SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 406

Upiti nad grafovima s podatcima i proširenja regularnih izraza

Mladen Mikša

Zagreb, lipanj 2012.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Računala svakodnevno skupljaju i obrađuju sve veće količine informacija. Kako bi se takvo skupljanje i obrada mogli uspješno izvršavati pronađeni su različiti načini pohranjivanja informacija u formi baza podataka. Pritom neki tipovi baza podataka omogućavaju prirodniji način pohrane određenih podataka od preostalih tipova. Pojavom društvenih mreža, semantičkog weba i izraženijim proučavanjem bioloških mreža, pojavila se i potreba za tipom baza podataka koji uspješno predstavlja takve informacije. Te se informacije pritom sastoje od objekata s njihovim svojstvima te vezama između njih. Novi tip baze podataka koji na prirodan način predstavlja takve informacije su grafovi s podatcima.

U bazama podataka efikasnost postavljanja upita nam je jedno od najvažnijih svojstava te je potrebno definirati načine postavljanja upita čija složenost je zadovoljavajuća. U ovom se radu promatraju neka svojstva različitih načina postavljanja upita kako bi mogli odvojiti efikasne načine od onih čiji se upiti sporo evaluiraju. Također, posebno promatramo skupinu upita zadanih s dva ekvivalentna načina predstavljanja koji se temelje na teoriji automata i formalnih jezika. Odnosno, upite gradimo korištenjem posebnih konačnih automata i regularnih izraza te promatramo njihova svojstva.

Rad je organiziran na sljedeći način. U sljedećem poglavlju definiramo kontekstno neovisnu gramatiku, osnovne pojmove iz teorije složenosti te detaljnije proučavamo neka pitanja teorije konačnih automata i regularnih jezika. Poglavlje 3 uvodi grafove s podatcima i formalizme za definiranje upita nad njima te proučava neka njihova svojstva. Konačni automati s registrima i regularni izrazi s memorijom definiraju se u poglavlju 4. Dokazujemo ekvivalentnost takvih regularnih izraza i automata, te ispitujemo složenosti evaluacije upita zadanih s njima. Na kraju dajemo zaključak rada s osvrtom na moguća preostala pitanja vezanih uz upite nad grafovima s podatcima.

2. Osnovni pojmovi

2.1. Formalni jezici i kontekstno neovisna gramatika

Računala prilikom rješavanja problema ne djeluju izravno nad strukturom na kojoj je definiran problem, već na određenom zapisu te strukture koja je podložnija za računanje. Kako bi takve zapise mogli formalno definirati i grupirati s obzirom na neka interesantna svojstva služimo se formalnim jezicima. Pritom se svaki zapis sastoji od niza znakova nekog unaprijed određenog skupa. Takav skup nazivamo alfabetom i označavamo sa Σ te ga formalno definiramo kao proizvoljan konačni skup čije elemente nazivamo znakovima. Uz definiran alfabet, moguće je definirati pojmove riječi i jezika nad danim alfabetom. Definicije prezentirane u ovom odjeljku uobičajene su definicije te se mogu naći u većini knjiga o teorijskom računarstvu, primjerice (Sipser, 2006).

Definicija 2.1 (Riječi i jezici)**.** *Riječ w alfabeta Σ je svaki konačan niz znakova iz Σ, pri čemu se broj znakova od kojih se riječ sastoji naziva duljina riječi, u oznaci $|w|$. S ϵ označavamo praznu riječ duljine 0.*

Jezik L nad alfabetom Σ je neki skup riječi alfabeta Σ. Odnosno, L je podskup od Σ^*, gdje sa Σ^* označavamo skup svih riječi alfabeta Σ.

Jedan od jednostavnih načina definiranja jezika je korištenjem formalnih gramatika. Formalne gramatike našle su mnoge primjene jer omogućavaju izgradnju stabla nad riječi što nam može pružiti dodatne informacije o svojstvima same riječi. U ovom radu gramatike i njihova svojstva se ne proučavaju detaljno, već se koristi jedan poseban tip gramatike kako bi jednostavno definirali nama zanimljive izraze. Tip gramatike koji koristimo je kontekstno neovisna gramatika čija je definicija dana u nastavku. U daljnjem tekstu prilikom spominjanja gramatike podrazumijevat ćemo da je riječ o kontekstno neovisnoj gramatici.

Definicija 2.2 (Kontekstno neovisna gramatika)**.** Kontekstno neovisna gramatika G *je uređena četvorka* (V, Σ, P, S) *gdje je*

– *V konačan skup čije elemente nazivamo nezavršnim znakovima,*

– *Σ konačan skup, disjunktan sa V, čije elemente zovemo završnim znakovima,*

– *P konačni skup, čije elemente nazivamo pravila i zapisujemo ih u obliku:*

$$A \to w, \text{ pri čemu je } A \in V, w \in \Sigma^*,$$

– *$S \in V$ element kojeg nazivamo početnim nezavršnim znakom.*

U slučaju kada imamo više pravila koja sadrže isti nezavršni znak A na lijevoj strani, $A \to w_1$, $A \to w_2$, …, $A \to w_k$, tada taj niz pravila zapisujemo u obliku:

$$A \to w_1 \mid w_2 \mid \ldots \mid w_k.$$

Kako nam sama definicija kontekstno neovisne gramatike još ne pruža poveznicu s riječima nekog jezika, potrebno je definirati što znači da gramatika izvodi riječ. Uz tu definiciju moći ćemo definirati i jezik gramatike, odnosno skup svih riječi koji pripadaju zadanoj gramatici. Ti pojmovi formalizirani su sljedećom definicijom.

Definicija 2.3 (Kontekstno neovisni jezici)**.** *Neka je $G = (V, \Sigma, P, S)$ neka kontekstno neovisna gramatika, a $u_1, u_2 \in (V \cup \Sigma)^*$ neki nizovi nezavršnih i završnih znakova.* Primjenu pravila $A \to v \in P$ *gramatike G nad nizom $u_1 A u_2$ definiramo kao zamjenu tog niza s nizom $u_1 v u_2$, u oznaci $u_1 A u_2 \Rightarrow u_1 v u_2$. Pritom kažemo da smo primjenom pravila $A \to v$ iz niza $u_1 A u_2$ dobili niz $u_1 v u_2$.*

Za kontekstno neovisnu gramatiku G kažemo da izvodi *riječ $w \in \Sigma^*$, ako je moguće iz početnog nezavršnog znaka S, uz konačno mnogo primjena pravila gramatike G, dobiti riječ w. Skup svih riječi $w \in \Sigma^*$ koje izvodi gramatika G nazivamo* jezikom gramatike *G i označavamo s $L(G)$.*

Kontekstno neovisnim gramatikama mogu se definirati mnogi jezici od praktičnog interesa. Primjerice, kontekstno neovisnom gramatikom moguće je definirati formule propozicionalne logike zadane u konjunktivnoj normalnoj formi. Konjunktivna normalna forma jedan je od bitnih oblika zapisa logičkih formula koji se koristi u teorijskom računarstvu. Formalnu definiciju konjunktivne normalne forme moguće je pronaći u mnogim knjigama o logici ili računarstvu, primjerice (Vuković, 2009). Primjer korištenja kontekstno neovisne gramatike pri definiranju logičkih formula dan je u nastavku.

Primjer 2.4. *Definirajmo gramatiku za generiranje logičkih formula u konjunktivnoj normalnoj formi, pri čemu nećemo razlikovati sudove te ćemo ih sve označavati s p.*

Gramatika $G = (V, \Sigma, P, S)$ sadrži tri nezavršna znaka $V = \{S, K, L\}$ te završne znakove za oznaku suda, logičke veznike i zagrade $\Sigma = \{p, \wedge, \vee, \neg, (,)\}$. Pravila P gramatike su:

$$S \to (K) \mid (K) \wedge S$$
$$K \to L \mid L \vee K$$
$$L \to p \mid \neg p$$

Nezavršni znak S prvo generira konjunkciju klauzula koje se šire korištenjem pravila pridijeljenih nezavršnom znaku K. Iz nezavršnog znaka L dobivamo literale, odnosno sudove ili njihove negacije. Primjer generiranja logičke formule $(p \vee \neg p) \wedge (p)$ korištenjem gramatike G dan je sljedećim nizom primjena pravila P:

$$S \Rightarrow (K) \wedge S \Rightarrow (K) \wedge (K) \Rightarrow (L \vee K) \wedge (K) \Rightarrow (L \vee L) \wedge (K)$$
$$\Rightarrow (L \vee L) \wedge (L) \Rightarrow (p \vee L) \wedge (L) \Rightarrow (p \vee \neg p) \wedge (L) \Rightarrow (p \vee \neg p) \wedge (p).$$

U nastavku promatramo druge načine definiranja jezika, koji nam omogućavaju da govorimo o složenijim problemima te da točno ocijenimo kolika je zapravo njihova složenost. Zbog toga, u sljedećem odjeljku definiramo osnovne pojmove teorije izračunljivosti i složenosti.

2.2. Teorija složenosti

Osnovni model računala koji se koristi u teoretskom računarstvu je Turingov stroj. Turingov se stroj sastoji od konačnog skupa stanja i konačnog broja beskonačnih traka na kojima se provodi računanje. Na početku rada Turingovog stroja možemo imati ulazni podatak, odnosno riječ, na prvoj traci koju zovemo ulaznom trakom stroja. Odluku o tome što se zapisuje na trake, u koje se stanje prelazi i na koju stranu se pomiču glave, Turingov stroj donosi na temelju trenutnog stanja i znaka kojeg trenutno čita. Iako Turingovi strojevi ne moraju stati za svaku početnu riječ, ovdje promatramo samo takve Turingove strojeve i poistovjećujemo ih s pojmom algoritma. Također, obično su početne riječi Turingovog stroja kôdovi nekog matematičkog objekta. Za objekt O s $\langle O \rangle$ označavamo njegov kôd, a detalje samog kodiranja opisujemo prilikom definiranja objekta O. Formalne definicije Turingovog stroja i ostalih pojmova o kojima je riječ u ovom odjeljku preuzete su, te se mogu naći u (Sipser, 2006).

Glavna primjena Turingovog stroja je u prihvaćanju jezika. Odnosno, nakon računanja nad ulaznom riječi Turingov stroj staje i prelazi u prihvatljivo ili neprihvatljivo

stanje ovisno o tome je li riječ u jeziku. Kako su jezici uobičajeno kôdovi nekih matematičkih objekata s određenim svojstvima, u nastavku poistovjećujemo problem ispitivanja svojstva nekog objekta s odgovarajućim jezikom. Ako Turingov stroj uvijek staje za sve riječi alfabeta, onda nas zanima i koliko je resursa potrošio prije nego što je završio s radom. Odnosno, zanimaju nas vremenska i prostorna složenost Turingovog stroja $\mathcal{T}$ koje definiramo u nastavku.

Definicija 2.5 (Vremenska i prostorna složenost)**.** *Neka je zadan Turingov stroj $\mathcal{T}$.* Vremenska složenost Turingovog stroja $\mathcal{T}$ *je funkcija $t : \mathbb{N} \to \mathbb{N}$, pri čemu je $t(n)$ najveći broj koraka koje stroj $\mathcal{T}$ napravi prilikom računanja za sve riječi veličine n.* Prostorna složenost Turingovog stroja $\mathcal{T}$ *je funkcija $s : \mathbb{N} \to \mathbb{N}$, pri čemu je $s(n)$ najveći broj ćelija koje stroj $\mathcal{T}$ iskoristi prilikom računanja za sve riječi veličine n.*

Kako nas obično ne zanima točna funkcija vremenske složenosti, već samo približan iznos, koristimo asimptotske granice za određivanje složenosti Turingovih strojeva.

Definicija 2.6 (Asimptotska gornja granica)**.** *Neka su $f, g : \mathbb{N} \to \mathbb{N}$ dvije funkcije za koje postoje prirodni brojevi c i n_0 takvi da za svaki $n \geq n_0$ vrijedi $f(n) \leq cg(n)$. Kažemo da je funkcija g* asimptotska gornja granica *funkcije f, u oznaci $f(n) = \mathcal{O}(g(n))$.*

Posebno nas zanima slučaj kad je asimptotska gornja granica vremenske složenosti Turingovog stroja neki polinom. Jezike koje takvi Turingovi strojevi prihvaćaju grupiramo u posebnu klasu složenosti koju poistovjećujemo s efikasno rješivim problemima. Sličnu situaciju imamo i u slučajevima kada je asimptotska gornja granica prostorne složenosti polinom ili logaritam, pri čemu za definiranje logaritamske prostorne složenosti ne promatramo ulaznu traku. Formalne definicije tih osnovnih klasa složenosti dane su u nastavku.

Definicija 2.7 (Klase složenosti)**.** P *je klasa svih jezika za koje postoji Turingov stroj $\mathcal{T}$ koji u polinomnoj vremenskoj složenosti, tj. $t(n) = \mathcal{O}(n^i)$ za neki i, prihvaća taj jezik.* PSPACE *je klasa svih jezika za koje postoji Turingov stroj $\mathcal{T}$ koji u polinomnoj prostornoj složenosti, tj. $s(n) = \mathcal{O}(n^i)$ za neki i, prihvaća taj jezik.* L *je klasa svih jezika za koje postoji Turingov stroj $\mathcal{T}$ koji u logaritamskoj vremenskoj složenosti, tj. $s(n) = \mathcal{O}(\log n)$, prihvaća taj jezik.*

Prethodno definirane klase složenosti odnose se na deterministične Turingove strojeve, odnosno strojeve koji za svaki par stanja i znaka imaju točno jedan prijelaz. Možemo promatrati i nedeterminističke Turingove strojeve koji imaju više izbora prilikom određivanja prijelaza. Prilikom definiranja složenosti takvih strojeva promatra se posebno svaki niz nedeterminističkih izbora koje možemo napraviti za određenu početnu

riječ. Vremenska i prostorna složenost se definiraju promatranjem niza nedeterminističkih izbora koji vodi do najvećeg broja koraka, odnosno broja korištenih ćelija. Uz to možemo definirati nedeterminističke klase složenosti.

Definicija 2.8 (Nedeterminističke klase složenosti)**.** NP *je klasa svih jezika za koje postoji nedeterministički Turingov stroj $\mathcal{T}_N$ koji u polinomnoj vremenskoj složenosti, tj. $t(n) = \mathcal{O}(n^i)$ za neki i, prihvaća taj jezik.* NPSPACE *je klasa svih jezika za koje postoji nedeterministički Turingov stroj $\mathcal{T}_N$ koji u polinomnoj prostornoj složenosti, tj. $s(n) = \mathcal{O}(n^i)$ za neki i, prihvaća taj jezik.* NL *je klasa svih jezika za koje postoji nedeterministički Turingov stroj $\mathcal{T}_N$ koji u logaritamskoj vremenskoj složenosti, tj. $s(n) = \mathcal{O}(\log n)$, prihvaća taj jezik.*

Za nedeterminističke se strojeve smatra da mogu prihvatiti više jezika nego što ih mogu determinističke verzije uz korištenje jednako resursa. S druge strane, kod prostorne složenosti imamo teorem koji nam govori da ne trebamo iskoristiti mnogo više prostora kako bi ga riješili na determinističkom stroju. Teorem je dokazao Savitch (1970) i iskazujemo ga u nastavku.

Teorem 2.9. *Za svaki jezik L za koji postoji nedeterministički Turingov stroj $\mathcal{T}_N$ koji ga prihvaća u prostornoj složenosti $\mathcal{O}(f(n))$ postoji i deterministički Turingov stroj $\mathcal{T}$ koji prihvaća isti jezik u prostornoj složenosti $\mathcal{O}(f(n)^2)$.*

Primijetimo da iz tog teorema dobivamo jednakost NPSPACE = PSPACE jer kvadriranjem polinoma dobivamo opet polinom. Druga važna primjena Turingovih strojeva su Turingovi strojevi koji ne prihvaćaju neki jezik već na posebnoj traci zapisuju neki izlaz ovisan o ulaznoj riječi. Tu traku zovemo izlaznom trakom te ju također ne brojimo prilikom računanja prostorne složenosti. Uz takve Turingove strojeve možemo definirati i pojam redukcije s jezika L_1 na jezik L_2.

Definicija 2.10 (Redukcija s jezika L_1 na L_2)**.** *Turingov stroj $\mathcal{T}$ reducira jezika L_1 na jezik L_2 ako za svaku početnu riječ w_1 na izlaznoj traci zapisuje riječ w_2 za koju vrijedi $w_1 \in L_1$ ako i samo ako $w_2 \in L_2$.*

Za Turingov stroj logaritamske prostorne složenosti kažemo da radi u logaritamskom prostoru. Slično kažemo i u slučaju vremenske složenosti te kad složenosti nisu logaritamske, već polinomne ili eksponencijalne. Posebno, redukciju s jezika L_1 na L_2 koju provodimo u logaritamskom prostoru skraćeno označavamo s $L_1 \leq_l L_2$. Korištenjem redukcija možemo definirati posebne klase problema koji sadrže „najteže“ probleme neke od prije definiranih klasa složenosti $\mathcal{C}$.

Definicija 2.11 ($\mathcal{C}$-težina i $\mathcal{C}$-potpunost)**.** *Neka je zadan jezik L i klasa složenosti $\mathcal{C}$. Za jezik L kažemo da je* $\mathcal{C}$-težak *ako se svaki jezik iz $\mathcal{C}$ može u logaritamskom prostoru reducirati na jezik L. Dodatno, ako se jezik L nalazi u klasi složenosti $\mathcal{C}$ kažemo da je taj jezik* $\mathcal{C}$-potpun*.*

Iako tvrdnja nije očita, kompozicija dvije redukcije koje rade u logaritamskom prostoru također radi u logaritamskom prostoru. Dokaz te tvrdnje može se pronaći u knjizi (Sipser, 2006). Primijetimo da zbog toga jednom kad pronađemo neki $\mathcal{C}$-potpuni jezik L, $\mathcal{C}$-težinu drugog jezika L' možemo dokazati samo pronalazeći neku redukciju s jezika L na L' koja radi u logaritamskom prostoru, odnosno dokazujući $L \leq_l L'$.

2.3. Konačni automati i regularni izrazi

Najjednostavniji strojevi koji prepoznaju neke jezike i koji su našli široku primjenu su konačni automati. Glavno svojstvo konačnih automata je da ne mogu pamtiti podatke koje pročitaju te onda ne mogu provoditi složenije operacije nad riječima. Cijelo upravljanje radom konačnih automata temelji se na konačnom broju upravljačkih stanja i pravila kako se ta stanja mijenjaju ovisno o pročitanim znakovima. U slučaju kada za svako stanje i svaki znak koji pročitamo imamo definirano sljedeće stanje govorimo o deterministčkim konačnim automatima. Definicije automata i jezika koje prihvaćaju, prikazane u nastavku, mogu se naći u mnogim knjigama o automatima i teoretskom računarstvu, primjerice (Sipser, 2006).

Definicija 2.12 (Deterministički konačni automat)**.** *Za zadani alfabet Σ,* deterministički konačni automat *$\mathcal{M}$ je uređena četvorka $\mathcal{M} = (Q, q_0, F, \delta)$ gdje je:*

- *Q konačni skup čije elemente zovemo stanja automata,*
- *$q_0 \in Q$ element kojeg nazivamo početnim stanjem automata,*
- *$F \subseteq Q$ skup čije elemente nazivamo prihvatljivim stanjima automata i*
- *$\delta \colon Q \times \Sigma \to Q$ funkcija koju nazivamo funkcijom prijelaza.*

Iz definicije automata nije odmah vidljivo kako su oni povezani s jezicima. Kako bi se ta veza ostvarila, potrebno je definirati ponašanje automata na nekoj proizvoljnoj riječi w alfabeta Σ. Automat započinje svoj rad u početnom stanju q_0 te čita zadanu riječ w znak po znak mijenjajući pritom svoje stanje ovisno o funkciji prijelaza. Odnosno, ako se automat nalazi u stanju q, nakon čitanja znaka a automat će prijeći u stanje $\delta(q, a)$. Koristeći takve promjene stanja, automat računa hoće li prihvatiti riječ

ili ne ovisno o konačnom stanju u kojem se nalazi. Formalna definicija prihvaćanja riječi i jezika kod deterministčkog konačnog automata dana je u nastavku.

Definicija 2.13 (Jezik deterministčkog konačnog automata)**.** *Za deterministički konačni automat $\mathcal{M} = (Q, q_0, F, \delta)$ kažemo da prihvaća riječ $w \in \Sigma^*$ ako se automat, započinjući izvođenje u stanju q_0, nalazi u nekom prihvatljivom stanju $q \in F$ nakon čitanja riječi w. Skup svih riječi koje automat prihvaća nazivamo jezikom automata i označavamo s $L(\mathcal{M})$.*

U definiciji deterministčkog konačnog automata koristili smo funkciju prijelaza koja je morala biti totalna funkcija nad kartezijevim produktom stanja i simbola alfabeta. Prilikom samog konstruiranja automata to može predstavljati problem, jer je potrebno definirati sve prijelaze od kojih nam mnogi ne moraju biti uopće interesantni. Dodatno, ako bi automat imao veći broj stanja u koje može prijeći ovisno o pročitanom znaku, to bi mu pružilo veću izražajnu moć. Zbog toga ćemo definirati novi konačni automat u kojem se funkcija prijelaza zamjenjuje s relacijom prijelaza koja će zadovoljiti prethodne zahtjeve.

Definicija 2.14 (Nedeterministički konačni automat)**.** *Za zadani alfabet Σ, nedeterministički konačni automat $\mathcal{N}$ je uređena četvorka $\mathcal{N} = (Q, q_0, F, \delta)$ gdje je:*

- *Q konačni skup čije elemente zovemo stanja automata,*
- *$q_0 \in Q$ element kojeg nazivamo početnim stanjem automata,*
- *$F \subseteq Q$ skup čije elemente nazivamo prihvatljivim stanjima automata i*
- *$\delta \subseteq Q \times \Sigma \times Q$ relacija koju nazivamo relacijom prijelaza.*

Kako više nemamo totalnu funkciju, ponašanje nedeterminističkih automata za zadanu riječ w ne može se definirati jednako kao i u determinističkom slučaju. Kod nedeterminističkog automata, ako se nalazimo u stanju q i trenutno čitamo znak a možemo prijeći u bilo koje stanje q' za koje vrijedi da se trojka (q, a, q') nalazi u relaciji prijelaza. Ako u cijelom procesu računanja odredimo točno koje prijelaze je automat koristio za pojedine znakove riječi w, definirat ćemo jednu granu računanja nedeterminističkog automata nad riječi w. Odnosno, grana računanja je niz stanja $q_0, q_1, \ldots, q_n$, gdje za svaki q_{i-1} i q_i vrijedi da je (q_{i-1}, a, q_i) u relaciji prijelaza automata, a a je i-ti znak riječi w. Korištenjem pojma grane računanja možemo definirati prihvaćanje riječi i jezika za nedeterminističke konačne automate.

Definicija 2.15 (Jezik nedeterminističkog konačnog automata)**.** *Za nedeterministički konačni automat $\mathcal{N} = (Q, q_0, F, \delta)$ kažemo da prihvaća riječ $w \in \Sigma^*$ ako postoji*

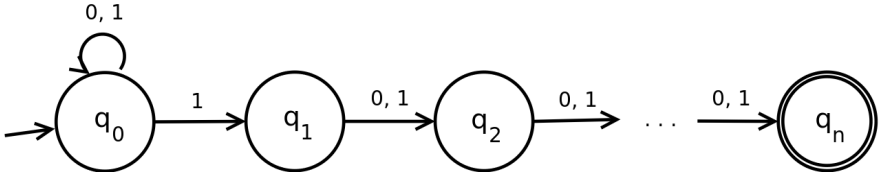
grana računanja automata takva da se automat, započinjući u stanju q_0, nalazi u nekom prihvatljivom stanju $q \in F$ nakon čitanja riječi w. Skup svih riječi koje automat prihvaća nazivamo jezikom automata i označavamo s $L(\mathcal{N})$.

S obzirom na veličinu ulazne riječi nedeterministički konačni automat ima eksponencijalno mnogo grana računanja od kojih je dovoljno da jedna bude prihvatljiva kako bi automat prihvatio riječ. Bez obzira na to, nedeterministički konačni automati prihvaćaju isti skup jezika kao i deterministički. Odnosno, za svaki nedeterministički konačni automat možemo konstruirati deterministički koji prihvaća isti jezik. Konstruirani deterministički automat pamtit će u svom stanju sva stanja nedeterminističkog automata do kojih nedeterministički automat može doći čitajući neki niz znakova. Prijelazi determinističkog automata definirani su tako da simuliraju prijelaze nedeterminističkog automata iz jednog skupa stanja u drugi. Korištenjem takve konstrukcije dobivamo da vrijedi sljedeći teorem (Rabin i Scott, 1959).

Teorem 2.16. *Neka je zadan nedeterministički konačni automat $\mathcal{N}$ nad alfabetom Σ. Tada postoji deterministički konačni automat $\mathcal{M}$ koji prihvaća isti jezik kao i nedeterministički automat $\mathcal{N}$, tj. $L(\mathcal{M}) = L(\mathcal{N})$.*

Iako ne mogu definirati nove jezike, nedeterministički automati nam pružaju veću ekspresivnost u odnosu na broj korištenih stanja. Odnosno, postoje jezici čiji deterministički konačni automati mogu imati bitno više stanja od ekvivalentnih nedeterminističkih automata. Primjer jednog takvog nedeterminističkog automata, preuzet iz (Hopcroft et al., 2001), prikazan je u nastavku.

Primjer 2.17. *Na slici 2.1 prikazan je primjer nedeterminističkog konačnog automata $\mathcal{N} = (Q, q_0, F, \delta)$. Stanja $Q = \{q_0, q_1, \ldots, q_n\}$ prikazana su kružnicama, pri čemu je početno stanja q_0 označeno strelicom, a prihvatljiva stanja su označena duplom kružnicom, pa je $F = \{q_n\}$. Prijelazi automata prikazani su strelicama nad kojima piše za koje simbole alfabeta vrijedi prijelaz. Automat je nedeterministički jer u stanju q_0 za pročitani znak 1 može prijeći natrag u stanje q_0 ili u stanje q_1.*



Slika 2.1: Primjer nedeterminističkog konačnog automata.

Prikazani nedeterministički konačni automat prihvaća sve riječi koje se sastoje od 0 i 1 te se znak 1 nalazi n znakova prije kraja riječi. Takvo ponašanje automata je

osigurano prijelazom iz stanja q_0 u q_1 koje je jedino moguće ostvariti u slučaju nailaska na znak 1. Ekvivalentan deterministički konačni automat morao bi u svojim stanjima pamtiti zadnjih n znakova kako bi mogao odlučiti je li prije n znakova pročitao 0 ili 1. Kako za pamćenje n znakova trebamo 2^n stanja, deterministički automat bi imao eksponencijalno veći broj stanja u odnosu na nedeterministički.

Jezici koje prihvaćaju konačni automati imaju razna povoljna svojstva, kao što je zatvorenost s obzirom na mnoge operacije. Jedna takva operacija je uzimanje komplementa jezika, odnosno skupa svih riječi zadanog alfabeta Σ koje se ne nalaze u jeziku. U nastavku je dan jednostavan dokaz tvrdnje da za svaki jezik L kojeg prihvaća neki konačni automat M postoji neki drugi konačni automat M' koji prihvaća komplement tog jezika.

Propozicija 2.18. *Neka je $\mathcal{M}$ neki deterministički konačni automat nad alfabetom Σ. Tada postoji deterministički konačni automat $\mathcal{M}'$ koji prihvaća komplement jezika automata $\mathcal{M}$, $L(\mathcal{M}') = \Sigma^* \setminus L(\mathcal{M})$.*

Dokaz. Definiramo konačni automat $\mathcal{M}' = (Q', q_0', F', \delta')$ takav da sadrži skup stanja, početno stanje i funkciju prijelaza koji su jednaki kao i kod automata $\mathcal{M}$, tj. $Q' = Q$, $q_0' = q_0$ i $\delta' = \delta$. Jedino u čemu se automati razlikuju je skup prihvatljivih stanja, gdje je F' komplement od F, $F' = Q \setminus F$. Lako se vidi da će svaka riječ w jezika $L(\mathcal{M})$ izvornog automata završiti u neprihvatljivom stanju automata $\mathcal{M}'$, dok će sve ostale riječi završiti u prihvatljivom stanju automata $\mathcal{M}'$. Slijedi da je jezik automata $\mathcal{M}'$ komplement jezika automata $\mathcal{M}$. Q.E.D.

U dokazu prethodne tvrdnje koristili smo determinističke konačne automate, ali zatvorenost na komplement vrijedi i za nedeterminističke automate po teoremu o ekvivalenciji s determinističkim (teorem 2.16). Nedeterministički automati razlikuju se od determinističkih u tome da konstrukcija prikazana u dokazu neće vrijediti za njih. Štoviše, u nekim slučajevima nedeterministički automat $\mathcal{N}'$ koji prihvaća komplement jezika drugog nedeterminističkog automata $\mathcal{N}$ imat će eksponencijalno više stanja. Kao primjer takvog jezika i nedeterminističkog automata možemo ponovno uzeti primjer 2.17, gdje smo komentirali da se sličan eksponencijalni rast događa prilikom prijelaza s nedeterminističkog na deterministički konačni automat.

Prethodnim dokazom pokazali smo da možemo iz zadanog konačnog automata dobiti drugi automat koji će prihvaćati komplement jezika izvornog automata. Općenito nas neće samo interesirati je li moguće generirati automat sa zadanim svojstvima, već i u kojoj složenosti ga možemo generirati. Zbog toga trebamo definirati neki način kodiranja konačnih automata. Jedan primjer kodiranja je dan sljedećom definicijom.

Definicija 2.19 (Kodiranje konačnih automata)**.** *Neka je zadan alfabet Σ te neki konačni automat $\mathcal{N} = (Q, q_0, F, \delta)$, pri čemu automat može biti i nedeterministički. Neka je skup stanja $Q = \{q_0, q_1, \ldots, q_k\}$ te alfabet $\Sigma = \{a_0, a_1, \ldots, a_l\}$. Stanjima i znakovima alfabeta pridjeljujemo prirodne brojeve tako da je stanju q_i pridijeljen prirodni broj $n_{q_i} = i$, a znaku a_i pridijeljen broj $n_{a_i} = i$.*

Relaciju prijelaza automata δ prikazujemo kao riječ w_δ koja se sastoji od niza trojki brojeva $(n_q, n_a, n_{q'})$ za svaki element $(q, a, q') \in \delta$, pri čemu su brojevi zapisani u binarnoj bazi. Kodiranje automata $\langle \mathcal{N} \rangle$ *je riječ koja se dobiva konkateniranjem riječi w_δ i riječi koja se sastoji od niza brojeva kojima su predstavljena prihvatljiva stanja automata.*

Primijetimo da pri takvom kodiranju automat čiji je kôd dužine n može imati najviše n stanja te isto toliko znakova alfabeta. Ali kako stanja i znakove kodiramo kao binarne brojeve, slijedi da će se svako stanje i znak moći kodirati nizom dužine $\mathcal{O}(\log n)$. Zbog toga kôdovi stanja i znakova alfabeta automata zauzimaju najviše logaritamski prostor u odnosu na dužinu kôda automata. Uz takvo kodiranje dokazujemo sljedeći teorem i njegov korolar o presjeku jezika dva nedeterministička konačna automata.

Teorem 2.20. *Postoji Turingov stroj $\mathcal{T}$ koji za svaka dva nedeterministička konačna automata $\mathcal{N}_1$ i $\mathcal{N}_2$ nad alfabetom Σ generira nedeterministički konačni automat $\mathcal{N}$ koji prihvaća presjek njihovih jezika, tj. $L(\mathcal{N}) = L(\mathcal{N}_1) \cap L(\mathcal{N}_2)$. Štoviše, Turingov stroj $\mathcal{T}$ radi u logaritamskom prostoru.*

Dokaz. Neka su $\mathcal{N}_i = (Q_i, q_0^{(i)}, F_i, \delta_i)$, $i = 1, 2$, dva nedeterministička konačna automata. Konstruiramo nedeterministički automat $\mathcal{N}$ čija su stanja uređeni parovi stanja izvornih automata $\mathcal{N}_1$ i $\mathcal{N}_2$, tj. $Q = Q_1 \times Q_2$. Početno stanje tog automata bit će uređeni par početnih stanja izvornih automata $q_0 = (q_0^{(1)}, q_0^{(2)})$. Skup prihvatljivih stanja F sadržavat će sve uređene parove prihvatljivih stanja početnih automata, tj. $F = F_1 \times F_2$.

Relacija prijelaza δ sadrži sve prijelaze iz jednog para stanja u drugi par stanja za znak a za koje vrijedi da i izvorni automati sadrže odgovarajuće prijelaze za isti znak a. Odnosno, vrijedi $((q_1, q_2), a, (q_1', q_2')) \in \delta$ ako je $(q_1, a, q_1') \in \delta_1$ i $(q_2, a, q_2') \in \delta_2$. Algoritam Turingovog stroja $\mathcal{T}$ koji u logaritamskom prostoru generira automat $\mathcal{N}$ dan je na slici 2.2.

Lako se vidi da algoritam generira prethodno opisani automat $\mathcal{N}$, pri čemu linija 1 generira početno stanja, linije 2–4 generiraju prijelaze, a linije 5–7 generiraju završna stanja automata. Tijekom rada algoritam treba pratiti najviše pet oznaka stanja i znakova alfabeta kako bi uspješno proveo petlje na linijama 2–4 i 5–7. Kako oznake

Ulaz: $\langle \mathcal{N}_1, \mathcal{N}_2 \rangle$, gdje su $\mathcal{N}_1$ i $\mathcal{N}_2$ nedeterministički konačni automati.

Izlaz: $\langle \mathcal{N} \rangle$, gdje je $\mathcal{N}$ nedeterministički konačni automat za koji vrijedi $L(\mathcal{N}) = L(\mathcal{N}_1) \cap L(\mathcal{N}_2)$.

1 Zapiši na izlaz $(q_0^{(1)}, q_0^{(2)})$ kao početno stanje.
2 **Ponovi** *za svaku četvorku stanja* $q_1, q_1' \in Q_1$, $q_2, q_2' \in Q_2$ *i svaki znak* $a \in \Sigma$
3 **Ako** *postoji* $(q_1, a, q_1') \in \delta_1$ *i* $(q_2, a, q_2') \in \delta_1$ **Onda**
4 Zapiši prijelaz $((q_1, q_2), a, (q_1', q_2'))$ na izlaz.
5 **Ponovi** *za svaki par stanja* $q_1 \in Q_1$, $q_2 \in Q_2$
6 **Ako** $q_1 \in F_1$ *i* $q_2 \in F_2$ **Onda**
7 Zapiši na izlaz stanje (q_1, q_2) kao prihvatljivo stanje.

Slika 2.2: Algoritam za presjek jezika nedeterminističkih konačnih automata.

stanja automata i znakova alfabeta zauzimaju najviše logaritamski prostor, slijedi da Turingov stroj generira automat $\mathcal{N}$ u logaritamskom prostoru. Q.E.D.

Korolar 2.21. *Skup svih jezika konačnih automata zatvoren je na konačne presjeke.*

Osim pitanja složenosti generiranja konačnog automata sa zadanim svojstvima, možemo postaviti i pitanje kolika je složenost određivanja ima li automat neko svojstvo. Jedno od najjednostavnijih takvih svojstava koje bi nas moglo zanimati i čiju složenost ćemo koristiti u nastavku je pitanje postoji li neka riječ koju automat ne prihvaća. Kako bi to definirali uvodimo pojam univerzalnosti automata koji nam govori da automat prihvaća sve riječi nekog alfabeta Σ.

Definicija 2.22 (Univerzalnost nedeterminističkog konačnog automata)**.** *Neka je zadan alfabet* Σ*. Za nedeterministički konačni automat* $\mathcal{N}$ *kažemo da je univerzalan za alfabet* Σ *ako prihvaća sve riječi jezika, odnosno vrijedi* $L(\mathcal{N}) = \Sigma^*$.

Primijetimo da će nas složenost ispitivanja postoji li riječ koju automat ne prihvaća zanimati samo u slučaju nedeterminističkih automata, jer je pitanje jednostavno za odgovoriti u deterministčkom slučaju. Razlog tome je da je tada pitanje ekvivalentno problemu dostiživosti u grafu, što je uobičajeni NL-potpuni problem. Dokaz NL-potpunosti problema dostiživosti u grafu može se naći u većini knjiga o teorijskom računarstvu i teoriji složenosti, primjerice (Sipser, 2006).

Formalno, problem ispitivanja postoji li neka riječ alfabeta Σ koju nedeterministički konačni automat ne prihvaća definira se kao

$$\mathrm{coUNIV}_{\mathrm{NKA}} = \{\langle \mathcal{N} \rangle \mid L(\mathcal{N}) \neq \Sigma^*\}.$$

Sljedećim teoremom dokazujemo gornju ogradu za složenost tog jezika, čiji je dokaz malo modificirana verzija sličnog dokaza iz Meyer i Stockmeyer (1972).

Teorem 2.23. *Problem ispitivanja postoji li riječ alfabeta Σ koju nedeterministički konačni automat $\mathcal{N}$ ne prihvaća je u klasi* PSPACE*, tj. kratko:*

$$\mathrm{coUNIV}_{\mathrm{NKA}} \in \mathsf{PSPACE}.$$

Dokaz. U dokazu prvo prezentiramo nedeterministički algoritam koji radi u linearnom prostoru. Odnosno, algoritam je u klasi $\mathsf{NSPACE}(n)$ i prihvaća samo one nedeterminističke automate koji ne prihvaćaju sve riječi jezika Σ^*. Tada je korištenjem Savitchevog teorema (Savitch, 1970) moguće konstruirati deterministički algoritam koji rješava isti problem u kvadratnom prostoru, tj. klasi $\mathsf{DSPACE}(n^2)$. Iz toga slijedi da se problem $\mathrm{coUNIV}_{\mathrm{NKA}}$ nalazi u klasi PSPACE. Algoritam je dan na slici 2.3.

Ulaz: $\langle \mathcal{N} \rangle$ gdje je $\mathcal{N} = (Q, q_0, F, \delta)$ nedeterministički konačni automat.
Izlaz: Prihvaća ulaz ako je $L(\mathcal{N}) \neq \Sigma^*$.

1 Stavi oznaku na početno stanje q_0 automata $\mathcal{N}$.
2 **Ponovi** 2^q *puta, gdje je* q *broj stanja u* $\mathcal{N}$
3 Nedeterministički izaberi neki znak $a \in \Sigma$.
4 Pomakni oznake na stanjima automata $\mathcal{N}$ tako da simuliraju čitanje znaka a.
5 Ako se nijedna od oznaka ne nalazi na stanju iz F, prihvati ulaz.
6 Odbaci ulaz.

Slika 2.3: $\mathsf{NSPACE}(n)$ algoritam za $\mathrm{coUNIV}_{\mathrm{NKA}}$ uz dokaz teorema 2.23.

Algoritam radi u linearnom prostoru jer samo treba čuvati trenutne oznake stanja kojih može najviše biti koliko i stanja automata. Uvedemo oznaku q, kao i u algoritmu, za broj stanja automata $\mathcal{N}$ i neka postoji riječ w duljine manje ili jednake 2^q koju automat $\mathcal{N}$ ne prihvaća. Tada će algoritam prihvatiti automat $\mathcal{N}$ jer će u jednoj nedeterminističkoj grani računanja simulirati ponašanje automata za tu riječ i utvrditi da ju automat ne prihvaća. Preostaje dokazati da u slučaju kada automat prihvaća sve riječi duljine manje ili jednake 2^q, slijedi da je automat univerzalan. Ali to vrijedi jer postoji samo 2^q različitih načina na koji se mogu postaviti oznake na stanja te svaki ulazni niz koji je duži od 2^q će ponoviti neko označavanje. Uklanjanjem dijela ulaznog niza između ponovljenih skupova oznaka, dobivamo niz koji je manji ili jednak 2^q i

kojeg automat prihvaća ako i samo ako prihvaća duži niz. Kako automat prihvaća sve takve nizove slijedi da mora biti univerzalan. Q.E.D.

Kako bi dokazali drugi smjer teorema, odnosno da je $\mathrm{coUNIV_{NKA}}$ PSPACE-težak, te da olakšamo zapis jezika koje možemo predstaviti konačnim automatima uvest ćemo pojam regularnih izraza. Ali, prije nego što možemo definirati regularne izraze i dokazati da predstavljaju iste jezike kao i konačni automati moramo uvesti nekoliko definicija. Započinjemo s definicijama regularnih operacija koje će nam biti osnova za konstruiranje regularnih jezika, pa i regularnih izraza. Definicije regularnih operacija i regularnih jezika su preuzete iz (Sipser, 2006).

Definicija 2.24 (Regularne operacije)**.** *Neka su L_1 i L_2 dva jezika. Regularne operacije* unije, konkatenacije *i* zvijezde *definirane su kao:*

Unija $L_1 \cup L_2 = \{x \mid x \in L_1$ *ili* $x \in L_2\}$,

Konkatenacija $L_1 \circ L_2 = \{xy \mid x \in L_1$ *i* $y \in L_2\}$ *i*

Zvijezda $L_1^* = \{x_1 x_2 \dots x_k \mid k \geq 0$ *i svaki* $x_i \in L_1\}$.

Uz regularne operacije možemo definirati regularne jezike, pri čemu uzimamo da su svi jezici koji se sastoje od samo jednog znaka alfabeta ili praznog niza regularni jezici. Preostale regularne jezike gradimo korištenjem regularnih operacija kako je opisano sljedećom definicijom.

Definicija 2.25 (Regularni jezik)**.** *Za zadani alfabet Σ, skup regularnih jezika definiramo rekurzivno kao najmanji skup koji zadovoljava sljedeće uvjete:*

a) prazan jezik $\emptyset$ i jednočlani jezici $\{a\}$, za sve $a \in (\Sigma \cup \epsilon)$, su regularni jezici,

b) ako su R_1 i R_2 regularni jezici, tada su i skupovi koji se dobivaju korištenjem regularnih operacija

$$R_1 \cup R_2, \quad R_1 \circ R_2 \quad i \quad R_1^*$$

regularni jezici.

Regularne jezike možemo jednostavno zapisati korištenjem regularnih izraza, koji će na intuitivan način predstaviti jezik koji možemo izgraditi korištenjem prethodne definicije. Odnosno, iz samih regularnih izraza bit će lako iščitati koji bi niz regularnih operacija nad elementima alfabeta trebali napraviti kako bi dobili željeni regularni jezik. Definicija regularnih izraza dana je u nastavku.

Definicija 2.26 (Regularni izrazi)**.** *Neka je Σ neki alfabet. Gramatika regularnih izraza $G_{\mathrm{REG}} = (V, \Sigma', P, S)$ je uređena četvorka gdje je $V = \{S\}$, $\Sigma' = \{\dot{a} \mid \forall a \in \Sigma\} \cup \left\{\dot{\emptyset}, \dot{\epsilon}, \dot{\cup}, \dot{^*}\right\}$, pri čemu skup P sadrži samo pravila:*

$$S \to \dot{\emptyset} \mid \dot{\epsilon} \mid \dot{a} \mid (S \dot{\cup} S) \mid (SS) \mid (S^{\dot{*}}), \quad \forall a \in \Sigma.$$

Jezik gramatike G_{REG} označavamo s $\mathrm{REG} = L(G_{\mathrm{REG}})$ i njegove elemente nazivamo regularnim izrazima.

Regularne izraze možemo povezati s regularnim jezicima pridajući svakom izrazu neki jezik korištenjem intuitivne semantike. Neka su r_1 i r_2 regularni izrazi kojima su pridjeljeni regularni jezici R_1 i R_2. Tada rekurzivno pridjeljujemo regularnom izrazu r regularni jezik R:

- $r = \dot{\emptyset}$ predstavlja jezik $R = \emptyset$,
- $r = \dot{\epsilon}$ predstavlja jezik $R = \{\epsilon\}$,
- $r = \dot{a}$ predstavlja jezik $R = \{a\}$ za svaki $a \in \Sigma$,
- $r = (r_1 \dot{\cup} r_2)$ predstavlja jezik $R = R_1 \cup R_2$,
- $r = (r_1 r_2)$ predstavlja jezik $R = R_1 \circ R_2$ i
- $r = (r_1)^{\dot{*}}$ predstavlja jezik $R = R_1^*$.

U nastavku rada nećemo pisati točku iznad simbola u regularnim izrazima te ćemo poistovjećivati regularne izraze i regularni jezik koji oni predstavljaju. Za neki regularni izraz r koristit ćemo oznaku $L(r)$ kako bi predstavili regularni jezik koji mu odgovara. Također, izostavljat ćemo zagrade u zapisu regularnih izraza te ćemo koristiti prioritet regularnih operacija. Pritom, najveći prioritet ima zvijezda, pa konkatenacija te na kraju unija.

Sljedeći teorem i njegov obrat povezuju konačne automate s regularnim izrazima, odnosno regularnim jezicima. Odnosno, dobivamo da je skup regularnih jezika točno onaj skup koji sadrži sve jezike koji se mogu predstaviti nekim konačnim automatom. Pritom, regularni izrazi nam predstavljaju jednostavniji zapis jezika od automata, dok su automati bliži samom načinu na koji računala prepoznaju jezike. Kako smo i prije spomenuli da nas zanimaju složenosti pojedinih operacija nad automatima, a sada i izrazima, u nastavku dokazujemo da u logaritamskom prostoru možemo iz regularnog izraza generirati pripadni nedeterministički konačni automat. Kodiranje regularnih izraza kako bi ih Turingov stroj mogao pročitati nije problem, jer su izrazi sami po sebi već riječi određenog jezika.

Teorem 2.27. *Postoji Turingov stroj $\mathcal{T}$ koji u logaritamskom prostoru za proizvoljni regularni izraz $r \in \mathrm{REG}$ na izlaznoj traci generira nedeterministički konačni automat $\mathcal{N}$ koji prihvaća isti jezik, tj. $L(\mathcal{N}) = L(r)$.*

Dokaz. Prvo ćemo induktivno po strukturi regularnog izraza definirati konstrukciju nedeterminističkog konačnog automata $\mathcal{N}$, nakon čega ćemo argumentirati da se takva konstrukcija može provesti na Turingovom stroju u logaritamskom prostoru. Konstrukcije nedeterminističkih konačnih automata ilustrirane su slikom 2.4. Plavom bojom označena su stanja i prijelazi koji se dodaju tijekom konstrukcije, dok je iscrtkanom linijom označeno da stanje gubi svoju prethodnu funkciju, odnosno nije više početno ili prihvatljivo stanje.

U slučajevima kad je regularni jezik prazan ili sadrži samo praznu riječ, automat $\mathcal{N}$ će imati će samo jedno stanje bez ikakvih prijelaza. Pritom će to stanje biti prihvatljivo u slučaju kada jezik sadrži praznu riječ. U slučaju kada imamo jednočlani jezik $L = \{a\}$, automat $\mathcal{N}$ će imati samo jedan prijelaz (q_0, a, q_1), pri čemu je q_0 početno, a q_1 prihvatljivo stanje. Potrebno je još promatrati slučajeve kada se regularni jezik R tvori korištenjem neke regularne operacije na proizvoljnim regularnim jezicima R_1 i R_2. Prilikom izgradnje takvog automata pretpostavljamo da već imamo automate $\mathcal{N}_i = (Q_i, q_0^{(i)}, F_i, \delta_i), i = 1, 2$ za jezike R_1 i R_2, pri čemu vrijedi $Q_1 \cap Q_2 = \emptyset$, pa je onda i $F_1 \cap F_2 = \emptyset$. Novi automat $\mathcal{N}$ koji prihvaća jezik R gradi se kombiniranjem automata $\mathcal{N}_1$ i $\mathcal{N}_2$.

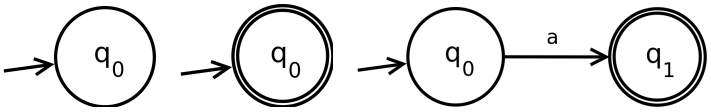
Automat $\mathcal{N}$ koji prihvaća uniju jezika dva automata gradi se tako da se u skup stanja dodaju sva stanja prvotnih automata te novo početno stanje, tj. $Q = Q_1 \cup Q_2 \cup \{q_0\}$. Prihvatljiva stanja novog automata čine uniju prihvatljivih stanja prvotnih automata $F = F_1 \cup F_2$. Relacija prijelaza sadrži sve njihove relacije te dodatne relacije koje povezuju novo početno stanje s automatima:

$$\delta = \delta_1 \cup \delta_2 \cup \left\{ (q_0, a, q) \mid (q_0^{(1)}, a, q) \in \delta_1 \vee (q_0^{(2)}, a, q) \in \delta_2 \right\}.$$

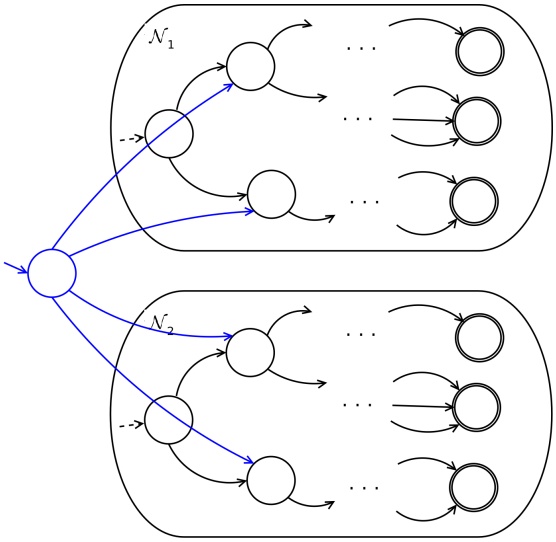
Dodatnu pozornost treba obratiti na slučaj kad je neko od početnih stanja izvornih automata prihvatljivo. U tom slučaju potrebno je dodati i novo početno stanje q_0 u skup prihvatljivih stanja F.

Automat za konkatenaciju jezika od dva automata gradimo uzimajući sva stanja izvornih automata $Q = Q_1 \cup Q_2$ i njihove prijelaze, te dodajući dodatne prijelaze između prvog i drugog automata:

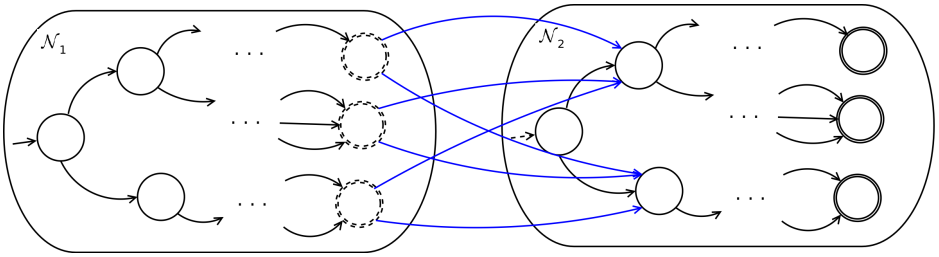
$$\delta = \delta_1 \cup \delta_2 \cup \left\{ (q, a, q') \mid q \in F_1 \wedge (q_0^{(2)}, a, q') \in \delta_2 \right\}.$$



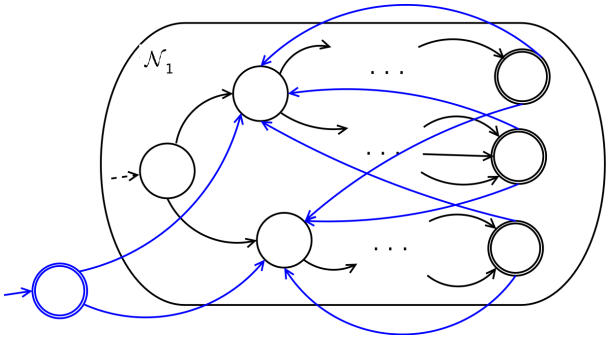
(a) (b) (c)



(d)



(e)



(f)

Slika 2.4: Prikazi konstrukcija nedeterminističkih konačnih automata za: (a) prazan skup; (b) praznu riječ; (c) znak a; (d) uniju regularnih jezika; (e) konkatenaciju regularnih jezika; (f) operator zvijezde nad regularnim jezikom.

Djelovanje operatora zvijezde nad jezikom nekog automata obuhvaćamo dodavanjem novog prihvatljivog početnog stanja kako bi se obuhvatilo prihvaćanje prazne riječi, tj. $Q = Q_1 \cup \{q_0\}$. Dodatno, potrebno je dodati prijelaze iz prihvatljivih stanja u stanja koja slijede početno stanje kako bi se ostvarila mogućnost ponavljanja riječi jezika izvornog automata. Formalno, relacija prijelaza definira se kao:

$$\delta = \delta_1 \cup \left\{(q_0, a, q) \mid (q_0^{(1)}, a, q) \in \delta_1\right\} \cup \left\{(q, a, q') \mid q \in F_1 \wedge (q_0^{(1)}, a, q') \in \delta_1\right\}.$$

Sada dokazujemo da postoji Turingov stroj $\mathcal{T}$ koji provodi opisano pridruživanje nedeterminističkog konačnog automata nekom regularnom izrazu, pri čemu $\mathcal{T}$ radi u logaritamskom prostoru. Prilikom provođenja algoritma svakom znaku i operatoru u regularnom izrazu pridjeljuje se njegov redni broj, te se znakovima pridjeljuju stanja korištenjem njihovog rednog broja. Kako jedan znak regularnog izraza uvodi najviše dva nova stanja, i-tom znaku regularnog izraza pripadaju stanja q_i i q_{i+1}. Generiranje prijelaza i prihvatljivih stanje je tada relativno lako provesti. Pritom Turingov stroj treba samo pamtiti za koji trenutno znak generira prijelaze, kao i konstantan broj pomoćnih varijabli koje služe prilikom pretraživanja izraza. Kako sve te varijable neće zauzimati više od logaritamskog prostora, Turingov stroj $\mathcal{T}$ provodi transformaciju u logaritamskom prostoru. Q.E.D.

Kao što smo spomenuli i prije, vrijedi i obrat prethodnog teorema, ali njega ovdje nećemo dokazivati jer nam neće biti od veće važnosti u nastavku. Za regularne izraze možemo postaviti ista pitanja kao i za konačne automate. Zbog toga možemo analogno jeziku $\mathrm{coUNIV}_{\mathrm{NKA}}$ definirati jezik $\mathrm{coUNIV}_{\mathrm{REG}}$ koji predstavlja sve regularne izraze za koje postoji riječ alfabeta Σ koja nije sadržana u njihovom jeziku. Po prethodnom teoremu vrijedi da u logaritamskom prostoru možemo iz regularnih izraza dobiti pripadni nedeterministički automat. Zbog toga možemo i problem $\mathrm{coUNIV}_{\mathrm{REG}}$ reducirati na problem $\mathrm{coUNIV}_{\mathrm{NKA}}$, tj. $\mathrm{UNIV}_{\mathrm{REG}} \leq_l \mathrm{UNIV}_{\mathrm{NKA}}$. Korištenjem te redukcije i teorema 2.23 slijedi da je ispitivanje postoji li riječ koja nije sadržana u jeziku zadanog regularnog izraza u klasi PSPACE.

Sada dokazujemo drugi smjer teorema, odnosno da je jezik $\mathrm{coUNIV}_{\mathrm{REG}}$ PSPACE-težak. Kao i u slučaju teorema 2.23, dokaz je malo izmijenjena verzija dokaza iz (Meyer i Stockmeyer, 1972).

Teorem 2.28. *Jezik* $\mathrm{coUNIV}_{\mathrm{REG}}$ *je* PSPACE*-težak.*

Dokaz. Slično kao i u dokazu Cook-Levinovog teorema NP-težine problema SAT, dokaz PSPACE-težine jezika $\mathrm{coUNIV}_{\mathrm{REG}}$ koristi tablicu izračunavanja. Primjer tablice

izračunavanja dan je slikom 2.5, pri čemu retci tablice predstavljaju stanje na traci Turingovog stroja $\mathcal{T}$ u nekom trenutku t. Stupci tablice predstavljaju pojedine ćelije trake Turingovog stroja. Kako promatramo jezike iz klase PSPACE broj stupaca tablice izračunavanja je za ulaz veličine n ograničen s n^k, za neki k. Za tablicu kažemo da je prihvatljiva ako se svaki redak, osim početnog, može dobiti primjenom nekog prijelaza Turingovog stroja $\mathcal{T}$ te se u nekoj ćeliji tablice nalazi prihvatljivo stanje q_{DA}.

	Prostor						
	0	1	p	$n-1$	n		n^k
Vrijeme 0	q_0	w_1	$\dots$	w_{n-1}	w_n	$\dots$	$\sqcup$
1	w'_1	q_1	$\dots$	w_{n-1}	w_n	$\dots$	$\sqcup$
$\vdots$	$\vdots$	$\vdots$	$\vdots$			$\vdots$	
t			$a\,b\,c$				
$t+1$			$d\,e\,f$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$			$\vdots$	
	$\dots$	$\dots$	q_{NE}			$\dots$	

Slika 2.5: Tablica izračunavanja.

Neka je $A \in$ PSPACE neki jezik kojeg prihvaća Turingov stroj $\mathcal{T}$ koji koristi najviše n^k ćelija trake te neka je Σ alfabet tog stroja $\mathcal{T}$. Za zadanu riječ $w \in \Sigma^*$ konstruirat ćemo regularni izraz r čiji jezik neće biti univerzalan ako i samo ako Turingov stroj $\mathcal{T}$ prihvaća riječ w. Kako bi mogli konstruirati regularni izraz r proširit ćemo alfabet Σ sa znakovima Q koji predstavljaju stanja stroja $\mathcal{T}$, znakovima za regularne operacije te znakom $\#$. Novi alfabet ćemo označavati s $\Delta = \Sigma \cup Q \cup \{\cup, ^*, \#\}$. Jezik regularnog izraza r sadržavat će sve riječi koje ne predstavljaju neku tablicu izračunavanja Turingovog stroja $\mathcal{T}$ nad riječi w takvu da stroj $\mathcal{T}$ prihvaća riječ w. Odnosno, skup $\Delta^* \setminus L(r)$ sadržavat će najviše jedan element koji će predstavljati prihvatljivu tablicu izračunavanja stroja $\mathcal{T}$ nad riječi w ako ona postoji. Tada možemo ispitivanjem univerzalnosti jezika od r utvrditi postojanje prihvatljive tablice izračunavanja, tj. vrijedi $\langle r \rangle \in \mathrm{coUNIV}_{\mathrm{REG}}$ ako i samo ako $w \in A$.

Kako bi tablica izračunavanja mogla biti tretirana kao riječ trebamo ju transformirati u niz, što radimo nadovezujući retke tablice jedne na druge. Pritom koristimo simbol $\#$, koji smo dodali u alfabet, kako bi odvojili retke tablice. Regularni izraz r unija je tri različita regularna izraza koji osiguravaju da se u $L(r)$ nalazi sve osim prihvatljive tablice računanja:

$$r = r_{los\_pocetak} \cup r_{los\_prijelaz} \cup r_{los\_kraj}.$$

Izrazom $r_{los\_pocetak}$ obuhvaćamo sve riječi w koje ne sadrže ispravni zapis prvog retka tablice izračunavanja. Predstavljamo ga kao uniju regularnih izraza koji će izražavati da u i-tom stupcu prvog retka tablice ne piše ispravan znak:

$$r_{los\_pocetak} = s_0 \cup s_1 \cup \ldots \cup s_n \cup s_{n+1} \cup \ldots \cup s_{n^k} \cup s_{\#}.$$

U prvom stupcu prvog retka tablice treba se nalaziti znak koji predstavlja početno stanje Turingovog stroja $\mathcal{T}$. Sve riječi koje ne zadovoljavaju taj uvjet obuhvaćamo izrazom $s_0 = \Delta_{-q_0}\Delta^*$, pri čemu sa Δ_{-s} označavamo uniju svih simbola alfabeta Δ osim simbola s. Kako sljedećih n stupaca prvog retka sadrži riječ w, korištenjem izraza $s_i = \Delta^i\Delta_{-w_i}\Delta^*, i \in \{1, 2, \ldots, n\}$, osiguravamo da se prihvaćaju zapisi tablice izračunavanja koji ne sadrže w u prvom retku. Pritom, s Δ^i označavamo konkatenaciju i izraza predstavljenih simbolom Δ. Potrebno je još definirati regularne izraze kojima se izražava da neki od preostalih stupaca ne sadrži prazno mjesto $\textvisiblespace$, koje bi trebao sadržavati, s izrazom $s_i = \Delta^i\Delta_{-\textvisiblespace}\Delta^*, i \in \{n+1, n+2, \ldots, n^k\}$. Također, definiramo regularni izraz koji kaže da se na $n^k + 1$ mjestu riječi ne nalazi znak $\#$ kojim razdvajamo retke tablice s $s_{\#} = \Delta^{n^k+1}\Delta_{-\#}\Delta^*$.

U ispravnoj tablici računanja svaki redak tablice treba slijediti iz prethodnog po pravilima funkcije prijelaza Turingovog stroja $\mathcal{T}$. Kako bi ispitali slijedi li neki niz od tri susjedna znaka def iz prethodnog retka potrebno je promatrati samo tri znaka abc koja se u prethodnom retku nalaze direktno iznad znakova def. Zbog toga možemo definirati regularni izraz kojim osiguravamo da riječ sadrži barem jedan nedozvoljeni prijelaz između redaka:

$$r_{los\_prijelaz} = \bigcup_{los(abc,def)} \Delta^* abc \Delta^{n^k-1} def \Delta^*,$$

gdje $los(abc, def)$ označava da ne možemo dobiti niz znakova def iz niza abc korištenjem funkcije prijelaza Turingovog stroja $\mathcal{T}$. Konačno, definiramo regularni izraz kojim izražavamo da Turingov stroj $\mathcal{T}$ ne završava u neprihvatljivom stanju q_{NE}:

$$r_{los\_kraj} = \Delta^*_{-q_{\mathrm{NE}}}.$$

Svi dijelovi regularnog izraza r imaju strukturu koja sadrži značajnu količinu ponavljanja u sebi te su najviše polinomno veći od duljine ulazne riječi w. Pritom nam za brojenje do n^k treba $\mathcal{O}(\log n)$ prostora jer brojače možemo zapisati u binarnom zapisu. Za pohranjivanje ostalih podataka potrebnih Turingovom stroju $\mathcal{T}$ nam također treba samo logaritamski prostor i zbog toga možemo konstruirati regularni izraz r u logaritamskom prostoru. Iz toga slijedi da je jezik $\mathrm{coUNIV_{REG}}$ PSPACE-težak. Q.E.D.

Korolar 2.29. *Jezik* $\mathrm{coUNIV}_{\mathrm{REG}}$ *je* PSPACE*-potpun.*

Tvrdnja prethodnog korolara vrijedi i za nedeterminističke konačne automate i jezik $\mathrm{coUNIV}_{\mathrm{NKA}}$, što dobivamo korištenjem redukcije iz teorema 2.27. Kako ćemo tu tvrdnju koristiti u nastavku da dokažemo PSPACE-potpunost nekih drugih jezika, iskazujemo ju sljedećim korolarom.

Korolar 2.30. *Jezik* $\mathrm{coUNIV}_{\mathrm{NKA}}$ *je* PSPACE*-potpun.*

3. Upiti nad grafovima s podatcima

Grafovi s podatcima pojavili su kao prirodan način predstavljanja baza podataka u kojima su objekti povezani u mrežnu strukturu različitim međusobnim vezama. Pritom se objekti predstavljaju kao čvorovi grafa, dok se odnosi između objekata predstavljaju usmjerenim vezama koje sadrže opis tipa veze, odnosno labelu. Opisujući bazu podataka s takvim grafom čuvamo sve odnose među objektima, ali gubimo informacije o njihovim svojstvima koja nas također mogu zanimati. Zbog toga je još potrebno svakom objektu, odnosno vrhu, dodijeliti podatke koji će opisivati njegova svojstva. U ovom radu promatra se situacija kada vrhovi sadrže samo jedan podatak, dok se slučaj s više podataka može lako modelirati korištenjem dodatnih čvorova i veza među njima. Pregled prethodno opisanog i još nekih drugih modela baza podataka koji koriste grafove može se naći u (Angles i Gutierrez, 2008). Opisani model baze podataka zovemo grafom s podatcima te ga formalno definiramo u nastavku (Libkin i Vrgoč, 2012a).

Definicija 3.1 (Grafovi s podatcima)**.** *Neka su zadani konačni alfabet Σ i prebrojiv skup podataka $\mathcal{D}$.* Graf s podatcima nad Σ i $\mathcal{D}$ *je trojka $G = (V, E, \rho)$ gdje je:*

- *V konačan skup čije elemente zovemo vrhovima,*
- *$E \subseteq V \times \Sigma \times V$ je skup čije elemente zovemo označenim bridovima i*
- *$\rho : V \to \mathcal{D}$ je funkcija koja svakom elementu iz V pridjeljuje podatak iz $\mathcal{D}$.*

Kao što je i prije spomenuto, u grafu s podatcima najviše nas zanimaju odnosi među vrhovima. Kako neposredne odnose između vrhova predstavljamo bridovima, do složenijih odnosa dolazimo praćenjem više bridova. Ideju praćenja više bridova kako bi povezali dva vrha formalno definiramo kao put u grafu s podatcima. Definicije sljedeća tri pojma koja su vezana uz put u grafu preuzeta su iz (Libkin i Vrgoč, 2012a).

Definicija 3.2 (Put)**.** *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$, te neka je zadan graf s podatcima G nad Σ i $\mathcal{D}$.* Put *u grafu G između vrhova v_1 i v_n je niz*

$$\pi_{v_1 v_n} = v_1 a_1 v_2 a_2 v_3 \ldots v_{n-1} a_{n-1} v_n,$$

takav da je svaka trojka $(v_i, a_i, v_{i+1}), i < n$, iz skupa bridova E grafa G.

Kada postavljamo upite nad bazama podataka obično nas više zanimaju svojstva objekata, odnosno podatci vrhova, nego konkretni vrhovi. Dodatno, i same oznake vrhova možemo promatrati kao jedan tip podatka koji taj vrh sadrži tako da se podatci vrha mogu promatrati kao proširenje pojma identifikatora vrha. Zbog toga uvodimo dodatni pojam podatkovnog puta u kojem se vrhovi zamjenjuju svojim podatcima. Uvođenjem podatkovnih puteva pratimo rad (Libkin i Vrgoč, 2012a) koji se razlikuje od uobičajene literature u kojoj se koristi sličan pojam podatkovnih riječi.

Definicija 3.3 (Podatkovni put)**.** *Neka je zadan graf s podatcima G nad alfabetom Σ i skupom podataka $\mathcal{D}$. Svakom putu $\pi_{v_1v_n} = v_1a_1v_2 \dots a_{n-1}v_n$ u grafu pridjeljuje se pripadni* podatkovni put

$$w_{\pi_{v_1v_n}} = \rho(v_1)a_1\rho(v_2)a_2\rho(v_3) \dots \rho(v_{n-1})a_{n-1}\rho(v_n),$$

koji se sastoji od niza alternirajućih podataka i znakova alfabeta, pri čemu početni i završni elementi niza moraju biti neki podatci. Skup svih podatkovnih puteva nad Σ i $\mathcal{D}$ označavamo sa $\Sigma[\mathcal{D}]^*$.

Iako smo posebnu pažnju dali uvođenju podataka u grafu, ponekad nas zanimaju samo odnosi među vrhovima bez obzira na njihove podatke. U slučaju takvih upita jednostavnije je promatrati samo niz oznaka koji preostaje izbacivanjem vrhova iz puta, odnosno podatkovnog puta. Takav niz oznaka nazivamo labelom puta.

Definicija 3.4 (Labela puta)**.** *Neka je zadan graf s podatcima G nad alfabetom Σ i skupom podataka $\mathcal{D}$.* Labela puta *λ je funkcija koja svakom putu $\pi_{v_1v_n} = v_1a_1 \dots a_{n-1}v_n$ ili pripadnom podatkovnom putu $w_{\pi_{v_1v_n}} = \rho(v_1)a_1 \dots a_{n-1}\rho(v_n)$ pridjeljuje riječ koju dobijemo uklanjanjem vrhova, odnosno podataka, iz puta:*

$$\lambda(\pi_{v_1v_n}) = a_1a_2 \dots a_{n-1} \in \Sigma^*.$$

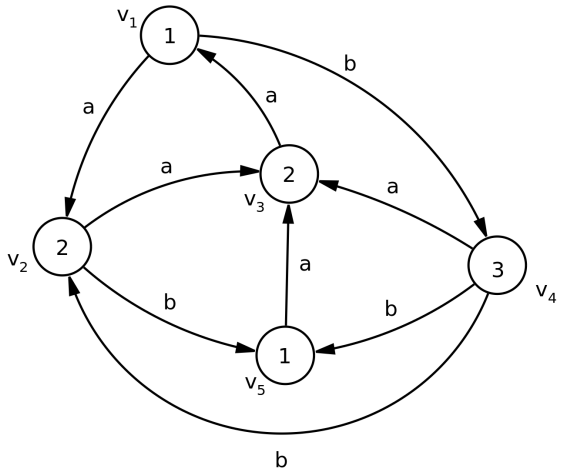
Prethodni pojmovi predstavljaju osnovne pojmove grafova s podatcima te nam omogućavaju da precizno govorimo o tipu veze između vrhova zadanog grafa. S druge strane, prilikom korištenja baza podataka ne želimo analizirati neke postojeće veze, već želimo unaprijed definirati neko svojstvo veze te dohvatiti sve parove vrhova koji su povezani takvom vezom. Zbog toga je potrebno definirati upite nad grafovima s podatcima, čime omogućavamo razlikovanje različitih tipova upita i analizu njihovih svojstava. Kako upiti za određeni graf trebaju vratiti sve parove vrhova povezanih vezom određenog svojstva, definiramo ih kao funkcije koje za zadani graf vraćaju upravo takve parove vrhova. Definicija upita je preuzeta iz (Libkin i Vrgoč, 2012a) uz nekoliko manjih modifikacija.

Definicija 3.5 (Upiti nad grafovima s podatcima)**.** *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$, te prebrojiv skup V. S $\mathcal{G}$ označimo skup svih grafova s podatcima nad Σ i $\mathcal{D}$ takvih da su vrhovi svih grafova G iz $\mathcal{G}$ elementi skupa V.*

Za zadani jezik $L \subseteq \Sigma[\mathcal{D}]^*$, funkciju $Q : \mathcal{G} \to P(V \times V)$ nazivamo upitom nad grafom s podatcima*, u oznaci $Q = x \xrightarrow{L} y$, ako svakom grafu $G \in \mathcal{G}$ pridjeljuje skup parova vrhova (v, v') takvih da vrijedi:*

$$(v, v') \in Q(G) \text{ ako i samo ako postoji podatkovni put } w_{\pi_{vv'}} \in L \text{ u grafu } G \text{ koji povezuje vrhove } v \text{ i } v'.$$

Primjer 3.6. *Neka su zadani alfabet $\Sigma = \{a, b\}$ i skup podataka $\mathcal{D} = \mathbb{N}$. Slika 3.1 prikazuje jedan primjer grafa s podatcima $G = (V, E, \rho)$ nad Σ i $\mathcal{D}$. Vrhovi grafa $V = \{v_1, v_2, \ldots, v_5\}$ prikazani su kružnicama, pri čemu su podatci grafa zapisani u kružnicama. Označeni bridovi grafa predstavljeni su strelicama s oznakom zapisanom pored strelice.*



Slika 3.1: Primjer grafa s podatcima.

Put $\pi_{v_5v_4} = v_5 a v_3 a v_1 b v_4$ predstavlja jedan primjer puta između vrhova v_5 i v_4, pri čemu je pripadni podatkovni put $w_{\pi_{v_5v_4}} = 1a2a1b3$, a labela puta $\lambda(\pi_{v_5v_4}) = aab$. Promatrajmo jezik $L_{eq} \subseteq \Sigma[\mathcal{D}]^*$ svih podatkovnih puteva koji sadrže dva jednaka podatka *te pripadni upit nad grafovima s podatcima $Q = x \xrightarrow{L_{eq}} y$. Prema primjeru puta $\pi_{v_5v_4}$ slijedi da je par vrhova (v_5, v_4) u rezultatu upita $Q(G)$, jer put sadrži vrhove v_5 i v_1 koji imaju isti podatak 1. Odnosno, u slučaju grafa G sa slike za svaki par vrhova (v, v') vrijedi da su rezultat upita $Q(G)$. To je posljedica toga da iz svakog vrha postoji put do svih preostalih vrhova. Zbog toga uvijek možemo pronaći put između dva vrha koji sadrži ciklus te onda i pripadni podatkovni put ima barem dva ista podatka.*

Uvođenjem grafova s podatcima definirali smo izgled željene baze podataka, dok smo s upitima definirali funkciju koja određuje sve parove vrhova između kojih postoji podatkovni put sa željenim svojstvom. Upiti kako su trenutno definirani ovise o zadanom jeziku te se za njihovo definiranje trebamo oslanjati na neki opis traženog jezika. Opisi upita korištenjem prirodnog jezika nisu pogodni za analizu njihovih svojstava i njihovo korištenje u računalu. Zbog toga definiramo formalizme za definiranje upita. Općenito, formalizmi nam omogućavaju da korištenjem strogo definiranih objekata i njima pridruženih jezika odredimo koji su tipovi upita uopće dozvoljeni. Uz tu definiciju možemo odrediti i neka ograničenja na ekspresivnost u opisu jezika korištenog u upitu uz koja će evaluacija upita biti efikasnija.

Definicija 3.7 (Formalizam za definiranje upita)**.** *Neka su zadani konačni alfabet* Σ *i prebrojiv skup podataka* $\mathcal{D}$*.* Formalizam $\mathcal{F} = (\mathcal{E}, L)$ *nad* Σ *i* $\mathcal{D}$ *je uređeni par pri čemu je:*

- $\mathcal{E}$ *prebrojiv skup i*
- $L : \mathcal{E} \to P(\Sigma[\mathcal{D}]^*)$ *funkcija koja svakom elementu iz* $\mathcal{E}$ *pridružuje neki jezik* $L' \subseteq \Sigma[\mathcal{D}]^*$*.*

Za element $e \in \mathcal{E}$ *jezik* $L(e)$ *nazivamo* jezikom elementa e*. Za formalizam* $\mathcal{F}$ *kažemo da može* izraziti *jezik* $L' \subseteq \Sigma[\mathcal{D}]^*$ *ako postoji element* $e \in \mathcal{E}$ *takav da je* $L' = L(e)$*.*

U nastavku poistovjećujemo formalizam $\mathcal{F}$ s njegovim elementima $\mathcal{E}$, koje ćemo ovisno o kontekstu nazivati izrazima, automatima, rečenicama, itd. Funkcija L koja će povezivati elemente s jezicima bit će implicitna iz konteksta i elemenata formalizma $\mathcal{F}$. Sada kada imamo definiciju formalizma možemo ju povezati s upitima te definirati što znači imati upit zadan u formalizmu $\mathcal{F}$.

Definicija 3.8 (Upiti nad grafovima s podatcima u formalizmu $\mathcal{F}$)**.** *Neka su zadani alfabet* Σ *i skup podataka* $\mathcal{D}$*, te formalizam* $\mathcal{F}$ *nad* Σ *i* $\mathcal{D}$*. Svakom elementu* $e \in \mathcal{E}$ *formalizma* $\mathcal{F}$ *pridružujemo upit* $Q = x \xrightarrow{L'} y$*, pri čemu je* L' *jednak jeziku* $L(e)$ *elementa* e*. Takve upite još označavamo s* $Q = x \xrightarrow{e} y$ *i kažemo da je* upit Q zadan u formalizmu $\mathcal{F}$*. Za formalizam* $\mathcal{F}$ *kažemo da može* izraziti *neki upit* $Q = x \xrightarrow{L'} y$ *ako formalizam može izraziti pripadni jezik* L'*.*

Definicija formalizma je općenita i može obuhvatiti prilično različite načine definiranja elemenata i pripadnih jezika. U nastavku su dana dva primjera formalizama za definiranje upita. Prvi se formalizam sastoji od jednostavnih elemenata o kojima je već

bilo riječ u ovom radu: regularnim izrazima. Korištenjem običnih regularnih izraza ne možemo izraziti uvjete nad podatcima te pitanje jesu li dva vrha rješenje upita ovisi samo o labelama puteva između ta dva vrha.

Primjer 3.9. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Definiramo formalizam $\mathrm{RPQ} = (\mathcal{E}, L)$ tako da su elementi skupa $\mathcal{E}$ svi regularni izrazi REG nad alfabetom Σ kako je zadano definicijom 2.26. Označimo s L_r regularni jezik pridijeljen regularnom izrazu r. Tada funkcija L pridjeljuje svakom regularnom izrazu r jezik $L(r) \subseteq \Sigma[\mathcal{D}]^*$ podatkovnih puteva čije labele su iz L_r:*

$$L(r) = \{w_\pi \in \Sigma[\mathcal{D}]^* \mid \lambda(w_\pi) \in L_r\}.$$

Primjerice, regularnim izrazima možemo izraziti upit koji kaže da su dva vrha povezana putem čiji svi bridovi imaju oznaku a, osim zadnjeg koji ima oznaku b. Regularni izraz koji iskazuje takav jezik je $r_1 = a^*b$ te je željeni upit $Q_1 = x \xrightarrow{r_1} y$. Put $\pi_{v_5v_4} = v_5av_3av_1bv_4$ iz primjera 3.6 je jedan put čiji je pripadni podatkovni put element jezika $L(r_1)$ te je i par vrhova (v_5, v_4) element upita $Q_1(G)$. Primijetimo da se upit $Q = x \xrightarrow{L_{eq}} y$ iskazan u primjeru 3.6 ne može iskazati regularnim izrazima, jer nam regularni izrazi ne govore ništa o podatcima. Premda oba upita sadrže par (v_5, v_4), kao jedan primjer gdje se oni razlikuju je par vrhova (v_1, v_3). Taj par nije u upitu Q_1 jer svi ulazni bridovi od v_3 imaju labelu a pa zadnja labela puta ne može biti b, dok smo u primjeru 3.6 da su svi parovi vrhova u upitu Q.

Najjednostavniji primjer upita kojeg možemo izraziti je postojanje proizvoljne veze između dva vrha. Takav upit $Q_2 = x \xrightarrow{r_2} y$ lako se izražava regularnim izrazom $r_2 = \Sigma^*$, gdje sa Σ podrazumijevamo uniju svih znakova a iz Σ. Računanje tog upita svodi se na rješavanje pitanja dostižívosti u zadanom grafu s podatcima G.

Segoufin (2006) opisuje više formalizama u kojima je moguće izraziti uvjete nad podatcima među kojima su fragment logike prvog reda i konačni automati s registrima. Kao drugi primjer formalizma u nastavku je opisan fragment logike prvog reda, dok su konačni automati s registrima detaljno obrađeni u sljedećem poglavlju. Opis korištenog fragmenta logike prvog reda je nešto drugačiji od opisa iz (Segoufin, 2006), jer se u tom radu promatraju podatkovne riječi, dok su ovdje promatrani podatkovni putovi.

Primjer 3.10. *Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$. Skup elemenata formalizma $\mathrm{FO}(\sim, <, +1)$ čine rečenice fragmenta logike prvog reda s tri binarne relacije $\sim^2$, $<^2$ i $+1^2$, unarnom relacijom d, te po jednom unarnom relacijom a^1 za svaki simbol alfabeta $a \in \Sigma$. Prilikom pridruživanja nekog jezika $L' \subseteq \Sigma[\mathcal{D}]^*$ rečenici ϕ, varijable*

rečenice ćemo interpretirati kao pozicije znaka ili podatka u podatkovnom putu. Za svaki $a \in \Sigma$ *relacija* $a(x)$ *je zadovoljena ako se na poziciji* x *nalazi znak* a*, dok s relacijom* $d(x)$ *označavamo da se na poziciji* x *nalazi proizvoljan podatak. Relacije* $x < y$ *i* $y = x + 1$ *su uobičajene relacije poretka i sljedbenika, dok dvije pozicije* x *i* y *zadovoljavaju relaciju* $x \sim y$ *ako sadrže isti podatak, odnosno znak. Uz takvu interpretaciju jezik* $L(\phi)$ *pridružen rečenici* ϕ *sastoji se od svih podatkovnih puteva* $w \in \Sigma[\mathcal{D}]^*$ *koji zadovoljavaju rečenicu* ϕ*.*

U primjeru 3.9 komentirali smo kako regularni izrazi ne mogu poslužiti pri definiranju upita L_{eq} *kojim iskazujemo da podatkovni put ima dva ista podatka. Promotrimo sada kako bi takav upit izrazili u logici prvog reda te stoga definiramo rečenicu:*

$$\phi_1 \equiv \exists x \, \exists y \, (((x < y) \vee (y < x)) \wedge d(x) \wedge (x \sim y)) \, .$$

Tom rečenicom iskazujemo da postoje dvije različite pozicije, jer x *mora biti ili strogo manje ili strogo veće od* y*, takve da je na prvoj poziciji podatak. Također, zbog* $(x \sim y)$ *mora vrijediti da je na pozicijama* x *i* y *jednak podatak. Vrijedi da jezik kojeg definira ta rečenica odgovara jeziku* L_{eq} *te definiramo upit* $Q_1 = x \xrightarrow{\phi_1} y$*. S druge strane, primijetimo da negiranjem prethodne rečenice* $\phi_2 \equiv \neg\phi_1$ *dobivamo drugi upit* $Q_2 = x \xrightarrow{\phi_2} y$ *koji iskazuje da su* svi podatci na putu različiti*. Taj upit bit će nam od posebne važnosti u nastavku ovog poglavlja te ćemo* jezik tog upita označavati s $\overline{L_{eq}}$.

Osim ekspresivnosti formalizma u iskazivanju različitih upita, posebno će nas zanimati u kojoj složenost možemo evaluirati upite formalizma. Kako bi to mogli formalno izraziti potrebno je definirati jezike čiju ćemo složenost ispitivati, a koji su povezani s intuitivnom idejom evaluacije upita. Pritom trebamo nekako kodirati grafove s podatcima kako bi ih mogli predstaviti kao ulaze Turingovog stroja. Vrhove grafa i veze između njih kodiramo slično kao što smo kodirali stanja i prijelaze konačnih automata u definiciji 2.19. Podatcima grafa pridružimo prirodne brojeve, što je moguće jer je skup podataka prebrojiv, te ih kodiramo kao niz parova vrha i pripadnog podatka. Također, prilikom utvrđivanja složenosti evaluacije upita trebamo naći neko kodiranje željenog formalizma, ali ono ovisi o samom formalizmu. Prvo definiramo kombiniranu složenost u kojoj se kao ulazi Turingovog stroja dovode i elementi formalizma. Definicije sljedećih složenosti uobičajene su definicije koje su uz manje modifikacije preuzete iz (Vardi, 1982).

Definicija 3.11 (Kombinirana složenost evaluacije upita)**.** *Neka su zadani alfabet* Σ *i skup podataka* $\mathcal{D}$*, te formalizam* $\mathcal{F} = (\mathcal{E}, L)$ *nad* Σ *i* $\mathcal{D}$*. Za formalizam* $\mathcal{F}$*,* kombiniranom složenosti evaluacije upita formalizma $\mathcal{F}$ *nazivamo klasu složenosti* $\mathcal{C}$ *kojoj*

pripada jezik

$$L_{cc} = \left\{ \langle e, G, v_s, v_t \rangle \mid \textit{za upit } Q = x \xrightarrow{e} y \textit{ vrijedi } (v_s, v_t) \in Q(G) \right\},$$

pri čemu je e element formalizma $\mathcal{F}$, G neki graf s podatcima, a v_s i v_t dva vrha iz G.

Jednostavniji slučaj je kad pretpostavimo da je upit unaprijed zadan. Tada kao ulaz Turingovog stroja imamo samo graf s podatcima i par vrhova te tu složenost zovemo podatkovnom složenosti.

Definicija 3.12 (Podatkovna složenost evaluacije upita)**.** *Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$, te neki jezik podatkovnih puteva $L \subseteq \Sigma[\mathcal{D}]^*$.* Podatkovna složenost evaluacije upita $Q = x \xrightarrow{L} y$ *je klasa složenosti $\mathcal{C}$ kojoj pripada jezik*

$$L_{dc,Q} = \{ \langle G, v_s, v_t \rangle \mid (v_s, v_t) \in Q(G) \},$$

pri čemu je G neki graf s podatcima, a v_s i v_t dva vrha iz G.

Za formalizam $\mathcal{F}$ kažemo da je podatkovna složenost evaluacije upita formalizma $\mathcal{F}$ u klasi $\mathcal{C}$ ako se za svaki upit $Q = x \xrightarrow{e} y$, kojeg formalizam može izraziti, jezik $L_{dc,Q}$ nalazi u klasi $\mathcal{C}$. Dodatno, ako formalizam može izraziti upit $Q = x \xrightarrow{e} y$ čija je podatkovna složenost $\mathcal{C}$-potpuna, kažemo da je i podatkovna složenost evaluacije upita formalizma $\mathcal{F}$ također $\mathcal{C}$-potpuna.

Korištenjem prethodnih definicija složenosti možemo govoriti o efikasnosti evaluacije upita te ju uzeti u obzir prilikom izbora formalizma kojeg ćemo koristiti. U slučaju da imamo upit koji govori da postoji proizvoljna veza između dva vrha lako možemo utvrditi njegovu podatkovnu složenost. Taj slučaj detaljnije promatramo u sljedećem primjeru.

Primjer 3.13. *Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$, te upit $Q_2 = x \xrightarrow{r_2} y$ uz $r_2 = \Sigma^*$ kao u primjeru 3.9. Taj upit nam govori da postoji proizvoljan put između vrhova v_s i v_t u grafu G. Odnosno, jezik $L_{dc,Q}$ podatkovne složenosti odgovara o problemu dostiživosti u grafu. Pritom je dostiživost u grafu uobičajeni* NL*-potpuni problem, a dokaz toga može se naći u svakoj knjizi o složenosti, primjerice (Sipser, 2006). Zbog toga je i podatkovna složenost evaluacije upita $Q_2 = x \xrightarrow{r_2} y$ također* NL*-potpuna.*

S prethodim smo primjerom dobili donju granicu za podatkovnu složenost formalizama, jer najmanji zahtjev koji možemo imati na formalizam je da može izraziti postojanje proizvoljne veze između vrhova. To se odnosi i na formalizme u ovom radu, pa

dokazivanjem da je podatkovna složenost evaluacije upita nekog formalizma $\mathcal{F}$ u klasi NL dokazali smo da je ta složenost i NL-potpuna. Bez obzira što smo s tim primjerom dobili donju granicu podatkovne složenosti za sve razumne formalizme, trebamo pronaći dodatna svojstva formalizama koja će nam omogućiti da eliminiramo formalizme s većom složenosti. U nastavku definiramo jedno takvo svojstvo koje nam omogućava eliminaciju formalizama, ali prvo trebamo definirati pomoćni pojam: zatvorenost formalizma za komplement.

Definicija 3.14 (Zatvorenost formalizma za komplement)**.** *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$ te formalizam $\mathcal{F} = (\mathcal{E}, L)$ nad njima. Za formalizam $\mathcal{F}$ kažemo da je zatvoren za komplement ako za svaki jezik L' koji $\mathcal{F}$ može izraziti, slijedi da $\mathcal{F}$ može izraziti i njegov komplement $\Sigma[\mathcal{D}]^* \setminus L$.*

Određeni formalizam ćemo eliminirati iz promatranja ako može izraziti upit za koji smo dokazali da mu je podatkovna složenost prevelika, npr. NP-teška. Jezik na temelju kojeg definiramo takav upit smo već prikazali u primjeru 3.10, gdje je on bio jezik drugog upita iz primjera. Naglasimo ovdje još jednom definiciju tog jezika:

$$\overline{L_{eq}} = \{w_\pi \in \Sigma[\mathcal{D}]^* \mid \text{svi podatci u podatkovnom putu } w_\pi \text{ su različiti}\},$$

i pripadnog upita $Q = x \xrightarrow{\overline{L_{eq}}} y$ čija je podatkovna složenost NP-teška, što je dokazano u nastavku ovog poglavlja.

Kako bi dokazali NP-težinu podatkovne složenosti upita $Q = x \xrightarrow{\overline{L_{eq}}} y$ provest ćemo redukciju s prethodno poznatog NP-potpunog problema. Problem koji koristimo kaže da običan graf sadrži dva puta između dva različita para vrhova, takve da ti putovi nemaju zajedničkih vrhova:

$$2-\mathsf{NEZAVISNA}-\mathsf{PUTA} = \{\langle G, v_{s_1}, v_{t_1}, v_{s_2}, v_{t_2}\rangle \mid \text{postoje putovi } \pi_{v_{s_1}v_{t_1}} \text{ i } \pi_{v_{s_2}v_{t_2}} \text{ takvi da je } \pi_{v_{s_1}v_{t_1}} \cap \pi_{v_{s_2}v_{t_2}} = \emptyset\},$$

čija je NP-potpunost dokazana u (Fortune et al., 1980). Redukcija tog problema na jezik podatkovne složenosti evaluacije upita $Q = x \xrightarrow{\overline{L_{\text{eq}}}} y$ provodi se udvostručavanjem izvornog grafa te korištenjem podataka kako bi se osigurala nezavisnost puteva. Formalni dokaz preuzet je iz (Libkin i Vrgoč, 2012a) te je dan sljedećom propozicijom.

Propozicija 3.15. *Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$, te jezik $\overline{L_{eq}}$. Podatkovna složenost evaluacije upita $Q = x \xrightarrow{\overline{L_{eq}}} y$ je* NP*-teška.*

Dokaz. Prilikom dokazivanja NP-težine podatkovne složenosti upita $Q = x \xrightarrow{\overline{L_{eq}}} y$ provodimo redukciju s NP-potpunog problema $2-\mathsf{NEZAVISNA}-\mathsf{PUTA}$. Neka je zadana neka instanca $\langle G, v_{s_1}, v_{t_1}, v_{s_2}, v_{t_2}\rangle$ problema $2-\mathsf{NEZAVISNA}-\mathsf{PUTA}$. Graf u toj instanci je običan graf $G = (V, E)$, pri čemu je skup vrhova $V = \{v_1, v_2, \ldots, v_n\}$, a bridovi nisu označeni, odnosno $E \subseteq V \times V$. Provodimo redukciju te instance na instancu $\langle G', v'_s, v'_t\rangle$ jezika $L_{dc,Q}$ podatkovne složenosti upita Q, gdje je $G' = (V', E', \rho)$ graf s podatcima nad alfabetom Σ i skupom podataka $\mathcal{D}$. Pritom se alfabet sastoji od samo jednog znaka $\Sigma = \{a\}$, a skup podataka je skup prirodnih brojeva $\mathcal{D} = \mathbb{N}$.

Novi graf s podatcima G' sastoji se od dvije kopije izvornog grafa povezane s jednim bridom između vrhova koji odgovaraju vrhovima v_{t_1} i v_{s_2} izvornog grafa. Prvo, vrhove novog grafa dobivamo kao uređene parove starih vrhova i indeksa 1 ili 2 koji označava o kojoj kopiji izvornog grafa se radi:

$$V' = (V \times \{1\}) \cup (V \times \{2\}).$$

Svaka kopija izvornog grafa sadrži jednake bridove uz dodatni brid koji spaja te dvije kopije preko vrhova $(v_{t_1}, 1)$ i $(v_{s_2}, 2)$:

$$E' = \{((v_i, k), a, (v_j, k)) \mid (v_i, v_j) \in E \text{ i } k \in \{1, 2\}\} \cup \{((v_{t_1}, 1), a, (v_{s_2}, 2))\}.$$

Podatci koji se pridružuju vrhovima novog grafa predstavljaju identifikatore vrhova izvornog grafa:

$$\rho((v_i, k)) = i, \quad \text{za } k \in \{1, 2\}.$$

Za potpuno definiranje instance jezika $L_{dc,Q}$ potrebno je još definirati vrhove v'_s i v'_t Vrhovi v'_s i v'_t nalaze se u različitim kopijama grafa i predstavljaju početak i kraj različitih disjunktnih puteva izvornog grafa, odnosno $v'_s = (v_{s_1}, 1)$ i $v'_t = (v_{t_2}, 2)$.

Dokažimo da u grafu G postoje dva disjunktna puta između parova vrhova (v_{s_1}, v_{t_1}) i (v_{s_2}, v_{t_2}) ako i samo ako u grafu G' postoji put između vrhova v'_s i v'_t u čijem su pripadnom podatkovnom putu svi podatci različiti. Pretpostavimo da postoje dva disjunktna puta π_1 i π_2 u grafu G između parova vrhova (v_{s_1}, v_{t_1}) i (v_{s_2}, v_{t_2}). U grafu G' put π' dobivamo tako da prvo slijedimo put π_1 u prvoj kopiji grafa, prijeđemo brid $((v_{t_1}, 1), a, (v_{s_2}, 2))$ i onda slijedimo put π_2 u drugoj kopiji grafa. Zbog disjunktnosti puteva π_1 i π_2 slijedi da za svaki vrh v izvornog grafa G put π' sadržava najviše jedan od vrhova $(v, 1)$ i $(v, 2)$. Iz toga slijedi da pripadni podatkovni put $w_{\pi'}$ između v'_s i v'_t ima sve podatke različite, odnosno $(v'_s, v'_t) \in Q(G')$.

Pretpostavimo sada da vrijedi $(v'_s, v'_t) \in Q(G')$, pri čemu je $w_{\pi_{v'_s v'_t}}$ podatkovni put koji pripada jeziku $\overline{L_{eq}}$. Dokažimo da postoje disjunktni putovi između parova vrhova

(v_{s_1}, v_{t_1}) i (v_{s_2}, v_{t_2}) izvornog grafa G. Kako put $\pi_{v'_s v'_t}$ povezuje vrhove $v'_s = (v_{s_1}, 1)$ i $v'_t = (v_{t_2}, 2)$ slijedi da mora sadržavati brid $((v_{t_1}, 1), a, (v_{s_2}, 2))$ jer jedino taj brid povezuje dvije kopije grafa. Također, zbog različitosti podataka na putu vrijedi da nijedna kopija vrha v izvornog grafa nije obiđena više od jednom. Rastavimo put $\pi_{v'_s v'_t}$ na dvije komponente π_1 i π_2, pri čemu je π_1 dio puta prije brida $((v_{t_1}, 1), a, (v_{s_2}, 2))$, a π_2 dio puta poslije tog brida. Tada komponenti puta π_1 odgovara put π'_1 između vrhova v_{s_1} i v_{t_1} u izvornom grafu, a komponenti π_2 odgovara put π'_2 između vrhova v_{s_2} i v_{t_2} u izvornom grafu. Pritom su putovi π'_1 i π'_2 disjunktni što dokazuje tvrdnju.

Opisana redukcija može se provesti u logaritamskom prostoru jer se većinom svodi na prepisivanje zapisa izvornog grafa uz dodavanje podataka svim vrhovima. Podatci stanu u logaritamski prostor jer predstavljaju binarne brojeve od 1 do $|V|$, pri čemu je $|V| = \mathcal{O}(n)$, pa podatci zauzimaju $\mathcal{O}(\log n)$ prostora. Q.E.D.

Korištenjem prethodne propozicije možemo eliminirati formalizme koji mogu izraziti upit $Q = x \xrightarrow{\overline{L_{eq}}} y$, jer će im podatkovna složenost biti prevelika za praktične svrhe. Kao jednostavni korolar dobivamo i drugo svojstvo prema kojem eliminiramo formalizme koje uzima u obzir zatvorenost formalizma za komplement.

Korolar 3.16. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$ te formalizam $\mathcal{F} = (\mathcal{E}, L)$ nad njima. Ako formalizam može izraziti jezik L_{eq} i zatvoren je za komplement, tada je podatkovna složenost evaluacije upita formalizma $\mathcal{F}$ u nekoj natklasi klase* NP.

Upit koji traži parove vrhova takve da na podatkovnom putu između njih postoje barem dva jednaka podatka je jednostavan i očekujemo da će ga svaki razumni formalizam moći izraziti. Korištenjem prethodnog korolara imamo da se većina formalizama koji su zatvoreni za komplement mogu eliminirati iz praktičnih razmatranja. Od formalizama koji su opisani u (Segoufin, 2006) konačni automati s registrima i podatkovni automati su jedini formalizmi za koje vrijedi da nisu zatvoreni za komplement. Prednost konačnih automata s registrima u odnosu na podatkovne automate je što su bliskiji dobro istraženom pojmu nedeterminističkih konačnih automata. Zbog toga su konačni automati s registrima, njihovi regularni izrazi i njihova svojstva daljnja tema ovog rada.

4. Konačni automati s registrima i proširenje regularnih izraza

4.1. Konačni automati s registrima

Slično kao i nedeterministički konačni automati, konačni automati s registrima sastoje se od konačnog skupa stanja i nedeterminističkih prijelaza između stanja. Jedina je razlika u tome što konačni automati s registrima mogu koristiti i podatke koji se pojavljuju u podatkovnom putu. Kako bi mogli iskoristiti podatke, konačni automati s registrima imaju konačan skup registara u koje mogu pohraniti pročitani podatak korištenjem posebnog tipa prijelaza. Također, taj tip prijelaza nam osim pohranjivanja podataka omogućava i usporedbu trenutno pročitanog podatka s podatcima pohranjenim u registrima. Pritom su dozvoljene usporedbe koje se tvore korištenjem logičkih veznika i varijabli koje predstavljaju jednakost pročitanog podatka s podatkom u nekom registru. Formalno, Libkin i Vrgoč (2012a) definiraju za zadani broj korištenih varijabli k skup uvjeta koje možemo izraziti.

Definicija 4.1 (Skup uvjeta)**.** *Neka je zadan prirodan broj k veći od 0 i skup varijabli $X = \{x_1, x_2, \ldots, x_k\}$.* Skup uvjeta $\mathcal{C}_k$ s k varijabli *je skup svih elemenata jezika $L_{\mathcal{C}_k}$ zadanog gramatikom*

$$C \to \epsilon \mid x_i^{=} \mid x_i^{\neq} \mid (C \wedge C) \mid (C \vee C) \mid (\neg C), \quad 1 \leq i \leq k.$$

Prethodna nam definicija daje sintaksu uvjeta koju moramo povezati sa pripadnom semantikom kako bi ju mogli iskoristiti. Pritom se zadovoljavanje uvjeta za logičke veznike definira kao što je uobičajeno u logici (Vuković, 2009), dok se posebno definiraju slučajevi zadovoljavanja izraza $x_i^{=}$, $x_i^{\neq}$ i ϵ. Formalno, neka su zadani prirodan broj k veći od 0, podatak $d \in \mathcal{D}$ i neka k-torka podataka $\tau = (d_1, d_2, \ldots, d_k) \in \mathcal{D}^k$. Zadovoljavanje uvjeta iz $\mathcal{C}_k$ definira se rekurzivno:

– $d, \tau \models \epsilon$ vrijedi za sve $d \in \mathcal{D}$ i $\tau \in \mathcal{D}^k$,

– $d, \tau \models x_i^{=}$ ako i samo ako vrijedi $d = d_i$,

– $d, \tau \models x_i^{\neq}$ ako i samo ako vrijedi $d \neq d_i$,

– $d, \tau \models (c_1 \wedge c_2)$ ako i samo ako vrijedi $d, \tau \models c_1$ i $d, \tau \models c_2$,

– $d, \tau \models (c_1 \vee c_2)$ ako i samo ako vrijedi $d, \tau \models c_1$ ili $d, \tau \models c_2$ i

– $d, \tau \models (\neg c_1)$ ako i samo ako vrijedi $d, \tau \not\models c_1$,

pri čemu su c_1 i c_2 proizvoljni uvjeti iz $\mathcal{C}_k$.

Korištenjem definicije uvjeta definiramo dodatne prijelaze koji nam omogućavaju prihvaćanje podatkovnih puteva. Ti novi prijelazi automata, koji obrađuju podatke, alterniraju s prethodno definiranim prijelazima, koji obrađuju znakove alfabeta, kako bi pratili alterniranje podataka i znakova u podatkovnom putu. Kako podatkovni putovi započinju i završavaju s podatkom, posebno se razlikuje koji tip prijelaza može biti prvi, odnosno zadnji. U nastavku koristimo skraćeni zapis $[k]$ kako bi prikazali skup brojeva $\{1, 2, \ldots, k\}$. Formalna definicija konačnog automata s registrima preuzeta je iz (Libkin i Vrgoč, 2012a).

Definicija 4.2 (Konačni automat s registrima)**.** *Neka je zadan alfabet Σ, skup podataka $\mathcal{D}$, neki prirodan broj k veći od 0 i skup uvjeta s k varijabli $\mathcal{C}_k$.* Konačni automat s k registara *je uređena petorka $\mathcal{A} = (Q, q_0, F, \tau_0, \delta)$ gdje je:*

– *$Q = Q_w \cup Q_d$, gdje su Q_w i Q_d dva konačna disjunktna skupa čije elemente zovemo znakovnim i podatkovnim stanjima,*

– *$q_0 \in Q_d$ je element kojeg nazivamo početnim stanjem,*

– *$F \subseteq Q_w$ je skup čije elemente nazivamo prihvatljivim stanjima,*

– *$\tau_0 \in \mathcal{D}^k$ je k-torka koju zovemo početnom konfiguracijom registara i*

– *$\delta = (\delta_w, \delta_d)$ je uređeni par relacija koje zovemo relacijama prijelaza:*

  - *$\delta_w \subseteq Q_w \times \Sigma \times Q_d$ zovemo znakovnom relacijom prijelaza i*
  - *$\delta_d \subseteq Q_d \times \mathcal{C}_k \times P([k]) \times Q_w$ zovemo podatkovnom relacijom prijelaza.*

Kako je bio i slučaj kod običnih konačnih automata, samim definiranjem automata još nismo odredili kako automat prihvaća određeni podatkovni put. U tu svrhu moramo definirati kako se provodi računanje s konačnim automatima s registrima. U slučaju znakovnih prijelaza to je računanje isto kao i kod običnih konačnih automata, dok u slučaju podatkovnih prijelaza imamo drugačije ponašanje koje ovisi o uvjetima i podatcima u registrima. Formalno, neka su zadani alfabet Σ, skup podataka $\mathcal{D}$, podatkovni put $w = d_0 a_0 d_1 \ldots d_n \in \Sigma[\mathcal{D}]^*$ i konačni automat s registrima $\mathcal{A}$. Konfiguracija

automata $\mathcal{A}$ nad podatkovnim putem w je uređena trojka (j, q, τ) pri čemu je j pozicija simbola u riječi w kojeg automat $\mathcal{A}$ čita, q je trenutno stanje automata, a $\tau \in \mathcal{D}^k$ je trenutna konfiguracija registara. Početna konfiguracija automata je $(0, q_0, \tau_0)$, dok su prihvatljive konfiguracije sve konfiguracije (j, q, τ) gdje je stanje q iz skupa prihvatljivih stanja F. Za podatkovni put w, iz konfiguracije $C = (j, q, \tau)$ možemo prijeći u konfiguraciju $C' = (j + 1, q', \tau')$ ako vrijedi jedno od sljedećeg:

- j-ti simbol podatkovnog puta w je znak a alfabeta Σ i postoji prijelaz $(q, a, q') \in \delta_w$ te vrijedi $\tau' = \tau$ ili
- j-ti simbol podatkovnog puta w je podatak d iz $\mathcal{D}$ i postoji prijelaz $(q, c, I, q') \in \delta_d$ takav da vrijedi $d, \tau \models c$ i konfiguracija registara τ' podudara se s konfiguracijom τ u svim vrijednostima osim što je i-ti element iz τ' jednak d svaki put kada je $i \in I$.

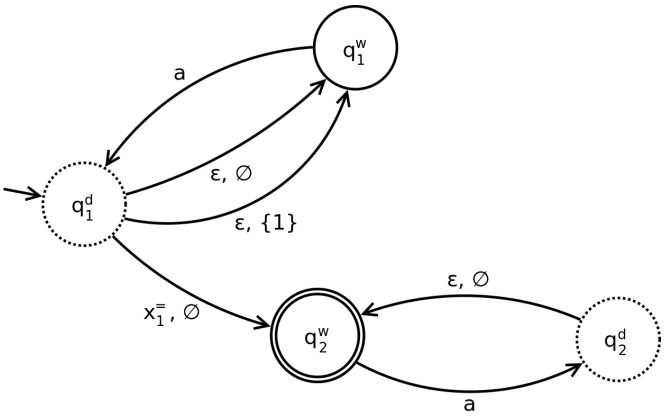
Uz prethodni opis načina računanja automata definiramo prihvaćanje podatkovnog puta, odnosno jezika, za konačne automate s registrima (Libkin i Vrgoč, 2012a).

Definicija 4.3 (Jezik konačnog automata s registrima)**.** *Neka su zadani alfabet Σ, skup podataka $\mathcal{D}$ i konačni automat s registrima $\mathcal{A} = (Q, q_0, F, \tau_0, \delta)$. Za zadani podatkovni put $w \in \Sigma[\mathcal{D}]^*$, niz konfiguracija $(C_0, C_1, \ldots, C_{|w|})$ automata $\mathcal{A}$ nad podatkovnim putem w nazivamo* prihvatljivim nizom konfiguracija *ako je C_0 početna, a $C_{|w|}$ neka prihvatljiva konfiguracija te iz svake konfiguracije $C_i, 0 \leq i < |w|$, možemo prijeći u konfiguraciju C_{i+1}.* Automat $\mathcal{A}$ prihvaća podatkovni put w *ako postoji neki prihvatljivi niz konfiguracija automata $\mathcal{A}$ nad podatkovnim putem w.*

Skup svih podatkovnih puteva koje automat prihvaća nazivamo jezikom automata *$\mathcal{A}$ i označavamo s $L(\mathcal{A})$. Posebno, s $L(\mathcal{A}, \tau, \tau')$ označavamo skup svih podatkovnih putova w_π za koje automat $\mathcal{A}$ s početnom konfiguracijom registara τ može završiti s konfiguracijom registara τ' nakon čitanja podatkovnog puta w_π.*

U prošlom smo poglavlju kod pitanja podatkovne složenosti formalizama detaljnije promatrali jezik L_{eq} podatkovnih putova koji imaju barem dva jednaka podatka. Također, dokazali smo da svi formalizmi koji mogu izraziti taj jezik, a zatvoreni su za komplement, imaju NP-tešku podatkovnu složenost. U nastavku dajemo primjer konačnog automata s registrima koji prihvaća jezik L_{eq} te dokaz da ne postoji konačan automat s registrima koji prihvaća njegov komplement $\overline{L_{eq}}$. Iz toga slijedi da konačni automati s registrima nisu zatvoreni za komplement te ih ne možemo eliminirati na temelju neefikasnosti evaluacije podatkovnih upita korištenjem korolara 3.16. Automat koji prihvaća jezik L_{eq} dan je sljedećim primjerom.

Primjer 4.4. *Slika 4.1 prikazuje konačni automat s registrima $\mathcal{A} = (Q, q_1^d, F, \tau_0, \delta)$ koji prihvaća jezik L_{eq}. Kao i kod nedeterminističkih konačnih automata, stanja automata prikazujemo kružnicama, pri čemu podatkovna stanja $Q_d = \{q_1^d, q_2^d\}$ prikazujemo iscrtkanim kružnicama, a znakovna stanja $Q_w = \{q_1^w, q_2^w\}$ punim kružnicama. Početno i prihvatljiva stanja označavamo jednako kao i kod nedeterminističkih konačnih automata, dok prijelaze razlikujemo na temelju zapisa pored strelice. Automat ima jedan registar $k = 1$, što nije direktno prikazano slikom, jednočlani alfabet $\Sigma = \{a\}$ i proizvoljan skup podataka $\mathcal{D}$. Početna konfiguracija registara je $\tau_0 = (\perp)$, odnosno registar x_1 sadrži podatak $\perp$ za koji znamo da se neće pojaviti u podatkovnom putu.*



Slika 4.1: Konačni automat s registrima koji prihvaća jezik L_{eq}.

Iz prijelaza $(q_1^d, x_1^=, \emptyset, q_2^w)$ automata $\mathcal{A}$ vidljivo je da možemo prijeći u prihvatljivo stanje q_2^w tek kada pročitamo podatak jednak podatku sadržanom u registru x_1. Kako prije toga mora postaviti registar x_1 na neku vrijednost, znamo da automat mora u nekom trenutku iskoristiti prijelaz $(q_1^d, \epsilon, \{1\}, q_1^w)$ umjesto prijelaza $(q_1^d, \epsilon, \emptyset, q_1^w)$. Iz toga slijedi da jedini način na koji automat može prihvatiti podatkovni put je taj da podatkovni put ima na dvije različite pozicije jednake podatke. Također, vrijedi da za svaki podatkovni put koji ima na dvije različite pozicije jednake podatke možemo definirati niz konfiguracija automata u kojem će zadnja konfiguracija biti prihvatljiva. Zbog toga slijedi da automat $\mathcal{A}$ prihvaća jezik L_{eq}.

U sljedećoj propoziciji dajemo dokaz da ne postoji konačni automat s registrima koji prihvaća komplement jezika L_{eq}. Posljedično vrijedi da konačni automati s registrima nisu zatvoreni za komplement.

Propozicija 4.5. *Neka su zadani alfabet Σ i beskonačno prebrojiv skup podataka $\mathcal{D}$. Ne postoji konačni automat s registrima $\mathcal{A}$ koji prihvaća jezik $\overline{L_{eq}}$.*

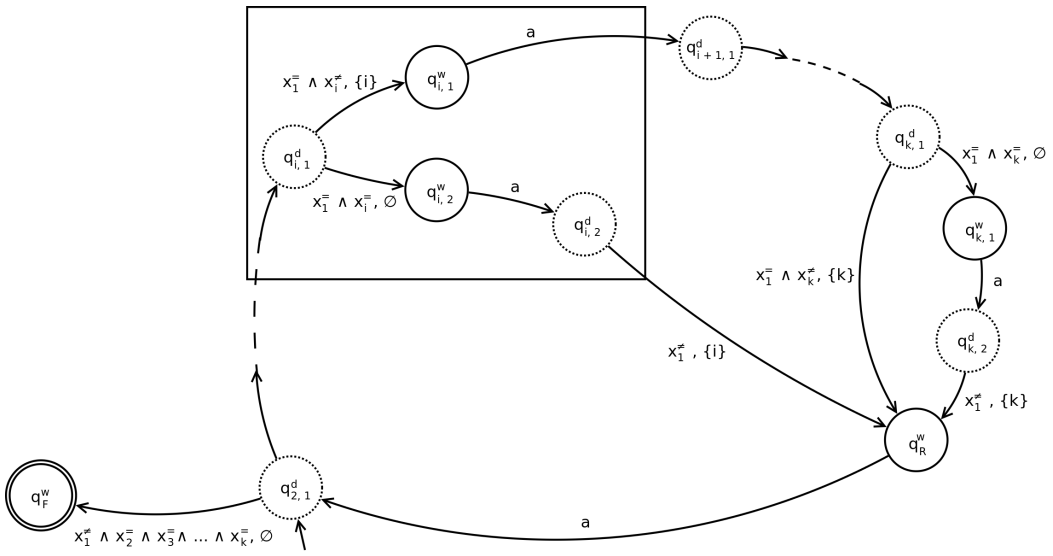
Dokaz. Pretpostavimo suprotno, odnosno da postoji konačni automat $\mathcal{A}$ s k registara koji prihvaća jezik $\overline{L_{eq}}$. Promatrajmo podatkovni put $w_\pi = d_0 a_0 d_1 \dots a_k d_{k+1}$ koji sadrži točno $k+2$ podatka, pri čemu su svi podatci različiti, tj. $d_i \neq d_j$ za $i \neq j$. Ako automat $\mathcal{A}$ ne prihvaća podatkovni put w_π, onda automat ne prihvaća sve podatkovne putove iz $\overline{L_{eq}}$ i dokaz je gotov. Pretpostavimo zbog toga da automat prihvaća podatkovni put w_π.

Promatrajmo prihvatljiv niz konfiguracija $C = (C_0, C_1, \dots, C_l)$ automata $\mathcal{A}$ nad podatkovnim putem w_π. Prilikom prijelaza iz stanja C_{l-1} u C_l automat čita zadnji podatak d_{k+1} podatkovnog puta w_π te je već pročitao $k+1$ različitih podataka. Kako automat ima samo k registara za pohranjivanje podataka, slijedi da postoji neki podatak d_i, $0 \leq i \leq k$, koji se ne pojavljuje u registrima konfiguracije C_{l-1}. Definiramo novi podatkovni put $w'_\pi = d_0 a_0 d_1 \dots a_k d_i$ u kojem smo zamijenili zadnji podatak s podatkom d_i. Tada za podatkovni put w'_π postoji prihvatljiv niz konfiguracija $C' = (C_0, \dots, C'_l)$ koji se podudara s nizom konfiguracija C u svemu, osim u zadnjoj konfiguraciji. Štoviše, zadnja konfiguracija C'_l razlikuje se od konfiguracije C_l samo u sadržaju registara. To vrijedi jer oba podatka d_i i d_{k+2} nisu sadržana u registrima konfiguracije C_{l-1} te zbog toga, zajedno s registrima τ_{l-1} konfiguracije C_{l-1}, zadovoljavaju iste atomarne uvjete $x_i^=$, odnosno $x_i^\neq$. Posljedično, oba podatka zadovoljavaju i iste općenite uvjete, pa onda automat može koristiti isti prijelaz prilikom njihova čitanja. Zbog toga automat $\mathcal{A}$ može prihvatiti i podatkovni put w'_π koji nije u jeziku $\overline{L_{eq}}$ te slijedi da automat $\mathcal{A}$ ne prihvaća jezik $\overline{L_{eq}}$. Q.E.D.

Kroz prethodnih nekoliko definicija i dokaza dali smo formalne osnove konačnih automata s registrima i ispitali neka njihova svojstva. U nastavku ispitujemo još jedno njihovo svojstvo prije nego što prijeđemo na proučavanje automata kao formalizma za definiranje upita. Svojstvo koje proučavamo je složenost ispitivanja je li jezik zadanog automata neprazan, odnosno prihvaća li automat barem jedan podatkovni put.

Ako bi automati koji prihvaćaju barem jedan podatkovni put morali prihvaćati i neki podatkovni put čija je dužina najviše polinomno veća od broja stanja automata, tada bi kao gornju ogradu složenosti ispitivanja je li jezika automata neprazan imali klasu NP. To bi vrijedilo zato jer bi u tom slučaju mogli pogoditi sve podatkovne putove polinomne veličine i onda ispitati prihvaća li automat neki od tih puteva. Ipak, može se dokazati da ta tvrdnja općenito ne vrijedi, odnosno da postoje automati kod kojih je dužina najkraćeg podatkovnog puta kojeg prihvaćaju eksponencijalno veća od broja stanja automata. Takav automat dan je sljedećim primjerom i on nam sugerira da ćemo za rješavanje prethodnog pitanja trebati promatrati neku natklasu klase NP.

Primjer 4.6. *Neka su zadani alfabet $\Sigma = \{a\}$ i skup podataka $\mathcal{D} = \{0, 1\}$. Primjer automata čiji najkraći podatkovni put kojeg automat prihvaća ima dužinu eksponencijalno veću od broja stanja automata dan je na slici 4.2. Pritom automat ima k registara za neki proizvoljni $k \geq 2$, a početna konfiguracija registara zadan je s $\tau_0 = (0, 0, \ldots, 0)$. Uokvireni se dio automata ponavlja za svaki i iz skupa $I = \{2, 3, \ldots, k-1\}$.*



Slika 4.2: Primjer konačnog automat s registrima koji se ponaša kao brojač.

Automat će prihvatiti riječ tek kada svi podatci u registrima, osim prvog, budu različiti od prvog podatka, odnosno kada budu jednaki 1. Vidljivo je da automat u stanju $q^d_{i,1}$ prilikom čitanja podatka d očekuje da je podatak jednak podatku sačuvanom u prvom registru, odnosno $d = 0$. Također, u tom stanju se promatra i i-ti registar te se ispituje je li njegova vrijednost jednaka trenutnom podatku $d = 0$ ili ne. U slučaju kad je različita od 0, odnosno jednaka 1, automat postavlja i-ti registar na vrijednost 0 i prelazi u stanje $q^d_{i+1,1}$ pratećі još jedan znakovni prijelaz. S druge strane kada je vrijednost i-tog registra jednaka 0, praćenjem nekoliko prijelaza automat ju postavlja na 1 i vraća se u početno stanje.

Primijetimo da smo s time opisali proces pribrajanja jedinice nekom binarnom broju. Zbog toga i uvjeta koji imamo za prijelaz u prihvatljivo stanje, prva podatkovna riječ koju automat prihvaća bit će dužine $\mathcal{O}(k2^k)$, gdje je k broj registara. Također, imamo da je broj stanja automata jednak $\mathcal{O}(k)$ jer za svaki registar imamo konstantan broj stanja koji ga obrađuje. Iz toga slijedi da je dužina najmanje riječi koju automat prihvaća eksponencijalno veća od broja stanja.

Kako bi dokazali kojoj točno klasi složenosti pripada problem ispitivanja je li je-

zik konačnog automata s registrima neprazan trebamo definirati dvije stvari: kodiranje automata i jezik s kojim ćemo predstavljati taj problem. Kodiranje konačnih automata s registrima provodimo slično kao i kodiranje nedeterminističkih konačnih automata zadano definicijom 2.19. Potrebno je jedino obratiti pažnju na podatkovne prijelaze (q, c, I, q'), ali njih možemo kodirati kao uređene četvorke, pri čemu se posebno treba obraditi kodiranje uvjeta c i skupa indeksa I. Kodiranje uvjeta c možemo obaviti proširivanjem alfabeta Turingovog stroja s logičkim simbolima i predstavljanjem uvjeta $x_i^{=}$ binarnim brojem i, dok se skup I predstavlja kao lista binarnih brojeva iz I.

Uz opisano kodiranje konačnih automata s registrima definiramo problem ispitivanja je li jezik konačnog automata s registrima neprazan kao jezik:

$$\mathrm{coE}_{\mathrm{AUT}} = \{\langle \mathcal{A} \rangle \mid L(\mathcal{A}) \neq \emptyset\}\,.$$

Sada kad imamo formalnu definiciju jezika možemo dokazati da se on nalazi u klasi PSPACE. U primjeru 4.6 prikazali smo konačni automat s registrima čiji najkraći podatkovni put kojeg automat prihvaća ima dužinu eksponencijalno veću od broja stanja automata. Sljedećom propozicijom dokazujemo da je to ujedno i gornja granica za duljinu najkraćeg podatkovnog puta sadržanog u jeziku automata. Iz toga slijedi da je za ispitivanje je li jezik automata neprazan dovoljno provjeriti podatkovne putove čija dužina je najviše eksponencijalno veća od broja stanja automata. Kako za zapis eksponencijalnog broja trebamo najviše polinomno prostora, strategijom pogađanja podatkovnog niza i provjerom nalazi li se taj niz u jeziku automata dobivamo da se problem nalazi u klasi PSPACE.

Propozicija 4.7. *Jezik* $\mathrm{coE}_{\mathrm{AUT}}$ *je u klasi* PSPACE.

Dokaz. Prvo opisujemo nedeterministički algoritam koji u $\mathcal{O}(n \log n)$ prostora prihvaća sve konačne automate s registrima čiji je jezik neprazan. Korištenjem Savitchevog teorema (Savitch, 1970) tada postoji deterministički algoritam koji u $\mathcal{O}(n^2 \log^2 n)$ prostora prihvaća iste automate. Time smo onda dobili i željenu tvrdnju da se jezik $\mathrm{coE}_{\mathrm{AUT}}$ nalazi u klasi PSPACE. Takav nedeterministički algoritam dan je slikom 4.3. Primijetimo da algoritam u svakom trenutku nedeterministički bira neki znak alfabeta a, odnosno podatak d iz konačnog skupa $\{0, 1, \ldots, k\}$, te bira jedan od dozvoljenih prijelaza. Zbog toga imamo da će automat sigurno pronaći neki podatkovni put, ako takav postoji, koji se sastoji od najviše $q(k+1)^k$ simbola i sadrži samo podatke iz skupa $D = \{0, 1, \ldots, k\}$. Preostaje nam još dokazati da ako automat prihvaća neki podatkovni put, onda mora prihvaćati i podatkovni put takvog oblika, te da je prostorna složenost algoritama jednaka $\mathcal{O}(n \log n)$.

Ulaz: $\langle \mathcal{A} \rangle$, gdje je $\mathcal{A} = (Q, q_0, F, \tau_0, \delta)$ konačni automat s registrima.
Izlaz: Prihvaća ulaz ako je $L(\mathcal{A}) \neq \emptyset$.

1 Stavi oznaku na početno stanje q_0 automata $\mathcal{A}$.
2 **Ponovi** $q(k+1)^k$ *puta, gdje je* q *broj stanja, a* k *broj registara automata* $\mathcal{A}$
3 Nedeterministički izaberi neki znak $a \in \Sigma$ ili podatak $d \in \{0, 1, \ldots, k\}$.
4 Nedeterministički pomakni oznaku stanja tako da simulira korištenje nekog prijelaza iz δ za izabrani znak a ili podatak d.
5 Ako oznaka dođe na prihvatljivo stanje, prihvati ulaz.
6 Odbaci ulaz.

Slika 4.3: $\mathsf{NSPACE}(n \log n)$ algoritam za $\mathrm{coE}_{\mathrm{AUT}}$ uz dokaz propozicije 4.7.

Prvo dokažimo da je korištenje samo podataka iz skupa $D = \{0, 1, \ldots, k\}$ dovoljno kada automat ima k registara. Pretpostavimo da imamo neki podatkovni put w_π duljine l kojeg automat prihvaća i čiji svi podatci nisu iz skupa D. Neka je zadan prihvatljiv niz konfiguracija $C = (C_0, C_1, \ldots, C_l)$ automata $\mathcal{A}$ nad tim podatkovnim putem te neka se u konfiguraciji C_i prvi put pojavljuje neki podatak $d \notin D$. Tada možemo zamijeniti podatak d u toj konfiguraciji i na pripadnoj poziciji podatkovnog puta s drugim podatkom $d' \in D$, pazeći pritom da se podatak d' ne nalazi u nijednom od registara konfiguracije C_i. Također, potrebno je zamijeniti svaki podatak d s podatkom d' u svim konfiguracijama i pripadnim pozicijama podatkovnog puta koje neposredno slijede konfiguraciju C_i, a sadrže podatak d u jednom od registara. Takva je zamjena moguća jer skup D ima $k+1$ podataka, a registara ima samo k. Novi niz konfiguracija je također valjan niz konfiguracija automata jer zadovoljava iste uvjete prijelaza kao i početni niz C. Provođenjem takvih zamjena sve dok ima podataka koji nisu iz D dobivamo novi podatkovni put kojeg automat prihvaća čiji su svi podatci iz D.

Uzimajući u obzir da možemo koristiti samo $k+1$ podataka za automat s k registara, možemo ograničiti broj parova stanja i konfiguracija registara u kojima automat može biti. Svaki od k registara može sadržavati jedan od $k+1$ podataka što ukupno ograničava broj različitih konfiguracija registara na $(k+1)^k$. Kako automat ima $q = |Q|$ stanja, slijedi da automat ima $q(k+1)^k$ parova stanja i neke konfiguracije registara, te će se nakon čitanja toliko simbola sigurno ponoviti neki par. Zbog toga je $q(k+1)^k$ ujedno i gornja granica dužine najmanjeg podatkovnog puta kojeg automat prihvaća.

Kako trebamo brojati do $q(k+1)^k$, brojač će nam zauzimati prostor veličine $\log q + k \log(k+1)$ što je jednako $\mathcal{O}(n \log n)$, pri čemu je veličina ulaza $\langle \mathcal{A} \rangle$ jednaka n. Pored

brojača algoritam treba čuvati i trenutnu konfiguraciju registara koja zauzima $k \log(k+1) = \mathcal{O}(n \log n)$ prostora, te trenutno stanje i pogođeni znak, odnosno podatak, od kojih svaki zauzima $\mathcal{O}(\log n)$ prostora. Zbog toga je i prostorna složenost algoritma jednaka $\mathcal{O}(n \log n)$, što smo i trebali dokazati. Q.E.D.

Prethodnim smo dokazom pokazali da se problem $\mathrm{coE}_{\mathrm{AUT}}$ nalazi u klasi PSPACE. U nastavku dokazujemo da vrijedi i jača tvrdnja, odnosno da je problem $\mathrm{coE}_{\mathrm{AUT}}$ ujedno i PSPACE-težak. Kako bi dokazali tu tvrdnju, opisujemo redukciju s PSPACE-potpunog problema $\mathrm{coUNIV}_{\mathrm{NKA}}$ na traženi problem $\mathrm{coE}_{\mathrm{AUT}}$. Pritom $\mathrm{coUNIV}_{\mathrm{NKA}}$ predstavlja problem ispitivanja postoji li riječ koju nedeterministički konačni automat ne prihvaća i čiju smo PSPACE-potpunost već dokazali u korolaru 2.30. Za zadani nedeterministički konačni automat, redukcija generira konačni automat s registrima koji simulira izvođenje nedeterminističkog konačnog automata nad proizvoljnom riječi. Ulazi tako generiranog konačnog automata s registrima su podatkovni putovi koji opisuju tijek simulacije nedeterminističkog automata. Pritom automat s registrima prihvaća jedino one podatkovne putove koji opisuju simulaciju nedeterminističkog konačnog automata u kojoj nedeterministički automat ne prihvaća pročitanu riječ.

Prilikom simulacije postavlja se pitanje kako za pročitani znak alfabeta simulirati prijelaz nedeterminističkog automata iz jednog skupa stanja u drugi skup. Kao rješenje koristimo ideju sličnu ideji koju smo koristili prilikom izgradnje konačnog automata s registrima iz primjera 4.6. U tom primjeru, automat s registrima simulira brojač tako da pri računanju sljedeće vrijednosti zasebno obrađuje svaki bit brojača. Slično, generirani automat s registrima ne određuje u jednom koraku novi skup stanja u koji nedeterministički automat treba prijeći, već se taj skup određuje zasebnom obradom svakog stanja početnog skupa. U nastavku su raspisani detalji tog dokaza čija je osnovna struktura preuzeta iz (Libkin i Vrgoč, 2012a).

Teorem 4.8. *Jezik* $\mathrm{coE}_{\mathrm{AUT}}$ *je* PSPACE*-težak.*

Dokaz. Provodimo redukciju s PSPACE-potpunog problema $\mathrm{coUNIV}_{\mathrm{NKA}}$ na željeni problem $\mathrm{coE}_{\mathrm{AUT}}$. Redukcija se provodi tako da zadanom nedeterminističkom konačnom automatu $\mathcal{N} = (Q, q_1, F, \delta)$ pridružujemo konačni automat s registrima $\mathcal{A} = (Q', q_0', F', \tau_0', \delta')$. Pritom je nedeterministički automat $\mathcal{N}$ definiran nad alfabetom Σ, dok je automat s registrima definiran nad alfabetom $\Sigma' = \Sigma \cup \{\#\}, \# \notin \Sigma$, i skupom podataka $\mathcal{D} = \{t, f\}$. U nastavku pretpostavljamo da je skup stanja nedeterminističkog konačnog automata zadan s $Q = \{q_1, q_2, \ldots, q_n\}$.

Označimo s $\lambda_{\#}$ funkciju koja podatkovnom putu w_π pridružuje riječ koju dobijemo uklanjanjem podataka i znakova $\#$ iz tog podatkovnog puta. Tada, pridruženi

automat s registrima $\mathcal{A}$ za pročitani podatkovni put w_π simulira rad nedeterminističkog automata $\mathcal{N}$ nad riječi $\lambda_\#(w_\pi)$. Automat s registrima $\mathcal{A}$ prihvaća jedino podatkovne putove čiju pridruženu riječ $\lambda_\#(w_\pi)$ nedeterministički automat $\mathcal{N}$ ne prihvaća. Zbog toga imamo da nedeterministički automat $\mathcal{N}$ ne prihvaća neku riječ alfabeta ako i samo ako postoji podatkovni put kojeg prihvaća automat s registrima $\mathcal{A}$.

Konačni automat s registrima $\mathcal{A}$ ima $2n + 2$ registara, pri čemu je n broj stanja nedeterminističkog konačnog automata $\mathcal{N}$. U registrima automata $\mathcal{A}$ se pohranjuje skup stanja nedeterminističkog automata $\mathcal{N}$ u koja je nedeterministički automat mogao prijeći čitajući neku riječ. Skup stanja se pohranjuje između drugog i $(n + 2)$-og registra tako da $(i + 2)$-gi registar automata $\mathcal{A}$ sadrži podatak t ako i samo ako se nedeterministički automat $\mathcal{N}$ nalazi u stanju q_i. Podatci u prva dva registra fiksirani su na t i f kako bi olakšali usporedbu prilikom određivanja novog skupa stanja nedeterminističkog automata $\mathcal{N}$. Između $(n + 3)$-eg i zadnjeg registra nalazi se pomoćni skup stanja pomoću kojeg se računa novi skup stanja.

Početno se automat $\mathcal{A}$ nalazi u stanju $q'_0 = q^d_\#$, a početna konfiguracija registara je $\tau'_0 = (t, f, t, f, f, \ldots, f)$. Tom smo konfiguracijom registara označili da se nedeterministički automat $\mathcal{N}$ na početku rada nalazi jedino u stanju q_1. Po završetku simulacije čitanja jednog znaka, automat $\mathcal{A}$ prelazi u jedno od znakovnih stanja iz skupa $Q^i_w = \{q^w_{DA}, q^w_{NE}\}$. Automat $\mathcal{A}$ prelazi u stanje q^w_{DA} jedino ako nedeterministički automat $\mathcal{N}$ prihvaća riječ $\lambda_\#(w_\pi)$, gdje je w_π pročitani podatkovni put. Inače, automat prelazi u stanje q^w_{NE}. Znakovno stanje q^w_{NE} ujedno je i jedino prihvatljivo stanje automata $\mathcal{A}$, tj. $F' = \{q^w_{NE}\}$, jer označava da smo pronašli riječ $\lambda_\#(w_\pi)$ koju nedeterministički automat $\mathcal{N}$ ne prihvaća. Posebnu pažnju obraćamo slučaju kada nedeterministički automat $\mathcal{N}$ prihvaća praznu riječ ϵ. Kako bi riješili taj slučaj koristimo posebni podatkovni prijelaz koji iz početnog stanja prelazi u jedno od stanja skupa Q^i_w:

$$\delta^\#_d = \left\{(q^d_\#, \epsilon, \emptyset, q^w_x) \mid (x = DA \Leftrightarrow q_0 \in F) \wedge (x = NE \Leftrightarrow q_0 \notin F)\right\},$$

pri čemu automat $\mathcal{A}$ prelazi u stanje q^w_{DA} jedino ako automat $\mathcal{N}$ prihvaća riječ ϵ.

Stanja skupa Q^i_w ujedno su i stanja iz kojih započinje simulacija čitanja jednog znaka alfabeta. Iz tih stanja automat s registrima $\mathcal{A}$ nedeterministički „bira“ znak alfabeta Σ kojeg će sljedećeg nedeterministički automat $\mathcal{N}$ pročitati:

$$\delta^i_w = \left\{(q^w_x, a, q^d_{a_1}) \mid \forall a \in \Sigma \wedge \forall x \in \{DA, NE\}\right\}.$$

Pritom za svaki znak a alfabeta Σ imamo podatkovna stanja $Q^a_d = \left\{q^d_{a_i} \mid \forall i \in [n]\right\}$ i pripadna znakovna stanja $Q^a_w = \left\{q^w_{a_i} \mid \forall i \in [n]\right\}$ preko kojih se provodi simulacija. Nakon izbora znaka a, automat s registrima pojedinačno prolazi oznake trenutnog

skupa stanja koji je pohranjen u njegovim registrima i određuje novi skup stanja, koji se pohranjuje u pomoćne registre. Ako je stanje q_i u trenutno aktivnom skupu stanja nedeterminističkog automata $\mathcal{N}$, odnosno $(i+2)$-gi registar automata $\mathcal{A}$ sadrži podatak t, tada u pomoćnim registrima automata $\mathcal{A}$ označavamo sva stanja u koja nedeterministički automat $\mathcal{N}$ može prijeći iz stanja q_i čitanjem znaka a. Odnosno, zapisujemo podatak t na pripadne indekse pomoćnih registara. U slučaju kada stanje q_i nije u trenutno aktivnom skupu stanja, preskačemo ga. Formalno, opisana podatkovna relacija prijelaza zadana je sa:

$$\delta_d^a = \left\{(q_{a_i}^d, x_1^{=} \wedge x_{i+2}^{=}, I, q_{a_i}^w) \mid \forall i \in [n] \wedge \forall j \left((q_i, a, q_j) \in \delta \Leftrightarrow (2j+2) \in I\right)\right\} \cup \\ \left\{(q_{a_i}^d, x_2^{=} \wedge x_{i+2}^{=}, \emptyset, q_{a_i}^w) \mid \forall i \in [n]\right\}.$$

Pripadna znakovna relacija prijelaza automata $\mathcal{A}$ samo propagira obradu s i-tog na $(i+1)$-vo stanje. U slučaju zadnjeg znakovnog stanja $q_{a_n}^w$ vezanog uz pročitani znak a, automat $\mathcal{A}$ prelazi na sljedeću fazu simulacije koja započinje stanjem $q_{c_1}^d$. Odnosno, formalno vrijedi:

$$\delta_w^a = \left\{(q_{a_i}^w, \#, q_{a_{i+1}}^d \mid \forall i \in [n-1]\right\} \cup \left\{(q_{a_n}^w, \#, q_{c_1}^d)\right\}.$$

U sljedećoj fazi simulacije prepisujemo novi skup stanja iz pomoćnih registara u glavne registre. To činimo korištenjem novih podatkovnih stanja $Q_d^c = \left\{q_{c_i}^d \mid \forall i \in [n]\right\}$ i znakovnih stanja $Q_w^c = \left\{q_{c_i}^w \mid \forall i \in [n]\right\}$. Podatkovni prijelazi koji prepisuju $(2i+2)$-ti podatak u $(i+2)$-gi zadani su s:

$$\delta_d^c = \left\{(q_{c_i}, x_{2i+2}^{=}, \{i+2\}, q_{c_i}^w) \mid \forall i \in [n]\right\},$$

dok su pripadni znakovni prijelazi prijelazi:

$$\delta_w^c = \left\{(q_{c_i}^w, \#, q_{c_{i+1}}^d) \mid \forall i \in [n-1]\right\} \cup \left\{(q_{c_n}^w, \#, q_e^d)\right\}.$$

Kao i u prošloj fazi simulacije, znakovni prijelazi nam služe samo kako bi propagirali obradu s i-tog na $(i+1)$-vo stanje, odnosno na sljedeću fazu simulacije.

Zadnja faza simulacije, koja započinje sa podatkovnim stanjem q_e^d, postavlja sve podatke iz pomoćnih registara na f kako bi omogućila novu iteraciju čitanja nekog znaka alfabeta Σ. Također, u toj fazi prelazimo u jedno od znakovnih stanja iz Q_w^i ovisno o tome sadrži li trenutno aktivni skup stanja nedeterminističkog automata $\mathcal{N}$ neko prihvatljivo stanje. Formalno, imamo podatkovni prijelaz:

$$\delta_d^e = \left\{\left(q_e^d, x_2^{=} \wedge \left(x_{j_1}^{\neq} \vee x_{j_2}^{\neq} \vee \ldots \vee x_{j_k}^{\neq}\right), I, q_{DA}^w\right), \right. \\ \left.\left(q_e^d, x_2^{=} \wedge \left(x_{j_1}^{=} \wedge x_{j_2}^{=} \wedge \ldots \wedge x_{j_k}^{=}\right), I, q_{NE}^w\right) \mid F = \{q_{j_1-2}, q_{j_2-2}, \ldots, q_{j_k-2}\}\right\},$$

pri čemu je skup I jednak skupu $\{n+3, n+4, \ldots, 2n+2\}$. Vraćanjem u neko od stanja iz skupa Q_w^i možemo ponovno započeti simulaciju čitanja nekog drugog znaka za nedeterministički automat $\mathcal{N}$.

Konačni se skup podatkovnih stanja automata $\mathcal{A}$ sastoji od unije prethodno definiranih skupova: $Q_d = \{q_\#^d\} \cup \left(\bigcup_{a\in\Sigma} Q_d^a\right) \cup Q_d^c \cup \{q_e^d\}$, a slično vrijedi i za skup znakovnih stanja: $Q_w = Q_w^i \cup \left(\bigcup_{a\in\Sigma} Q_w^a\right) \cup Q_w^c$. Podatkovni prijelazi su unija prethodno definiranih podatkovnih prijelaza: $\delta_d = \delta_d^\# \cup \left(\bigcup_{a\in\Sigma} \delta_w^a\right) \cup \delta_d^c \cup \delta_d^e$, a znakovni prijelazi su: $\delta_w = \delta_w^i \cup \left(\bigcup_{a\in\Sigma} \delta_w^a\right) \cup \delta_w^c$. Iz opisanog rada automata $\mathcal{A}$ lako se vidi da automat $\mathcal{A}$ za zadani znak alfabeta a simulira prijelaze nedeterminističkog automata $\mathcal{N}$ iz jednog skupa aktivnih stanja u drugi. Dolaskom u stanje q_{NE}^w automat prihvaća podatkovni put w_π te vrijedi da pripadna riječ $\lambda_\#(w_\pi)$ nije u jeziku automata $\mathcal{N}$.

Kako se struktura automata $\mathcal{A}$ ponavlja, prijelaze automata možemo generirati pamćenjem stanja za koje trenutno generiramo prijelaze te nekoliko pomoćnih znakova, stanja i brojača koji nam služe za određivanje prijelaza. Tih pomoćnih podataka ima konačno mnogo i zauzimaju najviše logaritamski prostor, pa postoji Turingov stroj $\mathcal{T}$ koji u logaritamskom prostoru pridjeljuje automat s registrima $\mathcal{A}$ zadanom nedeterminističkom automatu $\mathcal{N}$. Q.E.D.

Iz prethodnog teorema i propozicije 4.7 slijedi PSPACE-potpunost problema ispitivanja nepraznosti jezika konačnih automata s registrima $\mathrm{coE}_{\mathrm{AUT}}$.

Korolar 4.9. *Jezik* $\mathrm{coE}_{\mathrm{AUT}}$ *je* PSPACE*-potpun.*

Konačne smo automate s registrima uveli kako bi nam poslužili kao temelj za definiranje efikasnih upita nad grafovima s podatcima. Kako bi ostvarili taj cilj definiramo formalizam čiji su elementi konačni automati s registrima kojima pridružujemo pripadni jezik podatkovnih putova. Takve upite zovemo regularnim upitima nad podatkovnim putovima i formalno ih definiramo u nastavku.

Definicija 4.10 (Regularni upiti nad podatkovnim putovima)**.** *Neka su zadani alfabet* Σ *i skup podataka* $\mathcal{D}$*. Formalizam konačnih automata s registrima je formalizam* $\mathcal{F}_\mathcal{A} = (\mathcal{E}, L)$ *gdje je* $\mathcal{E}$ *skup svih konačnih automata s registrima nad* Σ *i* $\mathcal{D}$ *za sve brojeve registara* k*. Pritom, funkcija* L *pridružuje svakom automatu* $\mathcal{A} \in \mathcal{E}$ *njegov jezik* $L(\mathcal{A})$*.* Regularni upit nad podatkovnim putem (RDPQ) *je upit zadan u formalizmu* $\mathcal{F}_\mathcal{A}$*.*

Kako je glavna svrha upita njihovo evaluiranje nad nekim grafom i utvrđivanje koji mu vrhovi pripadaju, u nastavku promatramo postupak evaluacije regularnih upita nad podatkovnim putovima. Evaluacija upita provodi se tako da se grafu s podatcima

i konačnom automatu s registrima pridruže dva nedeterministička konačna automata. Ti nedeterministički automati prihvaćaju jedino riječi koje odgovaraju podatkovnim putovima koji postoje u grafu, odnosno koje konačni automat s registrima prihvaća. Tada evaluaciju upita možemo provesti traženjem presjeka tih automata i ispitivanjem postoji li riječ koju taj presjek prihvaća.

Iako nedeterministički konačni automati prihvaćaju riječi, u nastavku govorimo za generirane nedeterminističke automate da prihvaćaju neki podatkovni put. Pritom imamo na umu da nedeterministički automati zapravo prihvaćaju riječ koja odgovara nekom podatkovnom putu koji sadrži podatke iz nekog konačnog podskupa skupa podataka. Dokaz provodimo korištenjem dvije leme koje opisuju pridruživanje nedeterminističkog konačnog automata grafu s podatcima, odnosno konačnom automatu s registrima. Ti dokazi, kao i dokaz teorema o evaluaciji regularnih upita nad podatkovnim putovima koji iz njih slijedi, preuzeti su iz (Libkin i Vrgoč, 2012a). Prvo opisujemo generiranje nedeterminističkog konačnog automata koji prihvaća sve podatkovne putove između dva određena vrha zadanog grafa s podatcima. To je generiranje lako provesti pretvaranjem podataka u oznake bridova, nakon čega vrhove i bridove grafa poistovjećujemo sa stanjima i prijelazima nedeterminističkog konačnog automata.

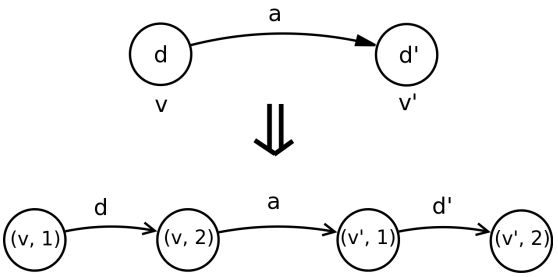
Lema 4.11. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Postoji Turingov stroj $\mathcal{T}$ koji u logaritamskom prostoru za ulaz $\langle G, v_s, v_t\rangle$ vraća izlaz $\langle \mathcal{N}\rangle$ pri čemu je:*

- *G graf s podatcima nad Σ i $\mathcal{D}$, a v_s i v_t su neki vrhovi grafa i*
- *$\mathcal{N}$ je nedeterministički konačni automat koji prihvaća sve podatkovne putove $w_{\pi_{v_s v_t}}$ koji odgovaraju nekom putu $\pi_{v_s v_t}$ između vrhova v_s i v_t u grafu.*

Dokaz. Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$. Zadanom grafu s podatcima $G = (V, E, \rho)$ nad Σ i $\mathcal{D}$, i vrhovima v_s i v_t pridružujemo nedeterministički konačni automat $\mathcal{N} = (Q, q_0, F, \delta)$. Graf s podatcima ima označene bridove, pa smo promatranjem vrhova kao stanja automata i bridova kao prijelaza vrlo blizu definiranja pripadnog nedeterminističkog konačnog automata $\mathcal{N}$. Jedini problem nam predstavljaju podatci grafa koje trebamo pretvoriti u prijelaze.

Označimo s D konačan podskup skupa podataka $\mathcal{D}$ koji sadrži sve podatke koji se nalaze u grafu G. Nedeterministički automat $\mathcal{N}$ tada koristi alfabet $\Sigma' = \Sigma \cup D$. Podatke grafa pretvaramo u prijelaze tako da vrh v, koji sadrži podatak d, rastavimo u dva stanja $(v, 1)$ i $(v, 2)$ s prijelazom između njih čija je oznaka d. Takvo rastavljanje vrha prikazano je slikom 4.4.

Formalno, skup stanja nedeterminističkog automata je unija dvije kopije vrhova grafa G, tj. $Q = (V \times \{1\}) \cup (V \times \{2\})$. Prijelaze gradimo tako da kopije istog vrha



Slika 4.4: Pridruživanje prijelaza automata vrhovima i bridovima grafa.

povezujemo prijelazom koji sadrži podatak pohranjen u tom vrhu, a različite parove vrhova povezujemo prijelazima koji imaju oznaku jednaku oznaci brida u grafu:

$$\delta = \{((v,1), d, (v,2)) \mid \rho(v) = d\} \cup \{((v,2), a, (v',1)) \mid (v, a, v') \in E\}.$$

Kako tražimo podatkovne putove iz v_s u v_t, početno stanje automata $\mathcal{N}$ odgovara vrhu v_s, tj. $q_0 = (v_s, 1)$, a prihvatljiva se stanja sastoje od samo jednog stanja koje odgovara vrhu v_t, tj. $F = (v_t, 2)$.

Automat $\mathcal{N}$ prihvaća podatkovni put w_π ako i samo ako postoji put između vrhova v_s i v_t u grafu G čiji je podatkovni put jednak w_π. U slučaju kada postoji podatkovni put w_π u grafu G, slijeđenjem pripadnih prijelaza nedeterminističkog automata $\mathcal{N}$ prijeći ćemo iz početnog u prihvatljivo stanje. Slično vrijedi i u drugom smjeru dokaza te tvrdnje. Lako je konstruirati Turingov stroj $\mathcal{T}$ koji za zadani graf s podatcima G vraća takav nedeterministički automat $\mathcal{N}$ jer se cijela konstrukcija temelji na prepisivanju bridova. Kako prilikom prepisivanja treba pamtiti najviše nekoliko stanja i znakova alfabeta, Turingov stroj $\mathcal{T}$ radi u logaritamskom prostoru. Q.E.D.

Prethodni teorem nam daje postupak pridruživanja nedeterminističkih konačnih automata grafovima s podatcima. Kako bi mogli do kraja provesti željenu konstrukciju pomoću koje evaluiramo upite, moramo opisati i postupak pridruživanja nedeterminističkih konačnih automata konačnim automatima s registrima. Općenito je takav postupak nemoguće provesti, jer je skup podataka beskonačan. Zbog toga pridruživanje provodimo za neki konačan podskup skupa podataka. Uz konačan skup podataka, nedeterministički automat može u svojim stanjima čuvati stanje automata s registrima i cijelu njegovu konfiguraciju registara. Složenost tog pridruživanja ovisi o tome je li broj korištenih registara promjenjiv ili ne. Odnosno, složenost pridruživanja ovisi o tome zadajemo li automat s registrima kao ulaz Turingovog stroja ili je automat s registrima fiksiran te je konačan skup podataka jedini ulaz.

Lema 4.12. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Postoji Turingov stroj $\mathcal{T}$ koji u polinomnom prostoru za ulaz $\langle \mathcal{A}, D \rangle$ vraća izlaz $\langle \mathcal{N} \rangle$ pri čemu je:*

– *$\mathcal{A}$ konačni automat s registrima nad Σ i $\mathcal{D}$,*

– *D je konačni podskup skupa podataka $\mathcal{D}$ i*

– *$\mathcal{N}$ je nedeterministički konačni automat koji prihvaća sve podatkovne putove w_π koji imaju podatke samo iz skupa D i prihvaća ih automat $\mathcal{A}$.*

Dodatno, ako unaprijed zadamo automat $\mathcal{A}$ i kao ulaz uzimamo samo skup podataka D, Turingov stroj radi u logaritamskom prostoru.

Dokaz. Zadanom konačnom automatu s registrima $\mathcal{A} = (Q, q_0, F, \tau_0, \delta)$ i konačnom podskupu podataka D pridružujemo nedeterministički automat $\mathcal{N} = (Q', q_0', F', \delta')$. Pritom nedeterministički automat $\mathcal{N}$ radi nad alfabetom Σ' koji je unija alfabeta Σ automata s registrima $\mathcal{A}$ i konačnog podskupa podataka D, tj. $\Sigma' = \Sigma \cup D$. Zbog korištenja konačnog podskupa skupa s podatcima, nedeterministički automat može u svojem stanju pohranjivati stanje automata s registrima i cijelu konfiguraciju registara, tj. $Q' = Q \times D^k$. Početno stanje nedeterminističkog automata $\mathcal{N}$ je stanje koje odgovara početnom stanju automata s registrima $\mathcal{A}$ i njegovoj početnoj konfiguraciji registara: $q_0' = (q_0, \tau_0)$. Prihvatljiva stanja nedeterminističkog automata $\mathcal{N}$ su sva stanja koja odgovaraju nekom prihvatljivom stanju automata s registrima $\mathcal{A}$ uz proizvoljnu konfiguraciju registara, tj. $F' = F \times D^k$.

Potrebno je još korištenjem relacije prijelaza automata s registrima $\mathcal{A}$ definirati relaciju prijelaza nedeterminističkog automata $\mathcal{N}$. Kako se relacija prijelaza automata s registrima $\mathcal{A}$ sastoji od dvije relacije prijelaza, definiranje relacije prijelaza nedeterminističkog automata $\mathcal{N}$ provodimo u dva dijela. Iz znakovnih prijelaza je lako definirati prijelaze nedeterminističkog automata, jer se znakovni prijelazi ne razlikuju od uobičajenih nedeterminističkih prijelaza te ne utječu na konfiguraciju registara:

$$\delta_w' = \{((q, \tau), a, (q', \tau)) \mid (q, a, q') \in \delta_w\}.$$

Prilikom definiranja prijelaza nedeterminističkog automata $\mathcal{N}$ koji odgovaraju podatkovnim prijelazima automata s registrima $\mathcal{A}$ trebamo obratiti pažnju i na konfiguraciju registara koju čuvamo u stanju nedeterminističkog automata. Odnosno, novi se prijelazi nedeterminističkog automata $\mathcal{N}$ definiraju u zavisnosti od uvjeta c i skupa I indeksa registara sadržanih u izvornim podatkovnim prijelazima kao:

$$\delta_d' = \{((q, \tau), d, (q', \tau')) \mid (q, c, I, q') \in \delta_d \wedge d, \tau \models c\},$$

te vrijedi da je τ' jednak τ u svim pozicijama, osim što su i-ti elementi od τ' jednaki d svaki put kad je i iz skupa I. Relacija prijelaza nedeterminističkog automata $\mathcal{N}$ je unija prošle dvije relacije prijelaza: $\delta = \delta_w' \cup \delta_d'$.

Potrebno je dokazati da nedeterministički automat $\mathcal{N}$ prihvaća točno one putove koje prihvaća i automat s registrima $\mathcal{A}$, a koji sadrže samo podatke iz konačnog skupa D. U slučaju kada nedeterministički automat $\mathcal{N}$ prihvaća neki podatkovni put, tada konačni automat s registrima $\mathcal{A}$ može slijediti prijelaze koji odgovaraju prijelazima nedeterminističkog automata te također završiti u prihvatljivom stanju. Slično dobivamo i u drugom smjeru dokaza kada pretpostavimo da automat s registrima $\mathcal{A}$ prihvaća neki podatkovni put čiji su svi podatci iz skupa D. Tada i nedeterministički automat $\mathcal{N}$ prihvaća taj podatkovni put jer može simulirati registre pohranjujući njihove vrijednosti u svojim stanjima.

Turingov stroj $\mathcal{T}$ prilikom generiranja pojedinog prijelaza treba pratiti kombinaciju trenutnog stanja automata i konfiguracije registara, te stanja automata i konfiguracije registara u koje prelazi. Pritom jedno stanje automata zauzima $\mathcal{O}(\log |Q|)$ prostora, dok jedna konfiguracija registara zauzima $\mathcal{O}(k \log |D|)$ prostora zbog k registara. Potrebno je još pamtiti i znak, odnosno podatak, za koji se prijelaz računa, što zauzima dodatnih $\mathcal{O}(\log |\Sigma|)$, odnosno $\mathcal{O}(\log |D|)$ prostora. Prilikom generiranja prijelaza nedeterminističkog automata koji odgovaraju znakovnim prijelazima nije potrebno čuvati nikakve dodatne podatke, jer se generiranje takvih prijelaza svodi na prepisivanje ulaza. Kod podatkovnih prijelaza je potrebno iskoristiti još prostora Turingovog stroja $\mathcal{T}$ kako bi se provjerio uvjet c i međuovisnost konfiguracija registara τ, τ' i skupa podataka I. Provjeru uvjeta prijelaza možemo provesti tako da zapišemo uvjet na traku i računamo onda nad njime, za što nam treba $\mathcal{O}(|\langle c \rangle|)$ prostora. Kod ispitivanja podudaranja konfiguracija registara τ i τ' u ovisnosti o skupu I možemo na traku zapisati skup I i onda provoditi usporedbu, za što će trebati $\mathcal{O}(|\langle I \rangle|)$ prostora.

U slučaju kada Turingov stroj kao ulaz dobiva kôd $\langle \mathcal{A}, D \rangle$ automata s registrima $\mathcal{A}$ i skupa podataka D, sve vrijednosti iz skupa $\{k, |Q|, |D|, |\Sigma|, |\langle I \rangle|, |\langle c \rangle|\}$ su veličine $\mathcal{O}(n)$, gdje je n duljina kôda $\langle \mathcal{A}, D \rangle$. Zbog toga je prostorna složenost takvog Turingovog stroja $\mathcal{O}(n \log n)$ te Turingov stroj radi u polinomnom prostoru. Primijetimo da će kôd generiranog nedeterminističkog automata $\mathcal{N}$ biti eksponencijalan u veličini ulaza, ali to nam ne utječe na prostornu složenost jer se korišteni prostor mjeri samo za radne trake Turingovog stroja.

U slučaju kada fiksiramo automat s registrima $\mathcal{A}$ i promatramo samo kôd skupa podataka D kao ulaz, vrijednosti iz skupa $\{k, |Q|, |\Sigma|, |\langle I \rangle|, |\langle c \rangle|\}$ su konstante veličine $\mathcal{O}(1)$, dok vrijednost $|D|$ ostaje veličine $\mathcal{O}(n)$. Zbog toga u tom slučaju Turingov stroj koristi $\mathcal{O}(\log n)$ prostora i cijelo se pridruživanje provodi u logaritamskom prostoru, čime je i naša tvrdnja dokazana. Q.E.D.

Korištenjem prethodnih lema možemo dobiti gornje granice za podatkovnu i kombiniranu složenost evaluacije upita zadanih korištenjem konačnih automata s registrima. Pritom koristimo i teorem 2.20, koji kaže da je moguće u logaritamskom prostoru generirati nedeterministički konačni automat koji prihvaća presjek jezika neka druga dva nedeterministička konačna automata. Iz toga slijedi da je podatkovna složenost evaluacije upita zadanih konačnim automatima s registrima u klasi NL, dok je kombinirana složenost u klasi PSPACE. Donju granicu podatkovne složenosti evaluacije upita jednostavno dobivamo iz razmatranja prezentiranih u primjeru 3.13, dok ćemo za donju granicu kombinirane složenosti evaluacije upita morati uvesti poseban tip regularnih izraza koji odgovaraju konačnim automatima s registrima.

Teorem 4.13. *Podatkovna složenost upita iz* RDPQ *je* NL*-potpuna. Kombinirana složenost upita iz* RDPQ *je u* PSPACE.

Dokaz. Korištenjem leme 4.11 imamo da proizvoljnom grafu s podatcima G i paru vrhova v_s i v_t možemo u logaritamskom prostoru pridružiti nedeterministički konačni automat $\mathcal{N}_{G,v_s,v_t}$ koji prihvaća sve podatkovne putove koji postoje između vrha v_s i v_t. S druge strane, iz leme 4.12 slijedi da i automatu s registrima $\mathcal{A}$ i zadanom konačnom podskupu podataka D možemo pridružiti nedeterministički automat $\mathcal{N}_{\mathcal{A},D}$ koji prihvaća iste podatkovne putove kao i automat $\mathcal{A}$. Uz ograničenje da se promatraju samo podatkovni putovi čiji su svi podatci u skupu D. Pridruživanje nedeterminističkog automata $\mathcal{N}_{\mathcal{A},D}$ automatu s registrima $\mathcal{A}$ možemo provesti u polinomnom prostoru, odnosno logaritamskom ako je automat $\mathcal{A}$ fiksiran.

Zbog toga, evaluaciju kombiniranog upita $\langle \mathcal{A}, G, v_s, v_t \rangle$ možemo provesti tako da grafu s podatcima i konačnom automatu s registrima pridružimo pripadne nedeterminističke konačne automate $\mathcal{N}_{G,v_s,v_t}$ i $\mathcal{N}_{\mathcal{A},D}$. Pritom se konačni podskup D skupa podataka $\mathcal{D}$ sastoji samo od podataka koji se pojavljuju u grafu G. Označimo s $\mathcal{T}_1$ Turingov stroj koji generira nedeterministički automat $\mathcal{N}_{G,v_s,v_t}$ te s $\mathcal{T}_2$ Turingov stroj koji generira nedeterministički automat $\mathcal{N}_{\mathcal{A},D}$. Korištenjem teorema 2.20 imamo da postoji Turingov stroj $\mathcal{T}_\cap$ koji za zadana dva nedeterministička automata generira novi nedeterministički automat koji prihvaća presjek jezika zadanih nedeterminističkih automata. Turingov stroj $\mathcal{T}_\cap$ pritom radi u logaritamskom prostoru u odnosu na veličinu kôda ulaznih nedeterminističkih automata. Kompozicijom prethodno definiranih strojeva $\mathcal{T}_1$, $\mathcal{T}_2$ i $\mathcal{T}_\cap$ dobivamo Turingov stroj $\mathcal{T}$ koji za zadani upit generira nedeterministički automat $\mathcal{N}$ koji prihvaća presjek jezika nedeterminističkih automata $\mathcal{N}_{G,v_s,v_t}$ i $\mathcal{N}_{\mathcal{A},D}$. Pritom, Turingov stroj $\mathcal{T}$ radi u polinomnom prostoru, jer se generiranje automata $\mathcal{N}_{\mathcal{A},D}$ provodi u polinomnom prostoru.

Vrhovi v_s i v_t grafa G zadovoljavaju upit definiran automatom s registrima $\mathcal{A}$ ako i samo ako jezik nedeterminističkog automata $\mathcal{N}$ nije prazan. Ako postoji podatkovni put u grafu G između vrhova v_s i v_t kojeg prihvaća automat s registrima $\mathcal{A}$, slijedi da ga moraju prihvaćati i nedeterministički automati $\mathcal{N}_{G,v_s,v_t}$ i $\mathcal{N}_{\mathcal{A},D}$. Zbog toga se taj podatkovni put nalazi i u presjeku jezika tih nedeterminističkih automata, odnosno u jeziku nedeterminističkog automata $\mathcal{N}$. Slično možemo dokazati i da svaki podatkovni put koji se nalazi u jeziku nedeterminističkog automata $\mathcal{N}$ odgovara nekom podatkovnom putu između vrhova v_s i v_t kojeg prihvaća automat s registrima $\mathcal{A}$.

Evaluaciju kombiniranog upita provodimo traženjem postoji li neki podatkovni put kojeg prihvaća nedeterministički automat $\mathcal{N}$. Ako nedeterministički automat $\mathcal{N}$ promatramo kao graf čija su stanja vrhovi, a prijelazi bridovi, taj problem svodimo na pitanje postoji li put između početnog i nekog od prihvatljivih stanja nedeterminističkog automata $\mathcal{N}$. Odnosno, imamo problem dostiživosti u grafu za koji znamo da je u klasi složenosti NL. Kôd nedeterminističkog automata $\mathcal{N}$ eksponencijalno je veći od kôda ulaza $\langle \mathcal{A}, G, v_s, v_t \rangle$, jer je i kôd nedeterminističkog automata $\mathcal{N}_{\mathcal{A},D}$ eksponencijalno veći, što smo i istaknuli u dokazu leme 4.12. Zbog toga pri računanju prostora tražimo logaritam eksponencijalne funkcije i problem evaluacije upita se nalazi u klasi složenosti NPSPACE, odnosno PSPACE.

U slučaju podatkovnog upita $\langle G, v_s, v_t \rangle$ i zadanog automata s registrima $\mathcal{A}$ generiranje automata $\mathcal{N}_{\mathcal{A},D}$ je u logaritamskom prostoru. Također, konačni nedeterministički automat $\mathcal{N}$ je polinomno veći od ulaza i sama evaluacija se može provesti u klasi NL. NL-potpunost podatkovne složenosti imamo jer se konačnim automatima s registrima može izraziti postojanje proizvoljnog puta, odnosno problem dostiživosti. Q.E.D.

U prethodnom dokazu da kombinirana složenost evaluacije regularnih upita nad podatkovnim putovima pripada klasi PSPACE za složenost je presudna bila mogućnost korištenja automata s proizvoljnim brojem registara. Kada bi fiksirali broj registara na neku unaprijed zadanu konstantu, kombinirana složenost upita bi bila u logaritamskom prostoru. To povećanje složenosti koje ostvarujemo korištenjem varijabilnog broja registara je također vidljivo i prilikom dokaza donje granice kombinirane složenosti. Kako bi tu granicu mogli dokazati, uvodimo poseban tip regularnih izraza koji odgovaraju konačnim automatima s registrima. Drugi razlog uvođenja tih regularnih izraza je taj što je ljudima lakše postavljati upite u obliku regularnih izraza, nego u obliku konačnih automata. U sljedećem odjeljku promatramo te regularne izraze.

4.2. Regularni izrazi s memorijom

Regularni su nam izrazi služili kao alternativan način opisivanja nedeterminističkih konačnih automata. S druge strane, kako radimo nad grafovima s podatcima koji sadrže i podatke, nedeterministički konačni automati nisu nam dovoljni. Zbog toga smo uveli konačne automate s registrima, ali definiranje upita preko njih je kompliciranо za ljude. Ljudima je lakše raditi s regularnim izrazima nego s automatima te nam je stoga cilj definirati pripadne regularne izraze za konačne automate s registrima. Jedina razlika između konačnih automata s registrima i nedeterminističkih konačnih automata je postojanje podatkovnih prijelaza. Zbog toga je potrebno definirati novi tip regularnog podizraza kojim možemo predstaviti podatkovne prijelaze. Regularne izraze s tim novim tipom podizraza nazivamo regularnim izrazima s memorijom te formalno definiramo po uzoru na (Libkin i Vrgoč, 2012b).

Definicija 4.14 (Regularni izrazi s memorijom)**.** *Neka je zadan alfabet Σ, prirodan broj k veći od 0, skup varijabli $X = \{x_1, x_2, \ldots, x_k\}$ i skup uvjeta $\mathcal{C}_k$ s k varijabli. Skup regularnih izraza s memorijom $\mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k])$ veličine k je skup svih elemenata jezika $L_{\mathrm{REG}(\Sigma[x_1,x_2,\ldots,x_k])}$ zadanog gramatikom:*

$$S \to \dot{\emptyset} \mid \dot{\epsilon} \mid \dot{a}[c] \downarrow I \mid (S \dot{\cup} S) \mid (SS) \mid (S^{\dot{*}}),$$

pri čemu je a proizvoljan simbol alfabeta $\Sigma_\triangleright = \Sigma \cup \{\triangleright\}, \triangleright \notin \Sigma$, c proizvoljan uvjet iz skupa uvjeta $\mathcal{C}_k$, a I proizvoljan podskup skupa X.

Primijetimo da smo u definiciji uveli novi znak alfabeta $\triangleright$ koji nam u nastavku služi kako bi označili početak podatkovnog puta. Uz definiciju sintakse regularnih izraza s memorijom potrebno je definirati i njihovu semantiku kako bi ih mogli koristiti. Za definiciju semantike potreban nam je novi pojam: konkatenacije podatkovnih putova. Neka su $w_\pi = d_0 a_0 \ldots a_{n-1} d_n$ i $w'_\pi = d_n a_n \ldots a_{m-1} d_m$ dva podatkovna puta nad istim alfabetom Σ i skupom podataka $\mathcal{D}$. Konkatenaciju putova w_π i w'_π definiramo kao podatkovni put $w_\pi w'_\pi = d_0 a_0 \ldots a_{n-1} d_n a_n \ldots a_{m-1} d_m$. Primijetimo da podatci na kraju podatkovnog puta w_π i na početku podatkovnog puta w'_π moraju biti jednaki kako bi konkatenacija bila moguća. Takva se definicija prirodno proširuje i na slučaj konkatenacije više podatkovnih putova. Ako podatkovni put w_π možemo zapisati kao konkatenaciju putova $w_\pi = w_{\pi_1} w_{\pi_2} \ldots w_{\pi_l}$, tada kažemo da se podatkovni put w_π rastavlja na podatkovne putove $w_{\pi_1}, w_{\pi_2}, \ldots, w_{\pi_l}$.

Neka su zadani alfabet Σ, skup podataka $\mathcal{D}$ i prirodan broj k veći od 0. Semantiku regularnih izraza definiramo korištenjem relacije izvoda $(e, w_\pi, \tau) \vdash \tau'$, gdje je e neki

regularni izraz iz $\mathrm{REG}(\Sigma[x_1, \ldots, x_k])$ koji koristi najviše k varijabli, w_π neki podatkovni put, a τ i τ' su k-torke podatka iz skupa $\mathcal{D}_\perp^k = (\mathcal{D} \cup \{\perp\})^k$. Pritom se $\perp$ ne nalazi u skupu podataka $\mathcal{D}$, a varijablu x_i poistovjećujemo s i-tom pozicijom u k-torci podataka τ. Relaciju izvoda definiramo rekurzivno:

- $(\dot{\emptyset}, w_\pi, \tau) \vdash \tau'$ nije zadovoljena za nijedan w_π, τ i τ',
- $(\dot{\epsilon}, w_\pi, \tau) \vdash \tau'$ ako i samo ako je $w = \epsilon$ i $\tau' = \tau$,
- $(\dot{a}[c] \downarrow I, w_\pi, \tau) \vdash \tau'$ ako i samo ako je $w = ad$, pri čemu je d podatak za koji vrijedi $d, \tau \models c$ i τ' dobivamo iz τ tako da pridružimo d svakom x_i iz I,
- $((e_1 \dot{\cup} e_2), w_\pi, \tau) \vdash \tau'$ ako i samo ako $(e_1, w_\pi, \tau) \vdash \tau'$ ili $(e_2, w_\pi, \tau) \vdash \tau'$,
- $((e_1 e_2), w_\pi, \tau) \vdash \tau'$ ako i samo ako je $w_\pi = w_{\pi_1} w_{\pi_2}$ i postoji k-torka podataka τ'' takva da vrijedi $(e_1, w_{\pi_1}, \tau) \vdash \tau''$ i $(e_2, w_{\pi_2}, \tau'') \vdash \tau'$, i
- $((e^{\dot{*}}, w_\pi, \tau) \vdash \tau'$ ako i samo ako
  - $w_\pi = \epsilon$ i $\tau = \tau'$ ili
  - $w_\pi = w_{\pi_1} w_{\pi_2}$ i postoji k-torka podataka τ'' takva da $(e, w_{\pi_1}, \tau) \vdash \tau''$ i $(e^{\dot{*}}, w_{\pi_2}, \tau'') \vdash \tau'$.

Uz tako definiranu semantiku relacije izvoda možemo definirati i pojam jezika regularnog izraza s memorijom. Pritom posebno koristimo novo uvedeni znak alfabeta za početak podatkovnog puta $\triangleright$.

Definicija 4.15. *Neka je zadan alfabet Σ, skup podataka $\mathcal{D}$ i prirodan broj k veći od 0. Također, neka su zadani regularni izraz s memorijom $e \in \mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k])$ i neki podatkovni put $w_\pi = d_0 a_0 \ldots a_{n-1} d_n$. Označimo s $w_{\pi,\triangleright}$ niz znakova i podataka koji dobijemo dodavanjem znaka $\triangleright$ na početak podatkovnog puta w_π, odnosno $w_{\pi,\triangleright} = \triangleright\, d_0 a_0 \ldots a_{n-1} d_n$. Niz znakova $w_{\pi,\triangleright}$ zovemo* podatkovni put s oznakom početka.

Za regularni izraz e kažemo da izvodi *k-torku podataka $\tau \in \mathcal{D}_\perp^k$ nad podatkovnim putom w_π ako vrijedi $(e, w_{\pi,\triangleright}, \perp^k) \vdash \tau$.* Jezik regularnog izraza s memorijom *e, u oznaci $L(e)$, je skup svih podatkovnih putova w_π za koje postoji neka k-torka podataka τ koju regularni izraza e izvodi. Posebno, s $L(e, \tau, \tau')$ označavamo skup svih podatkovnih putova w_π za koje vrijedi relacija izvoda $(e, w_{\pi,\triangleright}, \tau) \vdash \tau'$.*

Kao i u slučaju običnih regularnih izraza, u nastavku rada zanemarujemo točku iznad oznaka simbola te ne pišemo zagrade kad to nije potrebno. Pored toga, ignoriramo uvjet c i skup I u izrazu $a[c] \downarrow I$ ako je uvjet c jednak ϵ, a I prazan skup. Odnosno, izraz $a[\epsilon] \downarrow \emptyset$ zapisujemo jednostavno kao a. Sličnu situaciju imamo i u slučajevima

kada vrijedi samo jedno od to dvoje, odnosno kad je ili $c = \epsilon$ ili $I = \emptyset$, te tada u zapisu ignoriramo odgovarajući dio izraza. U slučaju kada je I jednočlani skup $I = \{x\}$ regularni izraz $a[c] \downarrow I$ zapisujemo kao $a[c] \downarrow x$ ili kao $a \downarrow x$, ako je $c = \epsilon$. Također, obično ignoriramo pojavu oznake početka podatkovnog puta $\triangleright$ i pretpostavljamo da se ona nalazi na početku regularnog izraza. U nastavku prikazujemo nekoliko primjera regularnih izraza s memorijom.

Primjer 4.16. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Ako želimo izraziti tvrdnju da podatkovni put ima sve podatke jednake, to možemo ostvariti korištenjem regularnog izraza $e_1 = \downarrow x(\Sigma[x^=])^*$. Pritom koristimo $\Sigma[x^=]$ kao skraćeni zapis izraza $(a_1[x^=] \cup a_2[x^=] \cup \ldots \cup a_n[x^=])$, pri čemu su $a_i, 1 \leq i \leq n$, svi simboli alfabeta Σ. Zbog podizraza $\downarrow x$, izraz e_1 pamti prvi podatak, nakon čega provjerava za svaki sljedeći podatak odgovara li podatku sačuvanom u varijabli x.*

Drugi primjer regularnog izraza s memorijom je izraz e_2 koji izražava tvrdnju da podatkovni put sadrži barem dva jednaka podatka. Odnosno, za jezik izraza e_2 vrijedi $L(e_2) = L_{eq}$, pri čemu je L_{eq} prethodno uvedena oznaka za jezik podatkovnih putova s dva jednaka podatka. Takav regularni izraz je zadan s $e_2 = \Sigma^*(\Sigma \downarrow x)\Sigma^*(\Sigma[x^=])\Sigma^*$, pri čemu koristimo oznaku $\Sigma \downarrow x$ kao skraćeni zapis izraza $(a_1 \downarrow x \cup a_2 \downarrow x \cup \ldots \cup a_k \downarrow x)$. Podatkovni put koji odgovara tom regularnom izrazu sastoji se od proizvoljnih podputova označenih sa Σ^*, između kojih se pohranjuje podatak u varijablu x te poslije ispituje jednakost trenutnog podatka s onim pohranjenim u x.

Kao treći primjer regularnog izraza s memorijom promatramo regularni izraz zadan s $e_3 = \downarrow x(a[x^=])^* a[x^\neq] \downarrow y\,(b[y^=])^* b[x^\neq \wedge y^\neq]$. Podatkovni putovi koji odgovaraju tom izrazu sastoje se od podputa $dada \ldots da$, podputa $d'bd' \ldots d'b$ i podatka d'' nadovezanih jedan na drugi. Pritom se svi podatci d, d' i d'' međusobno razlikuju. To je osigurano s uvjetima $a[x^\neq]$ i $b[x^\neq \wedge y^\neq]$, a jednakost podataka na podputovima osigurana je kao kod izraza e_1, primjerice $\downarrow x(a[x^=])^*$.

Regularne izraze s memorijom smo uveli kako bi nam predstavljali jednostavniji zapis jezika konačnih automata s registrima. Sada kad imamo definicije izraza i automata potrebno je dokazati ekvivalentnost ta dva prikaza. Prvo dokazujemo da se svakom regularnom izrazu može pridružiti neki konačni automat s registrima. Kako nas zanimaju i složenosti provođenja operacija nad izrazima i automatima, promatramo i složenost spomenutog pridruživanja. Prilikom određivanja složenosti potrebno je kodirati regularne izraze s memorijom kako bi Turingov stroj mogao raditi s njima. Kodiranje regularnih operacija provodi se kao i kod običnih regularnih izraza, dok se kodiranje uvjeta c i skupova I provodi kao kod konačnih automata s registrima. Dokaz

da regularnim izrazima s memorijom možemo pridružiti ekvivalentne konačne automate s registrima preuzet je iz (Libkin i Vrgoč, 2012b). Ideja dokaza slična je kao i kod teorema 2.27, u kojem smo dokazali analognu tvrdnju za regularne izraze i nedeterminističke konačne automate.

Teorem 4.17. *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Postoji Turingov stroj $\mathcal{T}$ koji u logaritamskom prostoru za ulaz $\langle e \rangle$ vraća izlaz $\langle \mathcal{A} \rangle$ pri čemu je:*

- *e regularan izraz s memorijom iz $\mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k])$, gdje je k proizvoljan prirodan broj veći od 0 i*
- *$\mathcal{A}$ konačni automat s k registara za koji vrijedi $L(\mathcal{A}, \tau, \tau') = L(e, \tau, \tau')$ za svaki $\tau, \tau' \in \mathcal{D}_\perp^k$.*

Dokaz. Zadanom regularnom izrazu $e \in \mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k])$ pridružujemo automat s registrima $\mathcal{A}$ za koji vrijedi $L(\mathcal{A}, \tau, \tau') = L(e, \tau, \tau')$. Taj automat gradimo u dvije faze. U prvoj se fazi regularnom izrazu e pridružuje modificirani konačni automat s registrima $\mathcal{A}_M$. Modificirani se automat razlikuje od običnih automata s registrima utoliko što se sastoji od samo jednog tipa stanja Q_M i prijelaza $\delta_M \subseteq Q_M \times \Sigma_\triangleright \times \mathcal{C}_k \times P([k]) \times Q_M$. Primijetimo da u modificiranom automatu jedan prijelaz obrađuje po jedan par (a, d) znaka a alfabeta $\Sigma_\triangleright$ i podatka d iz skupa $\mathcal{D}$. Prihvaćanje niza alternirajućih znakova i podatka $w = a_0 d_0 a_1 d_1 \ldots a_n d_n$ s automatom $\mathcal{A}_M$ definira se na prirodan način. Za modificirani automat $\mathcal{A}_M$ koji pridružujemo regularnom izrazu e vrijedi da $\mathcal{A}_M$ može promijeniti konfiguraciju registara iz τ u τ' prilikom čitanja riječi w ako i samo ako vrijedi $(e, w, \tau) \vdash \tau'$.

Konstrukcija modificiranog automata $\mathcal{A}_M$ provodi se kao i konstrukcija nedeterminističkog automata iz običnih regularnih izraza u teoremu 2.27. Jedino je potrebno posebno obraditi slučaj podizraza $a[c] \downarrow I$. Odnosno, u slučaju da je regularni izraz ϵ ili $\emptyset$, automat $\mathcal{A}_M$ sastoji se od samo jednog stanja koje je u slučaju prazne riječi ϵ prihvatljivo. Unija dva podizraza tvori se uzimanjem odgovarajućih automata $\mathcal{A}_{M,1}$ i $\mathcal{A}_{M,2}$ te njihovim povezivanjem s novim početnim stanjem iz kojeg se nedeterministički bira neki od prvih prijelaza automata $\mathcal{A}_{M,1}$ ili $\mathcal{A}_{M,2}$. Konkatenacija se tvori povezivanjem završnih stanja automata $\mathcal{A}_{M,1}$, koji odgovara prvom podizrazu, i prvih stanja u koje automat $\mathcal{A}_{M,2}$ može prijeći. Operatoru zvijezde pridružuje se automat s novim početnim stanjem i prijelazima iz svojih završnih stanja u stanja koja slijede početno prateći odgovarajuće prijelaze. Analognim zaključivanjem kao i u slučaju dokaza teorema 2.27, lako se vidi da tako stvoreni automati zadovoljavaju uvjete teorema.

Potrebno je još opisati slučaj kada je regularni izraz zadan s $e = a[c] \downarrow I$, pri čemu je to lako definirati za modificirane automate. Modificirani automat $\mathcal{A}_M$, koji je ekvi-

valentan izrazu e, sastoji se od dva stanja: početnog q_0^M i završnog stanja q_1^M koja su povezana prijelazom $\delta = \{(q_0^M, a, c, I, q_1^M)\}$. U slučaju kada vrijedi $(e, ad, \tau) \vdash \tau'$, u automatu $\mathcal{A}_M$ možemo slijediti njegov jedini prijelaz, pri čemu se njegova konfiguracija registara mijenja na identičan način kao i konfiguracija varijabli regularnog izraza e. Slično vrijedi i u drugom smjeru da ako možemo slijediti prijelaz automata, onda vrijedi i odgovarajuća relacija izvoda za izraz e. Za generiranje ovog prijelaza potreban je logaritamski prostor kako bi pohranili trenutno i buduće stanje. Kako je i za ostatak konstrukcije potreban logaritamski prostor, slično kao i u dokazu teorema 2.27, slijedi da možemo generirati cijeli modificirani automat $\mathcal{A}_M$ u logaritamskom prostoru.

Sada kad možemo izrazu e pridružiti modificirani automat $\mathcal{A}_M$, potrebno je samo još opisati pridruživanje običnog automata s registrima $\mathcal{A}$ modificiranom automatu. U toj se konstrukciji prijelazi (q, a, c, I, q') modificiranog automata $\mathcal{A}_M$ rastavljaju u dva nova prijelaza (q, a, q'') i (q'', c, I, q') običnog automata $\mathcal{A}$. Pritom su q i q' znakovna stanja novog automata, dok je q'' novo podatkovno stanje različito od ostalih. Takva se zamjena provodi za sve prijelaze (q, a, c, I, q') kod kojih je znak a različit od oznake početka $\triangleright$. Za početno stanje običnog automata $\mathcal{A}$ uvodimo novo podatkovno stanje q_0^d. Tom stanju pridajemo prijelaze (q_0^d, c, I, d) za svaki prijelaz oblika $(q_0^M, \triangleright, c, I, q)$ modificiranog automata $\mathcal{A}_M$, pri čemu je q_0^M početno stanje modificiranog automata. Kako prihvatljiva stanja F_M modificiranog automata $\mathcal{A}_M$ imaju odgovarajuća znakovna stanja u običnom automatu $\mathcal{A}$, ta stanja su im jednaka, tj. $F = F_M$.

Prilikom čitanja nekog podatkovnog puta s oznakom početka, za svaki prijelaz modificiranog automata $\mathcal{A}_M$ možemo pratiti odgovarajući par prijelaza običnog automata $\mathcal{A}$. Također, praćenjem nekog znakovnog prijelaza običnog automata $\mathcal{A}$ točno smo odredili sljedeći podatkovni prijelaz, te taj par prijelaza odgovara nekom prijelazu modificiranog automata $\mathcal{A}_M$. U slučaju prijelaza automata $\mathcal{A}$ prilikom čitanja podatka d postoje analogni prijelazi u modificiranom automatu $\mathcal{A}_M$ koji sadrže oznaku početka $\triangleright$. Zbog toga i u tom slučaju, oba automata $\mathcal{A}$ i $\mathcal{A}_M$ završavaju u analognim znakovnim stanjima. Konstrukcija običnog automata $\mathcal{A}$ iz modificiranog automata $\mathcal{A}_M$ svodi se na prepisivanje ulaza. Potrebno je dodatno pratiti najmanji indeks neiskorištenog stanja kako bi ga mogli pridijeliti novom stanju prilikom rastavljanja prijelaza modificiranog automata $\mathcal{A}_M$. Takvih novih stanja je najviše $\mathcal{O}(n)$, jedno po prijelazu običnog automata i jedno početno, pa se zbog toga taj indeks može čuvati u logaritamskom prostoru. Slijedi da i cijeli algoritam radi u logaritamskom prostoru. Q.E.D.

Prethodni nam teorem daje jednu poveznicu između regularnih izraza s memorijom i konačnih automata s registrima. Kako bi do kraja dokazali ekvivalenciju automata

i izraza, potrebno je još provesti drugi smjer dokaza. Odnosno, potrebno je opisati pridruživanje regularnih izraza s memorijom zadanim konačnim automatima s registrima. Ideja se temelji na uvođenju generaliziranih konačnih automata s registrima čiji prijelazi sadržavaju proizvoljne regularne izraze s memorijom. Do regularnog izraza koji odgovara početnom automatu dolazimo tako da odgovarajućem generaliziranom automatu uklanjamo pojedina stanja održavajući pritom jezik koji taj generalizirani automat prihvaća. Dokaz je preuzet iz (Libkin i Vrgoč, 2012b), a temelji se na ideji korištenoj u dokazu ekvivalentnosti nedeterminističkih konačnih automata i običnih regularnih izraza iz (Sipser, 2006).

Teorem 4.18. *Neka je zadan alfabet Σ i skup podataka $\mathcal{D}$. Postoji Turingov stroj $\mathcal{T}$ koji u eksponencijalnom vremenu za ulaz $\langle \mathcal{A} \rangle$ vraća izlaz $\langle e \rangle$ pri čemu je:*

- *$\mathcal{A}$ konačni automat s k registara, gdje je k prirodan broj veći od 0 i*
- *e regularan izraz s memorijom iz $\mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k])$ pri čemu vrijedi $L(e, \tau, \tau') = L(\mathcal{A}, \tau, \tau')$ za svaki $\tau, \tau' \in \mathcal{D}_\perp^k$.*

Dokaz. Uvodimo generalizirane konačne automate s registrima u kojima prijelazi sadrže proizvoljne regularne izraze. Takvi automati imaju dva posebna stanja koja označavaju početno i završno stanje automata, pri čemu nema prijelaza u početno te prijelaza iz završnog stanja. Odnosno, za zadani prirodan broj k veći od 0 imamo generalizirani konačni automat s k registara $\mathcal{A}_G = (Q_G, q_0^G, q_F^G, \tau^G, \delta_G)$. Pritom je q_0^G početno, a q_F^G završno stanje automata, dok je δ_G relacija prijelaza definirana kao:

$$\delta_G \subseteq \left(Q_G \setminus \left\{q_F^G\right\}\right) \times \mathrm{REG}(\Sigma[x_1, x_2, \ldots, x_k]) \times \left(Q_G \setminus \left\{q_0^G\right\}\right).$$

Generalizirani automat prihvaća riječi koje se sastoje od alternirajućih znakova alfabeta i podataka, slično kao i modificirani automat iz dokaza teorema 4.17. Prihvaćanje riječi definira se na prirodan način, odnosno generalizirani automat $\mathcal{A}_G$ prihvaća riječ w ako se w može rastaviti na niz riječi $w = w_1 w_2 \ldots w_m$, pri čemu postoji niz od $m + 1$ konfiguracija automata $C_i = (i, q_i, \tau_i), 0 \leq i \leq m$, i vrijedi:

- C_0 je početna konfiguracija, tj. $C_0 = (0, q_0^G, \tau^G)$,
- C_m je završna konfiguracija, tj. $C_m = (m, q_F^G, \tau_m)$ uz neki $\tau_m \subseteq \mathcal{D}_\perp^k$, i
- za svaki prirodan broj i manji od m vrijedi $(e_i, w_i, \tau_i) \vdash \tau_{i+1}$ gdje je e_i regularni izraz s memorijom za koji vrijedi $(q_i, e_i, q_{i+1}) \in \delta_G$.

Uz prihvaćanje riječi kod generaliziranog konačnog automata, na prirodan način definiramo i jezik generaliziranog konačnog automata te oznaku $L(\mathcal{A}_G, \tau, \tau')$.

Zadanom običnom automatu $\mathcal{A}$ s k registara pridružujemo generalizirani automat $\mathcal{A}_G$ s k registara. Stanja generaliziranog automata $\mathcal{A}_G$ odgovaraju znakovnim stanjima običnog automata $\mathcal{A}$ te generalizirani automat ima dva nova stanja: početno q_0^G i završno stanje q_F^G. Između svaka dva stanja generaliziranog automata $\mathcal{A}_G$ postoji najviše jedan prijelaz, a te prijelaze tvorimo iz prijelaza običnog automata sljedeći pravila:

- $(q_0^G, e, q) \in \delta_G$ pri čemu je e unija izraza oblika $(\triangleright [c] \downarrow I)$ za sve prijelaze (q_0^d, c, I, q) običnog automata $\mathcal{A}$,
- $(q, \epsilon, q_F^G) \in \delta_G$ ako je q prihvatljivo stanje običnog automata $\mathcal{A}$ i
- $(q, e, q') \in \delta_G$ pri čemu je e unija izraza oblika $(a[c] \downarrow I)$ za sve parove prijelaza (q, a, q'') i (q'', c, I, q') običnog automata $\mathcal{A}$, pri čemu je q'' neko proizvoljno podatkovno stanje.

Lako možemo vidjeti da obični automat $\mathcal{A}$ prihvaća podatkovni put w_π ako i samo ako generalizirani automat $\mathcal{A}_G$ prihvaća pripadni podatkovni put s oznakom početka $w_{\pi,\triangleright}$.

Ostatak dokaza provodimo opisivanjem postupka uklanjanja proizvoljnog stanja generaliziranog automata uz očuvanje jezika kojeg prihvaća. Uklanjanjem svih stanja, osim početnog i završnog stanja, dobivamo jedan prijelaz čija je oznaka regularni izraz e za koji vrijedi $L(e, \tau, \tau') = L(\mathcal{A}, \tau, \tau')$. Označimo stanje koje uklanjamo u jednom koraku s $q_{rip} \in Q_G \setminus \{q_0^G, q_F^G\}$. Novi generalizirani automat $\mathcal{A}'_G$ tvorimo tako da iz starog automata uklonimo stanje q_{rip}. Svaki par prijelaza (q, e_1, q_{rip}) i (q_{rip}, e_2, q') generaliziranog automata $\mathcal{A}_G$ mijenjamo s prijelazom $(q, e_1(e_3)^* e_2 \cup e_4, q')$, gdje je:

- e_3 regularni izraz uz prijelaz iz stanja q_{rip} u samo sebe, tj. (q_{rip}, e_3, q_{rip}), i
- e_4 regularni izraz uz prijelaz koji direktno povezuje stanje q i q', tj. (q, e_4, q').

Prijelazi koji povezuju stanja q i q', a koja nisu dodatno povezana preko stanja q_{rip}, koriste se i u novom generaliziranom automatu $\mathcal{A}'_G$.

U slučaju da automat $\mathcal{A}_G$ prihvaća riječ w, tada iz odgovarajućeg prihvatljivog niza konfiguracija možemo izbaciti sve konfiguracije koje sadrže stanje q_{rip}. Novi niz konfiguracija je prihvatljiv niz konfiguracija novog generaliziranog automata $\mathcal{A}'_G$, jer su prijelazi koji su bili sadržani u stanju q_{rip} opisani novim regularnim izrazom. Slično vrijedi i u slučaju kada novi generalizirani automat $\mathcal{A}'_G$ prihvaća riječ w. Odnosno, tada svaki par konfiguracija C_i i C_{i+1}, gdje se iz C_i prelazi u C_{i+1} korištenjem novog regularnog izraza, možemo zamijeniti s odgovarajućim nizom od nekoliko konfiguracija izvornog generaliziranog automata $\mathcal{A}_G$. Zbog toga za jezike takvih generaliziranih automata vrijedi $L(\mathcal{A}_G, \tau, \tau') = L(\mathcal{A}'_G, \tau, \tau')$. Pritom novi generalizirani automat $\mathcal{A}'_G$ ima jedno stanje manje od izvornog generaliziranog automata $\mathcal{A}_G$.

Postoji Turingov stroj $\mathcal{T}$ koji zadanom konačnom automatu s registrima pridružuje generalizirani konačni automat te provodi prije opisano uklanjanje stanja dok ne preostanu samo dva stanja. Svi takvi automati prihvaćaju isti jezik, uz uvjet da podatkovnim putovima koje prihvaća početni automat $\mathcal{A}$ dodamo oznake početka. Zbog toga izraz e koji je ekvivalentan početnom automatu $\mathcal{A}$ čitamo na kraju postupka kada imamo generalizirani automat s jednim prijelazom čiji je regularni izraz baš traženi izraz e. Prilikom generiranja generaliziranog automata, uklanjanjem nekog podatkovnog stanja q_i^d u koje ulazi j, a iz kojeg izlazi l prijelaza dobivamo novih $j \cdot l$ prijelaza. Slična situacija je i kod uklanjanja jednog stanja generaliziranog automata, samo što u tom slučaju ne dobivamo novih $j \cdot l$ prijelaza, već se za toliko poveća veličina regularnih izraza automata. Povećavaju se izrazi, a ne broj prijelaza, jer je između dva stanja q i q' već mogao postojati prijelaz i u tom slučaju mijenjamo taj prijelaz s novim, koji je unija starog i novog regularnog izraza.

Konačni regularni izraz je tada veličine $\mathcal{O}(\Pi_i l_i)$, pri čemu je l_i broj prijelaza koji izlaze iz i-tog stanja generaliziranog automata, a i se računa po svim stanjima. Kako stanja ima $\mathcal{O}(n)$ i prijelaza iz svakog stanja je također $\mathcal{O}(n)$, konačni regularni izraz je veličine $\mathcal{O}(n^n)$. Ta veličina je eksponencijalna te provođenje koraka algoritma ne zahtjeva puno više vremena od tog. Zbog toga opisanu konstrukciju provodimo u eksponencijalnom vremenu. Q.E.D.

Prethodna dva teorema nam daju ekvivalenciju regularnih izraza s memorijom i konačnih automata s registrima. Primijetimo da imamo puno veću složenost pridruživanja regularnih izraza konačnim automatima od složenosti pridruživanja konačnih automata regularnim izrazima. Dokaz ove ekvivalencije, pogotovo smjer iz teorema 4.17, nam je od velike koristi u nastavku rada pri promatranju upita zadanih s regularnim izrazima. Kako bi mogli ispitati složenost tih upita i njihova svojstva, moramo prvo definirati formalizam koji dobivamo korištenjem regularnih izraza s memorijom.

Definicija 4.19 (Regularni upiti s memorijom nad podatkovnim putovima)**.** *Neka su zadani alfabet Σ i skup podataka $\mathcal{D}$. Formalizam regularnih izraza s memorijom je formalizam $\mathcal{F}_e = (\mathcal{E}, L)$ gdje je $\mathcal{E}$ skup svih regularnih izraza s memorijom nad Σ i $\mathcal{D}$ za sve brojeve varijabli k. Pritom, funkcija L pridružuje svakom izrazu $e \in \mathcal{E}$ njegov jezik $L(e)$.* Regularni upit s memorijom nad podatkovnim putem ($\mathrm{RDPQ_{mem}}$) *je upit zadan u formalizmu $\mathcal{F}_e$.*

Sada promatramo podatkovnu i kombiniranu složenost regularnih upita s memorijom. Iz teorema 4.17 imamo da u logaritamskom prostoru možemo regularnom izrazu s memorijom pridružiti ekvivalentni konačni automat s registrima. Zbog toga prilikom

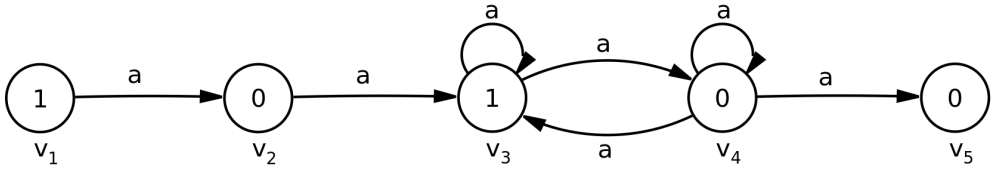
evaluacije upita zadanog s regularnim izrazom s memorijom možemo prvo provesti to pridruživanje, a onda evaluirati upit korištenjem dobivenog konačnog automata s registrima. Kako se pridruživanje provodi u logaritamskom prostoru, složenosti evaluacije upita koje smo dobili u teoremu 4.13 za konačne automate s registrima vrijede i za regularne izraze. Odnosno, uzimajući u obzir da s regularnim izrazima možemo izraziti i postojanje proizvoljnog puta, imamo sljedeći korolar.

Korolar 4.20. *Podatkovna složenost upita iz* $\mathrm{RDPQ}_{\mathrm{mem}}$ *je* NL*-potpuna. Kombinirana složenost upita iz* $\mathrm{RDPQ}_{\mathrm{mem}}$ *je u* PSPACE.

Preostaje nam još dokazati drugi dio tvrdnje, koji smo prethodno najavili, vezan uz kombiniranu složenost evaluacije upita zadanih s konačnim automatima s registrima. Odnosno, preostaje nam dokazati da je ta složenost PSPACE-potpuna. Kako bi to dokazali, prvo dokazujemo PSPACE-težinu kombinirane složenosti evaluacije upita zadanih s regularnim izrazima s memorijom. Kao i u slučaju dokaza PSPACE-težine problema $\mathrm{coE}_{\mathrm{AUT}}$ nepraznosti jezika konačnih automata s registrima provodimo redukciju s PSPACE-potpunog problema $\mathrm{coUNIV}_{\mathrm{NKA}}$. Pritom je $\mathrm{coUNIV}_{\mathrm{NKA}}$ prije definirani problem ispitivanja postoji li riječ koju zadani nedeterministički konačni automat $\mathcal{N}$ ne prihvaća. Za zadani graf i par vrhova, konstruirani regularni izraz jedino prihvaća podatkovni put koji odgovara simulaciji nedeterminističkog automata $\mathcal{N}$, a koja završava s neprihvaćanjem riječi. Dokaz je preuzet iz (Libkin i Vrgoč, 2012a).

Teorem 4.21. *Kombinirana složenost upita iz* $\mathrm{RDPQ}_{\mathrm{mem}}$ *je* PSPACE*-teška.*

Dokaz. Provodimo redukciju s PSPACE-potpunog problema $\mathrm{coUNIV}_{\mathrm{NKA}}$ u kojem ispitujemo prihvaća li neki nedeterministički konačni automat sve riječi nekog alfabeta. Zadanom nedeterminističkom konačnom automatu $\mathcal{N}$ pridružujemo graf s podatcima G, par vrhova (v_s, v_t) te regularni izraz s memorijom e. Pritom je graf G zadan slikom 4.5, pri čemu je vrh v_s jednak vrhu v_1, a v_t jednak vrhu v_5. U tom grafu postoji podatkovni put između vrhova v_1 i v_t koji se nalazi u jeziku regularnog izraza s memorijom e ako i samo ako nedeterministički konačni automat $\mathcal{N}$ ne prihvaća neku riječ alfabeta.



Slika 4.5: Prikaz grafa s podatcima uz dokaz teorema 4.21.

Regularni izraz e tvorimo slično kao i konačni automat s registrima u dokazu teorema 4.8. Odnosno, regularni izraz simulira ponašanje nedeterminističkog konačnog

automata korištenjem $2n + 2$ varijable, gdje je n broj stanja nedeterminističkog automata $\mathcal{N}$. Bez smanjenja općenitosti, pretpostavljamo da su stanja nedeterminističkog konačnog automata $\mathcal{N}$ zadana s $Q = \{q_1, q_2, \ldots, q_n\}$, pri čemu je q_1 početno stanje. Regularni izraz e definiramo nad jednočlanim alfabetom $\Sigma = \{a\}$ i skupom podataka $\mathcal{D} = \{0, 1\}$. Isto kao i u dokazu teorema 4.8, imamo dvije varijable t i f kako bi pohranili oznake s kojima označavamo nalazimo li se u nekom stanju. Također, imamo radne varijable $s_1, s_2, \ldots, s_n$ koje predstavljaju stanja automata te pomoćne varijable $w_1, w_2, \ldots, w_n$ koji nam služe u računanju novog skupa stanja automata $\mathcal{N}$.

Konačni regularni izraz e sastoji se od tri izraza povezana kao:

$$e = e_{pocetak}(e_{korak})^* e_{kraj},$$

pri čemu s $e_{pocetak}$ predstavljamo početnu inicijalizaciju varijabli, s e_{korak} jedan korak simulacije automata $\mathcal{N}$ te s e_{kraj} kraj simulacije automata $\mathcal{N}$ u kojoj smo došli u skup stanja od kojih nijedno nije prihvatljivo. Inicijalizacijski regularni izraz $e_{pocetak}$ zapisujemo kao:

$$e_{pocetak} = \downarrow t\,(a[t^{\neq}] \downarrow f)\,(a[t^{=}] \downarrow s_1)\,(a[f^{=}] \downarrow \{s_2, s_3, \ldots, s_n\}).$$

Slijedeći brid (v_1, a, v_2) grafa G prvo postavljamo varijable t i f na različite vrijednosti 1 i 0. Nakon toga, slijedeći brid (v_2, a, v_3) postavljamo varijablu s_1 na vrijednost varijable t, a varijable $s_2, s_3, \ldots, s_n$ na vrijednost varijable f korištenjem brida (v_3, a, v_4), Time smo označili da se automat $\mathcal{N}$ na početku nalazi samo u stanju q_1.

Kako bi definirali regularni izraz e_{korak} koji simulira čitanje jednog znaka alfabeta, prvo moramo definirati računanje skupa stanja u koje iz stanja q_i prelazi automat $\mathcal{N}$ čitajući znak b. Neka je zadan skup stanja $Q_{q_i,b} = \{q_{j_1}, q_{j_2}, \ldots, q_{j_m}\}$ automata $\mathcal{N}$ za koje vrijedi da je $(q_i, b, q_{j_l}), 1 \leq l \leq m$, prijelaz automata $\mathcal{N}$. Imamo regularni izraz

$$u_{q_i,b} = (a[t^{=} \wedge s_i^{=}]a[t^{=}] \downarrow \{w_{j_1}, w_{j_2}, \ldots, w_{j_m}\}) \cup (a[f^{=} \wedge s_i^{=}]),$$

koji provodi to računanje postavljanjem vrijednosti radnih varijabli koje odgovaraju skupu stanja $Q_{q_i,b}$ na vrijednost varijable t, u slučaju kada je stanje q_i aktivno. Pritom, $a[t^{=} \wedge s_i^{=}]$ ispituje aktivnost stanja q_i, a $a[t^{=}] \downarrow \{w_{j_1}, w_{j_2}, \ldots, w_{j_m}\}$ postavlja stanja $Q_{q_i,b}$. Ako je skup $Q_{q_i,b}$ prazan, odgovarajući regularni izraz zadajemo kao $u_{q_i,b} = \epsilon$. Varijable se ispituju i postavljaju korištenjem vrhova v_3 i v_4 grafa G, između kojih se u jednom koraku može doći do bilo kojeg od podataka 0 ili 1.

Korištenjem regularnih izraza $u_{q_i,b}$ kojima mijenjamo varijable ovisno o jednom stanju i znaku alfabeta možemo definirati promjenu neovisnu o znaku i stanju:

$$e_{promjena} = \bigcup_{b \in \Sigma} u_{q_1,b} u_{q_2,b} \ldots u_{q_n,b},$$

pri čemu s unijom nedeterministički biramo sljedeći znak b za koji se simulira prijelaz, a onda konkatenacijom izraza $u_{q_i,b}$ računamo prijelaze za svako stanje automata $\mathcal{N}$. Regularni izraz e_{korak} sadrži još i izraz koji poništava vrijednosti sačuvane u pomoćnim varijablama, te izraz koji na kraju računanja promjene iz izraza e_{korak} prepisuje pomoćne u radne varijable. Konačno, izraz e_{korak} je zadan kao:

$$e_{korak} = (a[f^{=}] \downarrow \{w_1, w_2, \ldots, w_n\}) e_{promjena} (a[w_1^{=}] \downarrow s_1)(a[w_2^{=}] \downarrow s_2) \ldots (a[w_n^{=}] \downarrow s_n),$$

gdje prvo postavljamo sve pomoćne varijable na vrijednost varijable f, nakon čega računamo promjenu i korištenjem izraza oblika $a[w_i^{=}] \downarrow s_i$ prepisujemo pomoćne u radne varijable. Podatkovni put koji slijedi ta postavljanja varijabli također možemo ostvariti korištenjem petlji između vrhova v_3 i v_4.

Zadnji regularni izraz e_{kraj} izražava da možemo završiti simulaciju u slučaju kada nijedno prihvatljivo stanje automata $\mathcal{N}$ nije aktivno. Odnosno, kada odgovarajuće radne varijable ne sadrže podatak jednak podatku iz varijable t. Za zadani skup prihvatljivih stanja $F = \{q_{h_1}, q_{h_2}, \ldots, q_{h_m}\}$ automata $\mathcal{N}$ imamo prijelaz:

$$e_{kraj} = aa[f^{=} \wedge s_{h_1} \wedge s_{h_2} \wedge \ldots \wedge s_{h_m}].$$

Koristimo dva znaka a jer se nakon prethodnog računanja koraka simulacije možemo nalaziti u vrhu v_3, pa korištenjem najmanje dva prijelaza možemo doći do vrha v_5.

Prethodno smo opisali kako povezujemo određene podizraze izraza e s podatkovnim putovima grafa G. Iz toga lako vidimo da podatkovni put između vrhova v_1 i v_5 koji se nalazi u jeziku izraza e postoji ako i samo ako automat $\mathcal{N}$ ne prihvaća neku riječ. Kako se struktura regularnog izraza ponavlja, vrijedi da pamćenjem nekoliko indeksa i znakova alfabeta možemo u logaritamskom prostoru konstruirati regularni izraz e. Korištenjem još konstantnog prostora koji nam je potreban za konstruiranje grafa G imamo da u logaritamskom prostoru možemo za zadani ulaz $\langle \mathcal{N} \rangle$ vratiti željeni izlaz $\langle e, G, v_s, v_t \rangle$. Time smo dokazali PSPACE-težinu kombinirane složenosti evaluacije upita iz $\mathrm{RDPQ}_{\mathrm{mem}}$. Q.E.D.

Primijetimo da smo u prethodnom dokazu imali fiksan graf s podatcima, jednočlani alfabet te da smo operator zvijezde koristili samo jednom. Iz toga možemo zaključiti da je donja granica kombinirane složenosti regularnih izraza s memorijom poprilično čvrsta te da su nam potrebna nešto veća ograničenja da se ona spusti. Kako za gornju granicu kombinirane složenosti evaluacije upita iz $\mathrm{RDPQ}_{\mathrm{mem}}$ vrijedi da je u klasi PSPACE slijedi i PSPACE-potpunost te kombinirane složenosti. Također, korištenjem rezultata teorema 4.17 o pridruživanju automata regularnim izrazima, imamo

i PSPACE-potpunost kombinirane složenosti evaluacije upita RDPQ zadanih s konačnim automatima s registrima. Formalno to iskazujemo kroz sljedeća dva korolara.

Korolar 4.22. *Kombinirana složenost upita iz* $\mathrm{RDPQ}_{\mathrm{mem}}$ *je* PSPACE*-potpuna.*

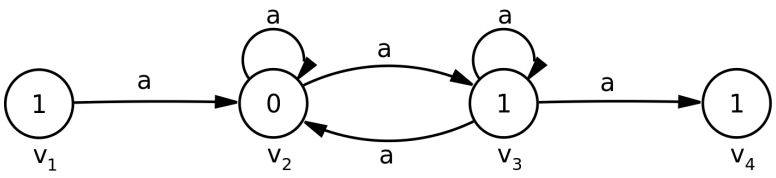
Korolar 4.23. *Kombinirana složenost upita iz* RDPQ *je* PSPACE*-potpuna.*

Konačne automate s registrima i regularne izraze s memorijom smo uveli kao formalizme u kojima je složenost upita prihvatljiva. Bez obzira na prihvatljivu podatkovnu složenost, PSPACE-potpunost kombinirane složenosti tih upita nam je još uvijek prevelika te bi željeli pronaći formalizam s još manjom složenosti. Već smo spomenuli da su potrebna veća ograničenja da se kombinirana složenost upita spusti. Zbog toga u nastavku promatramo jedno takvo ograničenje koje nam omogućava postavljanje upita čija kombinirana složenost je NP-potpuna. Ti su upiti zadani s regularnim izrazima s memorijom u kojima se ne koristi operator zvijezde. NP-potpunost kombinirane složenosti evaluacije tih upita dokazana je u (Libkin i Vrgoč, 2012a), pri čemu je kod dokaza NP-težine korištena redukcija s problema pronalaženja klike u grafu. U nastavku prezentiramo dokaz te tvrdnje. Pritom je dokaz da se kombinirana složenost nalazi u NP preuzet iz (Libkin i Vrgoč, 2012a), dok je dokaz NP-težine drugačiji od njihovog dokaza. U našem dokazu provodimo redukciju s poznatog NP-potpunog problema SAT ispunjivosti formule zadane u propozicionalnoj logici, čija je NP-potpunost dokazana u (Cook, 1971; Levin, 1973).

Propozicija 4.24. *Kombinirana složenost upita zadanih s regularnim izrazima s memorijom koji ne koriste operator zvijezde je* NP*-potpuna.*

Dokaz. Prvo dokazujemo da je kombinirana složenost takvog upita u NP, odnosno da je jezik $\langle e, Q, v_s, v_t \rangle$ u klasi NP. Kako se od regularnih operatora jedino korištenjem operatora zvijezde dobivaju proizvoljno veliki podatkovni putovi, slijedi da regularni izraz bez operatora zvijezde prihvaća podatkovne putove ograničene veličine. Odnosno, za takav regularni izraz e duljine n vrijedi da najdulji podatkovni put kojeg može prihvatiti ima najviše n znakova alfabeta. Nedeterministički Turingov stroj tada može nedeterministički izabrati put između vrhova v_s i v_t koji prati najviše n bridova. Također nedeterministički se bira koji se od elemenata svake unije iz regularnog izraza e koriste. Uz takve nedeterminističke odabire možemo u polinomnom vremenu provjeriti odgovara li podatkovni put zadanom regularnom izrazu, pa je kombinirana složenost evaluacije upita u klasi NP.

Dokaz NP-težine kombinirane složenosti evaluacije upita provodimo redukcijom s NP-potpunog problema SAT ispunjivosti formule propozicionalne logike. Zadanoj



Slika 4.6: Prikaz grafa s podatcima uz dokaz teorema 4.24.

formuli propozicionalne logike ϕ pridružujemo graf s podatcima G, vrhove v_s i v_t te regularni izraz s memorijom e koji ne sadrži operator zvijezde. Graf G jednak je grafu sa slike 4.6 te su vrhovi v_s i v_t jednaki $v_s = v_1$ i $v_t = v_4$. Ako formula ϕ ima k varijabli $x_1, x_2, \ldots, x_k$, tada će regularni izraz e imati istih k varijabli te dodatnu varijablu t. Varijabla t nam slično kao i prije služi za označavanje koje od preostalih varijabli su istinite. Skup podataka sastoji se od samo dva podatka $\mathcal{D} = \{0, 1\}$, a alfabet je jednočlani skup $\Sigma = \{a\}$.

Regularni izraz s memorijom e je zadan kao:

$$e = \downarrow t\, a(a \downarrow x_1)\,(a \downarrow x_2) \ldots (a \downarrow x_k)\, aa[t^{=} \wedge \phi'],$$

gdje je ϕ' logička formula jednaka formuli ϕ, pri čemu su varijable x_i zamijenjene s izrazima $x_i^{=}$. Regularni izraz nad grafom G prvo postavlja varijablu t na vrijednost 1, nakon čega proizvoljno postavlja preostale varijable x_i vrteći se u petljama između vrhova v_2 i v_3. Dva znaka a na početku izraza nam služe kako bi mogli doći do proizvoljnog vrha iz para vrhova v_2 i v_3, te postaviti varijablu x_1 na bilo koju od vrijednosti 0 ili 1. Zadnja dva a nam služe kako bi u slučaju da nas je zadnja varijabla koju smo postavili dovela u vrh v_2 mogli doći u vrh v_4.

U vrhu v_4 ispitujemo ispunjuje li proizvoljno postavljanje varijabli x_i logičku formulu ϕ'. Zbog povezanosti te formule i formule ϕ, podatkovni put između v_1 i v_4 koji se nalazi u jeziku izraza e postoji ako i samo ako je ϕ ispunjiva. Graf G je konstantan, a regularni izraz e tvorimo brojanjem varijabli x_i formule ϕ prilikom generiranja podizraza $(a \downarrow x_i)$, te prepisivanjem formule ϕ na kraju izraza. Zbog toga traženu redukciju s problema SAT provodimo u logaritamskom prostoru. Q.E.D.

Primijetimo da iako smo s takvom restrikcijom dobili regularne izraze čija bi kombinirana složenost trebala biti manja, takvi regularni izrazi nam nisu od izrazito velike koristi. Naime, kako je i spomenuto u dokazu, takvi regularni izrazi mogu izraziti samo svojstva podatkovnih putova konačne dužine. Zbog toga bi trebalo proučiti drugačija ograničenja koja bi mogla dovesti do praktičnijeg formalizma.

5. Zaključak

U ovom su se radu proučavali upiti nad grafovima s podatcima temeljeni na konačnim automatima i regularnim izrazima. Kao uvod u glavnu temu, prezentirani su neki osnovni teoremi iz teorije običnih konačnih automata i regularnih izraza. Posebice se promatrala računska složenost pojedinih problema vezanih uz te konačne automate i regularne izraze, čiji su se rezultati koristili u kasnijim dokazima. Definirali smo pojam grafa s podatcima i upita nad njime. Pritom nam je posebnu važnost predstavljao način postavljanja tih upita te nas je doveo do pojma formalizma. Eliminacijom pojedinih formalizama zbog prevelike podatkovne složenosti evaluacije upita, kao prihvatljiv formalizam smo izabrali konačne automate s registrima. Proučili smo neka svojstva konačnih automata s registrima i složenosti evaluacije upita. Također, uveli smo regularne izraze s memorijom i dokazali njihovu ekvivalenciju s konačnim automatima s registrima. Proučili smo i neka svojstva tih regularnih izraza te dokazali čvrste granice za složenosti evaluacije takvih upita.

Dobivena kombinirana složenost evaluacije upita zadanih s regularnim izrazima s memorijom, odnosno s konačnim automatima s registrima, još uvijek je prevelika. Zbog toga bi se u nastavku mogla proučavati neka ograničenja takvih regularnih izraza, kao što je primjerice korištenje regularnih izraza s jednakosti (Libkin i Vrgoč, 2012a). Kako smo i prije spomenuli, varijabilan broj registara jedna je od glavnih prepreka za nižu kombiniranu složenost. Zbog toga bi za praktične svrhe korištenja takvih izraza mogli promatrati i slučaj kada imamo najviše neki fiksiran broj registara. Također, postoji još pitanja vezanih uz regularne izraze i konačne automate koja ovdje nismo obradili, kao što je pitanje kada je jezik jednog regularnog izraza podskup jezika nekog drugog izraza. Složenosti tih pitanja za razne modifikacije regularnih izraza i konačnih automata predstavljaju još jedan mogući smjer proučavanja.

LITERATURA

Renzo Angles i Claudio Gutierrez. Survey of Graph Database Models. *ACM Computing Surveys*, 40(1), 2008.

Stephen A. Cook. The complexity of theorem-proving procedures. U *Proceedings of the third annual ACM symposium on Theory of computing (STOC)*, stranice 151–158, 1971.

Steven Fortune, John Hopcroft, i James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

John E. Hopcroft, Rajeev Motwani, i Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, poglavlje 2,3. Addison-Wesley, 2 izdanju, 2001.

Leonid A. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3): 115–116, 1973.

Leonid Libkin i Domagoj Vrgoč. Regular Path Queries on Graphs with Data. U *Proceedings of the 15th International Conference on Extending Database (ICDT)*, 2012a.

Leonid Libkin i Domagoj Vrgoč. Regular Expressions for Data Words. U *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, 2012b.

Albert R. Meyer i Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. U *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, stranice 125–129, 1972.

Michael O. Rabin i Dana S. Scott. Finite Automata and Their Decision Problem. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

Luc Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. U *Computer Science Logic*, stranice 41–57, 2006.

Michael Sipser. *Introduction to the Theory of Computation*, poglavlje 1–5,7–9. Thomson Course Technology, 2 izdanju, 2006.

Moshe Y. Vardi. The complexity of relational query languages. U *Proceedings of the fourteenth annual ACM symposium on Theory of computing (STOC)*, stranice 137–146, 1982.

Mladen Vuković. *Matematička logika*, poglavlje 1.4. Element, 2009.

Upiti nad grafovima s podatcima i proširenja regularnih izraza

Sažetak

Razvoj računala i prikupljanje sve veće količine informacija dovelo je do povećane važnosti različitih modela baza podataka. Pritom grafovi s podatcima predstavljaju posebno dobar model za opisivanje informacija dobivenih iz društvenih i bioloških mreža, te kao prikaz semantičkog weba. U ovom se radu proučava pitanje postavljanja efikasnih upita nad grafovima s podatcima. Ispitana su neka svojstva grafova s podatcima i pripadnih formalizama za postavljanje upita, među kojima su poglavito svojstva vezana uz složenost upita. Kao jedni efikasni tipovi formalizama za definiranje upita predstavljeni su konačni automati s registrima i regularni izrazi s memorijom. Dokazana je ekvivalentnosti ta dva načina definiranja upita i PSPACE-potpunost kombinirane složenosti evaluacije njihovih upita.

Ključne riječi: Regularni jezici, grafovi s podatcima, formalizmi za definiranje upita, podatkovna i kombinirana složenost, konačni automati s registrima, regularni izrazi s memorijom.

Queries on Graphs with Data and Extensions of Regular Expressions

Abstract

Development of computers and the acquisition of greater amounts of information led to an increase in importance of different database models. Graphs with data present an especially good model for describing information gathered from social and biological networks, and for representing the semantic Web. This work investigates the question of how to efficiently query graphs with data. Some properties of graphs with data and their formalisms for specifying queries were examined with the main focus being on properties related to query complexity. Register automata and regular expressions with memory were introduced as one efficient formalism. The equivalence of register automata and regular expressions with memory was proven, as well as PSPACE-completeness of the combined complexity of their queries.

Keywords: Regular languages, graphs with data, formalisms for specifying queries, data and combined complexity, register automata, regular expressions with memory.