kxy^kz, pa važi $uv^kxy^kz \in A$.

Pošto je R rekurzivna varijabla, vrijedi $v \neq \varepsilon$ ili $y \neq \varepsilon$, tj. $|vy| > 0$.

Dokažimo da možemo izvršiti podjelu tako da važi $|vxy| \leq p$. Neka je b najveći broj varijabli i terminala sa desne strane proizvoljnog pravila gramatike G. To znači da je faktor grananja bilo kojeg stabla raščlanjivanja manji ili jednak b. Neka je N broj varijabli gramatike G. Tada bilo koji put dužine veće $N+1$ u stablu sadrži varijablu koja se ponavlja, a to je ujedno i rekurzivna varijabla (podrazumijevamo da nemamo pravila oblika $X \to X$, koja su bespotrebna). Neka je $p_2 = b^{N+1}$. Uzmimo rekurzivnu varijablu R koja je najniže postavljena stablu raščlanjivanja stringa s. To znači da put od R od proizvoljnog terminala je dužine najviše $(N+1)$, te je broj terminala tog stabla najviše b^{N+1}, tj. R



Slika 3.9: **(a)** Stablo raščlanjivanja za string $uvxyz$. **(b)** Stablo raščlanjivanja za string uv^2xy^2z.

generiše podstring dužine najviše b^{N+1}. Podstring kojeg generišemo počinjući u R je vxy, pa je $|vxy| \leq p_2$.

Ako uzmemo $p = max\{p_1, p_2\}$, dobijamo tvrdnju leme. ■

Pumpajuću lemu za CFL ćemo koristiti da dokažemo da neki jezik A nije kontekstno nezavisan. Da bi to dokazali, pretpostavićemo suprotno, tj. da jeste kontekstno nezavisan. Zatim, odabraćemo neki string $\omega \in A$, te koristeći Pumpajuću lemu za kontekstno nezavisne jezike, konstruisati string $\omega_i \in A$. Sa druge strane, imaćemo da $\omega_i \notin A$. Na ovaj način ćemo dobiti kontradikciju sa pretpostavkom da je A kontekstno nezavisan.

■ **Primjer 3.20** Dokaži da jezik $A = \{a^n b^n c^n \mid n \geq 0\}$ nije kontekstno nezavisan, pri čemu je $\Sigma = \{a, b, c\}$.

Pretpostavimo suprotno, jezik A je kontekstno nezavisan. Tada važi Pumpajuća lema za CFL. Neka je p pumpajuća dužina. Uzmimo $s = a^p b^p c^p$ i neka je $s = uvxyz$ podjela iz Pumpajuće leme za CFL. Posmatrajmo slučajeve.

Slučaj 1. Neka je vxy sadržano u $a^p b^p$. Tada $s_2 = uv^2xy^2z$ sadrži više a-ova ili b-ova nego c-ova, pa $s_2 \notin A$, što je u kontradikciji sa Pumpajućom lemom za CFL.

Slučaj 2. Slučaj kada je vxy sadržano u $b^p c^p$ je analogan.

Podstring vxy ne može sadržati a i c, jer bi tada sadržao i b^p, pa bi imali $|vxy| > p$, što je u kontradikciji sa podjelom stringa s.

Iz dobijenih kontradikcija imamo da A nije kontekstno nezavisan. ■

K Posmatrajmo problem: "Da li je dati string oblika $a^n b^b c^n$". Iz prethodnog primjera zaključujemo da ovaj problem ne možemo riješiti pomoću potisnih automata.

Sada možemo dokazati da klasa kontekstno nezavisnih jezika nije zatvorena pod operacijama presjeka i komplementiranja. To ćemo uraditi slijedećim primjerima.

■ **Primjer 3.21** Iskoristi jezike $A = \{a^m b^n c^n \mid m,n \geq 0\}$, $B = \{a^m b^m c^n \mid m,n \geq 0\}$ da dokažeš da klasa kontekstno nezavisnih jezika nije zatvorena pod operacijom presjeka.

Iz primjera 3.7 imamo da su jezici A i B kontekstno nezavisni. Jezik $A \cap B = \{a^n b^n c^n \mid n \geq 0\}$ nije kontekstno nezavisan (primjer 3.20). Znači, klasa kontekstno nezavisnih jezika nije zatvorena pod presjekom. ■

■ **Primjer 3.22** Koristeći prethodni primjer, dokazati da klasa kontekstno nezavisnih jezika nije zatvorena pod operacijom komplementiranja.

Pretpostavimo suprotno, da klasa CFL jeste zatvorena pod komplementiranjem. Neka su A i B jezici iz primjera 3.21. Tada su $\overline{A}$ i $\overline{B}$ CFL, pa je i $\overline{A} \cup \overline{B}$ CFL. Iz De-Morganovih obrazaca imamo $\overline{A} \cup \overline{B} = \overline{A \cap B}$, pa je i $\overline{A \cap B}$ CFL. Pošto smo pretpostavili da je klasa CFL zatvorena pod komplementiranjem, imamo da je $\overline{\overline{A \cap B}} = A \cap B$ kontekstno nezavisan jezik, što je u kontradikciji sa primjerom 3.21.

Znači, klasa CFL nije zatvorena pod komplementiranjem. ■

Iako klasa kontekstno nezavisnih jezika nije zatvorena pod operacijom presjeka (primjer 3.21), ipak možemo iskazati nešto po tom pitanju.

Tvrdnja 3.9 Neka je A regularan i B kontekstno nezavisan jezik. Tada je $A \cap B$ kontekstno nezavisan jezik.

Dokaz. Dokaz je sličan konstrukciji presjeka dva DFA (primjer 2.19), pa predlažemo da prvo pogledate taj primjer.

Neka je $D = (Q_1, \Sigma, \delta_1, a_0, F_1)$ DFA koji prepoznaje A, $N = (Q_2, \Sigma, \Gamma, \delta_2, a_0, F_2)$ NFA koji prepoznaje B i $N' = (Q', \Sigma, \Gamma, \delta', q_0, F')$ NFA koji prepoznaje $A \cap B$, a koji tek treba da konstruišemo. Ideja je da string paralelno pustimo kroz oba automata D i N. Ako oba automata prihvate string, tada i N prihvata string.

Da bi imali paralelno računanje, treba da pratimo stanja od D i N istovremeno. To ćemo postići tako što uzmemo $Q' = Q_1 \times Q_2 = \{(a,b) \mid a \in Q_1, b \in Q_2\}$. Pošto N' prihvata string akko ga prihvataju D i F, to je $F' = F_1 \times F_2$.

Na koji načina N' računa, tj. kako izgleda funkcija tranzicije δ' ? Automat N' ima stek identičan steku automata N. Ako se nalazimo u stanju $(a,b) \in Q'$ i čitamo simbol α, δ' na prvu koordinatu (tj. a) djeluje na isti način kao δ_1, a na drugu koordinatu (tj. b) djeluje na isti način kao δ_2 i vrši iste operacije na steku. Formalno ovo možemo zapisati kao:

$$\delta((a,b),\alpha,\beta) = \bigcup_{(b',\beta') \in \delta_2(b,\alpha,\beta)} \{(\delta_1(a,\alpha),b',\beta')\}.$$

Na ovaj način simuliramo paralelan rad automata D i N. Prema tome konstruisali smo PDA koji prepoznaje $A \cap B$, pa je $A \cap B$ kontekstno nezavisan jezik. ■

■ **Primjer 3.23** Dokaži da jezik $A = \{0^n \mid n \text{ je prost broj}\}$ nije kontekstno nezavisan.

Pretpostavimo suprotno. Tada važi Pumpajuća lema za CFL i neka je $p > 0$ pumpajuća dužina. Neka je $k > p$ i k je prost broj. Uzmimo $s = 0^k = uvxyz \in A$. Tada je $s_i = uv^i xy^i z = 0^{k+i\cdot m}$, pri čemu je $m = |vy| > 0$. Pošto $s_i = uv^i xy^i z \in A$ za svako $i \geq 0$, to je $(k + i \cdot m)$ prost broj za svako $i \geq 0$.

Sa druge strane, za $i = k$ broj $k + i \cdot m = k + k \cdot m = k \cdot (1 + m)$ se može prikazati kao proizvod dva cijela broja veća od 1, pa nije prost broj. Dobijamo kontradikciju. Prema tome A nije CLF. ■

■ **Primjer 3.24** Neka je $B = \{\omega \mid \omega \text{ ima isti broj } a\text{-ova}, b\text{-ova i } c\text{-ova}\}$ jezik nad alfabetom $\Sigma = \{a,b,c\}$. Dokaži da B nije kontekstno nezavisan.

Imamo npr. da $bbacac \in B$ i $aabbc \notin B$. Pretpostavimo suprotno, neka je B kontekstno nezavisan jezik. Neka je $C = a^*b^*c^*$. Tada je C regularan jezik, jer smo ga predstavili pomoću regularnog izraza. Iz tvrdnje 3.9 imamo da je jezik $A = B \cap C = \{a^n b^n c^n \mid n \geq 0\}$ CFL, što je u kontradikciji sa primjerom 3.20.

Znači, jezik B nije CLF. ■

Uočite da u prethodnom primjeru nismo koristili Pumpajuću lemu za kontekstno nezavisne jezike, već tvrdnju 3.9.

■ **Primjer 3.25** Dokaži da jezik $A = \{\omega\omega\omega \mid \omega \in \Sigma^*\}$ nije kontekstno nezavisan.

Pretpostavimo suprotno, da jeste CFL. Neka je $p > 0$ pumpajuća dužina. Neka je $s = 0^p 10^p 10^p 1 \in A$ i neka je $s = uvxyz$ podjela iz Pumpajuće leme za CFL. Tada, na osnovu Pumpajuće leme za CFL, $s_2 = uv^2xy^2z \in A$.

Posmatrajmo slučajeve.

(a) Podstring vxy ne sadrži 1. Tada $v = 0^a$, $y = 0^b$ i $a + b > 0$. Znači, sa v^2xy^2 smo povećali broj nula i to u samo jednoj grupi 0^p, tj. $s_2 = 0^{p+a+b}10^p10^p1$ ili $s_2 = 0^p10^{p+a+b}10^p1$ ili $s_2 = 0^p10^p10^{p+a+b}1$.

Pretpostavimo da je $s_2 = 0^p10^{p+a+b}10^p1$ (ostala dva slučaja se dokazuju analogno). Zbog $s_2 \in A$ imamo $s_2 = \omega_1\omega_2\omega_3$ i $\omega_1 = \omega_2 = \omega_3$. Pošto je ω_3 sufiks od s_2, to ω_3 završava sa 1, pa ω_1 i ω_2 završavaju sa 1. U s_2 imamo tačno tri jedinice, pa mora biti $\omega_1 = 0^p1$, $\omega_2 = 0^{p+a+b}1$, $\omega_3 = 0^p1$, što je u kontradikciji sa $\omega_1 = \omega_2 = \omega_3$.

(b) Sada pretpostavimo da vxy sadrži 1. Pošto je $|vxy| \leq p$, to vxy sadrži tačno jednu 1. Naime, kada bi vxy sadržavalo barem dvije 1, onda bi sadržavao i 0^p, pa bi imali $|vxy| > p$, što je nemoguće.

(b.1) Neka v ili y sadrži 1. Znači, $s_2 = uvvxyyz$ sadrži dodatnu 1, tj. s_2 sadrži tačno 4 jedinice. Sa druge strane, $s_2 \in A$, pa $s_2 = \omega'\omega'\omega'$, a odavde imamo da broj jedinica koje s_2 sadrži mora biti djeljiv sa 3, što je u kontradikciji sa zaključkom da s_2 sadrži tačno 4 jedinice.

(b.2) Neka x sadrži 1. Tada je v sadržano u jednom 0^p, a y u susjednom 0^p. To znači da se sa $s_2 = uvvxyyz$ pumpaju dva 0^p. Tada je $s_2 = 0^{p+a}10^{p+b}10^p1$ ili $s_2 = 0^p10^{p+a}10^{p+b}1$. Slično kao u slučaju *(a)*, dobijamo kontradikciju (kako?).

Iz *(a)* i *(b)* imamo da A nije CFL. ∎

3.2.2 k-potisni automati

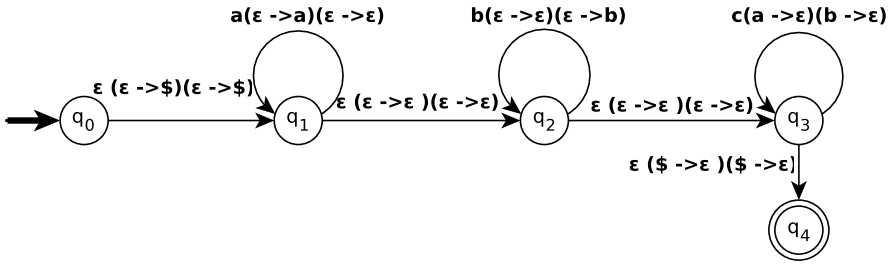
Vidjeli smo da nedeterminističke konačne automate možemo posmatrati kao potisne automate bez steka. Znači, dodavanjem jednog steka proširili smo skup jezika koje možemo prepoznati, tj. proširili smo skup regularnih jezika na skup kontekstno nezavisnih jezika.

Pod k-potisnim automatom podrazumijevamo potisni automat sa k stekova. Znači, 0-potisni automat je NFA, 1-potisni automat je PDA.

Postavlja se pitanje, šta ako imamo potisni automat sa više od jednog steka; da li je skup jezika, koje možemo prepoznati sa takvim automatima, širi od skupa kontekstno nezavisnih jezika? Odgovor je potvrdan i nalazi se u slijedećem primjeru.

■ **Primjer 3.26** Ovim primjerom ćemo dokazati da 2-potisni automati prepoznaju širu klasu jezika od "običnih" 1-potisnih automata. Posmatrajmo jezik $A = \{a^n b^n c^n \mid n \geq 0\}$. Iz Primjera 3.20 imamo da ne postoji potisni automat koji prepoznaje A.

Konstruisaćemo 2-PDA koji prepoznaje jezik A (slika 3.10). Ideja je da na prvi stek stavljamo sve simbole a, na drugi stek sve simbole b. Na kraju, za svako pročitano c, skidamo po jedan simbol sa prvog i drugog steka. String će biti prihvaćen ako je pročitan do kraja i oba steka su prazna.



Slika 3.10: 2-PDA koji prepoznaje $A = \{a^n b^n c^n \mid n \geq 0\}$.

■

Sada se postavlja pitanje da li 3-potisni automati prepoznaju širu klasu jezika od 2-potisnih automata. Odgovor je negativan i dokazan je u slijedećoj tvrdnji.

Tvrdnja 3.10 Klasa jezika koju prepoznaju k-potisni automati ($k \geq 3$) je jednaka klasi jezika koju prepoznaju 2-potisni automati.

Dokaz. Tvrdnju ćemo prvo dokazati za $k = 3$. Neka je A jezik kojeg prepoznaje neki 3-potisni automat. Dokažimo da postoji 2-potisni automat koji prepoznaje A.

Slika 3.11 opisuje kako se tranzicija u 3-PDA može predstaviti kao niz tranzicija u 2-PDA.

Neka je sada $k > 3$ i neka su $S_1, S_2, ..., S_k$ stekovi k-PDA. Odaberimo tri proizvoljna steka, recimo S_{k-2}, S_{k-1} i S_k. Analogno prethodnom slučaju, mi ova tri steka možemo zamijeniti sa dva steka: S'_{k-2} i S'_{k-1}. Na taj način dobijamo automat sa $k-1$stekova: $S_1, S_2, ..., S'_{k-1}$. Ponavljajući ovaj postupak,

snižavamo broj stekova za 1, dok ne dobijemo automat sa 2 steka, tj. 2-PDA. ■

Koja je to klasa jezika koju prepoznaju 2-potisni automati? Ta je klasa mnogo šira nego što se na prvi pogled može pretpostaviti. Naime, 2-potisni automati su ekvivalentni Turingovim mašinama, koje uvodimo u slijedećem poglavlju.

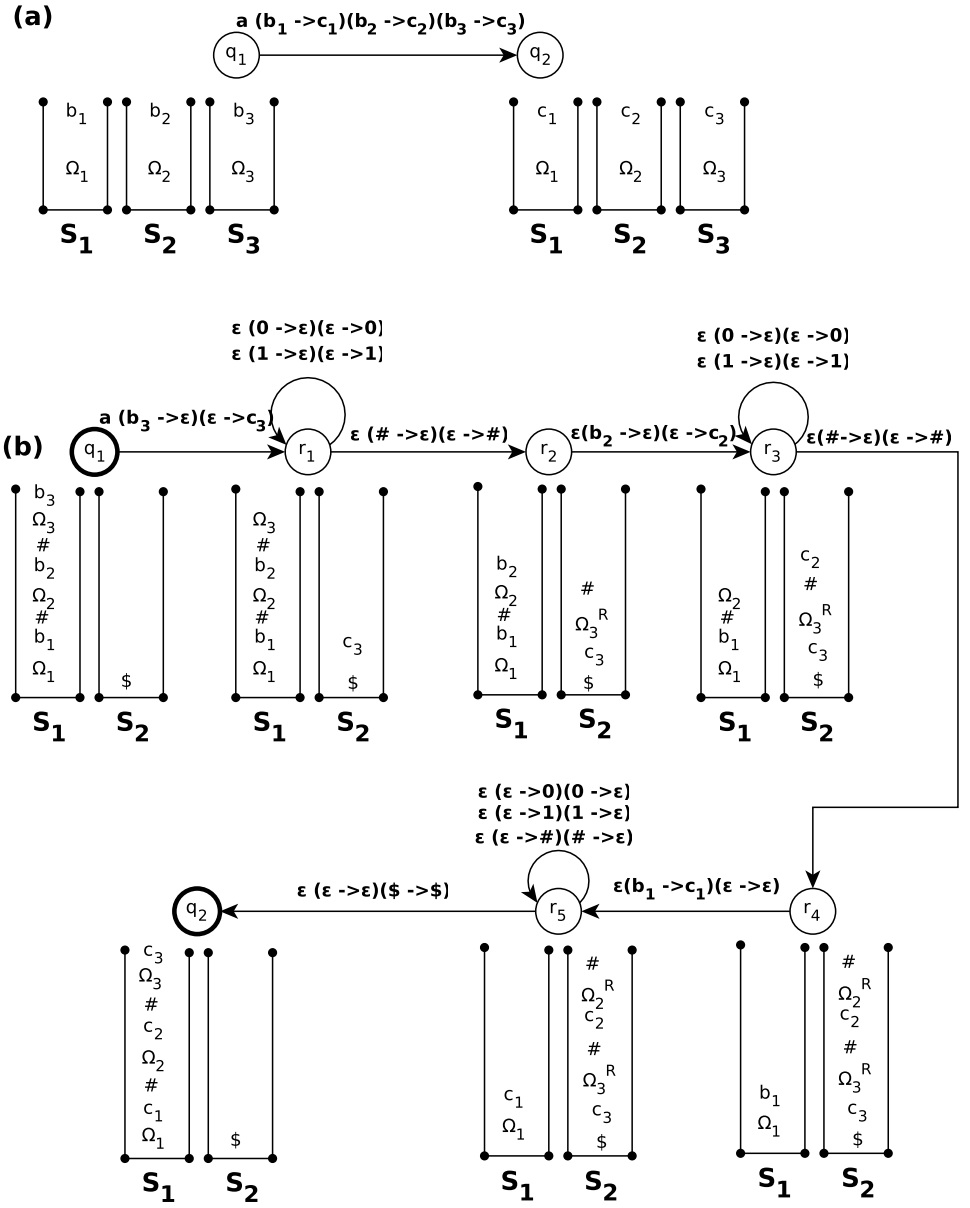
3.3 Primjena gramatika

Ovdje ćemo navesti jednu primjenu gramatike u programskim jezicima.

Programski jezik se sastoji od niza pravila sličnih onim iz zadatka 3.6 - vidi naredne zadatke za samostalan rad.

Kompajler programski kod tretira kao ulazni string ω. Jedan od zadataka koji obavlja je da provjeri da li gramatika, koja odgovara programskom jeziku, može generisati dati string ω, tj. da li može generisati dati programski kod. Ako ne može, onda javlja grešku (eng. *syntax error*). Ovaj proces obavlja dio kompajlera koji se naziva *parser*. U tom procesu, konstruiše i određenu vrstu stabla raščlanjivanja (*parse tree*).

Gramatika za C++ nije kontekstno nezavisna. Slika 3.12 prikazuje dio gramatike za C++17, koji se nalazi u [25]. Simbol ":" predstavlja "$\rightarrow$".



Slika 3.11: Transformacija 3-PDA u ekvivalentan 2-PDA.

init-statement:
expression-statement
simple-declaration

...
selection-statement:
if constexpr_opt (init-statement_opt condition) statement
if constexpr_opt (init-statement_opt condition) statement **else** statement
switch (init-statement_opt condition) statement

iteration-statement:
while (condition) statement
do statement while (expression) ;
for (init-statement condition_opt ; expression opt) statement
for (for-range-declaration : for-range-initializer) statement

Slika 3.12: Dio gramatike za C++.

Zadaci za samostalan rad

Zadatak 3.3 Neka je $\Sigma = \{(,)\}$. Simbol (nazivamo *lijeva zagrada*, a simbol) nazivamo *desna zagrada*.

Za string $\omega \in \Sigma$ kažemo da je *string pravilno balansiranih zagrada* ako je svaka lijeva zagrada uparena sa desnom zagradom koja dolazi poslije, a svaka desna zagrada je uparena sa lijevom zagradom koja dolazije prije. Dalje, ako lijeva zagrada l_1 dolazi prije lijeve zagrade l_2, tada r_1, desna zagrada uparena sa l_1, dolazi poslije r_2, desne zagrade uparene sa l_2, tj. važi: $...(_{l_1}...(_{l_2}...)_{r_2}...)_{r_1}$.

Zagrade koje se javljaju kod matematičkih izraza imaju ovu osobinu.

Zadatak je da se konstruiše kontekstno nezavisna gramatiku za jezik $A = \{\omega \mid \omega \text{ je string pravilno balansiranih zagrada}\}$. ■

Zadatak 3.4 Konstruiši kontekstno nezavisnu gramatiku koja generiše:
(*a*) $A = \{\omega \mid \omega \text{ ima isti broj 0 i 1}\}$;
(*b*) $B = \{\omega s \omega^R \mid \omega, s \in \Sigma^*\}$;
(*c*) $C = \{0^m 1^n \mid m \leq 2n\}$.

■

Zadatak 3.5 Konstruiši PDA za jezike iz prethodnog zadatka. ■

Zadatak 3.6 Data je kontekstno nezavisna gramatika:
START->ASSIGN | IF-THEN | IF-THEN-ELSE;
IF-THEN->if condition then START;
IF-THEN-ELSE->if condition then START else START;
ASSIGN->a:=1 | a:=2;
pri čemu je $\Sigma = \{if, condition, then, else, a := 1, a := 2\}$ i $V = \{START, IF-THEN, IF-THEN-ELSE, ASSIGN\}$.
Koristeći gornju gramatiku generiši string:

$$\omega = \text{"if condition then a:=1 else a:=2"}.$$

■

K Prethodni zadatak demonstrira kako se gramatika (ne nužno kontekstno nezavisna) može iskoristiti da se opišu pravila nekog programskog jezika. Kompajler provjerava da li se programski kod (tj. dati string) može generisati pomoću pravila gramatike koja opisuje programski jezik. Ako ne može, prijavljuje grešku.

Zadatak 3.7 Dokaži da slijedeći jezici nisu kontekstno nezavisni (podrazumijevamo $\Sigma = \{0, 1\}$):
(*a*) $A = \{0^p \mid p \text{ je prost broj}\}$;
(*b*) $B = \{1^{n^2} \mid n \geq 0\}$;
(*c*) $C = \{0^{2^n} \mid n \geq 0\}$;
(*d*) $D = \{0^p 1^n \mid p \text{ je prost broj i } n > p\}$.

■

Prazna stranica.

4. Turingove mašine

Automati koje smo spominjali imaju niz ograničenja: ograničenu memoriju, nemogućnost da čitaju ulazni string u svim smjerovima, nemogućnost fleksibilnog upisa. U ovom poglavlju uvodimo pojam Turingove mašine, koja nema ova ograničenja i predstavlja teoretski model klasičnog kompjutera.

Vidjeli smo da smo imali jednostavne probleme koje nismo mogli riješiti pomoću DFA, NFA ili PDA. Turingova mašina može riješiti bilo koji problem, koji se može riješiti algoritamskim putem.

4.1 Definicija Turingove mašine

Karakteristike Turingove mašine su:

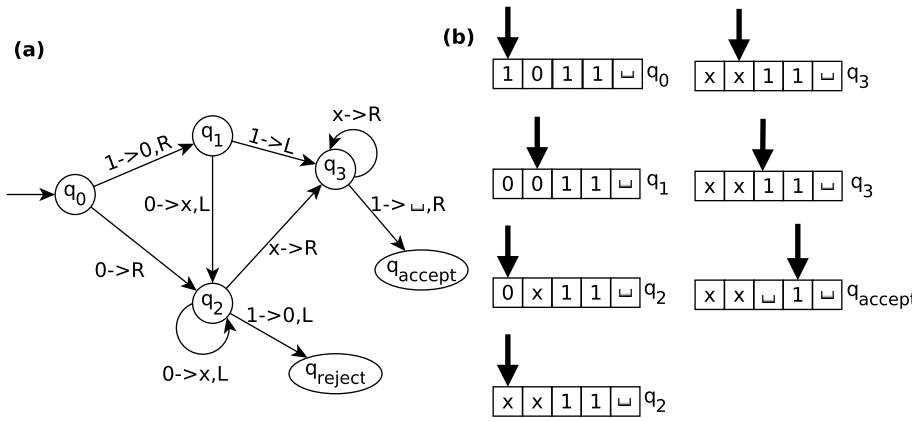
- ima memorijsku traku, koja je sa lijeve strane ograničena, a sa desne je beskonačna (dovoljno velika);
- ima glavu koja čita i piše simbole na traku;
- ulazni string uvijek počinje na prvom polju trake i glava se na početku nalazi na prvom polju;
- glava se u svakom potezu pomiče lijevo ili desno (L/R);

- ako je glava došla na početak trake i pomakne se ulijevo, ostaje gdje je i bila;
- mašina ne zna kada je došla na početak trake;
- podrazumijevamo da je deterministička (nema grananja).

Polje, tj. memorijska ćelija, Turingove mašine može da ne sadrži simbol, tj. da bude prazno. Simbol za prazno mjesto je $\sqcup$. Uočite da je $\varepsilon \neq \sqcup$.

Turingova mašina ćemo skraćeno pisati TM.

■ **Primjer 4.1** Za datu Turingovu mašinu i ulazni string $\omega = 1011$, ispiši niz stanja. Navesti parametre Turingove mašine (skup stanja, funkciju tranzicije, prihvatno i odbijajuće stanje).



Slika 4.1: Primjer računanja Turingove mašine. **(a)** Turingova mašina iz primjera 4.1. **(b)** Niz konfiguracija na ulaznom stringu $\omega = 1011$.

Niz konfiguracija je dat sa:
$q_0 1011$,
$0 q_1 011$,
$q_2 0 \times 11$,
$q_2 \times \times 11$,
$\times q_3 \times 11$,
$\times \times q_3 11$,
$\times \times \sqcup q_{accept} 1$.

Za razliku od DFA, NFA i PDA, string ne mora biti pročitan do kraja. Čim dođemo u stanje q_{accept} ili q_{reject}, string se prihvata ili odbija. Ako u trenutnom stanju nije definisana akcija za pročitani simbol, string se automatski odbija.

Tako, npr. string $\omega_1 = 11$, imamo: $q_0 11$, $0q_1 1$, $q_3 01$, $0q_{reject}1$. Pošto u q_3 nemamo definisanu akciju za simbol 0, automatski string se odbija, tj. podrazumijevamo da za nedefinisani simbol postoji strelica prema stanju q_{reject}. Pri tome, isti simbol ostaje na traci i glava za čitanje se pomjera udesno (mada nije bitno u kojem smjeru se pomjera glava za čitanje, jer je string odbijen).

Skup stanja je dat sa $Q = \{q_0, q_1, q_2, q_3, q_{accept}, q_{reject}\}$. Alfabet ulaznog stringa je dat sa $\Sigma = \{0,1\}$, dok je alfabet trake dat sa $\Gamma = \{0,1,\times,\textvisiblespace\}$.

Funkcija tranzicije $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ je data slijedećom tabelom:

δ	q_0	q_1	q_2	q_3	q_{accept}	q_{reject}
0	$(q_2,0,R)$	$(q_2,\times,L)$	$(q_2,\times,L)$	$(q_{reject},0,R)$	$(q_{accept},0,R)$	$(q_{reject},0,R)$
1	$(q_1,0,R)$	$(q_3,1,L)$	$(q_{reject},0,L)$	$(q_{accept},\textvisiblespace,R)$	$(q_{accept},1,R)$	$(q_{reject},1,R)$
$\times$	$(q_{reject},\times,R)$	$(q_{reject},\times,R)$	$(q_3,\times,R)$	$(q_3,\times,R)$	$(q_{accept},\times,R)$	$(q_{reject},\times,R)$
$\textvisiblespace$	$(q_{reject},\textvisiblespace,R)$	$(q_{reject},\textvisiblespace,R)$	$(q_{reject},\textvisiblespace,R)$	$(q_{reject},\textvisiblespace,R)$	$(q_{accept},\textvisiblespace,R)$	$(q_{reject},\textvisiblespace,R)$

Uvodimo i novi pojam, koji nismo ranije koristili; pojam *konfiguracije Turingove mašine*. U našem primjeru, konfiguracije su:

$(\varepsilon, q_0, 1011)$,
$(0, q_1, 011)$,
$(\varepsilon, q_2, 0\times 11)$,
$(\varepsilon, q_2, \times\times 11)$,
$(\times, q_3, \times 11)$,
$(\times\times, q_3, 11)$,
$(\times\times\textvisiblespace, q_{accept}, 1)$. ∎

Sada ćemo dati formalnu definiciju Turingove mašine.

Definicija 4.1 — Turingova mašina. Turingova mašina je uređena sedmorka: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ pri čemu važi:

1. Q je konačan skup stanja;
2. Σ je konačan alfabet ulaznog stringa i $\textvisiblespace \notin \Sigma$;
3. Γ je konačan alfabet trake, $\textvisiblespace \in \Gamma$ i $\Sigma \subset \Gamma$;
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ je funkcija tranzicije;
5. $q_0 \in Q$ je početno stanje;
6. $q_{qccept} \in Q$ je prihvatno stanje;
7. $q_{reject} \in Q$ je odbijajuće stanje i $q_{reject} \neq q_{accept}$.

Da bi definicija Turingove mašine bila potpuna, trebamo definisati računanje nad ulaznim stringom. Prije toga ćemo definisati pojam *konfiguracije Turingove mašine*.

Definicija 4.2 — Konfiguracija Turingove mašine. Uređena trojka (u,q,v) se naziva *konfiguracija Turingove mašine*, ako se Turingova mašina nalazi u stanju q, na traci je string uv i glava pokazuje na prvi simbol stringa v.

Definicija 4.3 — Računanje Turingove mašine nad ulaznim stringom. Neka je $T = (Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$ Turingova mašina i ω ulazni string. Neka je $S_i = (u_i,q_i,v_i)$ konačan ili beskonačan niz konfiguracija tako da važi:

- $u_0 = \varepsilon$, q_0 je početno stanje od T, $v_0 = \omega$;
- $(q_{i+1},c_i,M) = \delta(q_i,b_i)$ tako da važi:
  - ako je $M = L$ (slika 4.2a), tada je $u_i = u_i'a_i$, $v_i = b_iv_i'$, $u_{i+1} = u_i'$, $v_{i+1} = a_ic_iv_i'$;
  - ako je $M = R$ (slika 4.2b), tada je $v_i = b_iv_i'$, $u_{i+1} = u_ic_i$, $v_{i+1} = v_i'$;

  pri čemu $a_i,b_i,c_i \in \Gamma$, $u_i,v_i,u_i',v_i' \in \Gamma^*$.

Ako je S_i $(i = 0,\ldots,k)$ konačan niz, tada $q_k \in \{q_{accept},q_{reject}\}$, $q_i \notin \{q_{accept}, q_{reject}\}$ $(i = 0\ldots,k-1)$. Za $q_k = q_{accept}$ kažemo da T *prihvata*, dok za $q_k = q_{reject}$ kažemo da T *odbija* string ω.

Ako je S_i beskonačan niz, tada $q_i \notin \{q_{accept},q_{reject}\}$ $(\forall i)$ i kažemo da T *ulazi u petlju*.

Niz S_i nazivamo *računanje Turingove mašine T nad ulaznim stringom ω*.

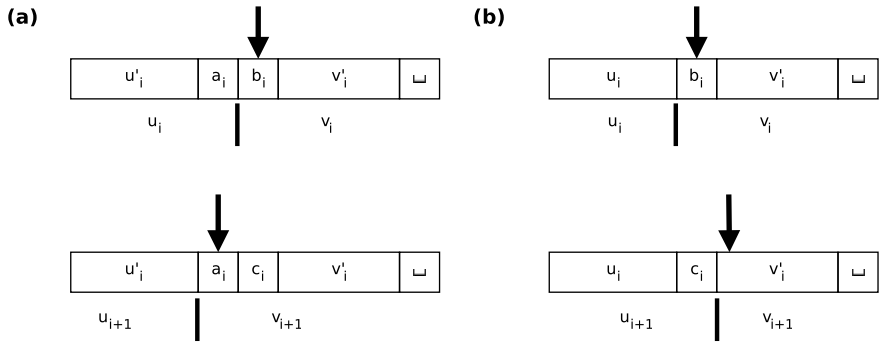
Prethodna definicija kaže da Turingova mašina ne mora vratiti *accept* ili *reject*, već da se može izvršavati bez prestanka. Slijedeći primjer daje jednu takvu Turingovu mašinu.

■ **Primjer 4.2** Turingova mašina opisana na slici 4.3 ne vraća ni *accept* ni *reject*, bez obzira na ulazni string, već ulazi u petlju.

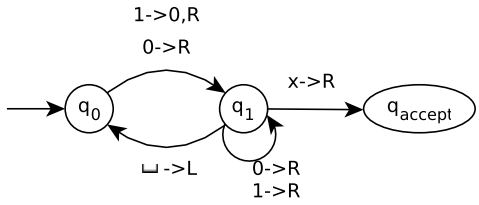
■

Znači, Turingova mašina može da vrati *accept*, *reject* ili da uđe u petlju. Ako kažemo da *Turingova mašina ne vraća accept* (odnosno da *ne prihvata string*), to znači da vraća *reject* ili ulazi u petlju. Slično, ako kažemo da *Turingova mašina ne vraća* reject, znači da vraća *accept* ili ulazi u petlju.

Naravno, nama su korisnije mašine koje uvijek vraćaju rezultat. Zato posmatramo podklasu Turingovih mašina - *odlučivače* (eng. *decider*) ili



Slika 4.2: Prelazak iz jedne konfiguracije u drugu. **(a)** Glava umjesto b_i upisuje c_i i pomjera se ulijevo. **(b)** Glava umjesto b_i upisuje c_i i pomjera se udesno.



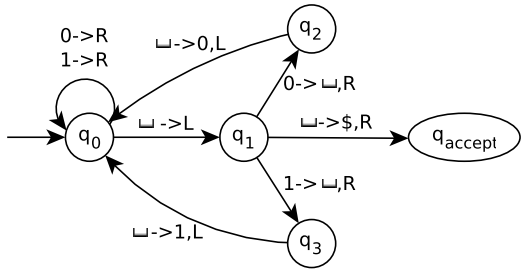
Slika 4.3: Primjer Turingove mašine koja nikada ne vraća rezultat, tj. koja ne staje ni za jedan ulazni string.

totalne Turingove mašine (eng. *total Turing machine*).

Definicija 4.4 — Odlučivač. Turingova mašina koja uvijek vraća rezultat, tj. uvijek vraća *accept* ili *reject*, se naziva *odlučivač* ili *totalna Turingova mašina*.

Prepoznavanje jezika se definiše na isti način kao u definiciji 2.5, tj. Turingova mašina M prepoznaje jezik A, ako je A skup svih stringova koje M prihvata. U tom slučaju pišemo $L(M) = A$.

Definicija 4.5 — Turing-prepoznatljiv i odlučiv jezik. Za jezik kažemo da je *Turing-prepoznatljiv* ili *rekurzivno prebrojiv* ako postoji Turingova mašina koja ga prepoznaje. Za jezik kažemo da je *odlučiv*, ako postoji odlučivač koji ga prepoznaje.



Slika 4.4: Turingova mašina koja pomjera ulazni string za jedno mjesto udesno, a na početak stavlja $.

Svaki odlučiv jezik je Turing-prepoznatljiv. Obratno ne mora da važi, kao što ćemo vidjeti u odjeljku *4.4 Osobine Turing-prepoznatljivih i odlučivih jezika*.

Vidjeli smo da Turingova mašina nema funkcionalnost da prepozna da se nalazi na početku trake. Ako nam je potrebno, na početak trake možemo staviti oznaku $, slično kao kod potisnih automata.

■ **Primjer 4.3** Konstruisati dijagram Turingove mašine, koja ulazni string pomjera za jedno mjesto udesno, a na početak trake stavlja $.

Slika 4.4 prikazuje rješenje. Mašina prvo ide do kraja stringa – čita 0 i 1, dok ne dođe do ␣. Zatim, pomjera se jedno mjesto ulijevo. Čita simbol (0 ili 1), umjesto njega upisuje ␣ i pomjera se jedno mjesto udesno. Zatim, na ␣ upisuje simbol koji je pročitan[1] i ide ulijevo.

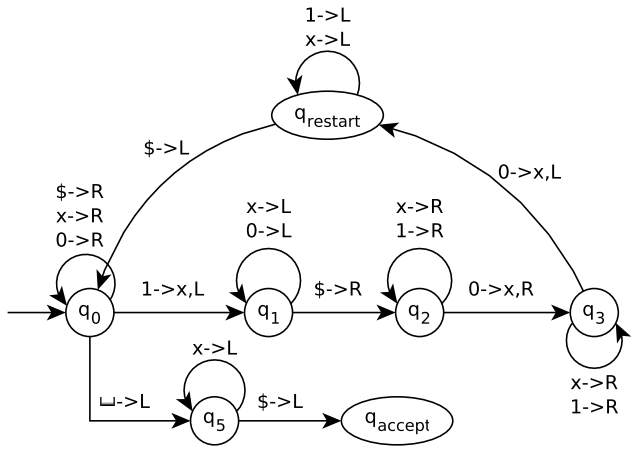
Glava je sada na ␣ (stanje q_0). Ide jedno mjesto ulijevo i dolazi na slijedeći simbol. Postupak se ponavlja, dok se cijeli string ne pomjeri jedno mjesto udesno. Kako mašina zna da je došla na početak trake?

Nakon što glava dođe na ␣ (stanje q_0), dobija naredbu da se pomjeri jedno mjesto ulijevo. Pošto je na početku trake, glava se ne pomjera, te i dalje pokazuje na ␣. U tom slučaju, stavlja simbol $ i prelazi u stanje q_{accept}.

■

K Ako nam je potrebno, podrazumijevaćemo da string počinje sa $. Nećemo posebno

[1]Mašina pamti koji je simbol pročitan na osnovu stanja q_2 i q_3; ako je pročitana 0 ide u q_2, a ako je pročitano 1 ide u q_3.



Slika 4.5: Turingova mašina koja provjerava da li string ima tačno dva puta više 0 od 1.

naglašavati kako se string pomjera jedno mjesto udesno, osim ako se eksplicitno ne traži u problemu.

■ **Primjer 4.4** Konstruiši Turingovu mašinu koja provjerava da li ulazni string ima tačno dva puta više 0 od 1. Ako ima, vraća *accept*, inače vraća *reject*.

Jedno od mnogih mogućih rješenja je dato na slici 4.5. Podrazumijevamo da string počinje sa $. Mašina skenira traku i za svake dvije prekrižene nule, prekriži po jednu jedinicu. Ako je našla jednu nulu, a nije drugu, vraća *reject*. Ako je našla dvije nule, a nije jedinicu, vraća *reject*. Ako nije našla nijednu nulu, a jeste jedinicu, vraća *reject*. Ako su ostali samo simboli $\times$, vraća *accept*.

■

4.2 Church-Turingova teza

Intuitivno, *algoritam* možemo shvatiti kao tačno definisan, konačan niz koraka koji vodi do rješenja problema. Church-Turingova teza, u svom najjednostavnijem obliku, kaže da je intuitivno poimanje algoritma ekvivalentno Turingovoj mašini.

Rezimirajući dosadašnje ekvivalencije mašina sa drugim strukturama, imamo tabelu:

mašina	ekvivalentna struktura
DFA, NFA	regularni izrazi
PDA	kontekstno nezavisne gramatike
TM	algoritam; neograničene gramatike

Neograničene gramatike ćemo obraditi kasnije.

Church-Turingovu tezu ćemo prihvatiti kao tačnu. Znači, ako samo želimo dokazati egzistenciju Turingove mašine, bez konstrukcije dijagrama, dovoljno je konstruisati algoritam, tj. pseudokod algoritma.

U slijedećem primjeru posmatramo jezik koji nije kontekstno nezavisan.

■ **Primjer 4.5** Dokaži da postoji Turingova mašina koja prepoznaje jezik $A = \{a^n b^n c^n \mid n \geq 0\}$.

U primjeru se traži da se samo dokaže egzistencija Turingove mašine. Znači, dovoljno je konstruisati algoritam. Rješenje je dato pseudokodom algoritam 4.1. U primjeru 3.20 smo dokazali da jezik A nije kontekstno nezavisan, tj. ne postoji PDA (a samim tim ni NFA ni DFA) koji prepoznaje A.

■

4.3 Varijante Turingove mašine

U ovom odjeljku ćemo dati razne varijante Turingovih mašina koje su ekvivalentne sa "standardnom" Turingovom mašinom. Radi jednostavnosti, nećemo navoditi formalne definicije, ali njih nije teško dobiti iz formalne definicije "standardne" Turingove mašine.

4.3.1 Turingova mašina sa nepomičnom glavom

Vidjeli smo da Turingova mašina, na svakoj tranziciji, dobije naredbu da pomjeri glavu ulijevo ili udesno (L/R). Možemo posmatrati i Turingovu mašinu koja ima opciju da ne pomjera glavu (S). Nju ćemo nazvati *Turingova mašina sa nepomičnom glavom*.

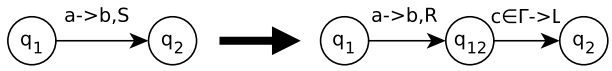
> **Tvrdnja 4.1** Za svaku Turingovu mašinu sa nepomičnom glavom postoji ekvivalentna Turingova mašina.

Dokaz. Slika 4.6 opisuje dokaz. Svaku tranziciju sa nepomičnom glavom možemo prikazati kao dvije tranzicije: jednom pomjeranje udesno, a zatim pomjeranje ulijevo.

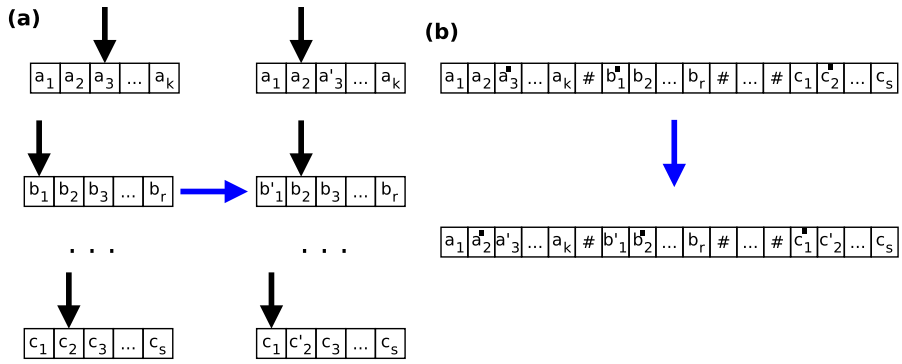
■

Algoritam 4.1 Turingova mašina koja prihvata stringove oblika $a^n b^n c^n$.

```
1:  procedure M(ω)
2:      n ← ω.length();                          ▷ Dužina stringa ω.
3:      n_a ← 0                                  ▷ Brojači.
4:      n_b ← 0;
5:      n_c ← 0;
6:      for i = 0; i < n; ++i do
7:          if ω[i] == a then
8:              if n_b > 0 or n_c > 0 then
9:                  return (reject);
10:             ++n_a;
11:         if ω[i] == b then
12:             if n_c > 0 then
13:                 return (reject);
14:             ++n_b;
15:         if ω[i] == c then
16:             ++n_c;
17:     if n_a == n_b and n_a == n_c then
18:         return (accept);
19:     else
20:         return (reject);
```



Slika 4.6: Svaku tranziciju sa nepomičnom glavom možemo prikazati kao dvije "obične" tranzicije.



Slika 4.7: TM sa više glava je ekvivalentna "standardnoj" TM.

4.3.2 Turingova mašina sa više traka

Turingova mašina ima samo jednu memorijsku traku. Možemo posmatrati verziju i sa više traka, koju nazivamo *Turingova mašina sa više traka*.

> **Tvrdnja 4.2** Za svaku Turingovu mašinu sa više traka postoji ekvivalentna Turingova mašina sa jednom trakom.

Dokaz. Dokaz je opisan na slici 4.7. Stringove sa svih traka smjestimo na jednu traku i odvojimo ih posebnim simbolom, recimo #. Iznad simbola nad kojim se nalazi glava za čitanje/pisanje stavimo novi poseban simbol, npr. • .

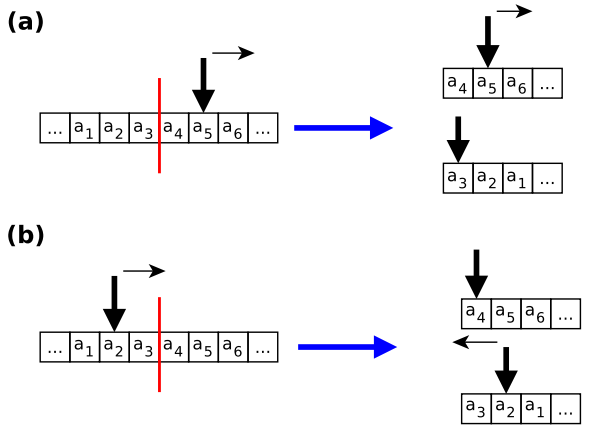
Jedna tranzicija TM sa više traka, zahtijeva više tranzicija TM sa jednom trakom. Naime, TM sa jednom trakom mora da prođe cijeli string i ažurira parametre: izmijenjeni simboli, pozicije specijalnih simbola • i #, te po potrebi pomjeri dijelove stringa.

■

4.3.3 Turingova mašina sa dvostruko beskonačnom trakom

Turingova mašina ima memorijsku traku koja je ograničena sa lijeve strane. Možemo posmatrati verziju mašine koja ima memorijsku traku neograničenu i sa lijeve i sa desne strane. Ovaku mašinu nazivamo *Turingova mašina sa dvostruko beskonačnom trakom*.

> **Tvrdnja 4.3** Za svaku Turingovu mašina sa dvostruko beskonačnom trakom postoji ekvivalentna Turingova mašina.



Slika 4.8: TM sa dvostruko beskonačnom trakom se može prikazati kao TM sa dvije trake.

Dokaz. Turingovu mašinu sa dvostruko beskonačnom trakom možemo transformisati u TM sa dvije trake (slika 4.8), koja je ekvivalentna sa "standardnoj" TM (tvrdnja 4.2).

Na koji način TM sa dvostruko beskonačnom trakom se može transformisati u TM sa dvije trake? Na bilo kojoj poziciji dvostruko beskonačne trake postavimo granicu. Dio stringa koji se nalazi sa desne strane granice postavimo da prvu traku, a dio stringa koji se nalazi sa lijeve strane granice postavimo na drugu traku, ali u obrnutom poretku.

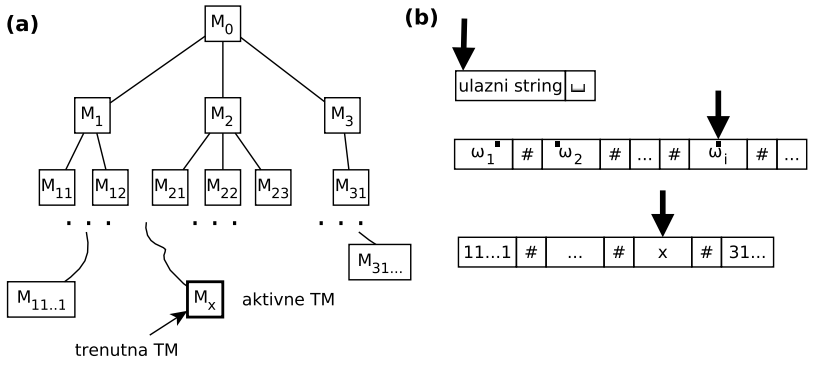
Ako se glava nalazi sa desne strane, tada njoj odgovara glava na prvoj traci, a glava na drugoj traci se nalazi na početku i u nepokretna je. Ako se glava početne TM pomjeri lijevo/desno, tada se prva glava druge TM pomjera lijevo/desno.

Ako se glava nalazi sa lijeve strane, tada njoj odgovara glava na drugoj traci, a glava na prvoj traci se nalazi na početku i u nepokretna je. Ako se glava početne TM pomjeri lijevo/desno, tada se druga glava druge TM pomjera desno/lijevo.

Obje TM upisuju isti simbol na odgovarajućim pozicijama. ■

4.3.4 Nedeterministička Turingova mašina

Slično kao i NFA, nedeterministička TM, označimo je sa M, se može granati na više nezavisnih kopija. Ako barem jedna kopija prihvati string, kažemo da



Slika 4.9: Nedeterministička TM je ekvivalentna TM sa tri trake.

M prihvata string. Ako sve kopije odbiju string, kažemo da *M* odbija string. Ako nijedna od kopija ne vrati *accept* i barem jedna uđe u petlju, kažemo da je *M ušla u petlju*.

Tvrdnja 4.4 Za svaku nedeterminističku Turingovu mašinu postoji ekvivalentna Turingova mašina.

Dokaz. Svaku nedeterminističku TM možemo simulirati pomoću Turingove mašine sa tri trake (slika 4.9).

Sve kopije Turingovih mašina čine stablo slično računanju NFA nad ulaznim stringom, koje smo uveli ranije. Nazovimo ga *stablo kompjutacije*. Listovi ovog stabla su trenutno aktivne mašine. Kopije Turingovih mašina možemo označiti nizom brojeva na osnovu pozicije u stablu (slika 4.9a).

Posmatrajmo sada (determinističku) Turingovu mašinu sa tri trake (slika 4.9b). Prva traka čuva ulazni string. Druga traka čuva stringove koji su pridruženi svakoj od aktivnih TM-a. Slično kao kod TM-a sa više traka, svaki string sadrži simbol • iznad simbola nad kojim se nalazi glava za čitanje/pisanje u odgovarajućoj Turingovoj mašini iz stabla kompjutacije. Treća traka čuva redne oznake aktivnih TM-ova iz stabla kompjutacije. Glava za čitanje/pisanje pokazuje na trenutno aktivnu TM-u iz stabla (označimo je sa M_x).

Stablo kompjutacije istražujemo pomoću Breadth First Search (BFS) – ovo bi neko nazvao i level-order obilazak. Svakom tranzicijom, trenutna TM prihvata string, odbija ga ili se grana na jednu ili više TM-ova.

Ako postoji TM u stablu koje vraća *accept*, tada ćemo sa BFS naići na tu poziciju, te TM sa tri strake vraća *accept*. Ukoliko istražimo cijelo stablo i u svakom listu imamo *reject*, tada i TM sa tri trake će vratiti *reject*. Ukoliko je stablo beskonačno i nijedan list ne vraća *accept*, tada će TM sa tri trake ući u petlju. ■

Tvrdnja 4.5 Za svaku nedeterminističku Turingovu mašinu sa više traka postoji ekvivalentna Turingova mašina.

Dokaz. Dokaz slijedi iz tvrdnji 4.2 i 4.4. ■

4.3.5 k-PDA

Potisne automate sa k stekova nazivamo k-PDA. Primjerom 3.26 smo dokazali da su 2-PDA moćniji od PDA. Dokazali smo da k-PDA, za $k \geq 3$, ima istu moć kao i 2-PDA (tvrdnja 3.10). Znači, 2-PDA su prilično moćni automati. Moćniji su nego što bi se reklo na prvi pogled. O tome govori slijedeća tvrdnja.

Tvrdnja 4.6 Za svaku Turingovu mašinu postoji ekvivalentan 2-potisni automat. Za svaki 2-potisni automat postoji ekvivalentna Turingova mašina.

Dokaz. (a) Pokazaćemo na koji način možemo simulirati rad Turingove mašine pomoću 2-PDA. Neka je (u,q,v) konfiguracija TM-a. Da se podsjetimo, uv je string prisutan na traci, q je stanje TM-a, a glava pokazuje na prvi simbol od v.

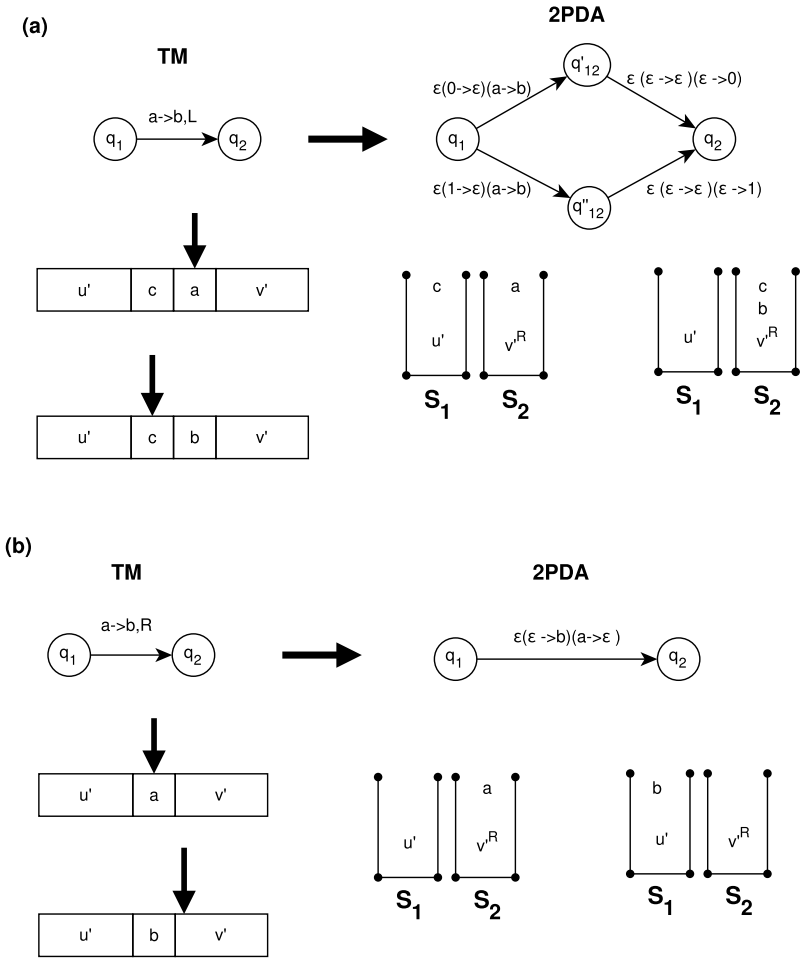
Ideja je da string u stavimo na prvi, a string v^R na drugi stek.

Prvi korak je da 2-PDA pročita cijeli ulazni string i stavi ga na drugi stek. Ostatak rada 2-PDA se sastoji u simulaciji datog TM-a, kako je opisano na slici 4.10.

Neka TM čita simbol a, umjesto njega upisuje b i pomjera glavu ulijevo. Ovo možemo simulirati pomoću 2-PDA, tako što sa drugog steka skinemo a i upišemo b, te sa prvog steka prebacimo simbol na drugi stek.

Ako TM pomjera glavu udesno, tada sa drugog steka skidamo simbol a, a na prvi stek upisujemo simbol b.

(b) Pokažimo kako 2-PDA možemo simulirati pomoću nedeterminističke Turingove mašine sa dvije trake. Postupak je opisan na slici 4.11 (pretpostavljamo da je $a,b,c,d \neq \varepsilon$).

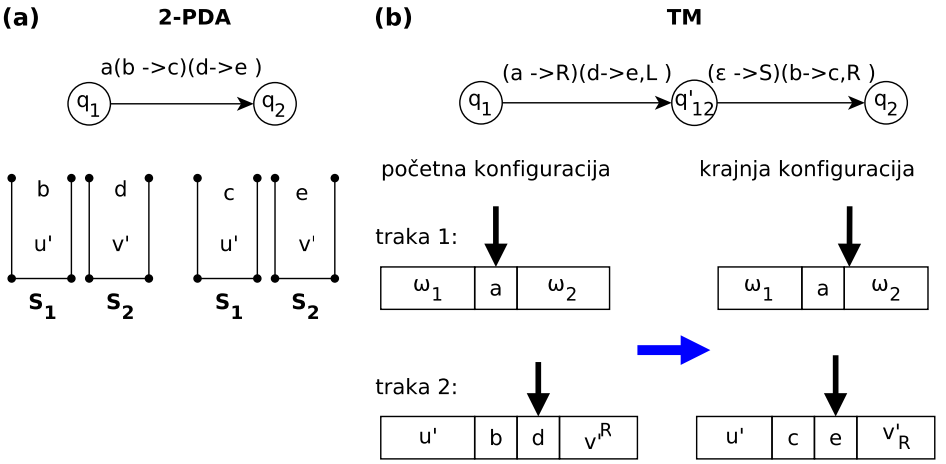


Slika 4.10: Turingova mašina se može prikazati pomoću 2-PDA. **(a)** Način na koji tranziciju TM možemo prikazati pomoću tranizicija 2-PDA, ako se glava na memorijskoj traci pomjera ulijevo. **(b)** Način na koji tranziciju TM možemo prikazati pomoću tranizicija 2-PDA, ako se glava na memorijskoj traci pomjera udesno.

Ulazni string se nalazi na prvoj traci TM-a. Zatim, iskopiramo ulazni string na drugu traku. Druga traka će nam čuvati konfiguraciju stekova, kao što je opisano pod (a).

Ako imamo tranziciju $a(b \to c)(d \to e)$ na 2-PDA, tada ovu tranziciju možemo simulirati pomoću TM na slijedeći način. Sa prve trake čitamo a i

pomjeramo glavu udesno. Na drugoj traci, umjesto d upišemo e i pomjerimo glavu ulijevo. Zatim, umjesto b upišemo c i pomjerimo glavu udesno.



Slika 4.11: Tranzicija 2-PDA prikazane pomoću dvije tranzicije Turingove mašine. Pretpostavljamo da je $a, b, c, d \neq \varepsilon$.

Na ovaj način na drugoj traci opet dobijamo konfiguracije stekova. ■

Zadatak 4.1 — Za samostalan rad. Posmatrati slučajeve kada su neki od parametara a, b, c, d jednaki ε, te dati odgovarajuće tranzicije u dokazu tvrdnje 4.6. ■

4.3.6 Primjeri

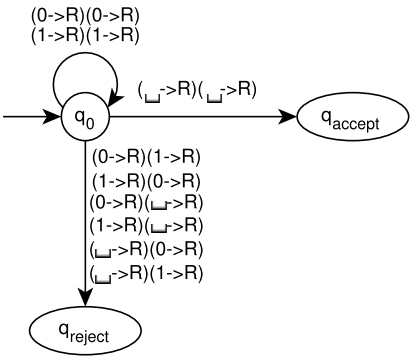
U ovom odjeljku ćemo uraditi nekoliko primjera sa raznim varijantama TM-a.

■ **Primjer 4.6** Konstruiši varijantu TM-a koja provjerava da li su dva stringa jednaka. ■

Rješenje. Slika 4.12 daje rješenje. Konstruisaćemo TM sa dvije trake. Na prvoj traci čuvamo jedan string, a na drugoj drugi string.

Glave čitaju na obje trake u svakoj tranziciji čitaju po jedan znak sa svake trake. Ako su znakovi jednaki, postupak se ponavlja dok obje glave ne naiđu na $\sqcup$. U tom slučaju ulaz se prihvata.

Ukoliko dobijemo situaciju da glave pročitaju različite simbole, ulaz se odbija. ■



Slika 4.12: Turingova mašina koja utvrđuje da li su dva stringa jednaka.

■ **Primjer 4.7** Konstruiši varijantu TM-a koja sabira binarne brojeve. ■

Rješenje. Ponovite kako se sabiraju (binarni) brojevi.

Konstruisaćemo TM sa tri trake (slika 4.13). Na prve dvije trake se čuvaju ulazni stringovi, a na trećoj traci se ispisuje njihov zbir. Podrazumijevamo da se na početku trake nalazi znak \$.

Brojeve sabiramo sa desna nalijevo. Zato, u prvom koraku glave za čitanje namjestimo na kraj stringa. Glave na prve dvije trake se pomjeraju prema lijevo i kreiraju zbir znak po znak. Stanja a_0 i a_1 nam služe da pamtimo da li smo prenijeli znak 1 iz prethodne kolone; ako smo u stanju a_0, nismo prenijeli, a ako smo u stanju a_1 jesmo prenijeli 1 iz prethodne kolone.

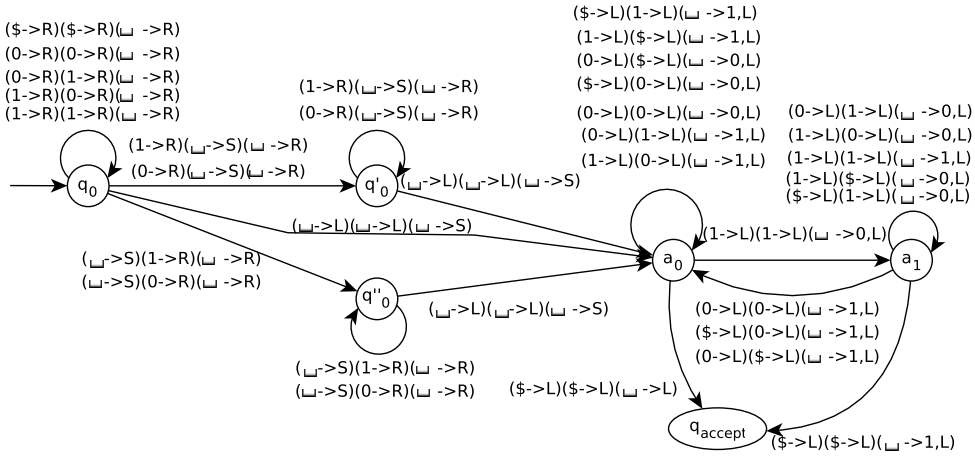
■

4.4 Osobine Turing-prepoznatljivih i odlučivih jezika

Jezik A je Turing-prepoznatljiv ako postoji TM M_A takva da važi $L(M_A) = A$. Znači, ako $\omega \in A$, tada $M_A(\omega) = accept$. Ako $\omega \notin A$, tada $M_A(\omega) = reject$ ili $M_A(\omega)$ ulazi u petlju.

Univerzalna Turingova mašina.

Turingova mašina M^* koja prihvata kao ulaz bilo koju Turingovu mašinu M i string ω, te simulira rad mašine M na ulazu ω se naziva *univerzalna Turingova mašina* (algoritam 4.2).



Slika 4.13: Turingova mašina sa tri trake koja sabira binarne brojeve.

Algoritam 4.2 Univerzalna Turingova mašina.

1: **procedure** $M^*(M,\omega)$
2: **return** $(M(\omega))$;

Tvrdnja 4.7 Klasa Turing-prepoznatljivih jezika je zatvorena pod operacijama:
(a) unije;
(b) presjeka;
(c) nadovezivanja;
(d) zvjezdice.

Dokaz. Neka su A i B Turing-prepoznatljivi jezici i neka su M_A i M_B Turingove mašine koje ih prepoznaju, tj. važi $L(M_A) = A$ i $L(M_B) = B$.

(a) Dokažimo da je $A \cup B$ Turing-prepoznatljiv jezik, tj. da postoji Turingova mašina M koja ga prepoznaje.

Pod "paralelno pokreni $M_A(\omega)$ i $M_B(\omega)$" podrazumijevamo da mašine $M_A(\omega)$ i $M_B(\omega)$ naizmjenično obavljaju po jednu tranziciju.

Algoritam 4.3 TM koja prepoznaje $A \cup B$.

```
1: procedure M(ω)
2:     paralelno pokreni M_A(ω) i M_B(ω);
3:     if neka od mašina M_A(ω) ili M_B(ω) vrati accept then
4:         return (accept);
5:     if obje mašine M_A(ω) i M_B(ω) vrate reject then
6:         return (reject);
```

Imamo da je

$$\begin{aligned}
&M(\omega) = accept \iff \\
&M_A(\omega) = accept \text{ ili } M_B(\omega) = accept \text{ (linija 3)} \iff \\
&\omega \in A \text{ ili } \omega \in B, \text{ jer } L(M_A) = A \text{ i } L(M_B) = B \iff \\
&\omega \in A \cup B.
\end{aligned}$$

Znači, $M(\omega) = accept$ akko $\omega \in A \cup B$, pa $L(M) = A \cup B$, tj. M prepoznaje $A \cup B$.

K Zašto smo paralelno pokrenuli $M_A(\omega)$ i $M_B(\omega)$? Da li smo umjesto paralelnog pokretanja mogli samo napisati uslov iz algoritma 4.4?

Algoritam 4.4

```
if M_A(ω) == accept or M_B(ω) == accept then
    return (accept);
```

Nismo. Uzmimo da $M_A(\omega)$ ulazi u petlju, a $M_B(\omega) = accept$. Tada $\omega \notin A$ i $\omega \in B$, te $\omega \in A \cup B$. Znači, želimo da $M(\omega)$ vrati *accept*. Da smo uzeli uslov iz algoritima 4.4, tada bi se mašina $M_A(\omega)$ pokrenula i nikada ne bi završila. Tj. nikada ne bi došli u situaciju da pokrenemo mašinu $M_B(\omega)$ i $M(\omega)$ bi ušlo u petlju.

(b) Dokažimo da je $A \cap B$ Turing-prepoznatljiv jezik. Dokaz je sličan dokazu pod (a). Neka je M TM i $L(M) = A \cap B$.

Algoritam 4.5 TM koja prepoznaje $A \cap B$.

```
1: procedure M(ω)
2:     if M_A(ω) == accept and M_B(ω) == accept then
3:         return (accept);
4:     return (reject);
```

Imamo da je

$$\begin{aligned}
&M(\omega) = accept \iff \\
&M_A(\omega) = accept \text{ i } M_B(\omega) = accept \text{ (linija 2)} \iff \\
&\omega \in A \text{ i } \omega \in B, \text{ jer } L(M_A) = A \text{ i } L(M_B) = B \iff \\
&\omega \in A \cap B.
\end{aligned}$$

Znači, $M(\omega) = accept$ akko $\omega \in A \cap B$, pa $L(M) = A \cap B$, tj. M prepoznaje $A \cap B$.

(c) Dokažimo da je AB Turing-prepoznatljiv jezik. Sa M označimo TM koja prepoznaje $AB = \{ab \mid a \in A, b \in B\}$. Znači, $M(\omega) = accept$ akko $\omega = ab$, $M_A(a) = accept$ i $M_B(b) = accept$.

Problem je, ako nam je dat ulazni string ω, kako ga podijeliti na dva dijela a i b? Mogli bi pokušati sve moguće podjele a i b, pa testirati $M_A(a)$ i $M_B(b)$. Ali, ovo neće funkcionisati jer bi u nekoj od podjela mogli dobiti da $M_A(a)$ ili $M_B(b)$ uđe u petlju, čime bi M "zaglavilo" na datoj podjeli. Rješenje je da ograničimo broj koraka koje mašine M_A i M_B mogu izvršiti.

Algoritam 4.6 TM koja prepoznaje AB.

```
1: procedure M(ω)
2:     n ← ω.length();                          ▷ ω = ω[1]ω[2]...ω[n].
3:     for k = 1,2,... do
4:         for i = 0; i <= n; ++i do
5:             a ← ω[1]...ω[i];
6:             b ← ω[i+1]...ω[n];
7:             if M_A prihvata a u najviše k koraka and M_B prihvata b u
   najviše k koraka then
8:                 return (accept);
```

Dokažimo da M prepoznaje AB.

Neka $M(\omega) = accept$. Tada $\omega = ab$, $M_A(a) = accept$ i $M_B(b) = accept$ u k' koraka, za neke a, b i k' (linije 7 i 8). Znači, $\omega \in AB$.

Neka $\omega \in AB$. Tada $\omega = ab$, $a \in A$, $b \in B$, pa $M_A(a) = accept$ i $M_B(b) = accept$. Tj. M_A i M_B prihvataju a i b u konačno mnogo koraka. Znači, postoji $k' \in \mathbb{N}$ tako da M_A i M_B prihvataju a i b u najviše k koraka. Znači, uslov u liniji 7 je ispunjen, pa $M(\omega)$ vraća *accept*.

Sve iteracije za $k = 1, ..., k' - 1$ su konačne, jer smo izvršavanje mašina M_A i M_B ograničili sa po k koraka.

Iz svega imamo $\omega \in AB$ akko $M(\omega) = accept$. Znači, $L(M) = AB$, tj. M prepoznaje AB.

(d) Dokažimo da je A^* Turing-prepoznatljiv jezik. Imamo $\omega \in A^*$ akko $\omega \in A^n$ za neko $n \in \mathbb{N}_0$. Posljednje važi akko $\omega = a_1 a_2 ... a_n$ i $a_i \in A$ $(i = 1, ..., n)$.

Sa M označimo TM koja prepoznaje A^*. Ideja je da $M(\omega)$ provjerava da li $\omega \in A^n$, redom za $n = 0, 1, 2, ...$ Takođe, koristićemo ideju iz prethodnog slučaja: ograničavaćemo broj koraka Turingovih mašina.

Algoritam 4.7 TM koja prepoznaje A^*.

```
1: procedure M(ω)
2:     n ← ω.length();                          ▷ ω = ω[1]ω[2]...ω[n].
3:     for k = 1, 2, ... do
4:         for m = 1, ..., n do
5:             for sve podjele ω = a_1 a_2 ... a_m, a_i ∈ A (i = 1, ..., m) do
6:                 if M_A(a_1) == ... == M_A(a_m) == accept u k koraka then
7:                     return (accept);
```

K Uočimo da u liniji 5 nismo opisali kako pravimo podjele $\omega = a_1 ... a_n$. Ovo nije teško uraditi, ali nam nije toliko bitno. Ono što je bitno je da se sve podjele mogu naći u konačno mnogo koraka. Ovo znamo jer ima konačno mnogo podjela.

Dokažimo da M prepoznaje A^*. Važi:

$\omega \in A^* \iff$
$\omega \in A^{m'}$, za neko $m' \iff$
$\omega = a_1 a_2 ... a_{m'}, a_1, a_2, ..., a_{m'} \in A \iff$
$\omega = a_1 a_2 ... a_{m'}$ i M_A prihvata $a_i (i = 1, ..., m')$ u konačno mnogo koraka $\iff$
$\omega = a_1 a_2 ... a_{m'}$ i M_A prihvata $a_i (i = 1, ..., m')$ u najviše k' koraka, za neko k'
$\iff M(\omega) = accept$.

Uočite da za $k = 1, ..., k' - 1$ mašina M neće ući u petlju, jer smo sva pozivanja mašine M_A ograničili sa k koraka.

Znači, $\omega \in A^*$ akko $M(\omega) = accept$, pa $L(M) = A^*$, tj. M prepoznaje A^*. Dobijamo da je A^* Turing-prepoznatljiv. ■

Zadatak 4.2 — Za samostalan rad. Dati pseudokod Turingove mašine koja kao ulaz prima string ω i $n \in \mathbb{N}_0$ i konstruiše sve moguće podjele $\omega = a_1 a_2 \dots a_n$. ■

Kasnije ćemo dokazati da klasa Turing-prepoznatljivih jezika nije zatvorena pod komplementiranjem.

Sada ćemo dokazati ekvivalentnu tvrdnju za odlučive jezike. Dokaz će biti još jednostavniji, jer kod odlučivih jezika odgovarajuća Turingova mašina (tj. odlučivač) uvijek vraća rezultat, tj. nikada ne ulazi u petlju. Da se podsjetimo, Turingova mašina D je odlučivač ako važi jedan od slijedećih uslova, koji su međusobno ekvivalentni:

- D za svaki ulaz vraća *accept* ili *reject*;
- D uvijek vraća rezultat;
- D nikada ne ulazi u petlju;
- D uvijek završava sa radom;
- D se izvršava u konačno mnogo koraka.

Tvrdnja 4.8 Klasa odlučivih jezika je zatvorena pod operacijama:
(a) unije;
(b) presjeka;
(c) komplementiranja;

(*d*) nadovezivanja;
(*e*) zvjezdice.

Dokaz. Neka su A i B odlučivi jezici i neka su D_A i D_B odlučivači koji prepoznaju A i B, tj. važi $L(D_A) = A$ i $L(D_B) = B$.

(a) Dokažimo da je $A \cup B$ odlučiv jezik. Sa D označimo odlučivač koji prepoznaje $A \cup B$. Mašina D treba da prihvati string akko ga prihvata D_A ili D_B. Konstrukcija mašine D je data slijedećim pseudokodom.

Algoritam 4.8 Odlučivač koji prepoznaje $A \cup B$.

```
1: procedure D(ω)
2:     if D_A(ω) == accept or D_B(ω) == accept then
3:         return (accept);
4:     else
5:         return (reject);
```

Dokažimo da je D odlučivač. Mašine D_A i D_B su odlučivači, pa operacije u liniji 2 će uvijek završiti u konačno mnogo koraka. Ostale operacije mašine D očigledno završavaju u konačno mnogo koraka. Prema tome, D je odlučivač.

Dalje imamo:

- $D(\omega) == accept \iff$
- $D_A(\omega) == accept$ ili $D_B(\omega) == accept$ (linija 2) $\iff$
- $\omega \in A$ ili $\omega \in B \iff$;
- $\omega \in A \cup B$.

Znači, D je odlučivač koji prepoznaje $A \cup B$, pa je $A \cup B$ odlučiv jezik.

(b) Konstruišimo odlučivač D koji prepoznaje $A \cap B$. Konstrukcija je skoro identična prethodnom slučaju.

Algoritam 4.9 Odlučivač koji prepoznaje $A \cap B$.

```
1: procedure D(ω)
2:     if D_A(ω) == accept and D_B(ω) == accept then
3:         return (accept);
4:     else
5:         return (reject);
```

Dokažimo da je D odlučivač. Mašine D_A i D_B su odlučivači, pa operacije u liniji 2 će uvijek završiti u konačno mnogo koraka. Ostale operacije mašine D očigledno završavaju u konačno mnogo koraka. Prema tome, D je odlučivač.

Dalje imamo:

- $D(\omega) == accept \iff$
- $D_A(\omega) == accept$ i $D_B(\omega) == accept$ (linija 2) $\iff$
- $\omega \in A$ i $\omega \in B \iff$
- $\omega \in A \cap B$.

Znači, D je odlučivač koji prepoznaje $A \cap B$, pa je $A \cap B$ odlučiv jezik.

(c) Konstruišimo odlučivač D koji prepoznaje jezik $\overline{A}$. Imamo da važi $\omega \in A \iff \omega \notin \overline{A}$.

Algoritam 4.10 Odlučivač koji prepoznaje $\overline{A}$.

```
1: procedure D(ω)
2:     return (\overline{D_A(ω)});
```

Mašina D_A je odlučivač, pa završava sa radom za bilo koji ulaz. Prema tome, operacije u liniji 2 će završiti u konačno mnogo koraka, pa je D odlučivač.

Važi

- $D(\omega) = accept \iff$
- $D_A(\omega) = reject \iff$
- $\omega \notin A \iff$
- $\omega \in \overline{A}$.

Znači, D je odlučivač koji prepoznaje $\overline{A}$, pa je $\overline{A}$ odlučiv jezik.

(d) Konstruišimo odlučivač D koji prepoznaje $AB = \{ab \mid a \in A, b \in B\}$. Za ulazni string ω, posmatraćemo sve moguće podjele na dva dijela: $\omega = ab$. Za svaku podjelu, provjeravamo da li $a \in A$ i $b \in B$, tj. da li $D_A(a) = accept$ i $D_B(b) = accept$.

Mašine D_A i D_B su odlučivači, pa će postupak u liniji 6 (algoritam 4.11) završiti u konačno mnogo koraka. Petlja u liniji 3 se izvršava u konačno mnogo iteracija. Ostale linije koda se očigedno izvršavaju u konačno mnogo koraka. Znači, D je odlučivač.

Važi:

- $D(\omega) = accept \iff$
- uslov u liniji 6 je ispunjen u nekoj iteraciji $\iff$

Algoritam 4.11 Odlučivač koji prepoznaje AB.

```
1: procedure D(ω)
2:     n ← ω.length();                                  ▷ ω = ω[1]...ω[n]
3:     for i = 0; i ≤ n; ++i do
4:         a ← ω[1]...ω[i];
5:         b ← ω[i+1]...ω[n];
6:         if D_A(a) == accept and D_B(b) == accept then
7:             return (accept);
8:     return (reject);
```

- postoji podjela $\omega = ab$, tako da $D_A(a) = D_B(b) = accept \iff$
- postoji podjela $\omega = ab$, tako da $a \in A, b \in B \iff$
- $\omega \in AB$.

Dobijamo da je D odlučivač koji prepoznaje AB, pa je AB odlučiv jezik.

(e) Konstruišimo odlučivač D koji prepoznaje A^*. Imamo da važi: $\omega \in A^* \iff \omega \in A^k$, za neko $k \in \mathbb{N}_0 \iff \omega = a_1 \dots a_k$, za neko $k \in \mathbb{N}_0; a_1, \dots, a_k \in A$.

Ideja je da pustimo $k = 1, 2 \dots n$ i pokušavamo sve podjele stringa ω na k dijelova.

Algoritam 4.12 Odlučivač koji prepoznaje A^*.

```
1: procedure D(ω)
2:     n ← ω.length();                                  ▷ ω = ω[1]...ω[n]
3:     for k = 0; k ≤ n; ++k do
4:         for all podjele ω = a_1 ... a_k do
5:             if D_A(a_1) == D_A(a_2) == ... == D_A(a_k) == accept then
6:                 return (accept);
7:     return (reject);
```

Pošto string ω možemo podijeliti na k dijelova na konačno mnogo načina, to iteracija iz linije 4 ima konačno mnogo. Takođe, iteracija u liniji 3 ima konačno mnogo. Pošto je D_A odlučivač, to se operacije u liniji 5 izvršavaju u konačno mnogo koraka. Znači, D je odlučivač.

Važi:

- $D(\omega) == accept \iff$
- $D_A(a_1) == D_A(a_2) == \dots == D_A(a_k) == accept$ za neko $k \leq n$ i $\omega = a_1 a_2 \dots a_k \iff$

- $a_1, a_2, \ldots, a_k \in A$ za neko $k \leq n$ i $\omega = a_1 a_2 \ldots a_k \iff$
- $\omega \in A^k$ za neko $k \leq |\omega| \iff$
- $\omega \in A^*$.

Znači, D je odlučivač koji prepoznaje A^*, pa je A^* odlučiv jezik. ■

4.5 Primjeri odlučivih jezika

Sada ćemo navesti neke osnovne odlučive jezike.

Slijedeća tvrdnja kaže da za bilo koji DFA D i string ω, u konačno mnogo koraka možemo utvrditi da li D prihvata ili ne prihvata ω.

Tvrdnja 4.9 Jezik $A_{DFA} = \{(D, \omega) \mid D \text{ je DFA koji prihvata string } \omega\}$ je odlučiv.

Dokaz. Dokaz je direktan. Konstruišemo odlučivač M koji simulira rad datog *DFA* (slično univerzalnoj Turingovoj mašini).

Algoritam 4.13 Turingova mašina koja simulira rad DFA.

```
1: procedure M(D, ω)
2:     return (D(ω));
```

Pošto svaki DFA završava rad u konačno mnogo koraka, za bilo koji ulazni string, to imamo konačno mnogo operacija u liniji 2. Znači, M je odlučivač i vraća *accept* akko $D(\omega)$ vraća *accept*, tj. akko $(D, \omega) \in A_{DFA}$. Prema tome, D prepoznaje A_{DFA}, pa A_{DFA} je odlučiv jezik. ■

Iako NFA može ući u beskonačnu petlju (vidi zadatak 2.3), slijedeća tvrdnja kaže da za bilo koji NFA N i string ω u konačno mnogo koraka možemo utvrditi da li N prihvata ili ne prihvata ω.

Tvrdnja 4.10 Jezik $A_{NFA} = \{(N, \omega) \mid N \text{ je NFA koji prihvata string } \omega\}$ je odlučiv jezik.

Dokaz. Odlučivač za A_{NFA} je dat algoritmom 4.14.

Svaki NFA možemo konvertovati u ekvialentan DFA u konačno mnogo koraka (teorema 2.1). Prema tome, linija 2 se izvršava u konačno mnogo koraka. Pošto svaki DFA završava rad u konačno mnogo koraka, za bilo koji ulazni string, to imamo konačno mnogo operacija u liniji 3. Znači, M je

Algoritam 4.14 Turingova mašina koja simulira rad NFA.

1: **procedure** M(N, ω)
2: konstruiši DFA D, koji je ekvivalentan sa N;
3: **return** ($D(\omega)$);

odlučivač i vraća *accept* akko $D(\omega)$ vraća *accept*, tj. akko $N(\omega)$ vraća *accept*, tj. akko $(N, \omega) \in A_{NFA}$. Prema tome, M prepoznaje A_{NFA}, pa A_{NFA} je odlučiv jezik. ■

Posljedica 4.5.1 Ako je D DFA ili NFA, tada je $L(D)$ odlučiv jezik.

Dokaz. Imamo da $\omega \in L(D) \iff D(\omega) = accept \iff (D, \omega) \in A_{DFA}$. Ovo ćemo iskoristiti da konstruišemo odlučivač za $L(D)$.

Algoritam 4.15 TM koja simulira rad DFA D.

1: **procedure** M(ω)
2: **return** ($D(\omega)$);

Pošto je D DFA ili NFA, procedura u liniji 2 završava u konačno mnogo koraka, pa je M odlučivač.

Imamo da važi:

- $M(\omega) = accept \iff$
- $D(\omega) = accept \iff$
- $\omega \in L(D)$.

Znači, M je odlučivač koji prihvata $L(D)$, pa je $L(D)$ odlučiv jezik. ■

Teorema 4.1 Svaki regularan jezik je odlučiv.

Dokaz. Neka je A regularan jezik. Iz teoreme 2.2, postoji DFA D tako da je $L(D) = A$, tj. D prepoznaje A. Iz posljedice 4.5.1, $L(D)$ je odlučiv jezik, pa je $A = L(D)$ odlučiv jezik, što je i trebalo dokazati. ■

Slijedeća tvrdnja kaže da za bilo koji regularan izraz R i string ω, u konačno mnogo koraka možemo utvrditi da li R generiše ili ne generiše ω.

Tvrdnja 4.11 Jezik $A_{REX} = \{(R, \omega) \mid R$ je regularan izraz koji generiše string $\omega\}$ je odlučiv.

Dokaz.

Algoritam 4.16 Odlučivač koji prepoznaje A_{REX}.

```
1: procedure M(R, ω)
2:     konstruiši NFA N: L(N) = L(R);        ▷ Tj. N je ekvivalentan sa R.
3:     return (N(ω));
```

Iz tvrdnje 2.3, procedura u liniji 2 (algoritam 4.16) će završiti u konačno mnogo koraka. Pošto se svaki NFA uvijek izvršava u konačno mnogo koraka, to u liniji 3 imamo konačno mnogo operacija. Znači, M je odlučivač.

Dalje:

- $M(R, \omega) = accept \iff$
- $N(\omega) = accept \iff$
- R generiše ω (jer su R i N ekvivalentni) $\iff$
- $(R, \omega) \in A_{REX}$.

Imamo da je M odlučivač koji prepoznaje A_{REX}, pa je A_{REX} odlučiv jezik. ■

Tvrdnja 4.12 Jezik $E_{DFA} = \{(D) \mid D$ je DFA i $L(D) = \varnothing\}$ je odlučiv.

Dokaz. DFA ne prihvata nijedan string akko ne postoji put od početnog do završnih stanja.

Algoritam 4.17 Odlučivač koji prepoznaje E_{DFA}.

```
1: procedure M(D)
2:     počevši od početnog stanja, obići graf koji predstavlja D i označi sva
       posjećena stanja (za obilazak možete koristiti npr. DFS ili BFS);
3:     if neko završno stanje je označeno then
4:         return (reject);
5:     else
6:         return (accept);
```

Pošto je graf koji predstavlja automat M je konačan, procedura u liniji 2 će se završiti u konačno mnogo koraka. Znači, M je odlučivač.

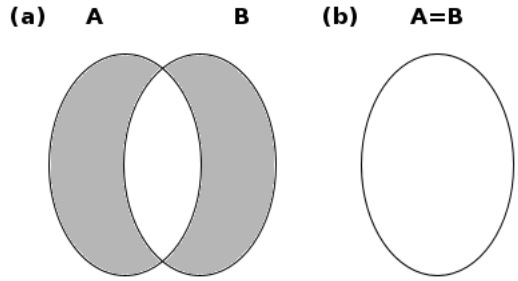
Postoji put od početnog do nekog završnog stanja akko je to završno stanje označeno prilikom pretrage grafa. Znači,

- $M(D) = accept \iff$
- nijedno završno stanje nije označeno $\iff$
- ne postoji put of početnog do nekog završnog stanja $\iff$;
- automat ne prihvata nijedan string $\iff$
- $L(D) = \varnothing \iff$
- $(D) \in E_{DFA}$.

Znači, M je odlučivač koji prepoznaje E_{DFA}, pa je E_{DFA} odlučiv jezik. ■

Tvrdnja 4.13 Jezik $EQ_{DFA} = \{(D_1, D_2) \mid D_1 \text{ i } D_2 \text{ su DFA i } L(D_1) = L(D_2)\}$ je odlučiv.

Dokaz. Neka su A i B skupovi. Sa $A \triangle B = (A \setminus B) \cup (B \setminus A) = (A \cap \overline{B}) \cup (\overline{A} \cap B)$ označavamo *simetričnu razliku* skupova A i B. Važi $A = B$ akko $A \triangle B = \varnothing$.



Slika 4.14: **(a)** Simetrična razlika skupova A i B. **(b)** Simetrična razlika je prazan skup akko $A = B$.

Prethodnu činjenicu i tvrdnju 4.12 ćemo iskoristiti da konstruišemo odlučivač za EQ_{DFA} (algoritam 4.18).

Ranije smo pokazali kako se konstruiše presjek, unija i komplement DFA (čitaoc neka navede postupke). Vidjeli smo da se ove procedure mogu završiti u konačno mnogo koraka. Znači, postupak u liniji 2 možemo završiti u konačno mnogo koraka.

Iz tvrdnje 4.12 imamo da provjeru uslova u liniji 3 možemo završiti u konačno mnogo koraka.

Iz svega imamo da je M odlučivač.

Dalje je:

Algoritam 4.18 Odlučivač koji prepoznaje EQ_{DFA}.

1: **procedure** M(D_1, D_2)
2: konstruišemo DFA D_3 tako da važi: $L(D_3) = L(D_1) \triangle L(D_2) = (L(D_1) \cap \overline{L(D_2)}) \cup (\overline{L(D_1)} \cap L(D_2))$;
3: **if** $L(D_3) == \varnothing$ **then**
4: **return** *accept*;
5: **else**
6: **return** (*reject*);

- $M(D_1, D_2) = accept \iff$
- $L(D_3) = \varnothing \iff$ (jer je $L(D_3) = L(D_1) \triangle L(D_2)$)
- $L(D_1) \triangle L(D_2) = \varnothing \iff$
- $L(D_1) = L(D_2) \iff$
- $(D_1, D_2) \in EQ_{DFA}$.

Znači, M je odlučivač koji prepoznaje EQ_{DFA}, pa je EQ_{DFA} odlučiv jezik. ■

Tvrdnja 4.14 Jezik $A_{CFG} = \{(G, \omega) \mid G \text{ je CFG koja generiše } \omega\}$ je odlučiv jezik.

Dokaz. Algoritam 4.19 daje odlučivač koji prepoznaje A_{CFG}.

Svaka CFG se može konvertovati u ekvivalentnu Chomsky normalnu formu u konačno mnogo koraka (tvrdnja 3.4). Znači, linija 3 se izvršava u konačno mnogo koraka. Imamo konačno mnogo stringova koje možemo generisati u tačno $2n-1$ koraka, pa je lista L konačna, pa se linija 4 izvršava u konačno mnogo koraka. Opet, pošto je lista L konačna, provjera uslova u liniji 5 se izvršava u konačno mnogo koraka. Iz svega imamo da se M izvršava u konačno mnogo koraka za proizvoljni ulaz, pa je M odlučivač.

Dalje imamo:

- $M(G, \omega) = accept \iff$
- $\omega \in L \iff$
- G_1 generiše ω u $2|\omega| - 1$ koraka $\iff$ (jer je G_1 u Chomsky normalnoj formi)
- G_1 generiše $\omega \iff$ (jer $L(G_1) = L(G)$)
- G generiše $\omega \iff$
- $(G, \omega) \in A_{CFG}$.

Algoritam 4.19 Odlučivač koji prepoznaje A_{CFG}.

```
1: procedure M(G, ω)
2:     n ← |ω|;
3:     konstruiši CGF G_1 koja je Chomsky normalna forma gramatike G;
4:     L ← lista svih stringova gramatike G_1 koji se mogu generisati u tačno
       2n − 1 koraka;
5:     if ω ∈ L then
6:         return (accept);
7:     else
8:         return (reject);
```

Znači, M je odlučivač koji prepoznaje A_{CFG}, pa je A_{CFG} odlučiv jezik. ■

Iako PDA može ući u beskonačnu petlju (vidi zadatak 3.1), ipak se problem da li PDA prihvata neki string može riješiti u konačno mnogo koraka (vidi zadatak 4.10).

4.6 Problem zaustavljanja

Ranije smo vidjeli da Turingova mašina može vratiti *accept*, *reject* ili ući u petlju. Treća opcija znači da Turingova mašina nikada neće prestati sa radom. Znači, Turingova mašina može prestati sa radom (i vratiti *accept* ili *reject*) ili nikada ne prestati da se izvršava.

Uzmimo da imamo Turingovu mašinu M i string ω. Sada pokrenimo mašinu $M(\omega)$. Ona će da radi neko vrijeme. Da li mi unaprijed možemo znati da će Turingova mašina prestati sa radom? Tj. da li postoji način da na osnovu neke analize ili algoritma mi možemo za svaku Turingovu mašinu M i za svaki ulazni string ω utvrditi da će $M(\omega)$ vratiti rezultat?

Problem je sličan situaciji kada imate programski kod i pokrenete ga. Prođe određeno vrijeme i program ne staje sa radom. Vi se počnete pitati da li program radi ono što treba ili je ušao u neku beskonačnu petlju.

Dati problem nazivamo *Problem zaustavljanja* (eng. *The Halting Problem*) i formalno je dat sa

$$HALT_{TM} = \{(M, \omega) \mid M \text{ je Turingova mašina koje se zaustavlja sa ulazom } \omega\}.$$

Prije nego analiziramo ovaj problem, posmatrajmo sličan problem.

Tvrdnja 4.15 Jezik $A_{TM} = \{(M, \omega) \mid M$ je Turingova mašina koja prihvata $\omega\}$ je Turing-prepoznatljiv problem.

Dokaz. Dokaz je direktan i jednostavan. Jednostavno pokrenemo $M(\omega)$ i vidimo da li će vratiti *accept*. Konstruišimo M^*, Turingovu mašinu koja prepoznaje A_{TM} (algoritam 4.20).

Algoritam 4.20 TM koja prepoznaje A_{TM}.

```
1: procedure M*(M, ω)
2:     return (M(ω));
```

Važi:

- $M^*(M, \omega) = accept \iff$
- $M(\omega) = accept \iff$
- $(M, \omega) \in A_{TM}$.

Znači, M^* je Turingova mašina koja prepoznaje A_{TM}, pa je A_{TM} Turing-prepoznatljiv jezik.

■

K Prisjetimo se da smo mašinu M^*, iz prethodnog dokaza, nazvali *univerzalna Turingova mašina*.

Problem koji odgovara jeziku A_{TM} možemo formulisati kao: "Da li data Turingova mašina prihvata dati string?". Drugim riječima, da li u konačno mnogo koraka za svaku Turingovu mašinu M i svaki ulazni string ω možemo utvrditi da li M prihvata ω? Teorema 4.2 konstatuje da je ovaj problem nerješiv.

Neka je M neka mašina. Sa $\langle M \rangle$ označavamo zapis te mašine u binarnom obliku. Znači, $\langle M \rangle$ je string. Nešto slično imate svakodnevno dok programirate. Vaš programski kod je Turingova mašina (označavamo kao M). Kada ga sačuvate na disk, on je zapisan u binarnom obliku (označavamo sa $\langle M \rangle$). Na ovaj način ste Turingovoj mašini M pridružili binarni string $\langle M \rangle$.

K Paradoks seoskog brijača se koristi da se pokaže kako ne možemo definisati skupove na proizvoljan način.

U jednom selu živi jedan brijač. On brije sve ljude iz tog sela koji ne briju sami sebe i samo njih brije. Da li brijač brije sebe?

Ako brijač brije sebe, onda je to u kontradikciji sa tvrdnjom da brije samo one koji ne briju sebe. Ako brijač ne brije sebe, onda je u kontradikciji sa tvrdnjom da brije sve ljude iz sela koji ne briju sebe. Prema tome, takav brijač ne postoji.

Slično zaključivanje ćemo koristiti i u dokazu slijedeće tvrdnje.

Teorema 4.2 A_{TM} je neodlučiv jezik.

Dokaz. Pretpostavimo suprotno, A_{TM} je odlučiv i neka je D odlučivač za A_{TM}, tj. važi:

- $D(M,\omega) = accept$, ako $M(\omega) = accept$;
- $D(M,\omega) = reject$, ako $M(\omega) = reject$ ili $M(\omega)$ ulazi u petlju.

Konstruišimo slijedeću Turingovu mašinu (algoritam 4.21).

Algoritam 4.21

```
1: procedure M*((M))
2:     return (D(M,(M)));
```

Posmatrajmo $M^*((M^*))$.

Neka je $M^*((M^*)) = accept$. Tada $D(M^*,(M^*)) = reject$, pa $M^*((M^*)) = reject$ ili $M^*((M^*))$ ulazi u petlju, što je kontradikcija.

Neka je $M^*((M^*)) = reject$. Tada $D(M^*,(M^*)) = accept$, pa $M^*((M^*)) = accept$, što je kontradikcija.

Znači, dobili smo kontradikciju sa pretpostavkom da je A_{TM} odlučiv, pa A_{TM} nije odlučiv. ■

Prethodna teorema nam je osnov za većinu budućih dokaza da neki jezik nije odlučiv. Npr. ako moramo dokazati da neki jezik A nije odlučiv, često ćemo imat slijedeći postupak:

- pretpostavimo suprotno, da A jeste odlučiv;
- prema tome, postoji odlučivač D koji odlučuje A;
- koristeći D, konstruišemo odlučivač za A_{TM};
- znači, A_{TM} je odlučiv, što je u kontradikciji sa teoremom 4.2.

Slijedeća tvrdnja konstatuje da ne postoji algoritam koji za proizvoljnu Turingovu mašinu M i proizvoljan string ω utvrđuje da li $M(\omega)$ završava sa radom.

Teorema 4.3 $HALT_{TM}$ je neodlučiv jezik.

Dokaz. Pretpostavimo suprotno, neka je $HALT_{TM}$ odlučiv jezik i D je odlučivač koji prepoznaje $HALT_{TM}$. Konstruisaćemo D', odlučivač koji prepoznaje A_{TM} (algoritam 4.22).

Algoritam 4.22

```
1: procedure D'(M, ω)
2:     if D(M, ω) == accept then
3:         return (M(ω));
4:     else
5:         return (reject);
```

Pošto je D odlučivač, provjera uslova u liniji 2 će završiti u konačno mnogo koraka. Procedura u liniji 3 će se aktivirati samo ako je uslov u liniji 2 zadovoljen. U tom slučaju, $M(\omega)$ će se zaustaviti, te će linija 3 završiti u konačno mnogo koraka. Ostale linije su elementarne, te se završavaju u konačno mnogo koraka. Prema tome, D' je odlučivač.

Dalje imamo:

- $D'(M,\omega) = accept \iff$
- $D(M,\omega) = accept \wedge M(\omega) = accept \iff$ (jer $M(\omega) = accept \implies D(M,\omega) = accept$, pa je $D(M,\omega) = accept$ suvišan uslov)
- $M(\omega) = accept \iff$
- $(M,\omega) \in A_{TM}$.

Znači, D' odlučuje A_{TM}, pa je A_{TM} odlučiv jezik. Ovo je u kontradikciji sa teoremom 4.2, pa $HALT_{TM}$ nije odlučiv jezik.

■

Tvrdnja 4.16 $HALT_{TM}$ je Turing-prepoznatljiv jezik.

Dokaz. Dokaz je skoro indentičan dokazu analogne tvrdnje za A_{TM}; pokrenemo $M(\omega)$ i vidimo da li će se zaustaviti. Konstruišimo M^*, Turingovu mašinu koja prepoznaje $HALT_{TM}$ (algoritam 4.23).

Važi:

- $M^*(M,\omega) = accept \iff$
- $M(\omega) == accept$ ili $M(\omega) == reject \iff$
- M se zaustavlja sa ulazom $\omega \iff$

Algoritam 4.23 TM koja prepoznaje $HALT_{TM}$.

```
1: procedure M*(M, ω)
2:     if M(ω) == accept or M(ω) == reject then
3:         return (accept);
```

- $(M, \omega) \in HALT_{TM}$.

Znači, M^* je Turingova mašina koja prepoznaje $HALT_{TM}$, pa je $HALT_{TM}$ Turing-prepoznatljiv jezik. ■

K Rezimirajmo teoremu 4.3, i tvrdnju 4.16. Pretpostavimo da imamo Turingovu mašinu (programski kod) M i ulazni string (ulazni podatak) ω. Izvršimo programski kod M nad ulaznim podatkom ω, tj. pokrenimo $M(\omega)$.

Tvrdnja 4.16 govori da, ako programski kod vraća rezultat, rezultat ćemo dobiti u konačno mnogo koraka, što je očigledno i bez tvrdnje.

Teorema 4.3 tvrdi da, sve dok se programski kod izvršava, unaprijed ne možemo znati da li će program vratiti rezultat ili će ući u neku beskonačnu pretlju. Sve dok se program izvršava, ne možemo unaprijed znati da li će ikada prestati sa radom. Drugim riječima, ne postoji teorija ili algoritam, koji bi nam za svaku Turingovu mašinu M i za svaki ulazni string ω, u konačno mnogo koraka, dali odgovor da li će $M(\omega)$ prestati sa radom ili ne.

Znači, problem $HALT_{TM}$ se ne može riješiti u konačno mnogo koraka.

Isto važi i za problem A_{TM}. Generalno, bilo koji problem kojem odgovara neodlučiv jezik se ne može riješiti u konačno mnogo koraka.

4.7 Primjeri neodlučivih i Turing-neprepoznatljivih jezika

Slijedeće dvije tvrdnje će nam pomoći da dokažemo da su neki jezici neodlučivi ili Turing-neprepoznatljivi.

Tvrdnja 4.17 Jezik A je odlučiv akko je $\overline{A}$ odlučiv.

Dokaz. ***(a)*** Neka je A odlučiv i neka je D njegov odlučivač. Konstruišimo $\overline{D}$, koji će biti odlučivač za $\overline{A}$ (algoritam 4.24).

Pošto je D odlučivač, to je i $\overline{D}$ odlučivač. Važi:

- $\overline{D}(\omega) = accept \iff$

Algoritam 4.24

```
1: procedure D̄(ω)
2:     return (D(ω));
```

- $D(\omega) = reject \iff$
- $\omega \notin A \iff$
- $\omega \in \overline{A}$.

Znači, $\overline{D}$ odlučuje $\overline{A}$, pa je $\overline{A}$ odlučiv jezik.

(b) Neka je sada $\overline{A}$ odlučiv jezik. Iz *(a)* imamo da je $\overline{\overline{A}}$ odlučiv jezik. Pošto je $\overline{\overline{A}} = A$, dobijamo da je A odlučiv jezik. ■

Definicija 4.6 — Ko-Turing-prepoznatljiv jezik. Za jezik A kažemo da je *ko-Turing-prepoznatljiv*, ako je $\overline{A}$ Turing-prepoznatljiv.

Tvrdnja 4.18 Jezik je odlučiv akko je Turing-prepoznatljiv i ko-Turing-prepoznatljiv.

Dokaz. ***(a)*** Neka je jezik A odlučiv i neka je D njegov odlučivač. Pošto je svaki odlučivač ujedno i Turingova mašina, A je očigledno Turing-prepoznatljiv jezik.

Iz tvrdnje 4.17 imamo da je $\overline{A}$ odlučiv, pa je i Turing-prepoznatljiv. Znači, A je ko-Turing-prepoznatljiv.

(b) Neka je A Turing-prepoznatliv i ko-Turing-prepoznatljiv i neka su M_1 i M_2 Turingove mašine koje prepoznaju A i $\overline{A}$. Konstruišimo D, odlučivač za A (algoritam 4.25).

Algoritam 4.25 Odlučivač za A.

```
1: procedure D(ω)
2:     paralelno (naizmjenično) pokreni M1(ω) i M2(ω);
3:     if M1(ω) == accept then
4:         return (accept);
5:     if M2(ω) == accept then
6:         return (reject);
```

Pošto za svaki string važi $\omega \in A$ ili $\omega \in \overline{A}$ imamo da će za svako ω uslov u liniji 3 ili 5 biti ispunjen, pa će M uvijek vratiti rezultat. Znači, D je odlučivač.

Dokažimo da D prepoznaje A. Imamo:

- $D(\omega) = accept \iff$
- $M_1(\omega) = accept \iff$
- $\omega \in A$.

Znači, D prepoznaje A, pa je A odlučiv jezik.

Prvu ekvivalenciju je potrebno dodatno pojasniti. Ako je $M_1(\omega) = accept$, tada $\omega \in A$, pa $\omega \notin \overline{A}$, pa $M_2(\omega) \neq accept$. Znači, $D(\omega) = accept$. ■

K Iz $A_{TM} = \{(M,\omega) \mid M \text{ je Turingova mašina koja prihvata } \omega\}$, imamo da je

$$\overline{A_{TM}} = \Sigma^\star \backslash A_{TM} = \{(M,\omega) \mid M \text{ je Turingova mašina koja ne prihvata } \omega\} \cup$$
$$\cup \{\omega' \mid \omega' \text{ ne predstavlja opis } (M,\omega)\}.$$

Drugi skup unije, skup stringova koji ne predstavlja opis (M,ω) ni za jedno M i ω, nije relevantan za problem da li Turingova mašina prihvata određeni string. Zato ćemo uzimati:

$$\overline{A_{TM}} = \{(M,\omega) \mid M \text{ je Turingova mašina koja ne prihvata } \omega\}.$$

Slično vrijedi i za komplemente ostalih jezika, koje ćemo susretati u nastavku.

Slijedeća tvrdnja daje primjer jezika koji nije Turing-prepoznatljiv.

Tvrdnja 4.19 $\overline{A_{TM}}$ nije Turing-prepoznatljiv jezik.

Dokaz. Imamo $\overline{A_{TM}} = \{(M,\omega) \mid M \text{ je Turingova mašina koja ne prihvata } \omega\}$.

Pretpostavimo suprotno, $\overline{A_{TM}}$ jeste Turing-prepoznatljiv. Pošto je i A_{TM} Turing-prepoznatljiv (tvrdnja 4.15), to je A_{TM} odlučiv (tvrdnja 4.18). Ovo je u kontradikciji sa činjenicom da A_{TM} nije odlučiv (teorema 4.2).

Znači, $\overline{A_{TM}}$ nije Turing-prepoznatljiv. ■

K Prethodna tvrdnja nam daje primjer jezika koji nije Turing-prepoznatljiv. Ranije smo vidjeli da neodlučive jezike/probleme ne možemo riješiti u konačno mnogo koraka. Znači, radi se o teškim problemima.

Neki neodlučivi jezici su Turing-prepoznatljivi, što nam omogućava da ih riješimo djelimično. Turing-neprepoznatljivi jezici/problemi su teži od neodlučivih Turing-prepoznatljivih problema. Naime, ako je A neodlučiv i Turing-prepoznatljiv jezik, tada postoji algoritam M koji vraća rezultat za svako $\omega \in A$ (za $\omega \notin A$ procedura $M(\omega)$ može ući u petlju). Za Turing-neprepoznatljiv jezik B ovakav algoritam ne postoji, tj. za bilo koju Turingovu mašinu M' postoji $\omega \in B$ tako da $M'(\omega)$ ulazi u petlju.

Tvrdnja 4.20 Jezik $E_{TM} = \{(M) \mid M \text{ je Turingova mašina i } L(M) = \varnothing\}$ je neodlučiv.

Dokaz. Pretpostavimo suprotno, E_{TM} je odlučiv jezik i neka je D njegov odlučivač. Konstruisaćemo D', odlučivač za A_{TM} (algoritam 4.27).

Prije toga ćemo pokazati kako se konstruiše pomoćna Turingova mašina $M'_{M,\omega}$ (algoritam 4.26).

Algoritam 4.26 Pomoćna Turingova mašina.

```
1: procedure M'_{M,ω}(x)
2:     if M(ω) == accept then
3:         return (accept);
4:     else
5:         return (reject);
```

Mašina $M'_{M,\omega}$ prihvata sve stringove akko $M(\omega) = accept$. Ako $M(\omega) \neq accept$, tada $M'_{M,\omega}$ ne prihvata nijedan string. Znači, $L(M'_{M,\omega}) \neq \varnothing$ akko $M(\omega) = accept$.

Algoritam 4.27

```
1: procedure D'(M, ω)
2:     konstruiši M'_{M,ω};
3:     return (D̄(M'_{M,ω}));
```

Pošto je $M'_{M,\omega}$ Turingova mašina, može se konstruisati u konačno mnogo koraka, pa linija 2 se izvršava u konačno mnogo koraka. Pošto je D odlučivač, to se i linija 3 izvršava u konačno mnogo koraka. Znači, D' je odlučivač.

Dalje imamo:

- $D'(M,\omega) = accept \iff$
- $D(M'_{M,\omega}) = reject \iff$
- $(M'_{M,\omega}) \notin E_{TM} \iff$
- $L(M'_{M,\omega}) \neq \varnothing \iff$
- $M(\omega) = accept \iff$
- $(M,\omega) \in A_{TM}$.

Dobijamo da D' odlučuje A_{TM}, pa je A_{TM} odlučiv jezik, što je u kontradikciji sa teoremom 4.2. Znači, naša pretpostavka nije tačna, pa je E_{TM} neodlučiv jezik. ■

Tvrdnja 4.21 Jezik $REGULAR_{TM} = \{(M) \mid M$ je Turingova mašina i $L(M)$ je regularan jezik$\}$ je neodlučiv.

Dokaz. Pretpostavimo suprotno. Neka je $REGULAR_{TM}$ odlučiv jezik i neka je D njegov odlučivač. Konstruišimo D', odlučivač za A_{TM} (algoritam 4.29).

Ideja je slična kao u prethodnom dokazu. Za dato M i ω, želimo konstruisati mašinu $M'_{M,\omega}$ tako da je $L(M'_{M,\omega})$ regularan jezik akko $M(\omega) = accept$. Za mjesto regularnog jezika $L(M'_{M,\omega})$ možemo uzeti bilo koji regularan jezik, recimo Σ^*. Takođe, za mjesto neregularnog jezika možemo uzeti bio koji jezik, recimo $\{0^n1^n \mid n \geq 0\}$ (pronađite primjer u kojem smo dokazali da ovaj jezik nije regularan).

Algoritam 4.28 Pomoćna Turingova mašina.

```
1: procedure M'_{M,ω}(x)
2:     if x == 0^n 1^n za neko n ≥ 0 then
3:         return (accept);
4:     else
5:         return (M(ω));
```

Algoritam 4.28 daje pseudokod za mašinu $M'_{M,\omega}$. Ako je $M(\omega) = accept$, tada $M'_{M,\omega}$ prihvata bilo koji string x, tj. važi $L(M'_{M,\omega}) = \Sigma^*$. Ako je $M(\omega) \neq accept$, tada $M'_{M,\omega}$ prihvata string samo ako je oblika 0^n1^n, tj. važi $L(M'_{M,\omega}) = \{0^n1^n \mid n \geq 0\}$. Prema tome, $L(M'_{M,\omega})$ je regularan akko $M(\omega) = accept$.

Svaka konstrukcija Turingove mašine je konačna, pa linija 2 se izvršava u konačno mnogo koraka. Pošto je D odlučivač i linija 3 se izvršava u konačno mnogo koraka. Iz svega imamo da je D' odlučivač.

Dalje je:

Algoritam 4.29

1: **procedure** $D'(M,\omega)$
2: konstruiši Turingovu mašinu $M'_{M,\omega}$;
3: **return** $(D(M'_{M,\omega}))$;

- $D'(M,\omega) = accept \iff$
- $D(M'_{M,\omega}) = accept \iff$ (jer je D odlučivač za $REGULAR_{TM}$)
- $(M'_{M,\omega}) \in REGULAR_{TM} \iff$ (jer $L(M'_{M,\omega})$ može biti Σ^* ili $\{0^n1^n \mid n \geq 0\}$)
- $L(M'_{M,\omega}) = \Sigma^* \iff$
- $M'_{M,\omega}$ prihvata svaki string $x \in \Sigma^* \iff$
- $M(\omega) = accept \iff$
- $(M,\omega) \in A_{TM}$.

Znači, D' odlučuje A_{TM}, pa je A_{TM} odlučiv jezik, što je nemoguće (teorema 4.2). Iz kontradikcije imamo da je naša pretpostavka netačna, tj. $REGULAR_{TM}$ je neodlučiv jezik. ■

Slijedeća tvrdnja daje primjer jezika koji nije Turing-prepoznatljiv (samim tim nije ni odlučiv) niti je ko-Turing-prepoznatljiv

Tvrdnja 4.22 Jezik $EQ_{TM} = \{(M_1,M_2) \mid M_1 \text{ i } M_2 \text{ su Turingove mašine i } L(M_1) = L(M_2)\}$
(*a*) je neodlučiv;
(*b*) nije Turing-prepoznatljiv;
(*c*) nije ko-Turing-prepoznatljiv.

Dokaz. ***(a)*** Tvrdnja slijedi direktno iz *(b)*, jer ako jezik nije Turing-prepoznatljiv, tada nije ni odlučiv.

(b) Pretpostavimo suprotno, postoji Turingova mašina M_1 koja prepoznaje EQ_{TM}. Konstruišimo Turingovu mašinu M' koja prepoznaje $\overline{A_{TM}}$ (algoritam 4.31).

Neka su M i ω proizvoljna Turingova mašina i string. Algoritam 4.30 daje pseudokod za $M^*_{M,\omega}$.

Znači, $M(\omega) \neq accept \iff M^*_{M,\omega}(x) \neq accept, \forall x \in \Sigma^* \iff L(M_\omega) = \varnothing$.

Linija 2 u kodu mašine M' se izvršava u konačno mnogo koraka. Dalje, imamo:

Algoritam 4.30 Pomoćna Turingova mašina.

```
1: procedure M*_{M,ω}(x)
2:     return (M(ω));
```

Algoritam 4.31

```
1: procedure M'(M, ω)
2:     M_2 ← Turingova mašina koja odbija sve stringove;
3:     return ( M_1(M*_{M,ω}, M_2) );
```

- $M'(M,\omega) = accept \iff$
- $M_1(M^*_{M,\omega}, M_2) = accept \iff$
- $L(M^*_{M,\omega}) = L(M_2) \iff$ (jer $L(M_2) = \varnothing$)
- $L(M^*_{M,\omega}) = \varnothing \iff$ (iz konstrukcije mašine $M^*_{M,\omega}$)
- $M(\omega) \neq accept \iff$
- $(M,\omega) \in \overline{A_{TM}}$.

Znači, M' prepoznaje $\overline{A_{TM}}$, pa je $\overline{A_{TM}}$ Turing-prepoznatljiv. Ovo je u kontradikciji sa tvrdnjom 4.19, pa EQ_{TM} nije Turing-prepoznatljiv.

(c) Dokaz je skoro identičan dokazu pod *(b)*. Pretpostavimo suprotno, postoji Turingova mašina M_1 koja prepoznaje $\overline{EQ_{TM}}$. Konstruišimo Turingovu mašinu M' koja prepoznaje $\overline{A_{TM}}$ (algoritam 4.33).

Neka su M i ω proizvoljna Turingova mašina i string. Konstruišimo $M^*_{M,\omega}$ (algoritam 4.32).

Algoritam 4.32 Pomoćna Turingova mašina.

```
1: procedure M*_{M,ω}(x)
2:     return (M(ω));
```

Znači,

$$M(\omega) \neq accept \iff M^*_{M,\omega}(x) \neq accept, \forall x \in \Sigma^* \iff L(M^*_{M,\omega}) = \varnothing,$$

te

$$M(\omega) = accept \iff M^*_{M,\omega}(x) = accept, \forall x \in \Sigma^* \iff L(M^*_{M,\omega}) = \Sigma^*.$$

Linija 2 u kodu mašine M' se izvršava u konačno mnogo koraka. Dalje, imamo:

Algoritam 4.33

```
1: procedure M'(M, ω)
2:     M_2 ← Turingova mašina koja prihvata sve stringove;
3:     return ( M_1(M*_{M,ω}, M_2) );
```

- $M'(M,\omega) = accept \iff$
- $M_1(M^*_{M,\omega}, M_2) = accept \iff$ (jer M_1 prepoznaje $\overline{EQ_{TM}}$)
- $L(M^*_{M,\omega}) \neq L(M_2) \iff$ (jer $L(M_2) = \Sigma^*$)
- $L(M^*_{M,\omega}) \neq \Sigma^* \iff$ (jer $L(M^*_{M,\omega}) = \Sigma^*$ ili $L(M^*_{M,\omega}) = \varnothing$)
- $L(M^*_{M,\omega}) = \varnothing \iff$ (iz konstrukcije mašine $M^*_{M,\omega}$)
- $M(\omega) \neq accept \iff$
- $(M,\omega) \in \overline{A_{TM}}$.

Znači, M' prepoznaje $\overline{A_{TM}}$, pa je $\overline{A_{TM}}$ Turing-prepoznatljiv. Ovo je u kontradikciji sa tvrdnjom 4.19, pa $\overline{EQ_{TM}}$ nije Turing-prepoznatljiv. ■

4.8 Riceov teorem

Inutitivno, slijedeća teorema tvrdi da skoro svaki jezik koji se sastoji od opisa nekih Turingovih mašina je neodlučiv.

Teorema 4.4 — Riceov teorem. Neka je $\mathscr{O}$ neka osobina Turingovih mašina i $R = \{(M) \mid M \text{ je Turingova mašina koja ima osobinu } \mathscr{O}\}$, tako da važi:

- R je netrivijalan jezik (sadrži barem jednu i ne sadrži sve Turingove mašine);
- ako $L(M_1) = L(M_2)$, tada $(M_1) \in R \iff (M_2) \in R$.

Tada je jezik R neodlučiv.

Dokaz. Pretpostavimo suprotno, R je odlučiv jezik i neka je D odlučivač koji prepoznaje R. Tada je i $\overline{R} = \{(M) \mid M \text{ je Turingova mašina koja nema osobinu } \mathscr{O}\}$ odlučiv jezik (tvrdnja 4.17). Pretpostavimo da $(\varnothing) \notin R$ - u suprotnom bi donji dokaz sproveli za $\overline{R}$.

Koristeći D, konstruišimo D' - odlučivač za A_{TM}.

Neka je M_1 proizvoljno odabrana, pa fiksirana Turingova mašina sa osobinom $\mathscr{O}$, tj. $(M_1) \in R$.

Prvo konstruišimo pomoćnu Turingovu mašinu $M'_{M,\omega}$ kao što je opisano

pomoću algoritma 4.34, pri čemu je M' proizvoljna TM, a ω proizvoljan string.

Algoritam 4.34 Pomoćna Turingova mašina.

```
1: procedure M'_{M,ω}(x)
2:     if M(ω) == accept then
3:         return (M_1(x))
4:     return (reject);
```

Pretpostavimo prvo da je $M(\omega) = accept$. Tada je $M'_{M,\omega}(x) = accept \iff M_1(x) = accept$, pa je $L(M_1) = L(M'_{M,\omega})$. Pošto $(M_1) \in R$, to i $(M'_{M,\omega}) \in R$, tj. $M'_{M,\omega}$ ima osobinu $\mathscr{O}$.

Neka je sada $M(\omega) \neq accept$. Tada je $M'_{M,\omega}(x) \neq accept, \forall x \in \Sigma^*$, tj. $L(M'_{M,\omega}) = \varnothing$, pa $(M'_{M,\omega}) \notin R$.

Znači, $(M'_{M,\omega}) \in R \iff M(\omega) = accept$.

Algoritam 4.35

```
1: procedure D'(M, ω)
2:     konstruiši M'_{M,ω};
3:     return (D(M'_{M,ω}));
```

Svaka TM je konačna struktura, pa se linija 2 izvršava u konačno mnogo koraka. Pošto je D odlučivač, tada se linija 3 izvršava u konačno mnogo koraka, pa je i D' odlučivač.

Važi:

- $D'(M,\omega) = accept \iff$
- $D(M'_{M,\omega}) = accept \iff$ (jer je D odlučivač za R)
- $(M'_{M,\omega}) \in R \iff$ (ranije dokazali)
- $M(\omega) = accept \iff$
- $(M,\omega) \in A_{TM}$.

Znači, D' je odlučivač koji prepoznaje A_{TM}, pa je A_{TM} odlučiv jezik, što je kontradikcija sa teoremom 4.2. Prema tome, R nije odlučiv jezik. Ovim smo dokazali Riceov teorem. ■

■ **Primjer 4.8** Neka je $A = \{(M) \mid M$ je Turingova mašina koja prihvata ω^R ako prihvata $\omega\}$. Dokažimo da je jezik A neodlučiv.

Koristićemo Riceov teorem. Sa $\mathscr{O}$ označimo osobinu: *"je Turingova mašina koja prihvata ω^R akko prihvata ω"*.

Postoji Turingova mašina sa ovom osobinom, recimo Turingova mašina koja prihvata sve stringove. Postoji i Turingova mašina bez ove osobine, npr. mašina koja prihvata samo string 10. Znači, A nije trivijalan jezik.

Neka su M_1 i M_2 Turingove mašine takve da je $L(M_1) = L(M_2)$. Važi:

- $(M_1) \in A \iff$
- $\omega \in L(M_1) \iff \omega^R \in L(M_1) \iff$ (jer $L(M_1) = L(M_2)$)
- $\omega \in L(M_2) \iff \omega^R \in L(M_2) \iff$
- $(M_2) \in A$.

Iz prethodnog imamo da A zadovoljava uslove Riceovog teorema, pa A nije odlučiv jezik. ∎

∎ **Primjer 4.9** Dokažimo da je jezik $A = \{(M) \mid M$ je Turingova mašina koja prihvata $1011\}$ neodlučiv.

Umjesto 1011 može da stoji bilo koji string. Uočimo da je ovaj problem specijalan slučaj problema A_{TM}, kada je string fiksiran.

Posmatrajmo osobinu $\mathscr{O}$: *"Turingova mašina prihvata string 1011"*. Očigledno je ova osobina netrivijalna, tj. postoji barem jedna mašina sa ovom osobinom i barem jedna mašina bez ove osobine.

Činjenicu da M prihvata 1011 možemo pisati i kao $1011 \in L(M)$. Neka su M_1 i M_2 Turingove mašine takve da je $L(M_1) = L(M_2)$. Važi:

- $(M_1) \in A \iff$
- $1011 \in L(M_1) \iff$ (jer $L(M_1) = L(M_2)$)
- $1011 \in L(M_2) \iff$
- $(M_2) \in A$.

Svi uslovi Riceovog teorema su zadovoljeni, pa je jezik A neodlučiv. ∎

Zadatak 4.3 — Za samostalan rad. Naći netrivijalnu osobinu $\mathscr{O}$ tako da za jezik $R = \{(M) \mid M$ je TM koja ima osobinu $\mathscr{O}\}$ ne važi: ako je $L(M_1) = L(M_2)$, tada $(M_1) \in R \iff (M_2) \in R$. ∎

4.9 Enumerator

Inutitivno, enumerator je mašina koja ispisuje stringove nekog jezika.

Definicija 4.7 — Enumerator. Za Turingovu mašinu E kažemo da *enumeriše* jezik A, ako na traku ispisuje samo stringove iz A i svaki string iz A će biti ispisan u nekom koraku. Ovakvu Turingovu mašinu nazivamo *enumerator*.

K Ako enumerator svaki string iz A ispisuje u nekom (konačnom) koraku, to ne znači da će ispisati sve stringove iz A. Npr. uzmite da imate mašinu koja ispisuje prirodne brojeve: 1,2,3,... Proizvoljan prirodan broj n će biti ispisan u nekom koraku, ali nećemo imati trenutak da su svi prirodni brojevi ispisani, jer ih ima beskonačno mnogo. Enumerator koji ispisuje beskonačan jezik nikada neće stati.

■ **Primjer 4.10 — Primjer enumeratora.** Konstruišimo pseudokod enumeratora za jezik $A = \{\omega \mid \omega \text{ je palindrom}\}$. Vidi algoritam 4.36.

Algoritam 4.36 Enumerator koji ispisuje palindrome.

```
1: ω ← ε;
2: while true do
3:     if ω je palindrom then
4:         print(ω);
5:     ω ← slijedeći string po leksikografskom poretku;
```

■

Vidimo da je konstrukcija u prethodnom primjeru generalna. Uslov u liniji 3 možemo zamijeniti bilo kojim uslovom koji opisuje jezik neki dati jezik A. Ako je A odlučiv jezik, tada se uslov može provjeriti odlučivačem. Ovo opažanje je formalizirano slijedećom tvrdnjom.

Slijedeće dvije tvrdnje daju vezu između enumeratora i Turingovih mašina.

Tvrdnja 4.23 Jezik je Turing-prepoznatljiv akko ga neki enumerator enumeriše.

Dokaz. ***(a)*** Pretpostavimo da je jezik A Turing-prepoznatljiv i neka je M Turingova mašina koja ga prepoznaje. Konstruišimo enumerator E koji će enumerisati A (algoritam 4.37).

Dokažimo da E enumeriše A. Neka je $\omega \in A$ i $|\omega| = k$. Pošto je $\omega \in A$, to M prihvata ω u konačno mnogo koraka (recimo m koraka). Neka je

Algoritam 4.37 Enumerator za Turing-prepoznatljiv jezik.

```
1: procedure E
2:     n ← 1;
3:     while 1 do
4:         L ← lista svih stringova dužine ne veće od n;
5:         for x ∈ L do
6:             if M prihvata x u najviše n koraka then
7:                 print(x);
8:         n ← n + 1;
```

$n = max\{k,m\}$. Dobijamo da u n-toj iteraciji $x \in L$, jer L sadrži sve stringove dužine ne veće od n. Pošto je $n \geq m$, uslov u liniji 6 će biti zadovoljen. Prema tome E će ispisati ω. Iz uslova u liniji 6 imamo da E ispisuje samo stringove iz A, pa E enumeriše A.

(b) Pretpostavimo sada da E enumeriše A. Konstruišimo M, Turingovu mašinu koja prepoznaje A (algoritam 4.38).

Algoritam 4.38 Turingova mašina za enumerabilni jezik.

```
1: procedure M(ω)
2:     pokreni enumerator E i prati ispis;
3:     if u nekom koraku je ispisan ω then
4:         return (accept);
```

Imamo da važi:

- $M(\omega) = accept \iff$
- E u nekom koraku ispisuje $\omega \iff$
- $\omega \in A$.

Znači, M je Turingova mašina koja prepoznaje A.

■

K Zašto enumerator E iz prethodnog dokaza nismo mogli konstruisati slijedeći način prikazan algoritmom 4.39?

Algoritam 4.39 Pogrešna konstrukcija enumeratora za Turing-prepoznatljiv jezik.

```
1: procedure E
2:     x ← ε;
3:     while 1 do
4:         if M(x) == accept then
5:             print(x);
6:         x ←slijedeći string u leksikografskom poretku;
```

Tvrdnja 4.24 Jezik je odlučiv akko ga neki enumerator enumeriše u leksikografskom poretku.

Dokaz. ***(a)*** Neka je A odlučiv jezik i neka je D odlučivač koji ga prepoznaje. Konstruišimo enumerator E koji enumeriše A u leksikografskom poretku (algoritam 4.40).

Algoritam 4.40 Ispis jezika A u leksikografskom poretku.

```
1: procedure E
2:     x ← ε;
3:     while 1 do
4:         if D(x) == accept then
5:             print(x);
6:         x ←slijedeći string u leksikografskom poretku;
```

Ako E ispisuje x, tada je uslov u liniji 4 ispunjen, pa $x \in A$. Znači, E ispisuje samo stringove iz A. Dokažimo da će svaki string biti ispisan u nekom koraku. Neka je $y \in A$ proizvoljan string i neka su $y_0 = \varepsilon, y_1, ..., y_m$ stringovi koji dolaze prije y u leksikografskom poretku. Pošto je D odlučivač, $D(y_i)$ se završava u konačno mnogo koraka ($i = 0, ..., m$), pa će varijabla x enumeratora E primiti vrijednost $x = y$ nakon konačno mnogo koraka. Znači, nakon konačno mnogo koraka u liniji 4 ćemo imati $D(x = y) = accept$ i E će ispisati y.

Iz svega imamo da E enumeriše A.

(b) Neka je A jezik i neka ga E enumeriše u leksikografskom poretku. Konstruišimo odlučivač D koji prepoznaje A (algoritam 4.41).

Ako je A konačan jezik, tada je i odlučiv, jer je svaki konačni jezik odlučiv. Pretpostavimo da je A beskonačan jezik.

Algoritam 4.41 Odlučivač za jezik A.

```
1: procedure D(ω)
2:     pokreni enumerator E i prati ispis;
3:     if u nekom koraku je ispisan ω then
4:         return (accept);
5:     if u nekom koraku je ispisan string duži od ω then
6:         return (reject);
```

Ako je $D(\omega) = accept$, tada E ispisuje ω, pa $\omega \in A$.

Ako je $D(\omega) = reject$, to znači da je E ispisao string ω', koji je duži od ω. Pošto E ispisuje stringove u leksikografskom poretku i $|\omega'| > |\omega|$, to se ω ne može naći u ispisu poslije ω', pa E ne ispisuje ω, tj. $\omega \notin A$.

Pošto stringova ne dužih od ω ima konačno mnogo, to će je dan od uslova u linijama 3 i 5 biti ispunjen u konačno mnogo koraka, pa će D vratiti *accept* ili *reject*.

Iz svega imamo da je D odlučivač i prepoznaje A, pa je A odlučiv jezik. ■

4.10 Neograničene gramatike

Ranije smo pokazali da su regularni izrazi ekvivalentni sa DFA i NFA; kontekstno nezavisne gramatike sa PDA. U ovoj sekciji ćemo uvesti pojam neograničene gramatike i dokazati da su ekvivalente sa Turingovim mašinama.

Definicija 4.8 — Neograničena gramatika. Neograničena gramatika je uređena četvorka (V, Σ, R, S) pri čemu je:

1. V je skup varijabli;
2. Σ je alfabet;
3. R je skup pravila kod kojih se jedan string iz $(V \cup \Sigma)^*$ mijenja sa stringom iz $(V \cup \Sigma)^*$;
4. S je početna varijabla.

Definicija neograničene gramatike ne daje na koji način gramatika generiše određeni string. Zato sada formalno uvodimo pojam generisanja stringa.

Definicija 4.9 — String kojeg generiše gramatika. Kažemo da gramatika $G = (V, \Sigma, R, S)$ generiše string $\omega \in \Sigma^*$, ako postoji niz stringova $\Omega_0, \Omega_1, \ldots, \Omega_k$ iz $(V \cup \Sigma)^*$, tako da važi:

1. $\Omega_0 = S$;
2. $\Omega_k = \omega$;
3. Ω_i i Ω_{i+1} se mogu napisati u obliku $\Omega_i = \alpha_i \rho_i \beta_i$, $\Omega_{i+1} = \alpha_i \rho_{i+1} \beta_i$ $(i = 0, \ldots, k-1)$, pri čemu $\alpha_i, \beta_i, \rho_i \in (V \cup \Sigma)^*$ i pravilo $\rho_i \to \rho_{i+1}$ pripada R.

Niz $\Omega_0, \ldots, \Omega_k$ nazivamo *generatorni niz* stringa ω nad gramatikom G.

■ **Primjer 4.11** Konstruišimo neograničenu gramatiku za jezik $A = \{a^n b^n c^n \mid n \geq 0\}$. Iz primjera 3.20 imamo da za A ne postoji CFG.

Neograničena gramatika je data sa:

$$
\begin{aligned}
&S \to ABCS \\
&S \to Z \\
&CA \to AC \\
&BA \to AB \\
&CB \to BC \\
&CZ \to Zc \\
&Z \to Y \\
&BY \to Yb \\
&Y \to X \\
&AX \to Xa \\
&X \to \varepsilon\,.
\end{aligned}
$$

Npr: $S \to ABCS \to ABCABCS \to ABCABCZ \to ABACBCZ \to AABCBCZ \to AABBCCZ \to \to AABBCZc \to AABBZcc \to AABBYcc \to AABYbcc \to AAYbbcc \to AAXbbcc \to AXabbcc \to \to Xaabbcc \to aabbcc = a^2b^2c^2$.

■

Teorema 4.5 Jezik je Turing-prepoznatljiv akko ga neka neograničena gramatika generiše.

Dokaz. ***(a)*** Neka neograničena gramatika $G = (V, \Sigma, R, S)$ generiše jezik A. Konstruišimo nedeterminističku Turingovu mašinu M koja prepoznaje A (algoritam 4.42).

Ako $\omega \in A$, tada postoji konačan niz pravila iz R pomoću kojeg iz S možemo

Algoritam 4.42 Nedeterministčka Turingova mašina za jezik A.

```
1: procedure M(ω)
2:     Ω ← S;
3:     while 1 do
4:         nedeterministički odaberi pravilo iz R i primjeni ga na Ω, ako je
   to moguće;
5:         if Ω == ω then
6:             return (accept);
```

dobiti ω. Pošto M nedeterministički ispituje sve mogućnosti, jedna grana komjutacije će dati rezultat ω u konačno mnogo korakai vratiti *accept*.

Neka je sada $M(\omega) = accept$, tada je mašina M našla niz pravila iz R koji vode od S do ω, pa $\omega \in A$. Znači, M prepoznaje A. Iz tvrdnje 4.4 imamo da je A Turing-prepoznatljiv jezik.

(b) Neka je M Turingova mašina koja prepoznaje jezik A. Dokažimo da postoji neograničena gramatika $G = (V, \Sigma, R, S)$ koja generiše A.

Prisjetimo se na koji smo način zapisivali konfiguraciju Turingove mašine. String $100q_2110$ nam je označavao da se mašina nalazi u stanju q_2, na traci se nalazi string 100110 i glava pokazuje na četvrti simbol (tj. 1). Svaka tranzicija Turingove mašine se može predstaviti na jedan od slijedećih načina:

$$q_i\alpha \to \beta q_{i+1}\,;$$
$$\alpha q_i\beta \to q_{i+1}\alpha\gamma\,,$$

pri čemu prvo pravilo odgovara pomjeranju glave udesno (i umjesto α upisujemo β), a drugo-ulijevo (i umjesto β upisujemo γ), te $\alpha, \beta, \gamma \in \Sigma, q_i \in Q(M), (i = 1, ..., |Q|)$.

Znači, rad Turingove mašine možemo predstaviti pomoću neograničene gramatike. ■

4.11 Chomskyeva hijerarhija formalnih jezika

Sada ćemo uvesti pojam mašine koja je slabija od Turingove mašine, a jača od PDA. Radi se o Turingovoj mašini sa ograničenom memorijom.

Definicija 4.10 — Linearni ograničeni automat. Linearni ograničeni automat je nedeterministička Turingova mašina kod koje glava za čitanje/pisanje se ne može pomjeriti van dijela koji sadrži ulazni string.

Skraćenica koju ćemo koristiti je LBA (*Linear Bounded Automaton*). Sada uvodimo gramatiku koja dogovara LBA.

Definicija 4.11 — Kontekstno zavisna gramatika. Kontekstno zavisna gramatika je uređena četvorka (V,Σ,R,S) pri čemu je:

1. V je skup varijabli;
2. Σ je alfabet;
3. R je skup pravila oblika: $\alpha A\beta \to \alpha B\beta$, pri čemu $A \in V, \alpha,\beta \in (V\cup\Sigma)^*$, $B \in (V\cup\Sigma)^+$;
4. S je početna varijabla.

Definicija 4.12 — String kojeg generiše gramatika. Kažemo da gramatika $G=(V,\Sigma,R,S)$ generiše string $\omega \in \Sigma^*$, ako postoji niz stringova $\Omega_0,\Omega_1,\ldots,\Omega_k$ iz $(V\cup\Sigma)^*$, tako da važi:

1. $\Omega_0 = S$;
2. $\Omega_k = \omega$;
3. Ω_i i Ω_{i+1} se mogu napisati u obliku $\Omega_i = \alpha_i\rho_i\beta_i$, $\Omega_{i+1} = \alpha_i\rho_{i+1}\beta_i$ $(i = 0,\ldots,k-1)$, pri čemu $\alpha_i,\beta_i \in (V\cup\Sigma)^*$ i pravilo $\rho_i \to \rho_{i+1}$ pripada R.

Sa $L(G)$ označavamo skup svih stringova iz Σ^* koje generiše gramatika G. Niz $\Omega_0,\Omega_1,\ldots,\Omega_k$ nazivamo *generatorni niz* stringa ω nad gramatikom G.

Teorema 4.6 Za jezik A postoji linearni ograničeni automat koji ga prepoznaje akko A generiše neka kontekstno zavisna gramatika.

Dokaz. Dokaz je sličan dokazu teoreme 4.5. Uočite da, kod kontekstno zavisnih gramatika, pravila imaju oblik: $\alpha A\beta \to \alpha B\beta$, pri čemu $A \in V, \alpha,\beta \in (V\cup\Sigma)^*$, $B \in (V\cup\Sigma)^+$. Prema tome, $B \neq \varepsilon$, pa nijedan član generatornog niza ne može biti duži od konačnog stringa koji se generiše.

(a) Neka kontekstno zavisna gramatika $G=(V,\Sigma,R,S)$ generiše jezik A. Konstruišimo LBA M koja prepoznaje A (algoritam 4.43).

Ako $\omega \in A$, tada postoji niz pravila iz R pomoću kojeg iz S možemo dobiti ω. Pošto M nedeterministički ispituje sve mogućnosti, jedna grana komjutacije

Algoritam 4.43 LBA za jezik A.

```
1: procedure M(ω)
2:     Ω ← S;
3:     while 1 do
4:         nedeterministički odaberi pravilo iz R i primjeni ga na Ω, ako je
   to moguće;
5:         if |Ω| > |ω| then
6:             return (reject);
7:         if Ω == ω then
8:             return (accept);
```

će dati rezultat ω i vratiti *accept*.

Neka je sada $M(\omega) = accept$. Tada je mašina M našla niz pravila iz R koji vode od S do ω, pa $\omega \in A$. Znači, M prepoznaje A. Iz tvrdnje 4.4 imamo da je A Turing-prepoznatljiv jezik.

Nijedan član generatornog niza nije duži od ulaznog stringa. Prema tome, iskoristićemo samo ograničeni dio memorije dužine ulaznog stringa, pa je M LBA.

(b) Neka je M LBA koja prepoznaje jezik A. Dokažimo da postoji kontekstno zavisna gramatika $G = (V, \Sigma, R, S)$ koja generiše A.

Konfiguraciju LBA ćemo zapisivati na isti način kao i konfiguraciju Turingove mašine. String $100q_2110$ nam je označavao da se mašina nalazi u stanju q_2, na traci se nalazi string 100110 i glava pokazuje na četvrti simbol (tj. 1). Svaka tranzicija LBA se može predstaviti na jedan od slijedećih načina:

$$q_i\alpha \to \beta q_{i+1}\,;$$
$$\alpha q_i\beta \to q_{i+1}\alpha\gamma\,,$$

pri čemu prvo pravilo odgovara pomjeranju glave udesno (i umjesto α upisujemo β), a drugo-ulijevo (i umjesto β upisujemo γ), te $\alpha, \beta, \gamma \in \Sigma, q_i \in Q(M), (i = 1, ..., |Q|)$. Pošto nemamo tranziciju koja mijenja varijablu sa ε, neće se dogoditi da dobijemo član generatornog niza duži od stringa kojeg generišemo. Prema tome, kompjutacija se odvija unutar trake čija dužina ne prelazi dužinu ulaznog stringa.

Znači, rad LBA možemo predstaviti pomoću kontekstno zavisne gramatike.

■

Ukratko ćemo spomenuti pojam *regularne gramatike*.

Definicija 4.13 — Regularna gramatika. *Regularna gramatika* je gramatika koja ima pravila oblika:

$$A \to aB;$$
$$A \to Ba;$$
$$A \to a;$$
$$A \to \varepsilon,$$

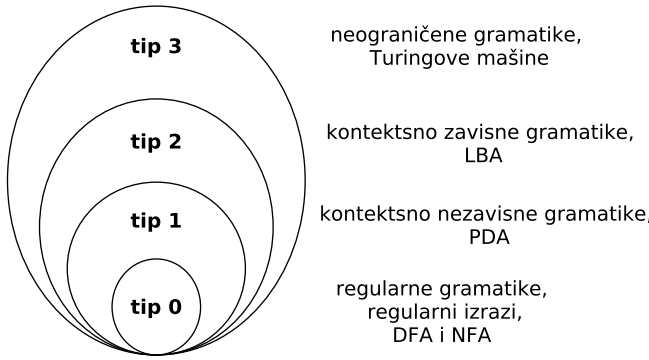
pri čemu su A i B varijable, $a \in \Sigma$.

Jednostavno se pokaže da su regularne gramatike ekvivalentne regularnim izrazima.

Sada možemo navesti Chomskyevu podjelu gramatika.

Definicija 4.14 — Chomskyeva hijerarhija. Gramatike možemo podijeliti u četiri grupe:
(*a*) tip 0: neograničene gramatike;
(*b*) tip 1: kontekstno zavisne gramatike;
(*c*) tip 2: kontekstno nezavine gramatike;
(*d*) tip 3: regularne gramatike.

Slika 4.15 daje prikaz Chomskyeve hijerarhije.



Slika 4.15: Chomskyeva hijerarhija.

4.12 Registarska mašina

Model registarskih mašina više odgovara radu standardnih kompjutera nego Turingove mašine. Registarska mašina ima konačno mnogo registara $R_0, R_1, ..., R_n$, koji omogućavaju direktan pristup memoriji u konstantnom vremenu. Formalno, R_i čuva prirodan broj, koji predstavlja dio procesa računanja.

Definicija 4.15 — Registarska mašina. Registarska mašina je data sa:
(*a*) konačno mnogo registara $R_0, R_1, ..., R_n$, pri čemu svaki može da sadrži prirodan broj;
(*b*) program koji se sastoji od konačne liste instrukcija: $L_0, L_1, ...$

Definicija 4.16 — Instrukcija registarske mašine. Instrukcija može imati jedan od slijedećih oblika:
(*a*) $R+ \to L'$ - dodaj jedan registru R i idi na instrukciju L';
(*b*) $R- \to L', L''$ - ako je sadržaj od R veći od nule, smanji ga za 1 i idi na instrukciju L'; inače idi na instrukciju L'';
(*c*) *stop* - završi sa radom.

Definicija 4.17 — Konfiguracija registarske mašine. Konfiguracija registarske mašine je data sa $c = (l, r_o, ..., r_n)$, pri čemu je l oznaka trenutne instrukcije, a r_i sadržaj registra R_i $(i = 0, ..., n)$.

Definicija 4.18 — Računanje registarske mašine. Računanje registarske mašine je konačan ili beskonačan niz konfiguracija $c_0, c_1, c_2, ...$ pri čemu:
(*a*) $c_0 = (0, r_0, ..., r_n)$ je početna konfiguracija;
(*b*) svaki $c = (l, r_0, ..., r_n)$ prelazi na slijedeću konfiguraciju izvršavajući instrukciju L_l sa registrima koji sadrže $r_0, ..., r_n$.
Za konačno računanje $c_0, c_1, ..., c_m$ zadnja konfiguracija c_m je završna konfiguracija ako je $L =$ stop ili $R+ \to L$ ili $R- \to L, L'$ sa $R > 0$ ili $R- \to L, L'$ sa $R = 0$ i ne postoji instrukcija L u programu ("zaustavljanje sa greškom").

Slijedeću teoremu navodimo bez dokaza.

Teorema 4.7 Registarska mašina je ekvivalentna Turingovoj mašini.

Zadaci za samostalan rad

Zadatak 4.4 Dati pseudokod algoritma koji, za ulazni string ω i $k \geq 1$, vraća sve moguće podjele stringa ω na k dijelova $a_1, a_2, \ldots, a_k$, tj. da važi $\omega = a_1 a_2 \ldots a_k$. ■

Zadatak 4.5 Konstruisati dijagram Turingove mašine koja prepoznaje jezik:
(*a*) $A = \{a^n b^n c^n \mid n \geq 0\}$;
(*b*) $B = \{0^{2^n} \mid n \geq 0\}$.
■

Zadatak 4.6 Napisati pseudokod algoritma koji ispisuje stringove u leksikografskom poretku. ■

Zadatak 4.7 Za sve jezike koje smo spominjali, a nisu kontekstno nezavisni, napisati neograničenu gramatiku koja ih generiše. ■

Zadatak 4.8 Neka je $f : \mathbb{N} \to \mathbb{N}$ izračunljiva funkcija. Dokaži da je $F : \mathbb{N} \to \mathbb{N}$ data sa $F(n) = f(f(...f(n)))$, pri čemu se f javlja n puta, izračunljiva funkcija. ■

Zadatak 4.9 Dokaži da svaki beskonačan Turing-prepoznatljiv jezik ima beskonačan odlučiv podskup. ■

Zadatak 4.10 Jezik $A_{PDA} = \{(P, \omega) \mid P \text{ je PDA koji prihvata } \omega\}$ je odlučiv jezik. ■

Zadatak 4.11 Neka su A i B jezici i $A \cap B = \varnothing$. Jezik C *razdvaja* jezike A i B ako $A \subseteq C$ i $B \subseteq \overline{C}$.

Dokaži da, za bilo koja dva ko-Turing-prepoznatljiva jezika, postoji odlučiv jezik koji ih razdvaja. ■

Zadatak 4.12 Neka je C jezik. Dokaži da je C Turing-prepoznatljiv akko postoji odlučiv jezik D tako da važi $C = \{x \mid \exists y : (x, y) \in D\}$. ■

5. NP-teški problemi

Do sada smo se bavili generalnim pitanjem da li se neki problem može riješiti u konačno mnogo koraka pomoću date mašine. U ovom poglavlju ćemo posmatrati probleme koji se mogu riješiti u konačno mnogo koraka, tj. odlučivi su. Pitanje koje ćemo postaviti je da li se mogu riješiti efikasno, tj. da li postoji brzi algoritam koji rješava dati problem?

5.1 Asimptotska notacija

Brzinu algoritma ćemo definisati pomoću broja koraka koje dati algoritam izvrši. Pri tome, dovoljno nam je odrediti približan broj koraka koje algoritam izvrši. Neki od razloga zašto ne dajemo tačan broj koraka koje algoritam izvršava su:

- broj koraka ne ovisi samo od algoritma, već i od njegove implementacije;
- nekada nije jasno koji koraci su elementarni, tj. da li se mogu se brojati kao jedan korak;
- lakše je odrediti približan broj koraka;
- za velike ulazne podatke je bitnije da znamo kako raste funkcija koja

predstavlja broj koraka.

Na koji način porediti broj koraka dva algoritma, ako koristimo metode za približno određivanje? Kako reći da jedan algoritam zahtijeva više koraka od drugog? To možemo ako uvedemo *asimptotsku notaciju*.

Drugi razlog za uvođenje asimptotske notacije je da nas često interesuje brzina algoritma samo za relativno velike n, pri čemu je n veličina ulaza. Zašto nas ne interesuje brzina za male n-ove? Jedan od razloga je da većina algoritama, implementirana na kompjuterima, je brza za male n, pa ne možemo steći realnu sliku koji je algoritam bolji.

Slijedeća definicija daje način da kažemo da je jedna funkcija veća ili jednaka od druge, za dovoljno veliki argument.

Definicija 5.1 — Veliko O. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Ako postoji $c > 0$ i $n_0 \in \mathbb{N}$ tako da važi:

$$f(n) \leq c \cdot g(n), \forall n \geq n_0,$$

tada pišemo $f(n) = O(g(n))$ ili $f(n) \in O(g(n))$. Kažemo da je $g(n)$ *asimptotska gornja granica* za $f(n)$ ili da je $f(n)$ *asimptotski manje ili jednako* od $g(n)$.

■ **Primjer 5.1** Neka je $f(n)$ polinom. Npr. $f(n) = 10n^4 + 3n^3 - 5n^2 + 2n + 10$. Jedan način da odredimo asimptotsku gornju granicu je da posmatramo najveći eksponent u polinomu. Dokažimo da vrijedi $f(n) \in O(n^4)$.

Trebamo dokazati da vrijedi $f(n) \leq cn^4$, $\forall n \geq n_0$, za neke $c > 0$ i $n_0 \in \mathbb{N}$. Imamo slijedeći niz ekvivalencija:

$$\begin{aligned} &10n^4 + 3n^3 - 5n^2 + 2n + 10 \leq cn^4 | : n^4 \iff \\ &\quad 10 + \frac{3}{n} - \frac{5}{n^2} + \frac{2}{n^3} + \frac{10}{n^4} \leq c. \end{aligned}$$

Ako pustimo da $n \to +\infty$, tada lijeva strana teži ka 10. Ako za vrijednost parametra c uzmemo bilo koju konstantu veću od 10, tada će posljednja relacija da važi za dovoljno veliko n, tj. postoji $n_0 = n_0(c)$ tako da posljednja relacija važi za svako $n \geq n_0$.

Pošto je posljednja relacija ekvivaletna prvoj relaciji, imamo i da prva relacija važi za svako $c > 10$ (npr. $c = 11$) i svako $n \geq n_0 = n_0(c)$, tj. $f(n) \in O(n^4)$. ■

Tvrdnja 5.1 Neka je $f_1(n) \in O(g_1(n))$, $f_2(n) \in O(g_2(n))$ i $g_1(n) \in O(g_2(n))$. Tada je

$$f_1(n) + f_2(n) \in O(g_2(n)).$$

Ovu činjenicu još pišemo i kao

$$O(g_1(n)) + O(g_2(n)) = O(g_2(n)).$$

Dokaz. Iz uslova tvrdnje imamo da postoje $c_i > 0$ i $n_i \in \mathbb{N}$ tako da važi:

$$f_i(n) \leq c_i g_i(n), \forall n \geq n_i, (i = 1,2).$$

Takođe važi

$$g_1(n) \leq c' g_2(n), \forall n \geq n'.$$

Neka je $n_0 = max\{n', n_1, n_2\}$. Dalje imamo
$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n) \leq c_1 c' g_2(n) + c_2 g_2(n) = (c_1 c' + c_2) g_2(n)$
$= c g_2(n), \forall n \geq n_0$. Tvrdnja je dokazana. ■

■ **Primjer 5.2** Iz tvrdnje 5.1 imamo da prilikom sabiranja funkcija, na asimptotsku notaciju utiče samo funkcija koja je asimptotski veća. Npr. važi: $O(n^4) + O(n^3) = O(n^4)$. ■

■ **Primjer 5.3** Iako se u praksi ne događa često, može se dogoditi da $f(n) \in O(g(n))$, ali da $\lim_{n \to +\infty} \frac{f(n)}{g(n)}$ ne postoji.

Npr. $f(n) = 2 + (-1)^n \in \{1,3\}$ i $g(n) = 5 + (-1)^n \in \{4,6\}$. Očigledno važi $f(n) \leq g(n), \forall n$, pa $f(n) \in O(g(n))$.

Sa druge strane, vrijednost $\frac{f(n)}{g(n)}$ oscilira između $\frac{1}{4}$ i $\frac{3}{6} = \frac{1}{2}$, pa $\lim_{n \to +\infty} \frac{f(n)}{g(n)}$ ne postoji. ■

Slijedeća tvrdnja opisuje relaciju O pomoću limesa.

Tvrdnja 5.2 Neka $f, g : \mathbb{N} \to \mathbb{R}^+$ i neka postoji $\lim_{n \to +\infty} \frac{f(n)}{g(n)}$. Tada

$$f(n) \in O(g(n)) \iff \lim_{n \to +\infty} \frac{f(n)}{g(n)} < +\infty.$$

Dokaz. ***(a)*** Neka $f(n) \in O(g(n))$. Tada je

$$f(n) \leq c \cdot g(n), \forall n \geq n_0, \text{ za neko } c > 0 \iff$$

$$\frac{f(n)}{g(n)} \leq c, \forall n \geq n_0, \text{ za neko } c > 0.$$

Ako pređemo na limes, dobijamo:

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} \leq \lim_{n\to+\infty} c = c < +\infty,$$

što je i trebalo dokazati.

(b) Neka je sada $\lim_{n\to+\infty} \frac{f(n)}{g(n)} < +\infty$. Tada je $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = c$. Iz definicije limesa imamo:

$$\left| \frac{f(n)}{g(n)} - c \right| \leq \varepsilon, \forall \varepsilon > 0, \forall n \geq n_\varepsilon.$$

Uzimajući $\varepsilon = 1$ imamo:

$$-1 \leq \frac{f(n)}{g(n)} - c \leq 1 \implies c - 1 \leq \frac{f(n)}{g(n)} \leq c + 1.$$

Na kraju imamo:

$$f(n) \leq (c+1)g(n), \forall n \geq n_1 \implies f(n) \in O(g(n)),$$

što je i trebalo dokazati. ■

Slijedeći definicija uvodi relaciju *asimptotski strogo manje*.

Definicija 5.2 — Malo o. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Ako je:

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0,$$

pišemo $f(n) \in o(g(n))$ ili $f(n) = o(g(n))$ i kažemo da je $f(n)$ *asimptotski strogo manje od* $g(n)$.

Definicija 5.2 govori da $f(n)$ raste dosta sporije nego $g(n)$.
Slijedeća tvrdnja daje karakterizaciju o notacije bez limesa.

Tvrdnja 5.3 Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Važi:

$$f(n) \in o(g(n)) \iff \forall c > 0, \exists n_c \in \mathbb{N} : f(n) \leq c \cdot g(n), \forall n \geq n_c.$$

Dokaz. Koristićemo definiciju limesa i o notacije.

Važi:

$$\begin{aligned}
f(n) \in o(g(n)) &\iff \\
\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0 &\iff \text{(definicija limesa)} \\
\forall c > 0, \exists n_c : \left|\frac{f(n)}{g(n)}\right| \leq c, \forall n \geq n_c &\iff \text{(jer } f(n) \geq 0, g(n) > 0) \\
\forall c > 0, \exists n_c : \frac{f(n)}{g(n)} \leq c, \forall n \geq n_c &\iff \text{(množimo sa } g(n) > 0) \\
\forall c > 0, \exists n_c : f(n) \leq c \cdot g(n), \forall n \geq n_c,
\end{aligned}$$

što je i trebalo dokazati. ■

Slijedeći primjer ilustruje da stepen logaritma je asimptotski strogo manji nego stepena funkcija, a ona je opet asimptotski strogo manja od eksponencijalne funkcije.

■ **Primjer 5.4** Važi:
(*a*) $log_a^k(n) \in o(n^b)$, za $a,b,k > 0$ i $a \neq 1$;
(*b*) $n^a \in o(b^n)$, za $a > 0$ i $b > 1$.

(a) Posmatrajmo $\lim_{n\to+\infty} \frac{log_a^k(n)}{n^b}$ za $k \in \mathbb{N}$ i $a > 1$. Primjenjujući L'Hospitalovo pravilo imamo:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{log_a^k(n)}{n^b} &= \lim_{n\to+\infty} \frac{k \cdot log_a^{k-1}(n)}{b \cdot n^{b-1}} \frac{1}{n \cdot ln(a)} = \\
&= \frac{1}{ln(a)} \lim_{n\to+\infty} \frac{k \cdot log_a^{k-1}(n)}{b \cdot n^b} = \\
&= \frac{1}{ln^2(a)} \lim_{n\to+\infty} \frac{k(k-1) \cdot log_a^{k-2}(n)}{b^2 \cdot n^b} = \\
&= \ldots = \\
&= \frac{1}{ln^k(a)} \lim_{n\to+\infty} \frac{k(k-1)\ldots 1}{b^k \cdot n^b} = 0.
\end{aligned} \tag{5.1}$$

Znači, $log_a^k(n) \in o(n^b)$, za $a,b > 0$, $k \in \mathbb{N}$ i $a > 1$.

Neka je $k > 0$ realni broj i $a > 1$. Tada postoji $k' \in \mathbb{N}$ i $k < k'$ i vrijedi:

$$0 \leq \lim_{n\to+\infty} \frac{log_a^k(n)}{n^b} \leq \lim_{n\to+\infty} \frac{log_a^{k'}(n)}{n^b} = 0,$$

pa iz teorema o uklještenju limesa imamo

$$\lim_{n\to+\infty} \frac{log_a^k(n)}{n^b} = 0,$$

što je i trebalo dokazati.

Ako je $0 < a < 1$, tada $\lim_{n\to+\infty} log_a^k(n) = 0$, pa je

$$\lim_{n\to+\infty} \frac{log_a^k(n)}{n^b} = 0,$$

tj.

$$log_a^k(n) \in o(n^b).$$

Ovime završavamo primjer pod (a).

(b) Prvo uzmimo da je $a \in \mathbb{N}$. Iz L'Hospitalovog pravila imamo.

$$\begin{aligned}
\lim_{n\to+\infty} \frac{n^a}{b^n} &= \lim_{n\to+\infty} \frac{a \cdot n^{a-1}}{b^n \cdot ln(b)} = \\
&= \frac{a}{ln(b)} \lim_{n\to+\infty} \frac{n^{a-1}}{b^n} = \\
&= \frac{a(a-1)}{ln^2(b)} \lim_{n\to+\infty} \frac{n^{a-2}}{b^n} = \\
&= ... = \\
&\frac{a(a-1)...1}{ln^a(b)} \lim_{n\to+\infty} \frac{1}{b^n} = 0.
\end{aligned} \tag{5.2}$$

Neka je $a > 0$ realan broj. Tada postoji $a' \in \mathbb{N}$ tako da je $a < a'$. Dalje imamo:

$$0 \leq \lim_{n\to+\infty} \frac{n^a}{b^n} \leq \lim_{n\to+\infty} \frac{n^{a'}}{b^n} = 0.$$

Iz teorema u ukliještenju imamo da je

$$\lim_{n\to+\infty} \frac{n^a}{b^n} = 0,$$

tj.

$$n^a \in o(b^n).$$

■

Definicija 5.3 — Asimptotski jednake funkcije. Neka $f, g : \mathbb{N} \to \mathbb{R}^+$. Ako $f(n) \in O(g(n))$ i $g(n) \in O(f(n))$, tada kažemo da su $f(n)$ i $g(n)$ *asimptotski jednake*, te pišemo $f(n) \in \Theta(g(n))$ i $g(n) \in \Theta(f(n))$ ili $f(n) = \Theta(g(n))$ i $g(n) = \Theta(f(n))$.

Slijedeći primjer ilustruje zašto nije bitna baza logaritma (sve dok je veća od 1) kada analiziramo asimptotsku kompleksnost.

■ **Primjer 5.5** Neka su $a, b > 1$. Tada je $log_a(n) \in \Theta(log_b(n))$.

Dokažimo ovu tvrdnju.

Prvo dokažimo da je $log_a(n) \in O(log_b(n))$, tj. da važi

$$log_a(n) \leq c \cdot log_b(n), \text{ za neko } c \text{ i za svako } n \geq n_0.$$

Koristeći formulu za pretvaranje iz jedne baze u drugu: $log_b(n) = \frac{log_a(n)}{log_a(b)}$, prethodna nejednakost je ekvivalentna sa:

$$log_a(n) \leq c \cdot \frac{log_a(n)}{log_a(b)}.$$

Ako uzmemo da je $c = log_a(b)$ dobijamo da je prethodna nejednakost tačna za sve n, pa je i prva nejednakost tačna.

Dokaz da $log_b(n) \in O(log_a(n))$ ide analogno.

Iz $log_a(n) \in O(log_b(n))$ i $log_b(n) \in O(log_a(n))$ dobijamo $log_a(n) \in \Theta(log_b(n))$.

■

5.2 Vremenska kompleksnost i klasa *P*

Kao što smo već napisali, u ovom poglavlju nas interesuje koji se problemi mogu riješiti brzo. Za to moramo uvesti pojam vremena koje algoritam potroši rješavajući neki problem.

Kako definisati vrijeme? Vrijeme ćemo definisati kao broj koraka Turingove mašine.

Definicija 5.4 — Vrijeme izvršavanja odlučivača. Neka je D odlučivač i $t : \mathbb{N} \to \mathbb{N}$. Za D kažemo da je $t(n)$ vremenska Turingova mašina (odlučivač), ako za svaki ulazni string dužine n, D završava sa radom u najviše $t(n)$ koraka.

Za D kažemo da je $O(t(n))$ vremenska Turingova mašina (odlučivač), ako za svaki ulazni string dužine n, D završava sa radom u $O(t(n))$ koraka.

U prethodnoj definiciji, umjesto $t : \mathbb{N} \to \mathbb{N}$ smo mogli uzeti i $t : \mathbb{N} \to \mathbb{R}^+$. Ako je D $O(t(n))$ vremenska Turingova mašina, tada postoji $c > 0$ i $n_0 \in \mathbb{N}$ tako da za svaki ulazni string dužine n, pri čemu je $n \geq n_0$, D završava sa radom u najviše $c \cdot t(n)$ koraka.

■ **Primjer 5.6** Neka je $A = \{\omega \mid \omega = \omega^R, \omega \in \Sigma^*\}$ skup svih palindroma. Konstruisaćemo algoritam koji odlučuje A (algoritam 5.1).

Algoritam 5.1 Odlučivač za skup svih palindroma.

```
1:  procedure D(ω)
2:      while true do
3:          a ← simbol na trenutnoj poziciji glave;
4:          upiši ×;
5:          pomjeraj glavu na drugi kraj stringa, dok ne dođeš do × ili ␣;
6:          if glava pomjerena za tačno jedno mjesto then
7:              return (accept);
8:          pomjeri glavu jedno mjesto unazad;
9:          b ← simbol na koji glava pokazuje;
10:         if a ≠ b then
11:             return (reject);
12:         upiši ×;
13:         pomjeri glavu jedno mjesto unazad;
```

Analizirajmo vrijeme izvršavanja algoritma 5.1. Imamo:

- u prvom prolazu: čita a (1), upisuje $\times$ (1), ide na drugi kraj stringa (pomjeranje glave, čitanje simbola, upoređivanje sa $\times$ i $\sqcup$: $4(n-1)$), vraća se jedno mjesto nazad (1), što je ukupno $4n-1$;
- u slijedećem prolazu string je kraći za dva simbola, pa ima $4(n-2)-1$ operacija;
- itd;

- ukupno: $4(n+(n-2)+...+1/2)-\lfloor\frac{n}{2}\rfloor \approx 4n\cdot\frac{n}{2}-\frac{n}{2}\left(\frac{n}{2}+1\right)-\frac{n}{2}\in O(n^2)$.

Znači, D je $O(n^2)$ vremenska Turingova mašina. ∎

U prethodnim primjeru smo operacije čitanja, pisanja, pomjeranja glave brojali kao 1, tj. kao elementarne operacije. Kompjutacijski model definiše koje su operacije elementarne, što opet utiče na procjenu vremena Turingove mašine.

U slijedećem primjeru ćemo uzeti isti jezik, ali drugačiji algoritam i neke druge operacije ćemo uzeti kao elementarne.

∎ **Primjer 5.7** Neka je $A=\{\omega \mid \omega=\omega^R, \omega\in\Sigma^*\}$ skup svih palindroma. Konstruisaćemo još jedan algoritam koji odlučuje A (algoritam 5.2).

Algoritam 5.2 Još jedan odlučivač za skup svih palindroma.

```
1: procedure D(ω)
2:     n ← |ω|;
3:     ω[i] ← (i+1)-i simbol od ω (i = 0,...,n−1);
4:     for i = 0,...,n−1 do
5:         if ω[i] ! = ω[n−1−i] then
6:             return (reject);
7:     return (accept);
```

Analizirajmo broj kraka po linijama koda:
(a) linija 2: 1 ili n koraka, zavisno od toga kako mjerimo dužinu stringa;
(b) linija 3: 1 ili n koraka, zavisno od implementacije;
(c) linija 4: n iteracija i svaka iteracija ima po najviše 2 koraka.

Jednostavnom analizom dobijemo da je vrijeme izvršavanja ovog algoritma jednako $O(n)$. ∎

Iz prethodna dva primjera vidimo da za isti problem imamo algoritme sa različitom kompleksnošću. Na kompleksnost utiče i šta podrazumijevamo pod *elementarnom operacijom*. U primjeru 5.6, da bi pristupili simbolu stringa, pomjerali smo glavu do tog simbola, što može zahtijevati $O(n)$ operacija. U primjeru 5.7 smo simbolu pristupali direktno sa $\omega[i]$, što je zahtijevalo jednu operaciju.

Čak i kada imamo isti algoritam i iste elementarne operacije, različitim analizama možemo dobiti različite procjene kompleksnosti. Ako analizom

dobijemo da se algoritam izvršava u vremenu $O(n^2)$, nekom kvalitetnijom analizom možemo dobiti da se algoritam izvršava u vremenu $O(n)$.

Zadatak 5.1 — Za samostalan rad. Petlju u algoritmu 5.2 optimiziraj tako da i ne ide do $n-1$. ■

K Ako se mašina D izvršava u vremenu $t(n)$, tada dužina ulaznog stringa ne može biti veća od $t(n)$. Naime, ulazni string ω mašina D mora da učita i pri tome će obaviti $O(n)$ operacija ($n = |\omega|$). Pošto D izvršava najviše $t(n)$ operacija, to je $t(n) \geq n = |\omega|$.

Iz istog razloga dužina stringa na traci, u bilo kojem trenutku, ne može biti veća od $t(n)$.

Slijedeće dvije tvrdnje govore kako više traka ili nedeterminizam utiče na vrijeme izvršavanja.

Tvrdnja 5.4 Neka je $t : \mathbb{N} \to \mathbb{N}$ i $t(n) \geq n$. Za svaku $t(n)$ vremensku Turingovu mašinu sa više traka, postoji ekvivalentna $O(t(n)^2)$ Turingova mašina sa jednom trakom.

Dokaz. Dokaz je sličan dokazu tvrdnje 4.2.

Posmatrajmo Turingovu mašinu M sa više traka (slika 4.7a). Za svaki simbol svakog stringa mašina mora obaviti barem jednu operaciju upisa. Pošto je ukupan broj operacija od M jednak $t(n)$, dužina ulaznog stringa, a i ukupna dužina stringova na svim trakama u bilo kojem trenutku ne može prelaziti $t(n)$.

Konstrukcija ekvivalentne Turingove mašine sa jednom trakom je objašnjena u sekciji 4.3.2, pa je sami objasnite na ovom mjestu.

Neka je M' ekvivalentna Turingova mašina sa jednom trakom (slika 4.7a). Iz prethodnog imamo da dužina trake ni u jednom trenutku nije veća od $O(t(n))$.

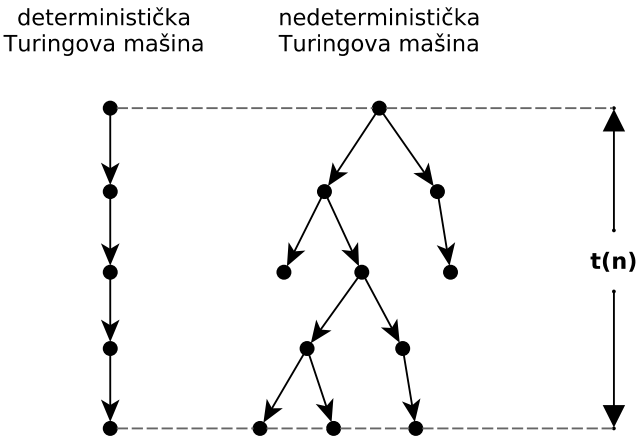
Za svaku operaciju od M, mašina M', u najgorem slučaju, mora obići cijelu traku određen broj puta i da napravi određeni broj upisa, tj. da obavi najviše $O(t(n))$ operacija. Pošto M izvršava $t(n)$ operacija, to M' izvršava $O(t(n)^2)$ operacija, što je i trebalo dokazati. ■

Kod "obične", nedeterminističke Turingove mašine, vrijeme izvršavanja smo definisali kao broj koraka. Kod nedeterminističke Turingove mašine

vrijeme izvršavanja se definiše na drugačiji način. Razlog je što u ovom slučaju imamo paralelno računanje, tj. više operacija se izvršava istovremeno.

Definicija 5.5 — Vrijeme izvršavanja nedeterminističkog odlučivača. Neka je D nedeterministički odlučivač. Vrijeme izvršavanje od D je preslikavanje $t : \mathbb{N} \to \mathbb{N}$, pri čemu je $t(n)$ najveći broj koraka koje koristi bilo koja grana kompjutacije, za bilo koji ulazni string dužine n.

Slika 5.1 opisuje prethodnu definiciju. Uočite da je $t(n)$ visina stabla kompjutacije nedeterminističke Turingove mašine.



Slika 5.1: Vrijeme izvršavanje determinističke i nedeterminističke Turingove mašine.

Jednostavno se dokaže da binarno stablo dubine/visine k ima najviše $2^{k+1} - 1$ čvorova (dokazati za samostalan rad).

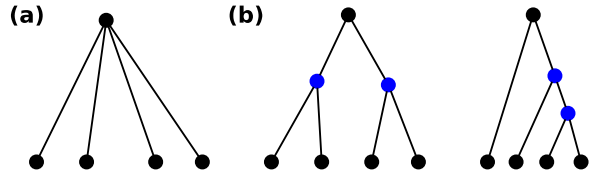
Tvrdnja 5.5 Neka je $t : \mathbb{N} \to \mathbb{N}$ i $t(n) \geq n$ vrijeme izvršavanja nedeterminističkog odlučivača. Tada postoji ekvivalentan deterministički odlučivač sa jednom trakom čije je vrijeme izvršavanja $2^{O(t(n))}$.

Dokaz. Dokaz je sličan dokazu tvrdnje 4.4.

Za samostalan rad, na ovom mjestu objasnite kako NTM transformišemo u ekvivalentnu TM.

Binarno stablo visine k ima najviše $2^{k+1} - 1$ čvorova. Šta ako stablo kompjutacije nedeterminističkog odlučivača M nije binarno? Nebinarno stablo se može na jednostavan način prikazati pomoću binarnog (slika 5.2). Na ovaj

način se visina stabla postaje najviše $c \cdot t(n)$, pri čemu je c najveći stepen čvora u stablu kompjutacije od M.



Slika 5.2: **(a)** Čvor sa k djece ($k \geq 3$). **(b)** Dodavanjem novih čvorova, dobijamo da svaki čvor ima tačno dvoje djece. Visina stabla se povećava za najviše $k-2$.

Novo binarno stablo ima najviše $2^{c \cdot t(n)+1} - 1 = 2^{O(t(n))}$ čvorova. Simulacijom prikazanom u dokazu tvrdnje 4.4 obilazimo sve ove čvorove, što daje $2^{O(t(n))}$ operacija. ■

Iz primjera 5.6 i 5.7 vidimo da za jedan problem može postojati više algoritama, sa različitim vremenom izvršavanja.

Definicija 5.6 — TIME(t(n)).

$$TIME(t(n)) = \{A \mid A \text{ je jezik odlučiv u vremenu } t(n)\}.$$

Znači, $A \in TIME(t(n))$ akko postoji odlučivač koji prepoznaje A i izvršava se u njaviše $t(n)$ koraka.

Kako neki algoritam opisati kao *"brzi algoritam"*? Brzina algoritma zavisi i od prirode problema. Za neke probleme je jednostavno naći brzi algoritam, dok za neke probleme nije to slučaj. Algoritmi tipa *brute force* nas ne interesuju, čak ni ako brzo rješavaju neke probleme, jer njihova konstrukcija ne zahtijeva značajniju kreativnost. Mogli bi reći da je algoritam *brz*, ako je *inteligentno konstruisan*. Opet se postavlja pitanje šta je *inteligentno konstruisan algoritam*?

Jedan od mogućih kriterija da kažemo da je neki algoritam brz (inteligentno konstruisan) je da se izvršava u polinomnom vremenu.

Definicija 5.7 — Klasa P. Klasa P je klasa jezika odlučivih u polinomnom

vremenu, tj.

$$P = \bigcup_{k=0}^{+\infty} TIME(n^k).$$

Primjer 5.6 daje jedan primjer jezika iz P.

■ **Primjer 5.8** Neka je SPANNINGTREE $= \{(G,k) \mid G$ je težinski graf koji ima pokrivajuće stablo težine $\leq k\}$. Konstruišimo polinomni odlučivač koji prepoznaje SPANNINGTREE, tj dokazaćemo da SPANNINGTREE $\in P$.

Koristićemo Primov algoritam (algoritam 5.3).

Algoritam 5.3 Primov algoritam.

Input: Povezan graf G i početni čvor v.
Output: Prim pokrivajuće stablo T od G.

```
1:  procedure PRIM(G, v)
2:      V(T) ← {v};                                  ▷ Inicijalizacija.
3:      E(T) ← ∅;
4:      RG(T) ← ∅ ;                                  ▷ Rubne grane.
5:      p(v) ← null;                                 ▷ Početni čvor nema roditelja.
6:      l(v) ← 0;                                    ▷ Oznaka čvora. Redni br. obrade.
7:      i ← 1;
8:      while V(T) ≠ V(G) do
9:          update RG(T);
10:         e = {u,w} ← grana iz RG(T) sa najmanjom težinom;;
11:         w ← čvor grane e koji nije u T;
12:         T ← T + e + w;          ▷ Dodajemo granu e i čvor w u stablo T.
13:         l(w) ← i;
14:         p(w) ← u;               ▷ Prethodnik/roditelj čvora w je čvor u.
15:         ++i;
16:     return (T);
```

Neka je $n = |V(G)|$ broj čvorova u grafu G. U svakoj iteraciji dodajemo po jedan čvor, pa imamo $O(n)$ iteracija. U svako iteraciji istražujemo skup rubnih grana, za što nam treba $O(m)$ vremena ($m = |E(G)|$). Znači, ukupna kompleksnost Primovog algoritma je $O(mn)$, pa SPANNINGTREE $\in P$. ■

Ako smo procjenom dobili da je algoritam iz $O(n^3)$, drugom boljom procjenom možemo dobiti da je isti algoritam iz $O(n^2)$. To važi i za SPANNINGTREE.

Procjena kompleksnosti algoritma ovisi i od memorijskih struktura koje koristimo za implementaciju. Da li koristimo matrice, liste, mape ili neke druge strukture utiče i na kompleksnost algoritma. Međutim, ovi detalji nam nisu bitni. Nas interesuje samo da li se neki problem može riješti u polinomnom vremenu.

Tvrdnja 5.4 govori da nam za polinomnost problema, tj. za pripadnost klasi P, nije bitan broj traka Turingove mašine koja rješava dati problem. Ako se algoritam izvršava u polinomnom vremenu na Turingovoj mašini sa više traka, tada će se izvršavati u polinomnom vremenu na Turingovoj mašini sa jednom trakom. Situacija je bitno drugačija ako je u pitanju nedeterministička Turingova mašina. Ako se se problem može riješiti u polinomnom vremenu pomoću nedeterminističke Turingove mašine, tada ne znamo ništa o tome da li se može riješiti u polinomnom vremenu pomoću "obične" Turingove mašine.

5.3 Polinomna redukcija

Da bi uveli pojam (polinomne) redukcije, prvo trebamo uvesti pojam *izračunljive funkcije*. Preslikavanje f je izračunljivo ako se svaka vrijednost $f(x)$ može izračunati u konačno mnogo koraka.

Definicija 5.8 — Izračunljivo preslikavanje. Za preslikavanje $f:\Sigma^*\to\Sigma^*$ kažemo da je *izračunljivo* ako postoji Turingova mašina koja za svaki ulaz ω se zaustavlja samo sa $f(\omega)$ na traci.

Neka su A i B problemi. Šta znači da smo problem A *reducirali* ili *sveli* na problem B? Sa redukcijom jednog problema na drugi se susrećete skoro uvijek kada rješavate neki problem.

■ **Primjer 5.9** Riješimo sistem jednačina:

$$\begin{array}{llll} x+y=2 & y=2-x & y=2-x & y=2-x \\ x\cdot y=1 & x\cdot y=1 & x\cdot(2-x)=1 & x^2-2x+1=0. \end{array}$$

Problem A (rješavanje sistema) smo postupkom f (algebarske transformacije) sveli na problem B (rješavanje kvadratne jednačine). ■

Sada formalno definišemo redukciju problema A na problem B (slika 5.3).

Definicija 5.9 — Redukcija jezika. Za preslikavanje $f : \Sigma^* \to \Sigma^*$ kažemo da je *redukcija jezika* A na *jezik* B ako važi:

$$f(\omega) \in B \iff \omega \in A, \forall \omega \in \Sigma^*.$$

Kažemo da je jezik A *reducibilan* na jezik B.

■ **Primjer 5.10** Posmatrajmo ponovo primjer 5.9. Sa A možemo označiti generalni problem rješavanja sistema jednačina, od koja je jedna jednačina linearna, a druga kvadratna. Sa B označavamo problem rješavanja kvadratne jednačine.

Preslikavanje f nam predstavlja niz algebarskih transformacija nad početnim sistemom jednačina. U datom primjeru, ω i $f(\omega)$ su konkretne instance problema, tj. $\omega = "x+y=2 \wedge xy=1"$, te $f(\omega) = "x^2-2x+1=0"$. ■

Pošto nas interesuje polinomno vrijeme izvršavanja, prethodne definicije ćemo obogatiti.

Definicija 5.10 — Polinomno izračunljivo preslikavanje. Za preslikavanje $f : \Sigma^* \to \Sigma^*$ kažemo da je *polinomno izračunljivo* ako postoji polinomno vremenska Turingova mašina koja za svaki ulaz ω se zaustavlja samo sa $f(\omega)$ na traci.

Definicija 5.11 — Polinomna redukcija jezika. Za polinomno izračunljivo preslikavanje $f : \Sigma^* \to \Sigma^*$ kažemo da je *polinomna redukcija jezika* A na *jezik* B ako važi:

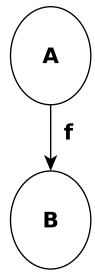
$$f(\omega) \in B \iff \omega \in A, \forall \omega \in \Sigma^*.$$

Kažemo da je jezik A *polinomno reducibilan* na jezik B i pišemo $A \leq_P B$.

Slijedeća tvrdnja objašnjava zašto koristimo oznaku $\leq_P$. Intuitivno, ako problem B možemo brzo (polinomno) riješiti i A možemo brzo (polinomno) svesti na B (slika 5.3), tada i problem A možemo brzo riješiti. Znači, iz brzog rješenja problema B, dobijamo brzo rješenja problema A. Zaključujemo da problem A nije teži od problema B, tj. $A \leq_P B$.

Tvrdnja 5.6 Neka su A i B jezici. Tada važi:

$$B \in P \wedge A \leq_P B \implies A \in P.$$



Slika 5.3: Redukcija problema A na problem B.

Dokaz. Neka je f polinomna redukcija sa A na B i D_2 polinomni odlučivač za B. Algoritam 5.4 daje pseudokod za D_1 - polinomni odlučivač za A.

Algoritam 5.4

```
1: procedure D1(ω)
2:     return (D2(f(ω)));
```

Pošto je D_2 polinomni odlučivač i f polinomno izračunljiva funkcija, to se linija 2 izvršava u polinomnom vremenu (jer je kompozicija dva polinoma polinom).

Važi:

- $D_1(\omega) = accept \iff$
- $D_2(f(\omega)) = accept \iff$ (jer je D_2 odlučivač za B)
- $f(\omega) \in B \iff$ (na osnovu definicije redukcije)
- $\omega \in A$.

Znači, D_2 je polinomni odlučivač koji prepoznaje A, pa $A \in P$, što je i trebalo dokazati. ■

5.4 Problemi zadovoljivosti logičkih formula

Prije nego nastavimo sa teorijom kompleksnosti, napravićemo malu digresiju da se upoznamo sa kompjutacijskim problemima iz matematičke logike. Ovi su problemi veoma značajni u teoriji NP-teških problema.

Slijedeća definicija nam daje pojam *logičke formule*. Pošto se sve logičke operacije mogu prikazati pomoću konjukcije ($\wedge$), disjunkcije ($\vee$) i negacije ($\neg$ ili $\overline{()}$), mi ćemo se ograničiti na ove tri operacije.

Redoslijed prioriteta je: zagrada, negacija, konjukcija, disjunkcija.

Definicija 5.12 — Logička varijabla, izraz (formula), literal. Za varijablu x_i kažemo da je *logička varijabla* ako može primiti samo dvije vrijednost: *tačno* i *netačno*. Ove vrijednosti se još označavaju sa 1 i 0.

Za R kažemo da je *logički izraz* ili *logička formula* ako je jednak jednom od slijedećih izraza:

(a) logičkoj varijabli;
(b) $R_1 \wedge R_2$;
(c) $R_1 \vee R_2$;
(d) $\overline{R_1}$,

pri čemu su R_1 i R_2 logički izrazi.

Logički izraz koji je jednak jednoj varijabli ili negaciji logičke varijable se naziva *literal*.

■ **Primjer 5.11** Posmatrajmo logičku formulu

$$\phi(x_1,x_2,x_3) = \phi = \neg(x_1 \wedge x_2 \vee \neg x_3) \wedge \neg x_1 \vee x_2 = \overline{(x_1 \wedge x_2 \vee \overline{x_3})} \wedge \overline{x_1} \vee x_2.$$

Ova formula ima tri varijable: x_1, x_2, x_3; te pet literala: $x_1, x_2, \overline{x_3}, \overline{x_1}, x_2$.

Važi:

$$\phi(0,1,0) = \overline{(0 \wedge 1 \vee \overline{0})} \wedge \overline{0} \vee 1 = \overline{0 \vee 1} \wedge 1 \vee 1 = \overline{1} \wedge 1 \vee 1 = 0 \wedge 1 \vee 1 = 0 \vee 1 = 1.$$

Mogli smo puno brže uočiti da je $\phi(0,1,0) = 1$ - kako?

Znači, postoje vrijednosti logičkih varijabli formule ϕ, tako da je $\phi = 1$. U tom slučaju kažemo da je ϕ *zadovoljiva* logička formula. ■

Definicija 5.13 — Zadovoljiva logička formula. Neka je $\phi(x_1,...,x_n)$ logička formula. Kažemo da je ϕ *zadovoljiva logička formula* ako postoje vrijednosti logičkih varijabli $x'_1,...,x'_n \in \{0,1\}$ tako da je

$$\phi(x'_1,...,x'_n) = 1.$$

Ako ovakve vrijednosti ne postoje, kažemo da je formula *nezadovoljiva*.

Uvedimo novi jezik, tj. novi problem:

$$SAT = \{(\phi) \mid \phi \text{ je zadovoljiva logička formula}\}.$$

SAT je skraćenica od *satisfiability*.

■ **Primjer 5.12** Formula ϕ iz primjera 5.11 je zadovoljiva logička formula. Formula $\phi_2 = x_1 \wedge \overline{x_1}$ nije zadovoljiva. Znači, $(\phi) \in SAT$ i $(\phi_2) \notin SAT$. ■

Svaka logička formula se može prikazati u jednostavnijem obliku.

Definicija 5.14 — Konjuktivna normalna forma. Neka je $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, pri čemu je C_i disjunkcija literala. Tada kažemo da je ϕ prikazana u *konjuktivnoj normalnoj formi* ili da je *cnf logička formula*. Izraz C_i se naziva *klauzula*.

Definicija 5.15 — 3-konjuktivna normalna forma. Neka je $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, pri čemu je C_i disjunkcija tačno tri literala. Tada kažemo da je ϕ prikazana u *3-konjuktivnoj normalnoj formi* ili da je *3-cnf logička formula*. Izraz C_i se naziva *klauzula*.

Slijedeću tvrdnju nećemo dokazivati.

Tvrdnja 5.7 Svaka logička formula ima odgovarajuću 3-konjuktivnu normalnu formu.

Uvedimo još nekoliko problema:

$$CNFSAT = \{(\phi) \mid \phi \text{ je zadovoljiva cnf logička formula}\};$$

$$3SAT = 3CNFSAT = \{(\phi) \mid \phi \text{ je zadovoljiva 3-cnf logička formula}\}.$$

■ **Primjer 5.13** Posmatrajmo 3-cnf logičku formulu:

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_4} \vee \overline{x_5}) \wedge (x_1 \vee x_3 \vee x_5).$$

Ova formula ima tri klauzule, pet varijabli i devet literala. Da bi ϕ bila zadovoljiva svaka od klauzula mora biti tačna. Ova je formula zadovoljiva, jer npr. imamo:

$$\phi(1,1,x_3,x_4,x_5) = (1 \vee \overline{1} \vee x_3) \wedge (1 \vee \overline{x_4} \vee \overline{x_5}) \wedge (1 \vee x_3 \vee x_5) = 1 \wedge 1 \wedge 1 = 1.$$

Znači, $(\phi) \in 3SAT$. ■

Na prvi pogled, da li logička formula pripada $3SAT$ (ili $CNFSAT$) je jednostavan problem – samo trebamo naći vrijednosti logičkih varijabli tako da je svaka klauzula tačna. Međutim, nije tako, kao što ćemo vidjeti kasnije.

Navedimo neka pravila za logičke izraze, koja ćemo koristiti kasnije:

1. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$;
2. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$;
3. $\overline{x \vee y} = \overline{x} \wedge \overline{y}$;
4. $\overline{x \wedge y} = \overline{x} \vee \overline{y}$.

Zadatak 5.2 — Za samostalan rad. Definiši 2-cnf logičku formulu i 2SAT. Dati polinomni algoritam koji rješava problem 2SAT, tj. dokazati da $2SAT \in P$. ■

5.5 Klasa NP

U ovoj skeciji posmatramo klasu jezika odlučivih u polinomnom vremenu pomoću nedeterminističke Turingove mašine.

Slično $TIME(t(n))$ uvodimo klasu $NTIME(t(n))$.

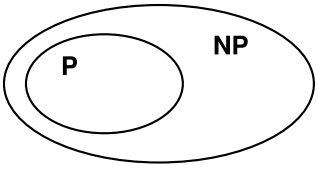
Definicija 5.16 — NTIME(t(n)). $NTIME(t(n)) = \{A \mid A$ je jezik odlučiv u vremenu $t(n)$ pomoću nedeterminističkog odlučivača$\}$.

Definicija 5.17 — Klasa NP. Klasa NP je klasa jezika odlučivih u polinomnom vremenu pomoću nedeterminističkog odlučivača, tj.

$$NP = \bigcup_{k=0}^{+\infty} NTIME(n^k).$$

Skraćenica NP dolazi od *nondeterministic polynomial*.

Očigledno je $P \subseteq NP$ (slika 5.4). Da li je $P \neq NP$ ili je $P = NP$ je još uvijek neriješen problem. Ovo je jedan od najpoznatijih problema u kompjuterskoj nauci. Većina naučnika vjeruje da je $P \neq NP$

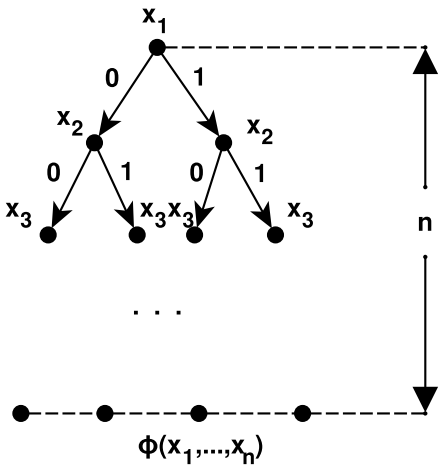


Slika 5.4: $P \subseteq NP$.

■ **Primjer 5.14** Dokažimo da je $(SAT) \in NP$. Dovoljno je konstruisati nedeterminističku Turingovu mašinu koja odlučuje *SAT* u polinomnom vremenu (algoritam 5.5 i slika 5.5).

Algoritam 5.5 Nedeterministički polinomni odlučivač za SAT.

```
1: procedure SAT(φ, x1, ..., xn)
2:     for i = 1; i ≤ n; i++ do
3:         (nedeterministički) granaj na dva slučaja: xi = 0 i xi = 1;
4:     (za ovu granu kompjutacije) return φ(x1, ..., xn)
5:     if neka grana kompjutacije vraća accept then
6:         return (accept);
7:     else
8:         return (reject);
```



Slika 5.5: Nedeterministička Turingova mašina za *SAT*.

Pošto u svakom koraku ispitujemo po jednu varijablu x_i i pošto imamo n varijabli, dubina stabla kompjutacije je jednaka n, pa se nedeterministička Turingova mašina izvršava u vremenu $O(n)$. Dobijamo $(SAT) \in NP$. ■

U prethodnom primjeru, "(nedeterministički) granaj na dva slučaja: $x_i = 0$ i $x_i = 1$" znači da se NTM grana na dvije mašine, pri čemu jedna uzima $x_i = 0$,

a druga $x_i = 1$.

K Da li $(SAT) \in P$, tj. da li se SAT može riješiti u polinomnom vremenu pomoću "obične" Turingove mašine je još uvijek neriješen problem. Naučnici ne znaju da li se može ili ne može riješiti, ali većina vjeruje da ne može.

U većini slučajeva, ideja koja stoji iza nedeterminističke Turingove mašine je *brute force* pristup. Prema tome, u većini slučajeva, nije teško dokazati da je problem iz NP, ako to zaista i jeste. Postoji još jednostavniji način da se dokaže da je problem iz NP, a to je pomoću *polinomnih verifikatora*.

Definicija 5.18 — Verifikator. *Verifikator* za jezik A je odlučivač V za koji važi:

$$A = \{\omega \mid V(\omega, c) = accept \text{ za neki string } c\}.$$

Ako se V izvršava u polinomnom vremenu u odnosu na $|\omega|$, nazivamo ga *polinomni verifikator* i za A kažemo da je *polinomno verifikabilan*.

String c se naziva *certifikat* ili *dokaz*.

Neka je

$$CLIQUE = \{(G,k) \mid G \text{ je graf sa klikom veličine } k\}.$$

Klika veličine k je podgraf K od G sa k čvorova, tako da je svaki čvor od K povezan sa svim ostalim čvorovima od K.

■ **Primjer 5.15** Odredimo polinomni verifikator za $CLIQUE$.

Šta bi bio string c? To bi bila neka struktura koja bi nas ubijedila da $(G,k) \in CLIQUE$, tj da graf G ima kliku veličine k. Jedan od mogućih kandidata za certifikat c je i sama klika od G veličine k.

Znači, neka je c klika od G veličine K. Konstruišimo odlučivač V (algoritam 5.6).

Procedura V vraća *accept* akko je c klika od G reda k, tj. akko $(G,k) \in CLIQUE$, pa važi:

$$CLIQUE = \{(G,k) \mid V(G,k,c) = accept\},$$

što znači da je V verifikator od $CLIQUE$.

Dokažimo da je V polinomni verifikator. Algoritam V ima dvije petlje: jedna sa k iteracija, a druga sa $(k-1)+(k-2)+\ldots+2+1 = \frac{k(k-1)}{2} \in O(k^2)$, pa je V polinomni verifikator. ■

Algoritam 5.6 Polinomni verifikator za CLIQUE.

```
1:  procedure V(G,k,c)
2:      n' ← broj čvorova od c;
3:      if n' ≠ k then
4:          return (reject);
5:      v_i ← čvorovi od c (i = 1,...,k);
6:      for i = 1; i ≤ k; i++ do
7:          if v_i nije čvor od G then
8:              return (reject);
9:      for i = 1; i ≤ k; i++ do
10:         for j = i+1; j ≤ k; j++ do
11:             if v_i i v_j nisu spojeni u G then
12:                 return (reject);
13:     return (accept);
```

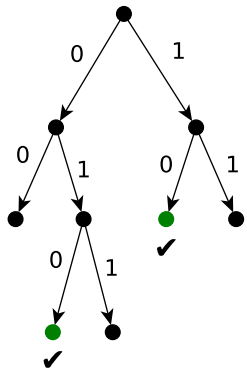
Problem koji odgovara jeziku *CLIQUE* je: "Da li dati graf G ima kliku reda k"? Rješenje ovog problema je konkretan primjer klike od G reda k. Upravo to nam je bio certifikat. U većini slučajeva certifikat će upravo biti rješenje problema.

Teorema 5.1 Jezik je u NP akko ima polinomni verifikator.

Dokaz. ***(a)*** Pretpostavimo da je $A \in NP$. Dokažimo da A ima polinomni verifikator. Neka je M polinomni nedeterministički odlučivač za A. Tada T, stablo kompjutacije od M, ima polinomnu visinu. Ako T nije binarno stablo, tada ga možemo pretvoriti u binarno i visina će ostati polinomna (vidi dokaz tvrdnje 5.5 i sliku 5.2).

Neka je $\omega \in A$. Tada je $D(\omega) = accept$ i neka c označava binarni put u stablu kompjutacije od korijena stabla do prihvatnog stanja (slika 5.6). Sa V označimo Turingovu mašinu koja kao ulaz prima (ω, c), te koristeći c prolazi kroz stablo kompjutacije do prihvatnog stanja, tj. simulira rad M, ali uzimajući samo one grane definisane putem c. Pošto stablo kompjutacije ima polinomnu visinu, to je put kojim prolazi V polinomne dužine, tj. V je polinomni algoritam po $|\omega|$.

(b) Pretpostavimo da jezik A ima polinomni verifikator V, koji se izvršava u vremenu $t(n) \in O(n^k)$. Pošto dužina ulaznog stringa ne može biti veća od



Slika 5.6: Binarno stablo kompjutacije za nedeterminističku Turingovu mašinu sa dva prihvatna stanja. Prvom prihvatnom stanju možemo pridružiti certifikat $c_1 = 010$, a drugom $c_2 = 10$.

vremena izvršavanja algoritma, dužina certifikata ne može biti veća od $t(n)$.

Dokažimo da $A \in NP$. Sa M označimo nedeterminističku Turingovu mašinu koju ćemo konstruisati (algoritam 5.7). Mašina M počinje od praznog stringa c, te u svakom koraku iteracije na kraj stringa c nedeterministički dodaje 0 ili 1. Ako uspije naći string c, koji predstavlja certifikat, vraća *accept*. U suprotnom, vraća *reject*.

Algoritam 5.7

```
1: procedure M(ω)
2:     n ← |ω|;
3:     c ← ε;
4:     for i = 1; i ≤ t(n); i++ do
5:         c ← c + nedeterministički 0 ili 1;
6:         if V(ω, c) == accept then
7:             return accept;
8:     return reject;
```

Pošto je V polinomni verifikator za A i pošto dužina certifikata nije veća od $t(n)$, algoritam 5.7 će ispitati sve moguće certifikate c. Prema tome, $M(\omega) = accept \iff$ postoji c tako da je $V(\omega, c) = accept \iff \omega \in A$, pa M prepoznaje A.

Dubina stabla kompjutacije mašine M je jednaka broju iteracija *for* petlje i iznosi $t(n)$. U svakom čvoru se izvršava verifikator V, a njegovo vrijeme izvršavanja je $t(n)$, pa je ukupno vrijeme izvršavanja mašine M jednako $t(n)^2 \in O(n^{2k})$, pa $A \in NP$, što je i trebalo dokazati. ■

U praksi, iz prethodne tvrdnje, imamo da se problem može riješiti u polinomnom vremenu pomoću nedeterminističke Turingove mašine akko možemo provjeriti rješenje u polinomnom vremenu pomoću "obične", determinističke Turingove mašine.

Pošto smo našli polinomni certifikat (primjer 5.15), imamo $CLIQUE \in NP$.

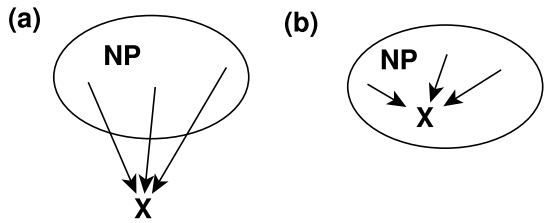
5.6 Cook-Levinova teorema

Postoji ogroman skup problema za niko nije uspio naći polinomni algoritam. Među tima problemima su: SAT, $CNFSAT$, $3SAT$, $CLIQUE$, itd. Naučnici su odlučili, ako već nemogu da ih riješe, da ih klasificiraju po težini.

Kada će problem A biti lakši od problema B. Taj odgovor smo dali ranije: ako važi $A \leq_P B$, tj. ako se A u polinomnom vremenu može reducirati na B. To bi značilo da ako dobijemo brzi algoritam za B da odmah imamo i brzi algoritam za A.

Da li postoji problem X na koji se mogu polinomno reducirati svi problemi iz NP (slika 5.7a)? Tj. da li postoji problem iz slijedeće definicije?

Definicija 5.19 — NP-težak jezik. Neka je X jezik na koji se u polinomnom vremenu može reducirati svaki jezik iz NP. Tada za X kažemo da je NP-težak problem.



Slika 5.7: **(a)** Jezik X je NP-težak, ako se svaki jezik iz NP može polinomno reducirati na X. **(b)** Jezik X je NP-kompletan ako je NP-težak i ako pripada NP.

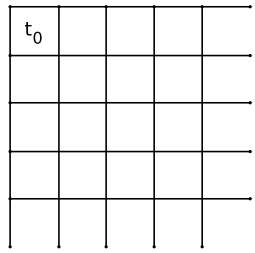
Ako bi ovakav problem X postojao, tada bi važilo $A \leq_P X, \forall A \in NP$, pa bi X bio neka vrsta gornje granice za NP. Takođe, egzistencija polinomnog algoritma za X, bi značilo da se svaki problem iz NP može riješiti u polinomnom vremenu.

Definicija 5.20 — NP-kompletan jezik. Za jezik X kažemo da je *NP-kompletan*, ako:
(a) X je NP-težak;
(b) $X \in NP$.

Skraćenica za NP-kompletan je NPC (*NP-complete*). Opet, postavlja se pitanje egzistencije ovakvog jezika. Odgovor na to daje Cook-Levinova teorema.

Teorema 5.2 — Cook-Levin. SAT je NP-kompletan problem.

Prije nego dokažemo Cook-Levinovu teorema, dokažimo NP-kompletnost jednog drugog problema - problema popločavanja.



Slika 5.8: Problem popločavanja.

Uzmimo da imamo dio ravni ograničen sa dvije okomite prave, nazovimo ga *kvadrant* i posmatrajmo mrežu sa cjelobrojnim koordinatama (slika 5.8). Dato nam je konačno mnogo vrsta pločica (skup T), a svaka vrsta ima beskonačno mnogo primjeraka. Tačno određena počica se mora postaviti u početno polje (pločica t_0). Samo određeni parovi pločica se mogu dodirivati horizontalno (parovi iz skupa H) i samo određeni parovi pločica se mogu dodirivati vertikalno (parovi iz skupa V). Problem se sastoji u određivanju popločavanja tako da su navedeni uslovi zadovoljeni.

U nastavku dajemo formalnu definiciju problema.

Definicija 5.21 — Problem popločavanja. Sistem popločavanja je uređena četvorka $\mathfrak{T} = (T, t_0, H, V)$, pri čemu je T konačan skup, $t_0 \in T$, $H, V \subseteq T \times T$. *Popločavanje sa* $\mathfrak{T}$ je preslikavanje $f : \mathbb{N}_0 \times \mathbb{N}_0 \to T$ tako da važi:
(*a*) $f(0,0) = t_0$;
(*b*) $(f(m,n), f(m,n+1)) \in H, \forall m,n \in \mathbb{N}_0$;
(*c*) $(f(m,n), f(m+1,n)) \in V, \forall m,n \in \mathbb{N}_0$.
Problem popločavanja je problem određivanja preslikavanja f za dato $\mathfrak{T}$.

Problem *ograničenog popločavanja* je popločavanje ograničenog kvadrata, umjesto kvadranta. Umjesto specificiranja samo prve pločice (t_0) u ograničenom problemu specificiramo cijelu prvu vrstu.

Definicija 5.22 — Problem ograničenog popločavanja. Sistem ograničenog popločavanja je uređena četvorka $\mathfrak{T} = (T, f_0, H, V)$, pri čemu je T konačan skup, $f_0 : \{0, 1, ..., s-1\} \to T$, $s \in \mathbb{N}$, $H, V \subseteq T \times T$. *Popločavanje sa* $\mathfrak{T}$ je preslikavanje $f : \mathbb{N}_0 \times \mathbb{N}_0 \to T$ tako da važi:
(*a*) $f(0,m) = f_0(m), \forall m < s$;
(*b*) $(f(m,n), f(m,n+1)) \in H, \forall m < s, n < s-1$;
(*c*) $(f(m,n), f(m+1,n)) \in V, \forall m < s-1, n < s$.
Problem ograničenog popločavanja je problem određivanja preslikavanja f za dato $\mathfrak{T}$.

Tvrdnja 5.8 Problem ograničenog popločavanja je NP-kompletan.

Dokaz. Problem je očigledno u NP. Certifikat je konkretno riješenje: popločavanje svih s^2 kvadrata.

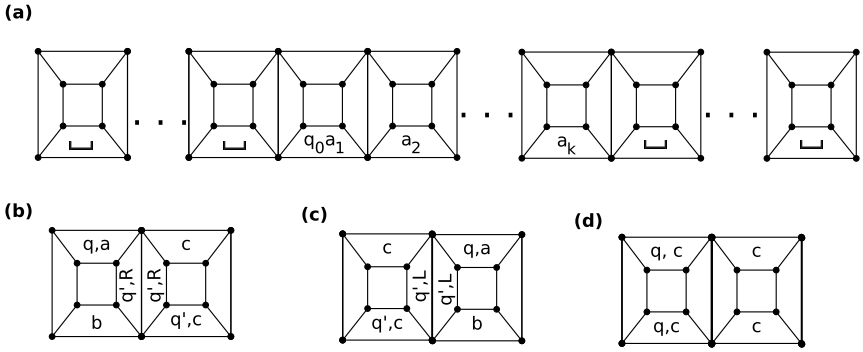
Dokažimo da se svaki jezik iz NP može pomoću polinomne redukcije svesti na problem ograničenog popločavanja.

Neka je M nedeterministička polinomna Turingova mašina sa vremenom izvršavanja $p(k)$. Uzmimo $x = a_1 a_2 ... a_k \in L(M)$, $n = 2p(k) + 1$ i q_0 početno stanje automata M.

Svaka vrsta simulira računanje mašine M kako je opisano na slici 5.9. Po vertikali, pločice se slažu samo ako dijelovi koji se dodiruju imaju identične simbole. Za ovako definisano popločavanje imamo da $M(x) = accept$ akko postoji ograničeno popločavanje.

Data redukcija je očigledno polinomna. Znači, svaki problem iz NP se može reducirati na problem popločavanja.

Iz svega imamo da je problem popločavanja NP-kompletan problem. ■



Slika 5.9: Popločavanje pridruženo nedeterminističkoj Turingovoj mašini. **(a)** Ukupno $n = 2p(k)+1$ pločica. Niz počinje sa $p(k)$ pločica sa praznim mjestom, a nastavlja sa $p(k)+1$ pločica koje sadrže: $q_0a_1, a_2, ..., a_k, \sqcup, ..., \sqcup$. **(b)** Primjer horizontalnog popločavanja za $(q', b, R) \in \delta(q, a)$. **(c)** Horizontalno popločavanje za $(q', b, L) \in \delta(q, a)$. **(d)** Popločavanje za dio trake koji se ne mijenja.

.

K Popločavanje opisano u dokazu prethodne teoreme se naziva *Wangovo popločavanje*, po matematičaru Hao Wangu.

Dokaz Cook-Levinove teoreme. Daćemo polinomnu redukciju sa problema ograničenog popločavanja na SAT problem. SAT je očigledno u NP, jer ima polinomni certifikat. Kao u skoro svim slulajevima, certifikat može biti konkretno rješenje: pridruživanje vrijednosti logičkim varijablama koji produciraju tačnu formulu. U ispravnost certifikata se možemo uvjeriti u polinomnom vremenu jednostavnim uvrštavanjem u logičku formulu.

Neka je dato: skup vrsta pločica T, broj n, prva vrsta popločavanja $t_1, ..., t_n$. Uvodimo logičke varijable $x_{k,l,t}$ za $k, l \in \{1, ..., n\}$ i $t \in T$. Imaćemo $x_{k,l,t} = 1$ akko se pločica t nalazi na poziciji (k, l). Logičku formulu Φ ćemo konstruisati na slijedeći način:

$$\Phi = \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4.$$

Formulu Φ_1 formiramo iz prve vrste $t_1, ..., t_n$:

$$\Phi_1 = \bigwedge_{k=1}^{n} x_{k,1,t_k}.$$

Iz uslova da na svakoj poziciji mora biti jedna pločica imamo:

$$\Phi_2 = \bigwedge_{k=1}^{n} \bigwedge_{l=1}^{n} \bigwedge_{t \neq t'} \overline{(x_{k,l,t} \wedge x_{k,l,t'})}.$$

Pločice susjedne po horizontali se moraju slagati:

$$\Phi_3 = \bigwedge_{k=1}^{n-1} \bigwedge_{l=1}^{n} \bigvee_{tt'} (x_{k,l,t} \wedge x_{(k+1),l,t'}).$$

Za vertikalno slaganje imamo:

$$\Phi_4 = \bigwedge_{k=1}^{n} \bigwedge_{l=1}^{n-1} \bigvee_{\substack{t \\ t'}} (x_{k,l,t} \wedge x_{k,(l+1),t'}).$$

Važi: Φ je zadovoljiva formula akko postoji ograničeno popločavanje.

Veličina formule Φ je polinomna po n, pa smo izvršili polinomnu redukciju.

Iz svega imamo da je SAT NP-kompletan problem. ■

5.7 Osobine i primjeri NP-kompletnih problema

Koristeći Cook-Levinovu teoremu i slijedeću tvrdnju, možemo dokazati NP-kompletnost za niz problema.

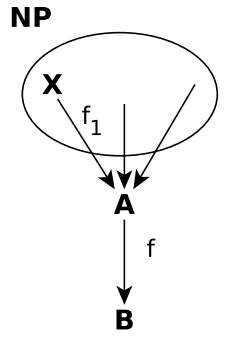
> **Tvrdnja 5.9** Neka je A NP-težak problem i $A \leq_P B$. Tada je B NP-težak problem.

Dokaz. Neka je $f : \Sigma^* \to \Sigma^*$ polinomna redukcija sa A na B. Tada za svaki string ω' važi:

$$\omega' \in A \iff f(\omega') \in B. \tag{5.3}$$

Neka je X proizvoljan jezik iz NP (slika 5.10). Pošto je A NP-težak, postoji polinomna redukcija f_1 sa X na A, pa važi:

$$\omega \in X \iff f_1(\omega) \in A. \tag{5.4}$$



Slika 5.10: Ako se A može reducirati na B i proizvoljno $X \in NP$ može reducirati na A, tada se X može reducirati na B kompozicijom redukcija.

Iz 5.3 i 5.4, stavljajući $\omega' = f_1(\omega)$, za proizvoljan string ω imamo:

$$\omega \in X \iff f(f_1(\omega)) \in B.$$

Znači, $f \circ f_1$ je redukcija sa X na B. Pošto su f i f_1 polinomno izračunljive funkcije, to je i njihova kompozicija $f \circ f_1$ polinomno izračunljiva.

Dobijamo da postoji polinomna redukcija sa proizvoljnog jezika iz NP na jezik B, pa je B NP-težak problem. ■

Prethodna tvrdnja nam omogućava da dokažemo slijedeću tvrdnju.

Tvrdnja 5.10 CNFSAT je NP-kompletan problem.

Dokaz. Dokažimo prvo da je CNFSAT NP-težak problem. Konstruišimo polinomnu redukcija sa SAT. Preciznije, svakoj logičkoj formuli ϕ pridružićemo cnf logičku formulu ϕ' tako da važi $(\phi) \in SAT \iff (\phi') \in CNFSAT$.

Koristićemo matematičku indukciju po dužini logičke formule ϕ. Pod "dužinom" od ϕ podrazumijevamo ukupan broj literala i logičkih operatora u ϕ.

Neka je $\phi = x_1'$, pri čemu je x_1' literal. Izraz ϕ je već u cnf obliku, pri čemu ima tačno jednu klauzulu.

Pretpostavimo da tvrdnja važi za sve logičke formule dužine manje ili jednako k.

Dokažimo da tvrdnja važi i za logičke formule dužine $k+1$. Imamo da je $\phi = \phi_1 \wedge \phi_2$ ili $\phi = \phi_1 \vee \phi_2$ ili $\phi = \overline{\phi_1 \wedge \phi_2}$ ili $\phi = \overline{\phi_1 \vee \phi_2}$. Iz induktivne prepostavke, logičkim formulama ϕ_1 i ϕ_2 možemo pridružiti cnf logičke formule: $\phi_1' = A_1 \wedge \ldots \wedge A_s$ i $\phi_2' = B_1 \wedge \ldots \wedge B_t$ tako da $(\phi_i) \in SAT \iff (\phi_i') \in CNFSAT$ $(i = 1,2)$. Možemo uzeti da logičke formule ϕ_1' i ϕ_2' imaju različite varijable (obrazložiti - za samostalan rad).

Slučaj 1. Neka je $\phi = \phi_1 \wedge \phi_2$. Uzmimo

$$\phi' = \phi_1' \wedge \phi_2' = A_1 \wedge \ldots \wedge A_s \wedge B_1 \wedge \ldots \wedge B_t.$$

Vidimo da je ϕ' cnf logička formula.

Važi:

- $(\phi) \in SAT \iff$
- $(\phi_1), (\phi_2) \in SAT \iff$ (na osnovu induktivne pretpostavke)
- $(\phi_1'), (\phi_2') \in CNFSAT \iff$ (pošto ϕ_1' i ϕ_2' imaju različite varijable)
- $\phi' = \phi_1' \wedge \phi_2' \in CNFSAT$,

što je i trebalo dokazati.

Slučaj 2. Neka je $\phi = \phi_1 \vee \phi_2$. Uzmimo

$$\phi' = \phi_1' \vee \phi_2' = (A_1 \wedge \ldots \wedge A_s) \vee (B_1 \wedge \ldots \wedge B_t) = \bigwedge_{\substack{1 \le i \le s; \\ 1 \le j \le t}} (A_i \vee B_j).$$

Vidimo da je ϕ' cnf logička formula.

Važi:

- $(\phi) \in SAT \iff$
- $(\phi_1) \in SAT$ ili $(\phi_2) \in SAT \iff$ (na osnovu induktivne pretpostavke)
- $(\phi_1') \in CNFSAT$ ili $(\phi_2') \in CNFSAT \iff$
- $\phi' = \phi_1' \vee \phi_2' \in CNFSAT$,

što je i trebalo dokazati.

Slučaj 3. Neka je $\phi = \overline{\phi_1 \wedge \phi_2}$ ili $\phi = \overline{\phi_1 \vee \phi_2}$. Pomoću $\phi = \overline{\phi_1 \wedge \phi_2} = \overline{\phi_1} \vee \overline{\phi_2}$ ili $\phi = \overline{\phi_1 \vee \phi_2} = \overline{\phi_1} \wedge \overline{\phi_2}$ ovaj slučaj se svodi na neki od prethodna dva slučaja.

Dokažimo da je prethodna redukcija polinomna. Dokaz ćemo opet sprovesti matematičkom indukcijom. Neka je $n_1 = |\phi_1|$ i $n_2 = |\phi_2|$ i pretpostavimo da ϕ_i' možemo dobiti iz ϕ_i u najviše $P(n_i)$ koraka, pri čemu je $P(x) = x^k$ i $k \ge 2$ $(i = 1,2)$.

Uočimo da važi $P(n_1) + P(n_2) + n_1 n_2 \le P(n_1 + n_2)$.

U prvom slučaju, za dobijanje ϕ', nam je potrebno najviše $P(n_1)+P(n_2) \leq P(n_1+n_2) = P(n)$ koraka.

U drugom slučaju, pošto je $s \leq n_1$ i $t \leq n_2$, potrebno je najviše $P(n_1) + P(n_2) + n_1 n_2 \leq P(n_1+n_2) = P(n)$ koraka.

Treći slučaj se svodi na jedan od prva dva slučaja.

Izvršili smo polinomnu redukciju sa SAT na CNFSAT. Pošto je SAT NP-težak problem, to je i CNFSAT NP-težak problem.

Dokažimo da $CNFSAT \in NP$. Dovoljno je naći polinomni certifikat. Neka su $x'_1, ..., x'_n$ vrijednosti logičkih varijabli za koje je $\phi = 1$. Da li možemo provjeriti ovu činjenicu u polinomnom vremenu? To možemo postići jednostavnim uvrštavanjem vrijednosti varijabli i računanjem vrijednosti logičke formule u $O(|\phi|)$ vremenu.

Pošto je $CNFSAT \in NP$ i pošto je NP-težak, to je i NP-kompletan. ■

Tvrdnja 5.11 3SAT je NP-kompletan problem.

Dokaz. Prvo izvršimo polinomnu redukciju sa CNFSAT na 3SAT. Neka je $\phi = C_1 \wedge ... \wedge C_m$ cnf formula.

Svaku klauzulu ćemo reducirati na konjukciju klauzula sa po tačno tri literala. Posmatrajmo slučajeve.

Slučaj 1. $C_i = x_1^i$, tj. klauzula C_i ima tačno jedan literal. Tada važi:

$$\begin{aligned} C_1 &= x_1^i \vee 0 \vee 0 = x_1^i \vee (y_1 \wedge \overline{y_1}) \vee (y_2 \wedge \overline{y_2}) = \\ &= (x_1^i \vee y_1 \vee y_2) \wedge (x_1^i \vee \overline{y_1} \vee y_2) \wedge (x_1^i \vee y_1 \vee \overline{y_2}) \wedge (x_1^i \vee \overline{y_1} \vee \overline{y_2}). \end{aligned} \tag{5.5}$$

Slučaj 2. $C_i = x_1^i \vee x_2^i$, tj. klauzula C_i ima tačno dva literala. Tada važi:

$$C_i = x_1^i \vee x_2^i = x_1^i \vee x_2^i \vee 0 = x_1^i \vee x_2^i \vee (y_1 \wedge \overline{y_1}) = (x_1^i \vee x_2^i \vee y_1) \wedge (x_1^i \vee x_2^i \vee \overline{y_1}). \tag{5.6}$$

Slučaj 3. $C_i = x_1^i \vee x_2^i \vee x_3^i$, tj. klauzula C_i ima tačno tri literala. Tada nemamo šta raditi.

Slučaj 4. $C_i = x_1^i \vee ... \vee x_{k_i}^i$ $(k_i \geq 4)$. Konstruišimo novi logički izraz: $C'_i = (x_1^i \vee x_2^i \vee y_1) \wedge (\overline{y_1} \vee x_3^i \vee y_2) \wedge (\overline{y_2} \vee x_4^i \vee y_3) \wedge ... \wedge (\overline{y_{k_i-4}} \vee x_{k_i-2}^i \vee y_{k_i-3}) \wedge (\overline{y_{k_i-3}} \vee x_{k_i-1}^i \vee x_{k_i}^i)$.

Tada važi: C_i je zadovoljiva logička formula akko C_i' zadovoljiva logička formula (dokazati za samostalan rad).

U svakom od slučajeva, klauzulu C_i smo sveli na konjukciju klauzula sa po tačno tri literala. U prva tri slučaja, to smo uradili u konstantnom broju koraka. U četvrtom slučaju, to smo uradili u $O(|C_i|)$ koraka.

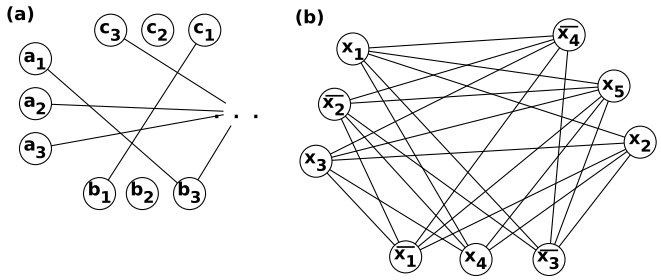
Znači, logičku formulu $\phi = C_1 \wedge ... \wedge C_m$ možemo svesti na 3cnf formulu ϕ' u $O(|\phi|)$ koraka, pri čemu važi da je ϕ zadovoljiva logička formula akko je ϕ' zadovoljiva logička formula, tj. $\phi \in CNFSAT \iff \phi' \in 3SAT$. Zaključujemo da je $3SAT$ NP-težak problem.

Dokažimo da $3SAT \in NP$. Certifikat za $\phi' = \phi'(x_1, x_2, ..., x_n)$ bi bile konkretne vrijednosti za $x_1, ..., x_n$ za koje je $\phi' = 1$. Da je $\phi' = 1$ možemo se uvjeriti u linearnom vremenu jednostavnim uvrštavanjem vrijednosti $x_1, ..., x_n$ u ϕ' i računanjem njegove logičke vrijednosti.

Dobijamo da je $3SAT$ NP-kompletan problem. ■

Tvrdnja 5.12 CLIQUE je NP-kompletan problem.

Dokaz. Izvršićemo redukciju sa $3SAT$. Neka je $\phi = (a_1 \vee a_2 \vee a_3) \wedge (b_1 \vee b_2 \vee b_3) \wedge ... \wedge (c_1 \vee c_2 \vee c_3)$ 3cnf formula sa m klauzula, pri čemu su $a_i, b_i, ..., c_i$ $(i = 1, 2, 3)$ literali. Ovoj formuli možemo pridružiti graf G (slika 5.11)



Slika 5.11: **(a)** Graf pridružen 3cnf formuli. **(b)** Graf pridružen formuli $\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_3}) \wedge (x_2 \vee x_5 \vee \overline{x_4})$.

Svakom literalu pridružimo po tačno jedan čvoru u grafu (ukupno $3m$ čvorova). Ne povezujemo čvorove iz iste klauzule i ne povezujemo čvorove sa suprotnim literalima (tj. ne povezujemo x_i sa $\overline{x_i}$). Grane možemo konstruisati u $O(m^2)$ koraka.

Dokažimo da je ϕ zadovoljiva akko G ima m-kliku.

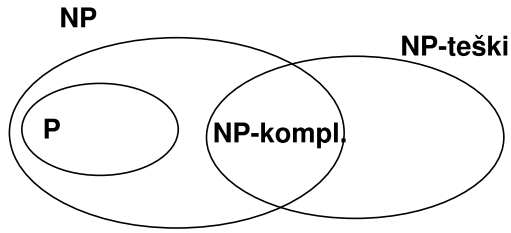
Neka je ϕ zadovoljiva formula. Tada postoje vrijednosti varijabli, tako da je svaka od m klauzula tačna. Tj. u svakoj klauzuli možemo izabrati po jedan literal koji je tačan: $a', b' ..., c'$. Ovih literala ima m. Pošto je svaki od njih tačan, nikoja dva nisu suprotna, tj. jedan nije negacija od drugog. Zbog ovoga i činjenice da pripadaju različitim klauzulama, imamo da su svaka sva od m literala $a', b', ..., c'$ povezana, tj. imamo m-kliku.

Neka sada graf G ima m-kliku, tj. postoji m čvorova $v_1, v_2, ..., v_m$ tako da su svaka dva povezana. Zbog pravila povezivanja, nikoja dva od ovih čvorova ne pripadaju istoj klauzuli. Znači, iz svake klauzule uzeli smo po tačno jedan čvor. Pošto su svi čvorovi povezani, nijedan nije negacija drugog, pa nema prepreka da uzmemo da su svi literali $v_1, v_2, ..., v_n$ tačni. Na ovaj način se osiguravamo da svaka klauzula ima po barem jedan literal tačan, tj. da svaka klauzula bude tačna, pa je $\phi = 1$.

Znači imamo $\phi \in 3SAT$ akko $(G, m) \in CLIQUE$.

Gornju redukciju smo izvršili u $O(m^2)$ koraka, pa se radi po polinomnoj redukciji. Dobijamo da je CLIQUE NP-težak problem.

Iz primjera 5.15 imamo da CLIQUE ima polinomni verifikator, pa $CLIQUE \in NP$ (teorema 5.1). Pošto je CLIQUE NP-težak problem i $CLIQUE \in NP$, to je CLIQUE NP-kompletan problem. ■



Slika 5.12: Odnos između klasa kompleksnosti pod pretpostavkom da je $P \neq NP$.

Slika 5.12 opisuje odnos između klasa kompleksnosti koje smo do sada radili. Pod pretpostavkom da je $P \neq NP$, ne postoji NP-težak problem koji je u P. Kada bi se dokazalo da je barem jedan NP-kompletan problem u P, tada bi direktno slijedilo $P = NP$. Postoje problemi koji su NP-teški, ali nisu u NP.

Takvi problemi su još teži od NP-kompletnih problema, jer nisu u NP, tj. ne mogu se riješiti u polinomnom vremenu pomoću nedeterminističke Turingove mašine. Jedan takav problem je $HALT_{TM}$, za kojeg znamo da je neodlučiv, tj. ne može se riješiti u konačno mnogo koraka. Problemi A_{TM} i $HALT_{TM}$ su NP-teški, ali nisu u NP, tj. nisu NP-kompletni.

K Naučnici su pokušavali naći inteligentno konstruisane (polinomne) algoritme za mnoge probleme. U neuspjehu su pokušali drugi pristup; da ih rangiraju po težini. Jedan od načina da se rangiraju po težini je pomoću polinomne redukcije. Problem B je teži ili jednak problemu A ako postoji polinomna redukcija sa A na B, tj. $A \leq_P B$. Na taj način su dobili klasu NP-kompletnih problema, kojoj pripada većina praktičnih problema.

Otvoreno je pitanje da li je $P \neq NP$, što je poznato kao *P vs NP problem*. Većina naučnika vjeruje da je zaista $P \neq NP$, tj. vjeruje da za NP-kompletne probleme ne postoji polinomni algoritam.

Praktični značaj ove teorije je da ako rješavate neki problem i dokažete da je NP-kompletan, onda možete odustati od traženja tačnog polinomnog algoritma, jer niko od naučnika ga nije uspio naći. To je razlog zašto se većina problema rješava heuristikom.

Ako se jednog dana dokaže da za neki NP-kompletan problem ne postoji polinomni algoritam, polinomnog redukcijom dobijamo da ni za jedan NP-kompletan problem ne postoji polinomni algoritam i $P \neq NP$. Ako se slučajno dokaže suprotno, da se neki NP-kompletan problem može riješiti u polinomnom vremenu, tada bi vrijedilo $P = NP$.

K Mi smo spomenuli tek nekoliko klasa kompleksnosti: P, NP, NPC. U trenutku pisanja ove knjige, registrovano je 545 klasa kompleksnosti. Ako želite saznati nešto više, proguglajte "complexity zoo".

K NP-kompletni problemi nisu najteži problemi koji postoje. Kao što smo već spomenuli, neodlučivi i Turing-neprepoznatljivi jezici su još teži, jer se ne mogu riješiti u konačno mnogo koraka.

Postoje klase problema, koji se mogu riješiti u konačno mnogo koraka, ali su teži od NP-kompletnih problema. Npr. EXPTIME-kompletni ili EXPSPACE-kompletni problemi se ne mogu riješiti u polinomnom vremenu - ovo je dokazana činjenica.

Jedan primjer EXPTIME-kompletnog problema je redukovana verzija Halting problema: "*Da li se data Turingova mašina na datom ulaznom stringu zaustavlja u najviše k koraka*". Ovaj problem se može riješiti u eksponencijalnom vremenu (pa pripada EXPTIME). Štaviše, ovaj problem je EXPTIME-kompletan, pa znamo da se ne može riješiti u polinomnom vremenu.

Razni problemi bazirani na igrama su EXPTIME-kompletni ili EXPSPACE-kompletni. Npr. igra *dame* (eng. *checkers*) ili $n \times n$ šah su EXPTIME-kompletni problemi.

5.8 Zadaci za samostalan rad

Zadatak 5.3 Dokaži da je klasa P zatvorena pod unijom, presjekom, nadovezivanjem, komplementom i zvjezdicom. ■

Zadatak 5.4 Dokaži da su slijedeći problemi NP-kompletni:

(a) VERTEX-COVER$=\{(G,k) \mid G$ je graf koji ima pokrivač veličine $k\}$[a];

(b) HAMILTON$=\{(G) \mid G$ je Hamiltonov graf$\}$[b];

(c) 3COLOR$=\{(G) \mid G$ je graf čije čvorove možemo obojiti sa tri boje$\}$[c];

(d) TSP$=\{(G,k) \mid G$ ima Hamiltonov ciklus dužine $\leq k\}$[d];

(e) KNAPSACK$=\{(S,k) \mid S$ je skup prirodnih brojeva koji ima podskup čiji je zbir jednak $k\}$.

■

[a] Pokrivač grafa G je skup čvorova S takav da je svak grana od G incidentna sa barem jednim čvorom iz S.

[b] Graf je Hamiltonov ako ima Hamiltonov ciklus, tj. ciklus koji sadrži sve čvorove grafa.

[c] Čvorovi iste boje nisu susjedni.

[d] *TSP - Traveling Salesman Problem.*

Zadatak 5.5 Dokaži da slijedeći problemi nisu u NP:

(a) A_{TM};

(b) $HALT_{TM}$.

■

Indeks

A_{CFG}, 123
A_{DFA}, 119
A_{NFA}, 119
A_{PDA}, 148
A_{REX}, 121
EQ_{DFA}, 122
EQ_{TM}, 133
E_{DFA}, 121
$HALT_{TM}$, 124
$REGULAR_{TM}$, 132
Σ^*, 10, 16
Σ_ε, 10
$\aleph_0$, 7
$*$ operator, 13
ε, 10
$\mathbb{N}$, 1
$\mathbb{N}_0$, 1
$\mathbb{Q}$, 1
$\mathbb{R}$, 1
$\mathbb{Z}$, 1
+ operator, 16
c, 9
2-potisni automat, 89
3-CNF, 166
3-konjuktivna normalna forma, 166
3-potisni automat, 89

alfabet, 10
algoritam, 101
asimptotska notacija, 149
automat
    2-potisni, 89
    3-potisni, 89
    deterministički konačni automat, 19
    DFA, 19
    enumerator, 138

k-PDA, 107
k-potisni, 88
k-potisni automat, 107
komplement DFA, 29
konfiguracija Turingove mašine, 98
LBA, 144
linearni ograničeni, 144
nadovezivanje NFA, 43
nedeterm. Turingova mašina, 105
nedeterministički konačni automat, 29
NFA, 29
odlučivač, 99
PDA, 71
potisni, 57
presjek DFA, 27
računanje Turingove mašine, 98
registarska mašina, 147
TM, 96
totalna Turingova mašina, 99
Turingova mašina, 95, 97
Turingova mašina sa dvostruko beskonačnom trakom, 104
Turingova mašina sa nepomičnom glavom, 102
Turingova mašina sa više traka, 104
unija DFA, 28
unija NFA, 43
univerzalna Turingova mašina, 110

beskorisno stanje, 36

certifikat, 169
CFG, 59
CFL, 60
Chomsky normalna forma, 68
Chomskyeva hijerarhija, 143, 146
Church-Turingova teza, 101
CLIQUE, 169
CNF, 166
CNFSAT, 166
Cook-Levinova teorema, 173

deterministički konačni automat, 19
DFA, 19
dvosmislen jezik, 68
dvosmislena gramatika, 67
dvosmisleni jezik, 67

element, 1
enumerator, 138

funkcija
izračunljiva, 162

gramatika
CFG, 59
dvosmislena, 67
kontekstno nezavisna, 57, 59
kontekstno zavisna, 144
nedvosmislena, 67
neograničena, 141

hipoteza kontinuuma, 9

inherentno dvosmislen jezik, 68
instrukcija registarske mašine, 147
izračunljivo preslikavanje, 162

jezik, 12
CFL, 60
dvosmislen, 68
dvosmisleni, 67
inherentno dvosmislen, 68
kontekstno nezavisan, 57, 59

odlučiv, 99
redukcija, 163
regularan jezik, 15, 17
rekurzivno prebrojiv, 99
Turing-prepoznatljiv, 99, 110

k-PDA, 107
k-potisni automat, 88, 107
Kleeneova zvijezda, 13
komplement DFA, 29
konfiguracija registarske mašine, 147
konfiguracija Turingove mašine, 98
konjuktivna normalna forma, 166
kontekstno nezavisan jezik, 57, 59
kontekstno nezavisna gramatika, 59
kontekstno nezavisna gramatika, 57
kontekstno zavisna gramatika, 144

LBA, 144
leksikografski poredak, 11
ligička varijabla, 165
linearni ograničeni automat, 144
literal, 165
logička formula, 165
logički izraz, 165

malo o, 152

nadovezivanje jezika, 12
nadovezivanje NFA, 43
nedeterministički konačni automat, 29
nedvosmislena gramatika, 67
neograničena gramatika, 141
NFA, 29
NP, 167
NP-kompletan jezik/problem, 173
NP-težak jezik/problem, 172
NTIME, 167

odlučiv jezik, 99
odlučivač, 99
operacija nadovezivanja, 12

P, 160
PDA, 71
petlja, 20
polinomna redukcija, 163
polinomni verifikator, 169
polinomno izračunljivo preslikavanje, 163
potisni automat, 57
prefix, 10
presjek DFA, 27
preslikavanje
bijekcija, 3
injekcija, 3
polinomno izračunljivo, 163
sirjekcija, 3
Primov algoritam, 161
problem ograničenog popločavanja, 174
problem popločavanja, 174
problem zaustavljanja, 124
Pumpajuća lema
za kontekstno nezavisne jezike, 84
za regularne jezike, 50

računanje registarske mašine, 147
računanje Turingove mašine, 98
redukcija jezika, 163
registarska mašina, 147
regularan izraz, 16
regularan jezik, 15, 17
rekurzivna varijabla, 83, 84
rekurzivno prebrojiv jezik, 99
Riceov teorem, 135
riječ, 10

SAT, 165
skup, 1
    beskonačan skup, 5
    broj elemenata skupa, 2
    jezik, 12
    komplement skupa, 3
    konačan skup, 5
    moć skupa, 5
    partitivni skup, 5
    podskup skupa, 2
    pravi podskup skupa, 2
    prazan skup, 2
    prebrojiv skup, 6
    presjek skupova, 2
    razlika skupova, 2
    skup cijelih brojeva, 1
    skup prirodnih brojeva, 1
    skup racionalih brojeva, 1
    skup realnih brojeva, 1
    skup svih stringova, 10
    unija skupova, 2
    univerzalni skup, 3
SpanningTree, 161
stablo
    stablo raščlanjivanja, 65
string, 10
    inverzni string, 14
    palindrom, 14, 74
    podstring, 10
    prazan string, 10
    prefix, 10
    skup svih stringova, 10
    sufix, 10
sufix, 10

TIME, 160
TM, 96
totalna Turingova mašina, 99
Turing-prepoznatljiv jezik, 99, 110
Turingova mašina, 95, 97
    nedeterministička, 105
    sa dvostruko beskonačnom trakom, 104
    sa nepomičnom glavom, 102
    sa više traka, 104
    univerzalna, 110

unija DFA, 28
unija NFA, 43

veliko O, 150
verifikator, 169
vrijeme izvršavanja
    nedeterminističke Turingove mašine, 159

Prazna stranica.

Bibliografija

[1] A.A., Puntambekar. *Formal Languages And Automata Theory*. 1st edition. Technical Publications, 2008. ISBN: 9788184314687.

[2] Arora, Sanjeev. *Computational Complexity : A Modern Approach*. Cambridge New York: Cambridge University Press, 2009. ISBN: 0521424267.

[3] Benoit, Anne; Robert, Yves, and Vivien, Frederic. *A guide to Algorithm Design : Paradigms, ;Methods, and Complexity Analysis*. CRC Press, 2014. ISBN: 1439825645.

[4] Blum, Lenore et al. *Complexity and Real Computation*. Springer, 1998. ISBN: 0387982817.

[5] Burgisser, Peter; Clausen, Michael, and Amin Shokrollahi, Mohammad. *Algebraic Complexity Theory*. Springer Berlin Heidelberg, 1997. ISBN: 3642082289.

[6] Carroll, John and Long, Darrell. *Theory of Finite Automata With an Introduction to Formal Languages*. 1st edition. 1989.

[7] Chandrasekaran, K.L.P. *Theory of Computer Science: Automata, Languages and Computation*. 3rd edition. PHI Learning, 2006.

[8] Chiswell, Ian. *A Course in Formal Languages, Automata and Groups*. London: Springer, 2009. ISBN: 1848009399.

[9] Ćirić, Miroslav and Ignjatović, Jelena. *Teorija algoritama, Automata i jezika - zbirka zadataka*. 1st edition. Prirodno-matematički fakultet Univerziteta u Nišu, 2012. ISBN: 978-86-83481-87-3.

[10] Ćirić, Miroslav; Petković, Tatjana, and Bogdanović, Stojan. *Jezici i automati*. 1st edition. Publisher, 2000. ISBN: 86-7455-457-1.

[11] Cohen, Daniel. *Introduction to Computer Theory*. Wiley, 1997. ISBN: 0471137723.

[12] Davis, Martin; Sigal, Ron, and J. Weyuker, Elaine. *Computability, Complexity, and Languages : Fundamentals of Theoretical Computer Science*. San Diego: Academic Press, Harcourt, Brace, 1994. ISBN: 0122063821.

[13] Dovedan, Zdravko. *Formalni jezici i prevodioci : Regularni izrazi, gramatike, automati*. 1st edition. Udžbenici Sveučilišta u Zagrebu, Element, 2012. ISBN: 978-953-197-617-6.

[14] Dovedan, Zdravko. *Formalni jezici i prevodioci : Sintaksna analiza i primjene*. 1st edition. Udžbenici Sveučilišta u Zagrebu, Element, 2012. ISBN: 978-953-197-618-3.

[15] Du, Ding-Zhu and Ko, Ker-I. *Problem Solving in Automata, Languages, and Complexity*. 1st edition. John Wiley & Sons, 2001. ISBN: 0-471-43960-6.

[16] Fortnow, Lance. *The Golden Ticket: P, NP, and the Search for the Impossible*. 1st edition. Princeton University Press, 2013. ISBN: 0691156492.

[17] Garey, Michael. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979. ISBN: 0716710455.

[18] Ginsburg, Seymour and Ullian, Joseph. "Ambiguity in Context Free Languages". In: *J. ACM* 13.1 (Jan. 1966), pages 62–89. ISSN: 0004-5411. DOI: 10.1145/321312.321318. URL: https://doi.org/10.1145/321312.321318 (cited on page 68).

[19] Goldreich, Oded. *Computational Complexity : A Conceptual Perspective*. Cambridge New York: Cambridge University Press, 2008. ISBN: 052188473X.

[20] Hollos, Stefan and Hollos, Richard. *Finite Automata and Regular Expressions : Problems and Solutions*. Abrazol, 2013. ISBN: 1887187162.

[21] Hopcroft, John; Motwani, Rajeev, and Ullman, Jeffrey. *Introduction to Automata Theory, Languages, and Computation*. 3rd edition. Boston: Pearson/Addison Wesley, 2007. ISBN: 0321455363.

[22] Hromkovič, Juraj. *Theoretical Computer Science : Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography*. Springer, 2004. ISBN: 3540140158.

[23] Ignjatović, Jelena and Ćirić, Miroslav. *Automati i Formalni jezici*. 1st edition. Univerzitet u Nišu, Prirodno-matematički fakulet, 2016. ISBN: 978-86-6275-056-3.

[24] Immerman, Neil. *Descriptive Complexity*. Springer, 1999. ISBN: 0387986006.

[25] *INTERNATIONAL STANDARD ISO/IEC 14882*. Fifth edition. ISO/IEC, 2017-12 (cited on page 90).

[26] Kozen, Dexter. *Automata and Computability*. New York: Springer, 1997. ISBN: 0387949070.

[27] Kuich, Werner and Salomaa, Arto. *Semirings, Automata, Languages*. Springer-Verlag, 1986. ISBN: 3540137165.

[28] Landsberg, J. M. *Geometry and Complexity Theory*. Cambridge University Press, 2017. ISBN: 1107199239.

[29] Lawson, Mark. *Finite Automata*. Chapman & Hall/CRC, 2004. ISBN: 1584882557.

[30] Lewis, Harry and Papadimitriou, Christos. *Elements of the Theory of Computation*. 2nd edition. Prentice-Hall, 1998. ISBN: 0132624788.

[31] Linz, Peter. *An Introduction to Formal Languages and Automata*. 6th edition. Burlington, MA: Jones & Bartlett Learning, 2017. ISBN: 1284077241.

[32] Listrovoy, Sergey. *On the Class of NP-Complete Problems and Rank Approach*. LAP LAMBERT Academic Publishing, 2014. ISBN: 3659549053.

[33] Mihov, Stoyan and Schulz, Klaus U. *Finite-State Techniques: Automata, Transducers and Bimachines*. 1st edition. Cambridge University Press, 2019.

[34] Moore, Cristopher and Mertens, Stephan. *The Nature of Computation*. Oxford University Press, 2011. ISBN: 0199233217.

[35] Nagpal, C. K. *Formal Languages and Automata Theory*. Oxford University Press, 2011. ISBN: 9780198071068.

[36] Nayak, Janmenjoy and H.S. Behera, Hadibandhu Pattnayak. *Formal Languages and Automata Theory*. 1st edition. Vikas, 2018. ISBN: 9325978598.

[37] Ognjanović, Zoran and Krdžavac, Nenad. *Uvod u teorijsko računarstvo*. 1st edition. Fakultet organizacionih nauka, 2004. ISBN: 86-7680-028-6.

[38] Papadimitriou, Christos. *Computational Complexity*. Addison-Wesley, 1994. ISBN: 0201530821.

[39] Papadimitriou, Christos and Steiglitz, Kenneth. *Combinatorial Optimization : Algorithms and Complexity*. Prentice Hall, 1982. ISBN: 0486402584.

[40] Reiter, Edna and Johnson, Clayton Matthew. *Limits of Computation : An Introduction to the Undecidable and the Intractable*. CRC Press, 2013. ISBN: 1439882061.

[41] Rhodes, John. *Applications of Automata Theory and Algebra : Via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*. Edited by Chrystopher L. Nehaniv. World Scientific, 2010. ISBN: 9812836977.

[42] Rich, Elaine. *Automata, Computability and Complexity : Theory and Applications*. Pearson Prentice Hall, 2008. ISBN: 0132288060.

[43] Rudich, Steven and Wigderson, Avi. *Computational Complexity Theory*. American Mathematical Society Institute for Advanced Study, 2004. ISBN: 082182872X.

[44] Salomaa, Arto. *Computation and Automata*. Cambridge University Press, 1985. ISBN: 0521302455.

[45] Saxena, Shivam. *Automata Theory, Languages of Machines and Computability*. 1st edition. I K International Publishing House, 2018.

[46] Sermutlu, Emre. *Automata, Formal Languages, and Turing Machines*. 1st edition. 2020. ISBN: 979-8690145385.

[47] Shallit, Jeffrey. *A second Course in Formal Languages and Automata Theory*. Cambridge New York: Cambridge University Press, 2009. ISBN: 0521865727.

[48] Simon, Matthew. *Automata Theory*. World Scientific, 1999. ISBN: 9810237537.

[49] Singh, Ajit. *Formal Language and Automata Theory*. 1st edition. 2019. ISBN: 1079108130.

[50] Sipser, Michael. *Introduction to the Theory of Computation*. 3rd. Cengage Learningg, 2013. ISBN: 13: 978-1-133-18781-3.

[51] Spasić, Irena and Janičić, Predrag. *Teorija algoritama, jezika i automata - zbirka zadataka*. 1st edition. Matematički fakultet, Beograd, 2000. ISBN: 86-7589-013-3.

[52] Srbljić, Siniša. *Jezični procesori 1 : Uvod u teoriju Formalnih jezika, Automata i gramatika*. 1st edition. Udžbenik Sveučilišta u Zagrebu, Element, Zagreb, 2000. ISBN: 953-197-129-3.

[53] Šupić, Haris. *Automata and Formal Languages*. 1st edition. Faculty of Electrical Engineering - University of Sarajevo, 2015. ISBN: 978-9958-629-62-4.

[54] Wegener, Ingo. *The Complexity of Boolean Functions*. B.G. Teubner J. Wiley, 1987. ISBN: 9780471915553.

[55] Wegener, Ingo. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer, 2005. ISBN: 3540210458.

[56] Wigderson, Avi. *Mathematics and Computation : A Theory Revolutionizing Technology and Science*. Princeton University Press, 2019. ISBN: 0691189137.