Univerzitet u Zenici
Pedagoški fakultet u Zenici

Student: Penjić Safet

Seminarski rad

Tema: L i NL klase jezika

Sadržaj

Uvod

- Razlikujemo dvije vrste problema, preciznije dvije vrste izlaza mogu biti za problem kao njegovo rješenje:
  1. jedan bit, koji odgovara na pitanje Da ili Ne;
     (ovo je model u kojem razmatramo samo proleme koje treba odlučiti – tj. problem koji za odgovor zahtjeva samo Da/Ne)
  2. niz simbola iz istog alfabeta iz kojeg je i ulaz
     (u ovom modelu također dozvoljavamo optimizaciju problema – tj. problem koji zahtjeva izračunavanje za najbolji odgovor iz nekog skupa mogućih odgovora)

- Razmatraćemo kompleksnost izračunavanja problema u zahtjevima količine prostora (memorije), koji oni zahtijevaju

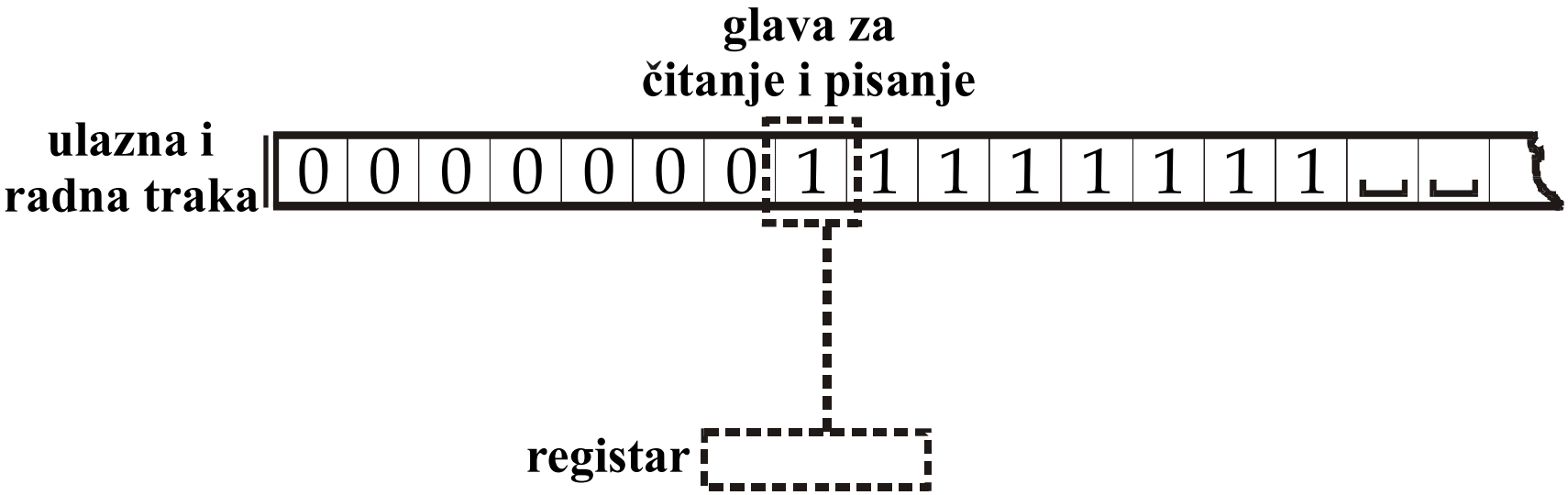
Vremenska kompleksnost

- Analiza problema u najgorem slučaju odnosi se na najduže potrebno vrijeme na svim ulazima određene dužine

Definicija

Neka je M deterministička Turing mašina koja staje na svim ulazima. **Potrebno vrijeme** ili **vremenska kompleksnost** od *M* je funkcija $f: \mathbf{N} \rightarrow \mathbf{N}$ gdje je *f(n)* maksimalni broj koraka koje M koristi na nekom ulazu dužine n. Ako je *f(n)* potrebno vrijeme od M, kažemo da *M* koristi *f(n)* vremena i da je *M* *f(n)* vremenska Turing mašina. Uobičajeno, koristimo *n* da predstavimo dužinu ulaza.

Model TM



Model TM

Definicija

Neka je t: N $\rightarrow$ R^+ funkcija. Definišemo **vremenski kompleksnu klasu**, **TIME(*t(n)*)** kao porodicu (kolekciju) svih jezika koje su odlučive sa $O(t(n))$ vremenskom Turing mašinom.

Klasa P

Definicija

P je klasa jezika koji su odlučivi u polinomijalnom vremenu na determinističkoj jednoj-traci Turing mašini. Drugim riječima,

$$P = \cup_k \text{TIME}(n^k).$$

- PATH = {<*G*, *s*, *t*> | G je orjentisan graf koji ima direktan put od s do t}
- RELPRIME = {<x, y> | x i y su relativno prosti}
- PATH i RELPRIME pripadaju P klasi jezika (objasniti zašto)

Klasa NP

Definicija

Provjeravač za jezik A je algoritam V, gdje

$$A=\{ w \mid V \text{ prihvata } \langle w, c\rangle \text{ za neki string } c\}.$$

Definicija

NP je klasa jezika koje imaju polinomijalno vremenski provjeravač.

Teorema

Jezik je u NP ako i samo ako je odlučiv pomoću neke nedeterminističke polinomijalno vremenske Turing mašine.

Klasa NP

Definicija

NTIME(*t(n)*) = {L | L je jezik odlučiv pomoću $O(t(n))$ vremenske nedeterminističke Turing mašine}.

Posljedica

$$NP = \cup_k NTIME(n^k).$$

- CLIQUE = {<*G*, *k*> | G je neorjentisan graf sa k-klikom}
- SUBSET-SUM = {<*S*, *t*> | S={x_1, ..., x_k} i za neki {y_1, ..., y_k} $\subseteq$ {x_1, ..., x_k} imamo $\Sigma y_i = t$ }
- CLIQUE i SUBSET-SUM pripadaju NP klasi jezika (objasniti zašto)

Klasa PSPACE

Definicija

Neka je M deterministička Turing mašina koja staje na svim ulazima. Prostorna kompleksnost od M je funkcija f: $\mathbf{N} \rightarrow \mathbf{R}^+$, gdje je $f(n)$ maksimalni broj ćelija trake koje M skenira na nekom ulazu dužine n. Ako je $f(n)$ prostorna kompleksnost od M, također kažemo da M koristi $f(n)$ prostora.

Ako je M nedeterministička Turing mašina gdje sve branše staju na svim ulazima, definišemo prostornu kompleksnost $f(n)$ kao maksimalni broj ćelija trake koje M skenira na nekoj branši svog izračunavanja za neki ulaz dužine n.

Klasa PSPACE

Definicija

Neka je f: $\mathbf{N} \rightarrow \mathbf{R}^+$ funkcija. Prostorne kompleksne klase SPACE(f(n)) i NSPACE(f(n)), definisane su kao:

SPACE(f(n)) = {L | L je jezik odlučiv sa O(f(n)) prostornom determinističkom Turing mašinom}.

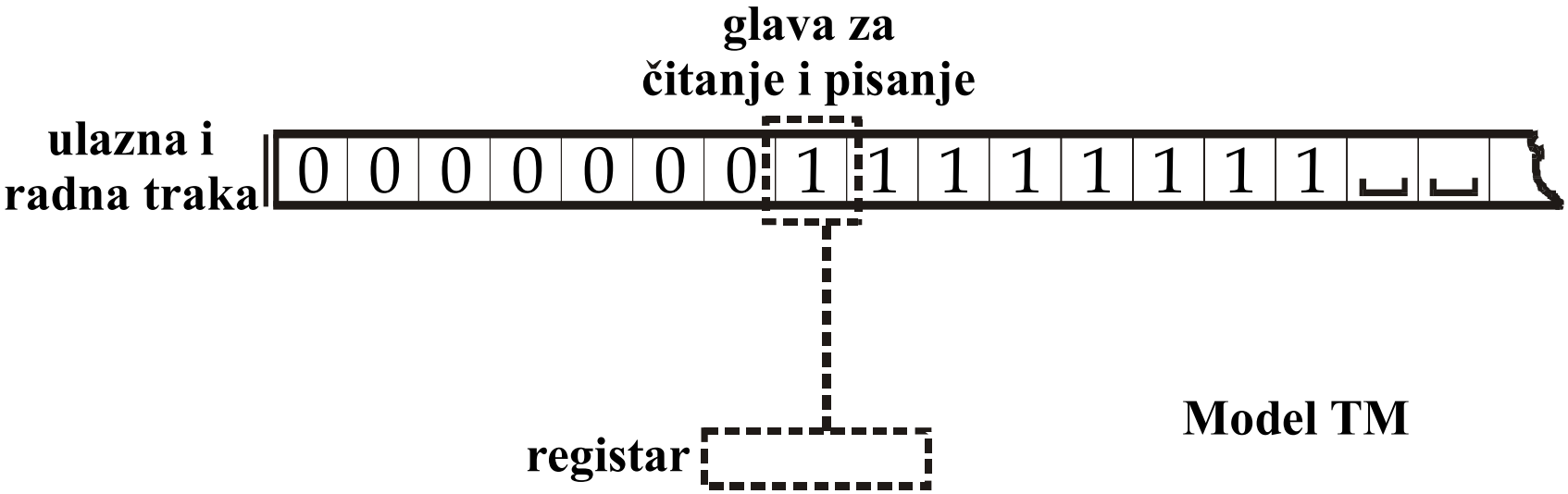
NSPACE(f(n)) = {L | L je jezik odlučiv sa O(f(n)) prostornom nedeterminističkom Turing mašinom}

Definicija

PSPACE je klasa jezika koji su odlučivi sa polinomijalno prostornom determinističkom Turing mašinom. Drugim riječima,

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k).$$

Klasa PSPACE

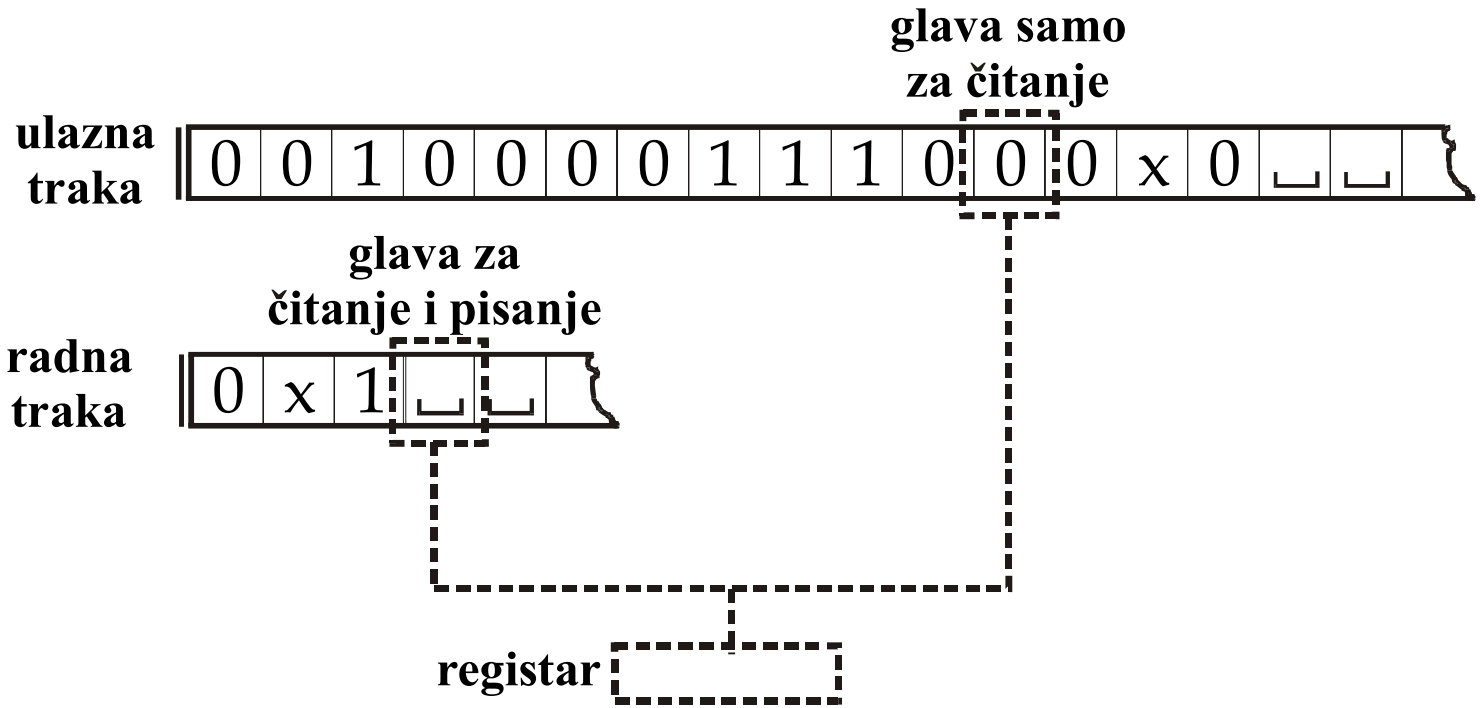


- SAT = {<*f*> | f je zadovoljavajuća Boolean formula}
- ALL_{NFA} = {<*A*> | A je NFA i L(A) = Σ^* }
- SAT i ALL_{NFA} pripadaju PSPACE klasi jezika (objasniti zašto)

Logaritamski prostor

- Kako je polinomijalni prostor tako moćan, prirodno se nameće razmatranja problema u ograničenijoj prostorno kompleksnoj klasi
- Čak je i linearni prostor dovoljan da riješi SAT problem
- Da bi dobili podlinearnu prostornu kompleksnost, zanemarujemo čelije koje sadrže ulaz (za koje pretpostavimo da su čelije samo za čitanje) i brojimo samo radni prostor

Logaritamski prostor



- Prostorno ograničeno izračunavanje. Jedino ćelije korištene na radnoj traci se broje u prostornoj kompleksnosti

Logaritamski prostor

- Možemo razmišljati o samo-čitaj ulaznoj traci kao o DVD-rom uređaju
- Za podlinearna prostorna ograničenja, koristimo samo model sa dvije trake

Definicija

L je klasa jezika koji su odlučivi u logaritamskom prostoru na determinističkoj Turing mašini

$$L = SPACE(\log n).$$

NL je klasa jezika koji su odlučivi u logaritamskom prostoru na nedeterminističkoj Turing mašini

$$NL = NSPACE(\log n).$$

Problemi u L i NL

Koje vrste problema su u L i NL?

U logaritamskom prostoru možemo zapamtiti
- fiksan broj brojača (koji idu sve do dužine ulaza)
- fiksiran broj pokazivača na poziciju u ulaznom stringu

Pa,
- L Sadrži sve probleme koji zahtjevaju samo fiksan broj brojača/pokazivača za rješenje
- NL sadrži sve probleme koji zahtjevaju samo fiksiran broj brojača/pokazivača za potvrdu datog rješenja

Primjer 1

- $A = \{<0^k 1^k> \mid k \geq 0\}$. A pripada L klasi jezika
- Riješimo zadatak u linearnom prostoru

Primjer 1

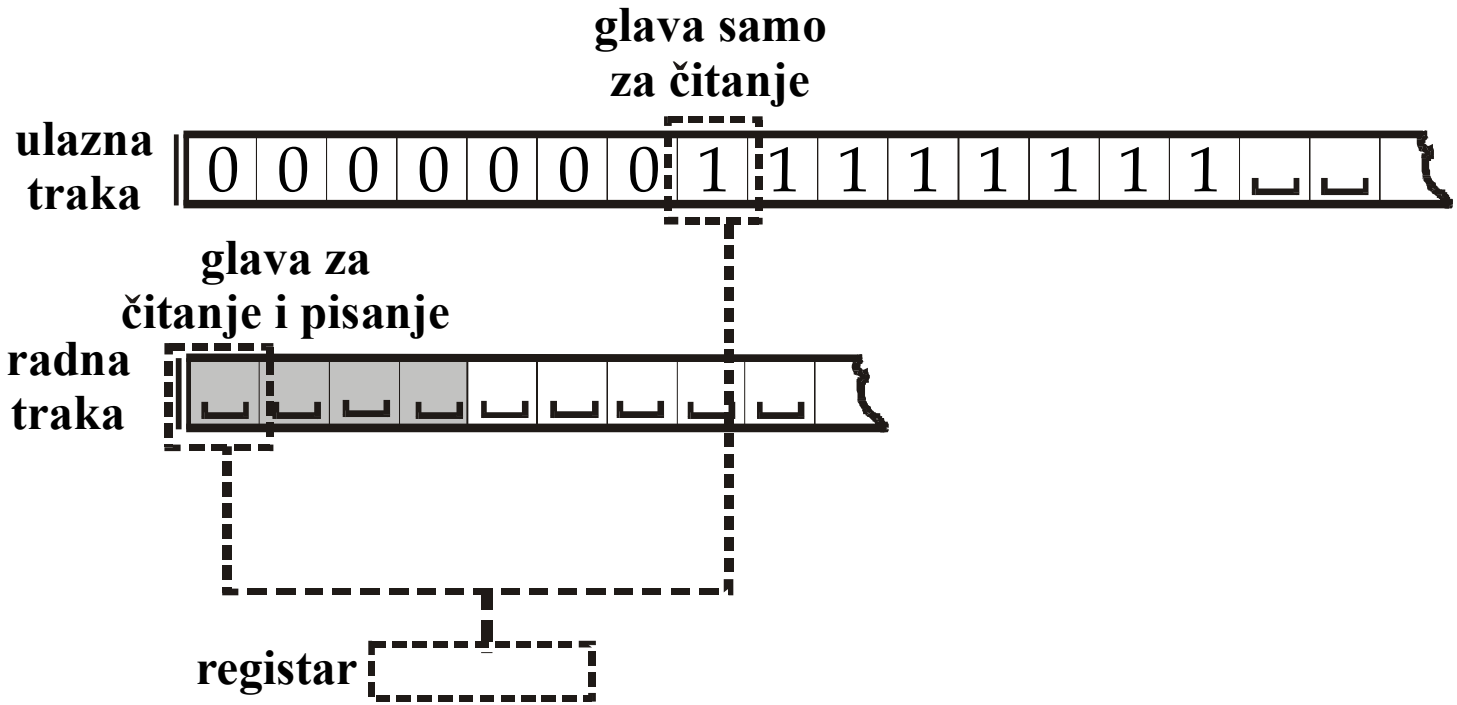
Algoritam:

M_1= "Na ulazu *w*, gdje je *w* neki string:

1. Skeniraj traku i *odbaci* ako nađeš 0 desno od 1.
2. Ponavljaj sve dok neka 0-la i neka 1-ca ostanu na traci
    3. Skeniraj traku, i provjeri da li je ukupan broj 0-la i 1-ca koje su ostale na traci paran ili neparan. Ako je neparan, *odbaci*.
    4. Ponovo skeniraj traku, križaj svaku drugu nulu počevši od prve, pa onda križaj svaku drugu 1-cu počevši od prve 1-ce.
5. Ako nema 0-la i nema 1-ca na traci, *prihvati*.
   U suprotnom, *odbaci*."

Primjer 1

- Logaritamsko prostorna TM za A ne može križati 0-le i 1-ce koje nađe na ulaznoj traci zato što je ta traka samo za čitanje. Umjesto toga...



Korak 1

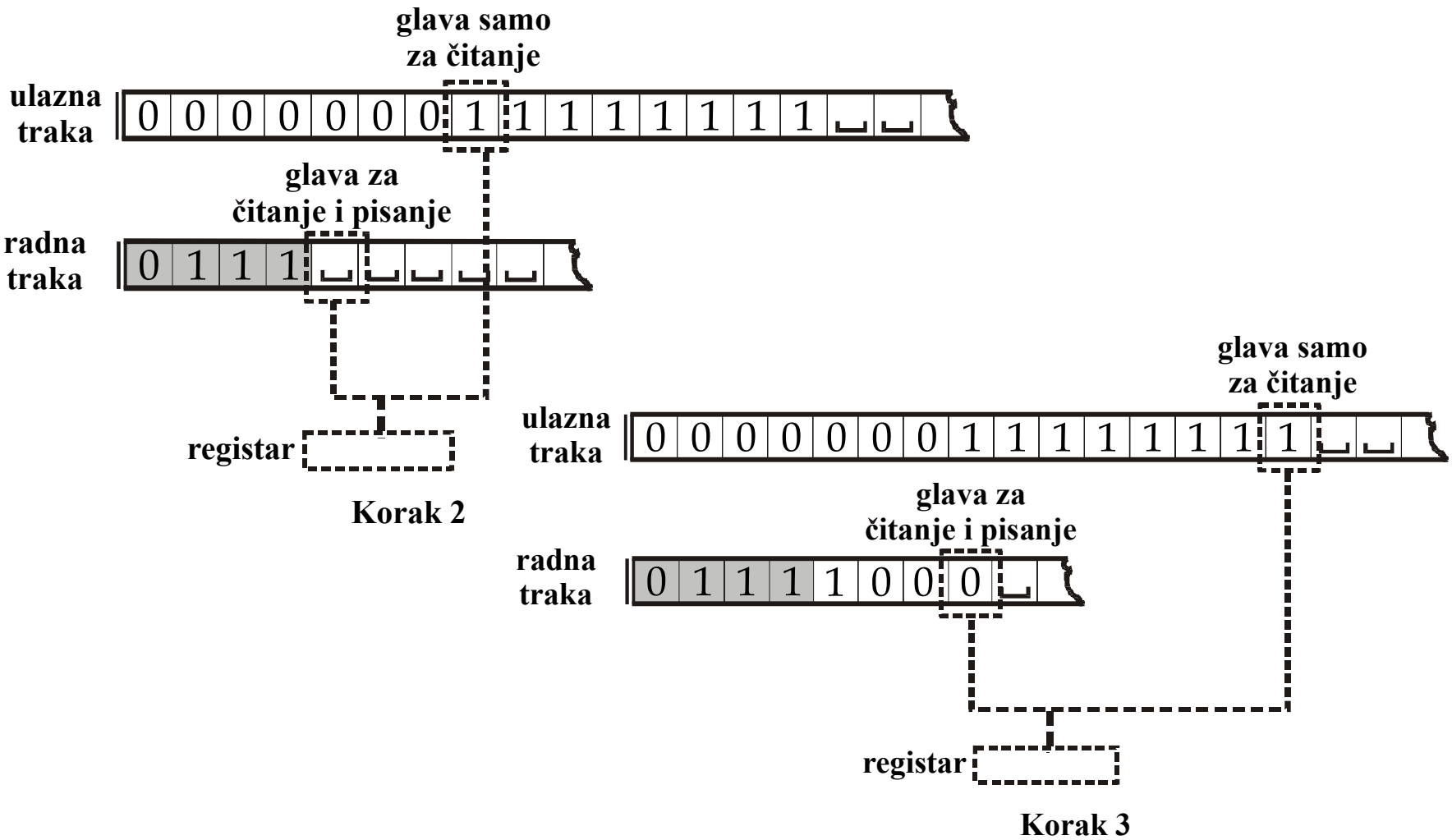
Primjer 1

Algoritam:

- Provjeri da li se 1-ca pojavljuje negdje iza 0-le. Ako je odgovor pozitivan *odbaci*.
  (Ovo ne zahtjeva radnog prostora)

2. Prebroj broj 0-la i broj 1-ca.
3. Ako su brojači isti *prihvati*, u suprotnom *odbaci*.

Primjer 1


glava samo
za čitanje
ulazna
traka
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
glava za
čitanje i pisanje
radna
traka
0 1 1 1
registar
Korak 2
glava samo
za čitanje
ulazna
traka
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
glava za
čitanje i pisanje
radna
traka
0 1 1 1 1 0 0 0
registar
Korak 3

Primjer 2

- $PATH = \{<G, s, t> \mid$ G je orjentisan graf koji ima direktan put od s do t$\}$

PATH pripada NL klasi jezika

Primjer 2

Algoritam:

M_2 = "Na ulazu <*G, s, t*>, gdje je G orjentisan graf sa vrhovima *s* i *t*:

1. Definiši brojač i inicijaliziraj ga na broj vrhova u grafu.
2. Definiši pokazivač da čuva "trenutni vrh" i inicijaliziraj ga na početni vrh s.
3. Dok je brojač različit od nule radi
    4. Ako je trenutni vrh jednak ciljanom vrhu *t*, *prihvati*.
    5. Nedeterministički izaberi vrh koji može biti dohvaćen iz trenutnog vrha.
    6. Zabilježi pokazivač na novi vrh (tj. izabrani vrh) i smanji brojač.
7. *Odbaci*. "

Primjer 3

- $A_{DFA} = \{<A, w> \mid A \text{ je DFA i } A \text{ prihvata ulazni string } w \}$

$A_{DFA} \in L$

Primjer 3

Algoritam:

M_3 = "Na ulazu <*A*, *w*>, gdje je *A* DFA i *w* neki string:

1. Definiši prostor za trenutno stanje i inicijaliziraj ga na početno stanje od A.
2. Definiši prostor za trenutnu poziciju glave i inicijaliziraj je na početak ulazne trake.
3. Simuliraj *A* na ulazu *w* i svaki put, kad za neki simbol od w, DFA *A* dođe u novo stanje sačuvaj to stanje i sačuvaj novu poziciju glave ulazne trake."
4. Kad se završi čitanje ulaznog stringa w, ako je sačuvano stanje prihvatljivo stanje, *prihvati*. U suprotnom *odbaci*.

Primjer 4

- $B = \{w \mid w \in \{ (\ ,\ ) \}^*$ je string koji sadrži ispravno zatvorene zagrade$\}$

$B \in L$

Primjer 4

Opis algoritma:

Dovoljno je da pokažemo da možemo prepoznati da je dati ulaz u obliku ispravno zatvorenih zagrada korištenjem samo jednog brojača. Stavimo brojač na nulu i čitamo ulaz sa lijeva na desno. Za svako "(" povečamo brojač za jedan. Za svako ")" smanjimo brojač za jedan. Ako brojač ikad dobije negativnu vrijednost, odbacimo ulaz (zato što smo zatvorili zagradu prije nego što smo je otvorili). Ako brojač nije nula kad završimo čitanje, tad isto odbacujemo ulaz (zato što broj otvorenih i zatvorenih zagrada nije jednak). U suprotnom, prihvatamo ulaz

Operacija unija

- NL klasa jezika je zatvorena pod operacijom unija
- Unija: $A_1 \cup A_2 = \{ x \mid x \in A_1 \text{ ili } x \in A_2 \}$
- A_1 i A_2 jezici koji su odlučivi sa NL-mašinama N_1 i N_2
- Nedeterminističke branše mašine N_U istovremeno simuliraju N_1 i simuliraju N_2. U bilo kojem slučaju, N_U prihvata ako bilo koja od simuliranih mašina prihvata

Operacija konkatanacija

- NL klasa jezika je zatvorena pod operacijom konkatanacija
- Konkatanacija: $A_1 \circ A_2 = \{ xy \mid x \in A_1 \text{ i } y \in A_2 \}$
- A_1 i A_2 jezici koji su odlučivi sa NL-mašinama N_1 i N_2
- Mašina N_o nedeterministički izabere poziciju na ulazu, koji će string *w* podjeliti na dva podstringa. Jedino pokazivač te pozicije je sačuvan na radnoj traci. Poslije toga N_o simulira N_1 na prvom podstringu, nedeterministički praveći branše koje su potrebne da simuliraju nedeterminizam od N_1. Ako neka od branši dospije u N_1 prihvatljivo stanje, tad N_o simulira N_2 na drugom podstringu. Ako bilo koja od tih branša dođe do N_2 prihvatljivog stanja, N_o prihvata

Operacija zvijezda

- NL klasa jezika je zatvorena pod operacijom zvijezda
- Zvijezda: $A_1^* = \{ x_1x_2...x_k \mid k \geq 0 \text{ i svaki } x_i \in A_1\}$
- A_1 jezik je odlučiv sa NL-mašinom N_1
- Konstruisat ću Turing mašinu N_* koja će odlučivati A_1^*.

Operacija zvijezda

Algoritam:

N_*= "Na ulazu w:

1. Inicijaliziraj pozicije dva ulazna pokazivača p_1 i p_2 na 0, i za date pozicije odmah razmatraj prvi ulazni simbol.
2. Prihvati ako nema ulaznog simbola iza p_1.
3. Pomjeri p_2 naprijed na nedeterministički izabranu ulaznu poziciju.
4. Simuliramo N_1 na podstringu od w, čiji je sadržaj od pozicije p_1, pa sve do pozicije p_2, nedeterministički praveći branše koje su potrebne za simuliranje nedeterminizma od N_1.
5. Ako ova branša simulacije dođe do N_1 prihvatljivog stanja, kopiraj sadržaj p_2 u p_1 i idi na korak broj 2."

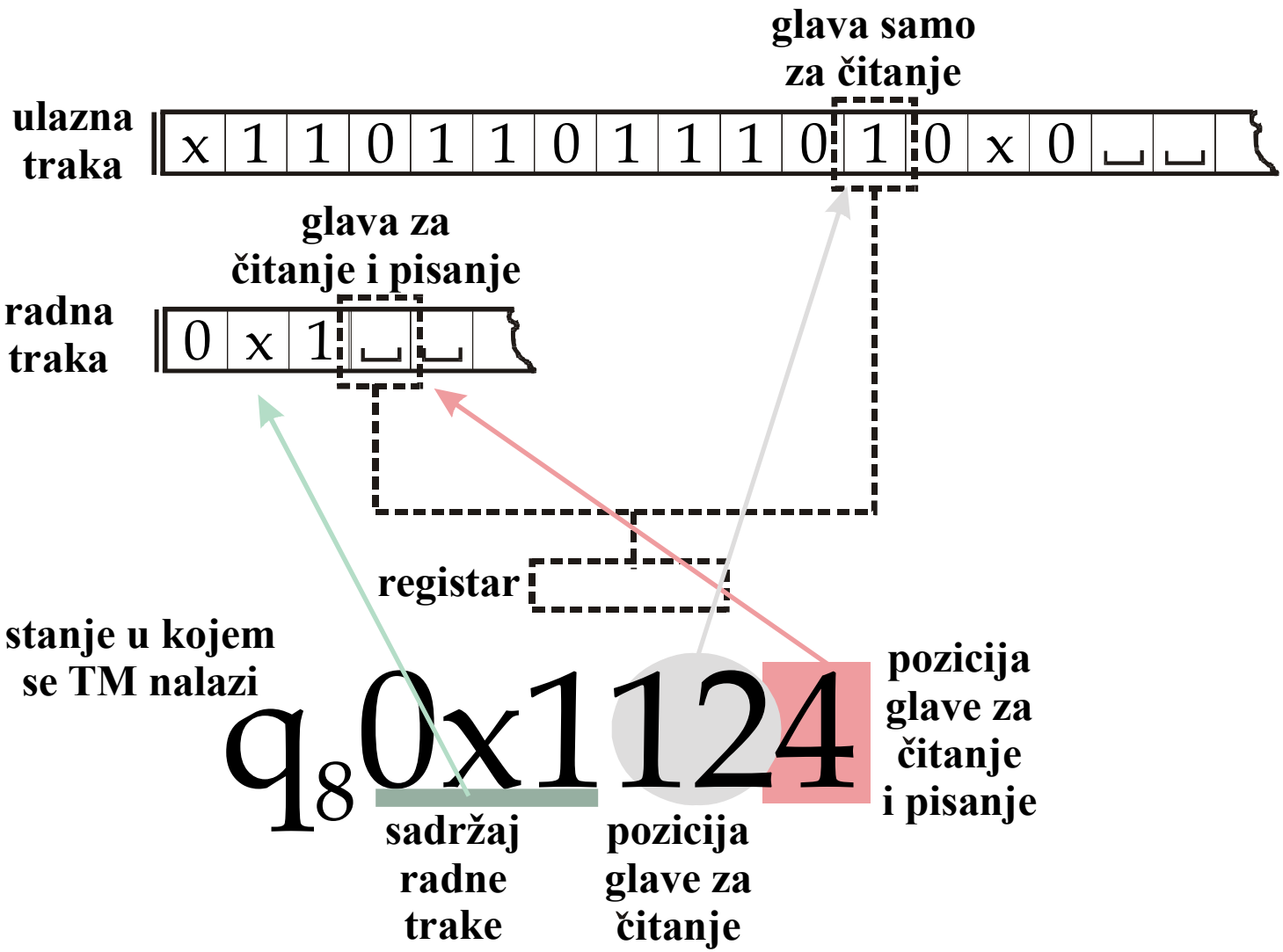
Konfiguracija od M na w za log TM

- $f(n)$ prostorno ograničena Turing mašina može koristiti najviše $2^{O(f(n))}$ vremena

Definicija

Ako je M Turing mašina koja ima odvojenu ulaznu traku samo za čitanje i w je ulaz, **kofiguracija od *M* na *w*** je podešenje koje se sastoji od trenutnog stanja mašine, sadržaja radne trake, i pozicije dvije glave na trakama. Ulaz w nije dio konfiguracije od M na w.

Konfiguracija od M na w za log TM



coNL

- coNL je klasa jezika J takvih da je $\overline{J} \in \text{NL}$.
- Navešćemo teorem za čiji dokaz nam treba poznavanje klase jezika NL-complete

Teorem

$$\text{NL} = \text{coNL}$$

Ideja za dokaz je

- ako je A u NL-complete tad je $\overline{A}$ u coNL-complete,
- Ako coNL-complete problem pripada NL tad je NL=coNL
- PATH je NL-complete

Prema tome dovoljno je pokazati da je $\overline{\text{PATH}}$ u NL
(Za detalje pogledati Sipser teorema 8.27)

2-SAT∈NL

- 2-SAT = {< ϕ > | ϕ je zadovoljavajuća 2CNF-formula}
- Od ranije znamo da je 2-SAT u P
- 2-SAT je u NL kalasi jezika

Dokaz:
Enkodiramo 2cnf-formulu u orjentisan graf G(ϕ) na sljedeći način:

- vrhove od G(ϕ) predstaviti ćemo sa varijablama od ϕ i njihovim negacijama

- postoji ivica (α, β) (čitaj iz α u β) ako i samo ako postoji klauzula (¬α, β) (ili (¬β, α)) u ϕ

Ideja za dokaz

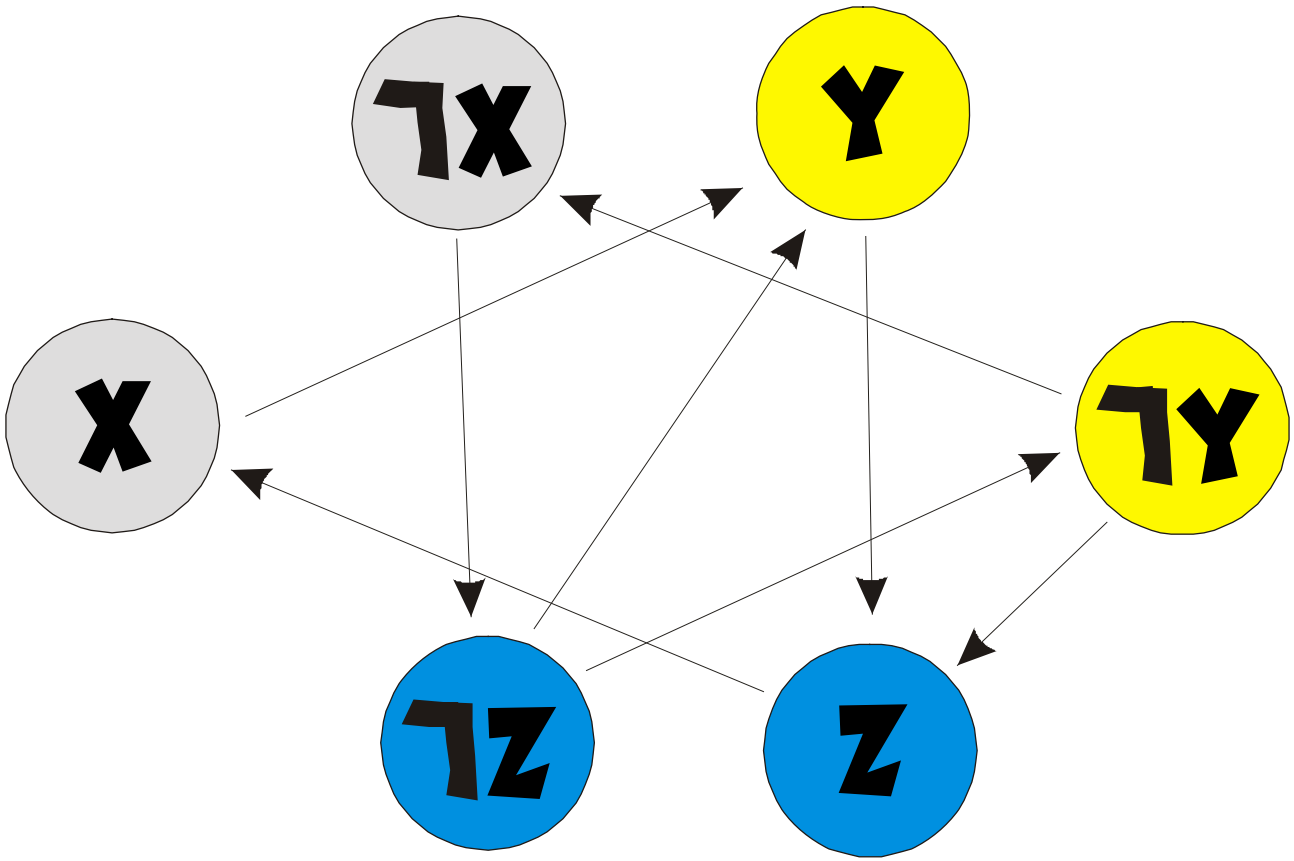
Intuitivno, ivice možemo predstaviti pomoću logičke implikacije ($\Rightarrow$)

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \equiv (\neg\alpha \vee \beta) \equiv (\beta \vee \neg\alpha).$$

G(ϕ) je simetričan graf, tj. ($\alpha \vee \beta$) je ivica ako i samo ako ($\neg\alpha \vee \neg\beta$) je ivica. Primjetimo i to da putevi u G(ϕ) odgovaraju ispravnim implikacijama (tranzitivnošću od $\Rightarrow$).

Primjer konstrukcije

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$



Korisna Lema

Lemma

ϕ je nezadovoljavajuća cnf formula ako i samo ako postoji varijabla x takva da postoji put iz x u $\neg x$ i iz $\neg x$ u x u grafu G(ϕ)

Dokaz: Potreban uslov. Pretpostavimo da postoji varijabla x takva da postoji put iz x u $\neg x$ i iz $\neg x$ u x u grafu G(ϕ) i pretpostavimo da ϕ može biti zadovojena sa nekom dodjelom T.

- Ako pretpostavio da je T(x) = Tačno tada je T($\neg x$) = Netačno.
- Kako postoji put iz x u $\neg x$ to postoji i ivica (α, β) na ovom putu takva da je T(α) = Tačno i T(β) = Netačno
- Ali (α, β) odgovara klauzuli ($\neg\alpha \vee \beta$) u ϕ
- Zaključujemo da ϕ nije zadovoljena sa T, kontradikcija

Ako pretpostavimo da je T(x) = Netačno, dokaz je sličan

Nastavak dokaza

Dovoljan uslov. Pretpostavimo da ne postoji varijabla x takva da postoji put iz x u $\neg x$ i iz $\neg x$ u x u grafu G(ϕ). Pokazačemo da tad ϕ može biti zadovojena sa nekom dodjelom.

Ponavljaj sljedeću proceduru dok svaki vrh ne dobije tačnu vrijednost:

- 1. Izaberimo neki vrh α kojem nismo dodijelili vrijednost (a prema pretpostavci za taj vrh ne postoji put od α do $\neg\alpha$).
- 2. Dodjelite Tačnu vrijednost vrhu α, i svim vrhovima koji se nađu na putu od α, i
- 3. Dodijeli Netačnu vrijednost svim vrhovima koji se nađu na putu iz vrha $\neg\alpha$.

Nastavak dokaza

Primjetite da

- x i $\neg x$ su uvijek dodjeljene vrijednosti u istom koraku;
- ne može se desiti nekakav sukob između navedenih argumenata, zato što bi u tom slučaju bilo, ako bi postojao put iz α u oba vrha β i $\neg\beta$, i tada bi, kako je G(ϕ) simetričan, postojali puteviiz $\neg\beta$ i β u $\neg\alpha$.(pa bi postojao put iz α u $\neg\alpha$, što je u kontradikciji sa pretpostavkom);
- ako bi postojao put iz α u vrh kome je dodjeljena vrijednost NETAČNO u bilo kojem ranijem koraku, tada bi, po trečem pravilu procedure, vrhu α bila dodjeljena vrijednost (NETAČNO) u tom ranijem koraku;

Nastavak dokaza

- nakon što je svim vrhovima dodjeljena vrijednost, ne postoji ivica koja vodi iz TAČNO u NETAČNO, prema konstrukciji. Tako da je, svaka klauzula zadovoljavajuća i ϕ je zadovoljavajuća cnf formula, što je i trebalo dokazati

Ova lema će nam pomoći da lakše zaključimo da je 2-SAT $\in$ NL. Ovaj jezik je u coNL zato što možemo prepoznati nezadovoljavajuću instancu u log prostoru, tako što ćemo pogađati varijablu x i put iz x u $\neg x$ i nazad
Konstrukcijom TM M_6 nam govori da je $\overline{\text{2-SAT}} \in$ NL, iz čega slijedi da je 2-SAT $\in$ coNL. Prema Teoremi koja kaže da je

$$\text{NL} = \text{coNL}$$

zaključujemo da je 2-SAT $\in$ NL, što je i trebalo dokazati

BIPARTITE ∈ NL

- Neorjentisan graf je **bipartitan** ako vrhove možemo podijaliti u dva skupa tako da sve ivice idu iz vrhova u jednom skupu u vrhove u drugom skupu
- Graf bipartitan ako i samo ako ne sadrži konturu koja ima neparan broj vrhova
- *BIPARTITE* = {<*G*> | *G* je bipartitan graf}
- Pokažimo da je *BIPARTITE* ∈ NL
- Konstruisat ćemo log prostornu nedeterminističku Turing mašinu M_7 koja odlučuje *BIPARTITE*.

BIPARTITE ∈ NL

- Algoritam:

M_7 = "Na ulazu <*G*>, gdje je G neki graf sa *n* vrhova:

1. Definiši prostor za brojač *k* i inicijalizirajmo ga na 1.
2. Definiši prostor za početni vrh i za vrh na putu.
3. Nedeterministički izaberimo početni vrh *v*.
4. Dok je $k \leq n$ radi sljedeće:
    5. Nedeterministički izaberi neki vrh *u*, koji se nalazi na putu iz prethodnog vrha.
    6. Ako su vrhovi *u* i *v* jednaki, i *k* je neparan broj, *prihvati*.
    7. Povečaj k za 1
8. *Odbaci*."

- NL=coNL, slijedi da je *BIPARTITE* ∈ NL, g.e.d.

Literatura

[MS] Michael Sipser,
Introduction to the theory of computation, second edition;
str. 303-334, *2006*

[MJ] Matthew Johnson,
http://www.dur.ac.uk/matthew.johnson2/teaching/acc/;
lecture 11-13, *2007*

[SA] Sanjeev Arrora, Boaz Barak,
Computation Complexity: A Modrern Approach
str.75-90, *2007*

[OG] Oded Goldreich,
Computation Complexity: A Conceptual Perspective;
str. 147-186, *2006*