ODGOVORI NA PITANJA

Svojstva regularnih jezika

Svojstva zatvorenosti regularnih jezika

• zatvorenost klase jezika definira se s obzirom na pojedine operacije nad jezicima:
– unije, nadovezivanja, Kleenovog operatora
– zatvorenost slijedi iz neposredno iz definicije regularnih izraza
• unija regularnih jezika L i N je regularni jezik za koji je N□moguće izgraditi DKA M takav da vrijedi L(M) = L – komplement, presjek, supstitucija
• Def: Klasa jezika je zatvorena s obzirom na neku operaciju ako primjenom te operacije na bilo koji jezik iz te klase dobijemo jezik koji je u istoj klasi

Svojstvo napuhavanja

- engl. pumping lema
- DKA $M=(Q, \Sigma, \delta, q_0, F)$ ima n stanja
- ulazni niz $a_1,a_2,..a_m$; $m > n$;
- $\forall$ i= 1..m : neka je $\delta(q_0, a_1,a_2,..a_i)= q_i$;
- budući da DKA ima samo **n** različitih stanja, nije moguće da je **n+1** stanja u nizu stanja $q_0,q_1,..q_n$ različito
  - 0 ... n .. i .. m
  - pigeonhole principle
- s obzirom da je $m > n$ **neka od stanja** u nizu $q_0,q_1,..q_n$ **se moraju ponoviti** (barem jedno stanje) odnosno

58

Svojstvo napuhavanja I

- za svaki ulazni niz $a_1,a_2,..a_m$ može se odrediti
- dva indeksa j i k: $0 \le j < k \le n$ i $q_j = q_k$:
  - $j<k$ i $k \le n$ za duljinu niza $a_{j+1},a_{j+2},..a_k$ vrijedi $1 \le | a_{j+1},a_{j+2},..a_k| \le n$
- Uvodi se oznaka z
  - ako $a_1,a_2,..a_m = z = uvw$ (u,v,w, podnizovi)
    $u= a_1,a_2,..a_j$;
    $v= a_{j+1},a_{j+2},..a_k$;
    $w= a_{k+1},a_{k+2},..a_m$;

60

Church-Turningova hipoteza

•izračunljive funkcije se poistovjećuju s klasom parcijalno rekurzivnih funkcija
–točnost hipoteze se ne dokazuje jer nema formalne definicije
–pokazuje se prikladnost i razumnost hipoteze
•parcijalno rekurzivne funkcije su izračunljive jer je za njih moguće izgraditi TS
•TS funkcije računa "mehaničkim" putem: korak po korak
•TS ne mora stati sa svaki ulazni niz

Izračunljivost

•intuitivana ("meka") definicija: –problem je izračunljiv ako postoji automat koji postupkom korak po korak (mehaničkim) rješava zadani problem
–nema ograničenja:
•broja koraka
•veličine spremnika
•postupak se ne mora zaustaviti
–potrebno samo raščlaniti rješavanje na slijed postepenih koraka

Odlučivost

•rekurzivni jezici su odlučivi
–jer ih prihvaćaju TS koji uvijek stanu i odluče o prihvaćanju ili neprihvaćanju niza
•rekurzivno prebrojivi jezici nisu odlučivi
–ne postoji TS koji uvijek stane

–ako niz nije u jeziku TS neće stati
–ako ne stane ne možemo odrediti da li ga prihvaćamo ili ne

<u>Odlučivost i izračunljivost</u>

•rekurzivni jezici su izračunljivi i odlučivi
•rekurzivno prebrojivi jezici su izračunljivi ali nisu odlučivi

<u>Svojstvo napuhavanja za KNJ i RJ</u>

• dokazivanje neregularnosti jezika
– ako jeziku ne možemo dokazati svojstvo napuhavanja onda je jezik neregularan
– svojstvo napuhavanja kaže da se svi nizovi jezika mogu napuhati ako prelaze određenu duljinu napuhavanja
• to znači da svaki niz sadrži podniz koji se može ponavljati bezbroj puta, a da rezultirajući niz bude u jeziku = regularan
– dokazivanje ispravnosti algoritama kojima se utvrđuje nepraznost regularnog jezika, beskonačnost regularnog jezika, itd.

AUTOMATI

Konačni automati

– DKA – deterministički konačni automat

– NKA – nedeterministički konačni automat

ε - NKA – nedeterm. konačni automat s ε prijelazima

– konačni automati s izlazom

• Mooreov automat

• Mealyev automat

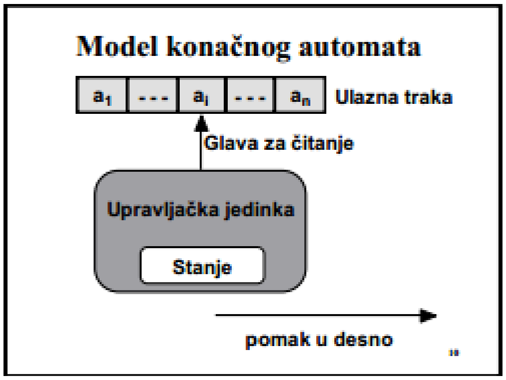
Odnos automata, regularnih izraza i regularnih jezika

• jezik je regularan ako postoji konačni automat koji ga prihvaća

• za svaki regularni jezik moguće je izgraditi konačni automat koji ga prihvaća

regularni izrazi opisuju regularne jezike

Model konačnog automata



Formalni postupak prihvaćanja niza

postupak prihvaćanja simbola raširimo na postupak prihvaćanja niza znakova koji nas iz stanja q dovedu u novo stanje p u tu svrhu uvodimo novu funkciju $\delta' : Q \times \Sigma^* \rightarrow Q$, gdje je

ε* skup svih mogućih nizova; uključujući i prazan niz Σ

x,y,w,z $\in \Sigma^*$ nizovi ulaznih znakova i a,b $\in \Sigma$

Definicija: prihvaćanja niza

Deterministički konačni automat , dka = (Q, Σ ,δ, q0 , F) prihvaća niz x ako je δ(q0 , x)=p

DKA prihvaća skup nizova

L(dka)={x|δ(q0,x) $\in$ F} koji je podskup skupa svih mogućih nizova

L(dka) $\subseteq \Sigma^*$

nizove koji nisu element skupa L(DKA): DKA NE prihvaća

DKA dijeli skup Σ^* na skup koji prihvaća prijelazom u jedno od prihvatljivih stanja, te na skup koji ne prihvaća jer se DKA nakon posljednjeg pročitanog znaka ne nalazi u prihvatljivu stanju

Minimizacija DKA

za svaki DKA je moguće izgraditi beskonačno mnogo drugih DKA koji prihvaćaju isti jezik, zbog učinkovitosti tražimo DKA s čim manjim brojem stanja

Za regularan jezik L moguće je izgraditi DKA M koji ima manji ili jednak broj stanja od bilo kojeg drugog DKA M´ koji prihvaća isti jezik L

Ispitivanje istovjetnosti stanja p i q

Ispitujemo dva uvjeta:

podudarnost: oba stanja moraju bit prihvatljiva ili oba neprihvatljiva

napredovanje: za bilo koji ulazni znak a vrijedi da su (q, a) istovjetnaδ(p, a) i δstanja
• 3 algoritma za određivanje istovjetnosti
– ispitivanjem 2 uvjeta
– dijeljenjem skupa stanja po uvjetu podudarnosti (u grupe)
– traženjem neistovjetnih stanja
Pojednostavljanje DKA minimizacijom
cilj smanjiti broj stanja zadanog DKA
grupa istovjetnih stanja zamjeni se jedinstvenim stanjem na slijedeći način:
– istovjetna stanja se označe zajedničkim imenom
– sve oznake istovjetnih stanja u funkciji prijelaza δ označe se novim zajedničkim imenom (tabela prijelaza)
– u skupu stanja Q se ostavi jedno od istovjetnih stanja a ostala se izuzmu
Određivanje istovjetnosti ispitivanjem dvaju uvjeta
za svaki par stanja DKA ispituju se uvjeti podudarnosti i napredovanja (algoritam je neučinkovit)
1.korak: tablica ispitivanja istovjetnosti za svaki ulazni znak ima svoj ulaz a u recima parove stanja koje ispitujemo
2. korak: ispitujemo uvjet podudarnosti za sva stanja u tablici par stanja iz prvog retka nije podudaran, stanemo$\Rightarrow$
– ako uvjet nije ispunjen
– inače odredimo nove parove stanja i nastavimo ispitivanje
3. korak: za novi par stanja
– ako su ista nema akcije
– ako su različita, ali postoji zapis u prethodnim recima nema akcije
– ako su različita, i NE postoji prethodni zapis, upišemo novi par u novi redak tablice

Određivanje istovjetnosti dijeljenjem u podskupove po podudarnosti
korak 1: skup stanja podijelimo u dva podskupa:
u 1. Grupi sva stanja prohvatljiva pi E F a u 2. Sva stanja neprihvatljiva qi $\notin$F
• korak 2: primjeni se algoritam na podjelu $\prod$
– za (sve grupe stanja Gj u podjeli $\prod$)
korak 3:
ako podjela ostaje ista $\prod$ nova = $\prod$ onda stop, stanja u istim grupama su istovjetna
inače $\prod$ nova različito $\prod$ nastavi korakom 2 dok uvjet $\prod$nvoa = $\prod$ nije sipunjen

Određivanje istovjetnosti traženjem neistovjetnih stanja

- označi sve parove (p, q) za koje vrijedi $p \in F$ i $q \notin F$;
- za (bilo koji par različitih stanja $(p,q) \in (F\times F)$ ili $(p,q) \notin (F\times F)$){
  ako (za neki znak a par ($\delta(p, a)$, $\delta(q, a)$) jest označen) {
    označi (p,q);
    rekurzivno označi sve neoznačene parove u listi koja je pridružena paru (p,q) i sve ostale parove u listama koje su pridružene parovima označenim u ovom koraku;}
  inače {
    za (svi znakovi a) {
      ako ($\delta(p, a)$!= $\delta(q, a)$)
      stavi (p,q) u listu koja je pridružena paru ($\delta(p, a)$, $\delta(q, a)$)
  }}}

Nedohvatljiva stanja

- stanje p DKA $M = (Q, \Sigma, \delta, q_0, F)$ je nedohvatljivo ako ne postoji niti jedan niz $w\in\Sigma^*$ za koji vrijedi $p= \delta(q_0 ,w)$
- odbacivanjem nedohvatljivih stanja dobije se istovjetan DKA s manjim brojem stanja
- Određivanje dohvatljivih stanja u listi DS:
  - u listu dohvatljivih stanja DS upiše se početno stanje q_0;
  - listu DS proširi se skupom stanja $\{p \mid p= \delta(q_0 ,a)$ za sve $a\in\Sigma\}$;
  - za sva stanja $q_i \in$ DS proširi se lista skupom stanja $\{p \mid p = \delta(q_i ,a)$, stanje p nije prethodno upisano u listu, za sve $a\in\Sigma\}$;
  - ako se lista DS proširi novim stanjem ponavlja se od koraka 3.

DKA s minimalnim brojem stanja

- **Algoritam**: minimalni DKA dobije se odbacivanjem
  - nedohvatljivih stanja **(1.)** i
  - istovjetnih stanja **(2.)**
  - **PAZI redoslijed**
- ne postoji niti jedan drugi DKA koji prihvaća isti jezik a ima manji broj stanja
- istovjetnost minimalnog DKA s originalnim dokazuje se pomoću dokaza o istovjetnosti njihovih početnih stanja
  - algoritma ispitivanjem uvjeta podudarnosti i napredovanja

Nedeterminizam
- izvođenje paralelnih procesa
– ako jedan od procesa dođe do prihvatljivog stanja svi se procesi prihvate
- UNIX: fork
- stablo mogućnosti, grananja
– počinjemo u korijenu, grananje na svakom koraku
– ako je jedna od grana u prihvatljivom stanju onda se prihvaća rad automata

Prihvaćanje niza DKA vs. NKA
- za bilo koji niz postoji samo jedan slijed prijelaza DKA (q0,w), ako slijedδiz početnog stanja q0 u stanje p= prijelaza završi u prihvatljivu stanju niz w se prihvaća
- za neki niz w NKA može imati više od jednog slijeda prijelaza, provjeravaju se svi slijedovi prijelaza i ako postoji barem jedan slijed prijelaza iz q0 u jedno od prihvatljivih stanja niz w se prihvaća

Konačni automati s izlazom
- izlaz je funkcija stanja automata:
– Mooreov automat
- izlaz je funkcija stanja automata i ulaznog znaka:
– Mealyev automat
- Mooreov i Mealyev automat su istovjetni ukoliko za bilo koji ulazni niz daju jednake izlazne nizove • za bilo koji MoDKA moguće je izgraditi istovjetni MeDKA i obrnuto
- Skriveni Markovljev model

Primjena automata s izlazom
- govorne tehnologije
– raspoznavanje govora
- za govorni signal zapisujemo tekst
– sinteza govora
- iz teksta generiramo govorni signal
– nadzor nad vođenjem dijaloga
- modeliranje mrežnih protokola
- modeliranje naprava – inteligentne sobe, kuće

Definicija SMM-a

- Diskretni skriven Markovljev model je definiran:
  - **Izlaznom abecedom** $O=\{o_1, o_2,..,o_M\}$, gdje je M broj simbola u abecedi;
  - **Skupom stanja** $S=\{s_1,..,s_N\}$, gdje je N broj stanja modela;
  - Matricom **vjerojatnosti prijelaza** $\mathbf{A=\{a_{ij}\}}$, gdje je a_{ij} vjerojatnost prijelaza iz stanja i u stanje j zapisana izrazom $a_{ij}=P(s_t=j|s_{t-1}=i)$;
  - Matricom **vjerojatnosti izlaznih simbola** $\mathbf{B=\{b_i(k)\}}$, gdje je $b_i(k)$ vjerojatnost pojave simbola o_k u stanju i;
    - Ako je $X=(X_1,..,X_t)$ izlazni niz procesa do trenutka t, i ako je slijed stanja $S=(s_1,s_2,..,s_t)$ koje je proces pritom zauzeo prikriven onda se $b_i(k)$ može definirati kao $b_i(k)=P(X_t=o_k|s_t=i)$,
  - **Vektorom početnih vjerojatnosti** $\Pi=\{\pi_i\}$; gdje je π_i vjerojatnost da se model u trenutku t_0 nalazi u stanju s_i, za koju vrijedi $\pi_i=P(s_0=i)$ za $1 \le i \le N$

Struktura monofonskog SMM-a fonema /a/

- SMM je označen oznakama <BEGINHMM> i <ENDHMM>,
- naziv SMM-a oznakom ~h "a",
- broj stanja oznakom <NUMSTATES>,
- matrica vjerojatnosti prijelaza među stanjima oznakom <TRANSP>,
- stanje oznakom <STATE>,
- a srednje vrijednosti i varijance svakog stanja oznakama <MEAN> i <VARIANCE>,
- duljina i vrsta vektora značajki oznakom <VECSIZE>.
- Broj Gaussovih mješavina kao i težinski faktor označeni su oznakama <NUMMIXES> i <MIXTURE>,

65

Tri problema

• problem procjene (evaluacije) vjerojatnosti niza simbola X (The Evaluation Problem)

• problem dekodiranja slijeda stanja za dani niz X (The Decoding Problem)

• problem ocjene (učenja, optimiranja) parametara PMM) (The Learninig/Estimation Problem).π=(A, B, Φ 69

Procjena

- problem procjene (evaluacije) vjerojatnosti niza simbola X *(The Evaluation Problem)*
- Ukoliko imamo model Φ=(A, B, π) i izlazni slijed opažanja X={X_1,X_2,..X_T}, koja je vjerojatnost da je upravo model Φ generirao ta opažanja P(X|Φ)?
  - izračunati vjerojatnost da je izlazni slijed nastao baš u tom modelu ili koliko je model usklađen s opažanjima.
  - problem određivanja koliko je model prilagođen opaženom izlaznom slijedu, odnosno možemo ga svesti na problem klasifikacije koji od potencijalnih modela najbolje odgovara opaženom izlaznom slijedu
- **algoritmi naprijed i natrag**

70

Dekodiranje

- Ukoliko imamo model Φ=(A, B, π) i izlazni slijed opažanja X={X_1,X_2,..X_T} koji je najvjerojatniji slijed stanja S={s_1,s_2,..s_T} u kojima je model generirao izlazni slijed?
  - problem prikrivenosti unutarnjeg procesa odnosno - znamo kako se prikriven proces ponaša.
  - otkriti optimalan slijed stanja koji nije nužno i "pravilan" slijed stanja
- **Viterbijev algoritam** određuje najvjerojatniji slijed stanja u PMM-u. Traži se slijed stanja S={s_1,s_2..s_T} takav da je vjerojatnost P(S, X|Φ) maksimalna

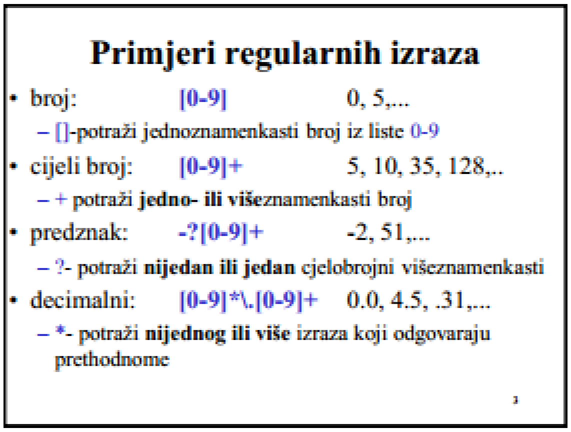
71

Procjena- Učenje

- Ukoliko imamo izlazni slijed opažanja X={X_1,X_2,..X_T} i neki početni model Φ=(A, B, π) kako ćemo odrediti parametre modela Φ'=(A',B', π'), koji maksimiziraju produkt vjerojatnost ?
  - Rješenje: na osnovi podataka s kojima učimo model ocijeniti parametre modela, zato se ovaj problem često naziva i problemom učenja.
  - Dakle parametre modela učenjem možemo maksimalno prilagoditi podacima na kojim učimo. To može dovesti do dodatnih problema prevelike prilagodbe podacima koji nastaju zbog premale količine podatak za učenje (*ovefitting*)
- kombinacija **Naprijed-natrag algoritama**
- **Baum–Welchev algoritam** odnosno
- **metoda očekivanja i popravaka** (***Expectation-Modification Method – EM algorithm***).

72

REGULARNI IZRAZI



Upotreba RI

–pretraživanje interneta
–u mnogim UNIX alatima: grep, sed, awk, gawk,...
–u programskim jezicima Perl, Python, Java, JavaScript,…
–programima za uređivanje teksta: vi, find izbornik u MS Wordu,...

Regularni izrazi

• regularni jezik L(r) opisujemo regularnim izrazima r
– ako je jezik moguće opisati regularnim izrazom onda je jezik regularan
– za bilo koji jezik L(r) definiran regularnim izrazom r moguće je izgraditi DKA M za koji vrijedi L(M)=L(r)
• regularni jezik je pravi podskup skupa svih jezika
• neregularne jezike nije moguće opisati regularnim izrazima i za njih ne možemo napraviti konačni automat
– za neregularne jezike koriste se modeli potisnih automata i Turingovi strojevi (omogućavaju pamćenje)

Pojednostavljivanje RI

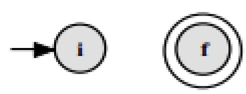
• rekurzivna pravila
• precedenca operatora
• asocijativnost
• algebarski zakoni
– komutativnost
– asocijativnost
– distributivnost
– idempotentnost

Rekurzivna pravila

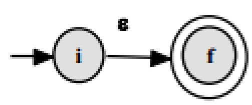
- $\varnothing$ je regularan izraz; $L(\varnothing)=\{\ \}$
- ε je regularan izraz; $L(\varepsilon)=\{\varepsilon\}$
- $\forall a\in\Sigma$, a je regularan izraz; $L(a)=\{a\}$
- ako su r i s regularni izrazi koji označavaju L(r) i L(s) onda:
  - (r)+(s) je regularan izraz [ili (r)|(s)]; $L((r)+(s))=L(r)\cup L(s)$ – unija jezika
  - (r)(s) je regularan izraz; $L((r)(s))=L(r)L(s)$ – nadovezivanje jezika, konkatenacija
  - (r)* je regularan izraz; $L((r)^*)=L(r)^*$ – Kleenov operator na jezikom L(r)

7 pravila konstrukcije ε-NKA iz regularnih izraza I

- **p1:** za regularan izraz $\varnothing$; $L(\varnothing)=\{\ \}$ konstruira se ε-NKA $\mathbf{M=(\{i,f\}, \Sigma, \{\},i,\{f\})}$

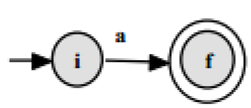


- **p2:** za regularan izraz ε; $L(\varepsilon)=\{\varepsilon\}$ konstruira se ε-NKA $\mathbf{M=(\{i,f\}, \Sigma, \{\delta(i, \varepsilon)=f\},i,\{f\})}$
  - za bilo koji $b\in\Sigma$, $\delta(f, b)=\{\}$
  - M prihvaća isključivo prazni niz ε



7 pravila konstrukcije ε-NKA iz regularnih izraza II

- **p3:** za regularan izraz a; $L(a)=\{a\}$ konstruira se ε-NKA $\mathbf{M=(\{i,f\}, \Sigma, \{\delta(i, a)=f\},i,\{f\})}$
  - za bilo koji $b\in(\Sigma\cup\{\varepsilon\})$ i $b\neq a$: $\delta(f, b)=\{\}$
  - M prihvaća isključivo niz a
  - M ne prihvaća niz ε

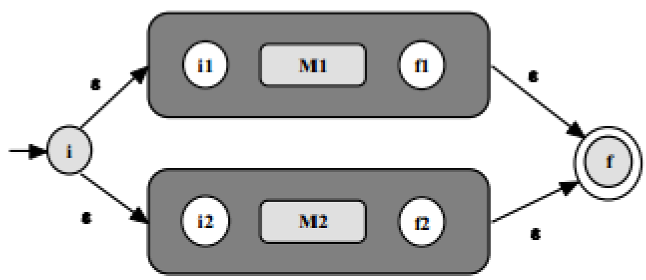


7 pravila konstrukcije ε-NKA iz regularnih izraza III

- **p4:** za regularan izraz r_1+r_2; $L(r_1+r_2)=L(r_1)\cup L(r_2)$ konstruira se ε-NKA $\mathbf{M=(Q_1\cup Q_2\cup\{i,f\}, \Sigma_1 \cup \Sigma_2, \delta, i, \{f\})}$
  - ukoliko su prije izgrađeni ε-NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ i $M_2=(Q_2, \Sigma_2, \delta_2, i_2, \{f_2\})$ takvi da je $L(M_1)=L(r_1)$ i $L(M_2)=L(r_2)$
  - i nema prijelaza iz stanja f_1 i f_2 niti za jedan ulazni znak i $Q_1\cap Q_2=\{\}$
- novo početno stanje je *i* a prihvatljivo stanje je *f*
  - stanja i_1 i_2 nisu više početna i stanja f_1 f_2 nisu više prihvatljiva
- i funkcija δ se određuje:
  - $\delta(i, \varepsilon)=\{i_1, i_2\}$
  - $\delta(f_1, \varepsilon)=\delta(f_2, \varepsilon)=\{f\}$
  - $\delta(q, a)=\delta_1(q, a)$: $\forall q\in(Q_1\backslash\{f_1\})$, $\forall a\in(\Sigma_1\cup\{\varepsilon\})$
  - $\delta(q, b)=\delta_2(q, b)$: $\forall q\in(Q_2\backslash\{f_2\})$, $\forall b\in(\Sigma_2\cup\{\varepsilon\})$

7 pravila konstrukcije ε-NKA iz regularnih izraza IV

- **p4 (II):**



7 pravila konstrukcije ε-NKA iz regularnih izraza V

- **p5:** za regularan izraz r_1r_2; $L(r_1r_2)=L(r_1)L(r_2)$ konstruira se ε-NKA $\mathbf{M=(Q_1\cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, i_1, \{f_2\})}$
  - ukoliko su prije izgrađeni ε-NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ i $M_2=(Q_2, \Sigma_2, \delta_2, i_2, \{f_2\})$ takvi da je $L(M_1)=L(r_1)$ i $L(M_2)=L(r_2)$
  - i nema prijelaza iz stanja f_1 i f_2 niti za jedan ulazni znak i $Q_1\cap Q_2=\{\}$
- novo početno stanje je i_1 a prihvatljivo stanje je f_2
  - stanje i_2 nije više početno i stanje f_1 nije više prihvatljivo
- i funkcija δ se određuje:
  - $\delta(f_1, \varepsilon)=\{i_2\}$
  - $\delta(q, a)=\delta_1(q, a)$: $\forall q\in(Q_1\backslash\{f_1\})$, $\forall a\in(\Sigma_1\cup\{\varepsilon\})$
  - $\delta(q, b)=\delta_2(q, b)$: $\forall q\in(Q_2\backslash\{f_2\})$, $\forall b\in(\Sigma_2\cup\{\varepsilon\})$

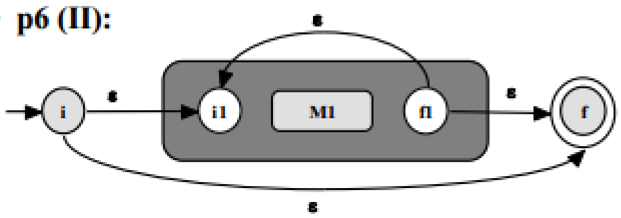
7 pravila konstrukcije ε-NKA iz regularnih izraza VII

- **p6:** za regularan izraz r_1*; $L(r_1^*)=L(r_1)^*$ konstruira se ε-NKA **M= ($Q_1 \cup$ {i,f}, Σ_1, δ, i, {f})**
  - ukoliko je prije izgrađeni ε-NKA $M_1=(Q_1, \Sigma_1, \delta_1, i_1, \{f_1\})$ takav da je $L(M_1)=L(r_1)$ i nema prijelaza iz stanja f_1 niti za jedan ulazni znak
  - novo početno stanje je *i* a prihvatljivo stanje je *f* (stanje i_2 nije više početno i stanje f_1 nije više prihvatljivo)
- i funkcija δ se određuje:
  - $\delta(i, \varepsilon)= \delta(f_1, \varepsilon)= \{i_1,f\}$
  - $\delta(q, a)= \delta_1(q, a)$: $\forall q\in(Q_1\backslash\{f_1\})$, $\forall a\in(\Sigma_1\cup \{\varepsilon\})$

21

7 pravila konstrukcije ε-NKA iz regularnih izraza VIII

- **p6 (II):**



- **p7:** budući da je $L((r))=L(r)$za ε-NKA M regularnog izraza r uzima se ε-NKA M_1 regularnog izraza r jer je $L(M_1) =L(r)=L((r))$

22

Svojstva regularnih jezika

Svojstva zatvorenosti regularnih jezika

• zatvorenost klase jezika definira se s obzirom na pojedine operacije nad jezicima:
– unije, nadovezivanja, Kleenovog operatora
– zatvorenost slijedi iz neposredno iz definicije regularnih izraza
• unija regularnih jezika L i N je regularni jezik za koji je N∪moguće izgraditi DKA M takav da vrijedi L(M) = L – komplement, presjek, supstitucija
• Def: Klasa jezika je zatvorena s obzirom na neku operaciju ako primjenom te operacije na bilo koji jezik iz te klase dobijemo jezik koji je u istoj klasi

Svojstvo napuhavanja

- engl. pumping lema
- DKA M=(Q, Σ, δ, q_0, F) ima n stanja
- ulazni niz $a_1,a_2,..a_m$; m > n;
- ∀ i= 1..m : neka je δ(q_0, $a_1,a_2,..a_i$)= q_i;
- budući da DKA ima samo **n** različitih stanja, nije moguće da je **n+1** stanja u nizu stanja $q_0,q_1,..q_n$ različito
  - 0 ... n .. i .. m
  - pigeonhole principle
- s obzirom da je m > n **neka od stanja** u nizu $q_0,q_1,..q_n$ **se moraju ponoviti** (barem jedno stanje) odnosno

58

Svojstvo napuhavanja I

- za svaki ulazni niz $a_1,a_2,..a_m$ može se odrediti
- dva indeksa j i k: 0≤j<k≤n i $q_j=q_k$:
  - j<k i k ≤ n za duljinu niza $a_{j+1},a_{j+2},..a_k$ vrijedi $1 \le | a_{j+1},a_{j+2},..a_k| \le n$
- Uvodi se oznaka z
  - ako $a_1,a_2,..a_m$ = z = uvw (u,v,w, podnizovi)

    u= $a_1,a_2,..a_j$;

    v= $a_{j+1},a_{j+2},..a_k$;

    w= $a_{k+1},a_{k+2},..a_m$;

60

Primjena svojstva napuhavanja

• dokazivanje neregularnosti jezika

– ako jeziku ne možemo dokazati svojstvo napuhavanja onda je jezik neregularan

– svojstvo napuhavanja kaže da se svi nizovi jezika mogu napuhati ako prelaze određenu duljinu napuhavanja

• to znači da svaki niz sadrži podniz koji se može ponavljati bezbroj puta, a da rezultirajući niz bude u jeziku = regularan

– dokazivanje ispravnosti algoritama kojima se utvrđuje nepraznost regularnog jezika, beskonačnost regularnog jezika, itd.

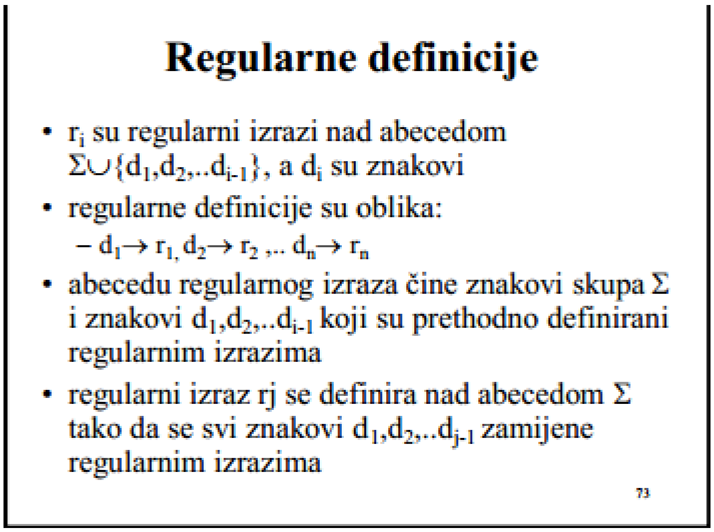
Neregularnost jezika

• nema odgovarajućeg konačnog automata

• ako je L – regularan jezik onda postoji cjelobrojna konstanta n: da je moguće bilo koji niz z iz L: |z|>n rastaviti na podnizove 0≥ n te za bilo koji i≤ |v| i |uv| ≤z=uvw: 1 niz uviw je također element L

• pokazuje se da n nije veći od broja stanja minimalnog DKA koji prihvaća jezik L

Nepraznost i beskonačnost regularnog jezika
• Nepraznost:
– Regularni jezik L(M) je neprazan ako i samo ako DKA M s n stanja prihvaća niz z duljine manje od n (|z|
– ako je u skupu dosegljivih stanja M barem jedno prihvatljivo stanje L(M) je neprazan
• Beskonačnost:
– Regularni jezik L(M) je beskonačan ako i samo ako M prihvaća niz duljine l, gdje je n ≤M
– ako je u dijagramu stanja M (bez neprihvatljivih stanja) barem jedna zatvorena petlja L(M) je beskonačan

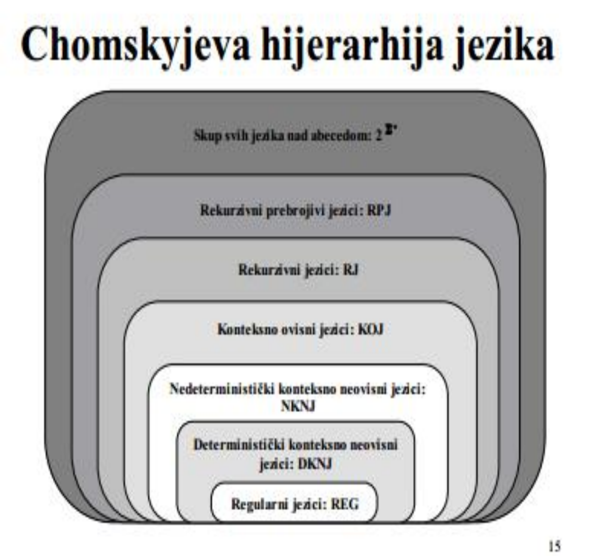


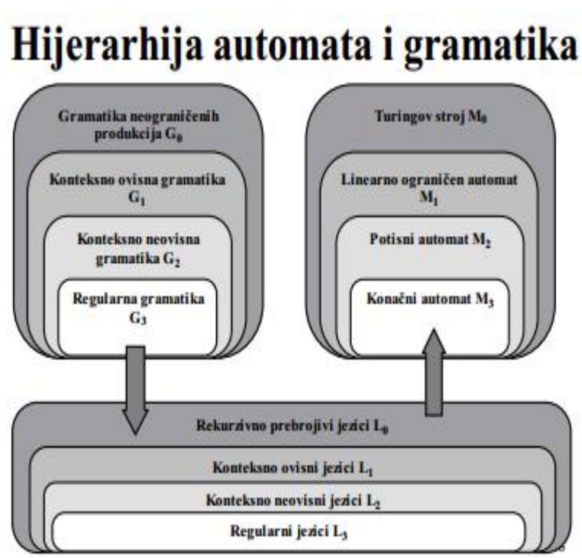
GRAMATIKA

• formalna gramatika koristi se u generiranju i analizi nizova znakova formalnog jezika
• regularna gramatika generira regularne jezike
• kontekstno neovisna gramatika je uređena četvorka G=(V, T, P, S)
– V – konačni skup nezavršnih znakova;
– T - konačni skup završnih znakova
– P – konačni skup produkcija (pravila)
• A je nezavršni znak, alfa je niz znakova skupa (VUT)* i alfa može biti prazan niz epsilon;
– S – početni, nezavršni znak
A, B, C, D, E, S – velika slova koriste se za nezavršne znakove, S – početni nezavršni znak
• a, b, c, d, e i brojke – mala slova i brojke su završni znakovi (vizualno podebljani)
• X, Y, Z su završni ili nezavršni znakovi
• u, v, w, x, y i z – mala slova označavaju nizove završnih znakova
• -mala grčka slova označavaju nizove završnih inezavršnih znakova
• | - znak za više produkcija (pravila) pridruženih istom znaku
Primjer BNF V-I
• definicija sintakse programskih jezika
• BNF (Backus-Naurov format)
• opis jezika zadaje se nizom pravila koja imaju lijevu i
desno stranu odvojenu znakom jednakosti
– lijeva strana ::= desna strana
• lijevu stranu čini točno jedna varijabla
• desnu stranu čini više izraza odvojenih operatorom

izbora |
– desna strana može biti prazna bez znaka (-produkcija)
• izraz je niz varijabli i konstanti
– varijabla v: <v>

Generativno stablo gramatike G
• gramatika G generira niz završnih znakova w i za gramatiku G je moguće izgraditi generativno stablo čiji su listovi isključivo označeni znakovima iz niza w i znakom
• čvorovi stabla označeni su znakovima iz VT{}
• korijen je označen početnim nezavršnim znakom S
• unutrašnji čvorovi su označeni nezavršnim znakovima AV
• neka su čvorovi n1,n2,..nk djeca čvora n; ako je n označen znakom Ai ako su čvorovi n1,n2,..nk označeni X1,X2,.. Xk onda jeA X1,X2..Xk produkcija iz skupa P
• znakom epsilon moguće je označiti isključivo list stabla; takav list jejedino dijete svojih roditelja odnosno dijete jednog od unutarnjihčvorova





Regularna gramatika => NKA

- neka su produkcije gramatike G=(V,T,P,S) tipa **A→aB** ili **A→ε**, A i B su nezavršni znakovi, **a** je završni znak za gramatiku G je moguće izgraditi NKA M=(Q,Σ,δ,q_0,F) za kojeg vrijedi L(M)=L(G):
  - skup ulaznih znakova Σ=T skup završnih znakova
  - skup stanja Q=V skup nezavršnih znakova
  - početnom stanju q_0=S početni nezavršni znak
  - iz produkcije A→aB gradi se prijelaz δ(A,a)=δ(A,a)∪B
    - npr. početno je δ(A,a)=∅; ako imamo produkcije A→aB i A→aC onda je δ(A,a)={B, C} (nedeterminizan je dozvoljen)
  - ako je u gramatici produkcija A→ε, A je završno stanje; A ∈ F

19

Desno-linearna (DL) gramatika
• ukoliko gramatika ima najviše jedan nezavršni znak na desnoj strani i to na krajnje desnom mjestu:
A->wB ili A->w
– A i B su nezavršni znakovi e V; w je niz završnih znakova e T* proizvoljne duljine uključujući i prazan niz
• desno linearna gramatika je regularna gramatika

Lijevo-linearna (LL) gramatika
• ukoliko gramatika ima najviše jedan nezavršni znak na desnoj strani i to na krajnje lijevom mjestu:
A->Bw ili A->w
– A i B su nezavršni znakovi e V; w je niz završnih znakova e T* proizvoljne duljine
uključujući i prazan niz
• lijevo linearna gramatika je regularna gramatika

jezik L je regularan ako i samo ako postoji desno linearna gramatika GD koja generira jezik L=L(GD)
– isto vrijedi i za lijevo linearnu gramatiku L=L(GL)
• DL gramatika ili LL gramatika se preurede tako da su sve produkcije oblika A⍰aB ili A⍰⍰ i
zatim se gradi NKA

Pojednostavljenje gramatike
• odbacivanje beskorisnih znakova
– odbacivanje mrtvih znakova
– odbacivanje nedohvatljivih znakova
• odbacivanje jediničnih produkcija
• odbacivanje epsilon-produkcija

Chomskyjev normalni oblik produkcija (CNO)

- neka gramatika G=(V, T, P, S) generira kontekstno neodvisni jezik L(G)\{ε}. Moguće je izgraditi istovjetnu gramatiku G'=(V', T', P', S') koja ima sve produkcija oblika **A→BC** ili **A→a**
- znakovi A, B, C su nezavršni znakovi gramatike a znak **a** je završni
- gramatika G nema:
  - beskorisnih znakova,
  - ε-produkcija i
  - jediničnih produkcija

38

Greibachov normalni oblik produkcija (GNO)

- neka gramatika G=(V, T, P, S) generira kontekstno neodvisni jezik L(G)\{ε}. Moguće je izgraditi istovjetnu gramatiku G'=(V', T', P', S') koja ima sve produkcija oblika **A→aα**; **a** je završni znak a **α niz nezavršnih znakova koji može biti i prazan**
- najprije se gramatika pretvori u CNO
- zatim se izvede algoritam zamjene krajnjeg lijevog nezavršnog znaka i algoritam razrješenja lijeve rekurzije
- izvede se pretvorba u GNO

40

PARSERI

• postupak prepoznavanja niza i gradnja generativnog stabla na temelju zadanih produkcija kontekstno neovisne gramatike
• želimo odrediti je li ulazni niz (program) generirala gramatika (je li u skladu sa sintaksnim pravilima zapisanima u gramatici)

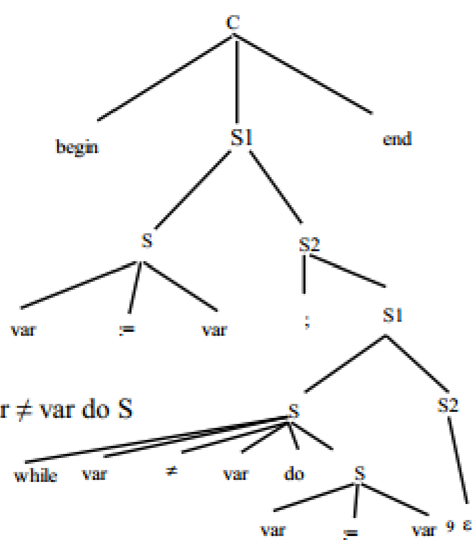
Parsiranje niza

• prepoznavanje niza – određivanje
– pripada li niz w jeziku L(G)?
– generira li gramatika G niz w?
– w – C program, SQL query, XML dokument, ...
• generativno stablo (stablo parsiranja) koristi se za interpretaciju niza (u listovima isključivo završni znakovi)
• parsiranje – prepoznavanje niza + gradnja generativnog stabla (za niz w i gramatiku G)
– od vrha prema dnu
– od dna prema vrhu

Generativno stablo

begin var:= var;
while var ≠ var
do
var:= var
end

C→ begin S1 end
S1→S S2
S2→ ; S1 | ε
S→var := var | while var ≠ var do S
| begin S1 end



Rekurzivni spust I

- uporaba rekurzije za parsiranje od vrha prema dnu
- nezavršnim znakovima pridružuju se potprogrami koji ispituju da li desna strana produkcije odgovara pročitanom nizu
- završni znakovi desnih stana produkcija uspoređuju se sa nizom
- T (okrenuto) - oznaka za kraj niza

Programska realizacija algoritma:

1. u glavnom programu pročita se prvi znak niza w i pozove se potprogram za početni nezavršni znak gramatike

– ako se pri izvođenju pročita oznaka za kraj niza ⍰ niz se prihvaća u suprotnom se ne prihvaća

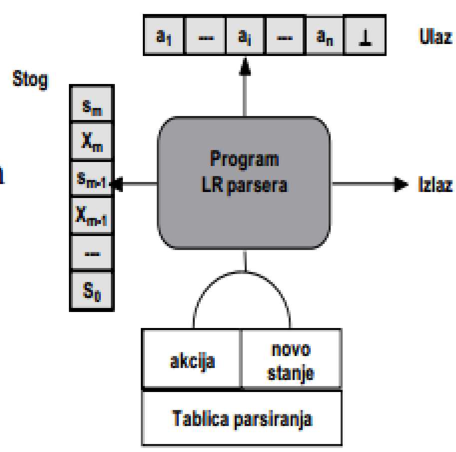
Parsiranje od dna prema vrhu

- gradnja započinje od listova, završnih znakova gramatike
- ako je međuniz niza w jednak desnoj strani produkcije onda se zamijeni lijevom
- na taj način gradi se stablo uključujući i korijen
- broj znakova se postepeno smanjuje prema vrhu zato produkcije nazivamo redukcijama
- metoda primjerena za generatore parsera
- LR(k) parsiranje –

– L niz čitamo s lijeva na desno
– R stablo gradimo s desna na lijevo
– k : najviše k znakova treba pročitati da se primjeni redukcija; k>1

Model LR parsera

- LR parser gradi se posebnim generatorom
- LR parseri različitih gramatika razlikuju se samo u **tablici parsiranja**
- u radu koristi potisni stog (LIFO)



20

Rad bottom-up parsera

- koristi se potisni stog i ulazni spremnik

– na jednom koraku se čita više znakova stoga (ne samo vrh)
– na stogu su završni i nezavršni nizovi
– u ulaznom spremniku je niz w i svi međunizovi koji se parsiraju

- oznaka dna stoga: Δ
- oznaka kraja niza:⊥
- akcije:

– Pomakni; Reduciraj; Prihvati; Odbaci

POTISNI AUTOMATI

-jezik je kontekstno neovisan ako i samo ako postoji potisni automat koji ga prihvaća

Model potisnog automata (PA)

ulaznoj traci, upravljačkoj jedinki i glavi za čitanje dodaje se potisni stog (LIFO stog)
• glava za čitanje čita ulazni znak sa trake i znak sa vrha potisnog stoga
• upravljačka jedinka nalazi se u jednom od konačnog broja stanja:
– prihvatljivih ili neprihvatljivih
• PA nakon čitanja znaka s vrha stoga i s ulazne trake
– s vrha stoga uzima pročitani znak na njegovo mjesto upisuje novi znak ili niz znakova
– te se glava pomiče za jedno mjesto u desno na ulaznoj traci
• ulazna traka je konačna

Rad PA

• upravljačka jedinka donosi odluku o promjeni sadržaja vrha stoga, pomaku glave za čitanje i promjeni stanja na osnovu:
– stanja upravljačke jedinke
– znaka na vrhu stoga i
– pročitanog znaka na traci

Rad PA II

• upravljačka jedinka može donijeti odluku o promjeni na osnovu stanja, ulaznog znaka i znaka na vrhu stoga (3 od 3)
– tada se glava za čitanje miče za jedno mjesto u **DESNO**
• upravljačka jedinka može donijeti odluku o promjeni i samo na osnovu stanja i znaka na vrhu stoga (2 od 3)
– tada se glava za čitanje **NE** miče za jedno mjesto u **desno**

Rad PA III

• upravljačka jedinka odlučuje koji niz se stavlja na vrh stoga, a može se staviti:
1. prazni niz e = **uzimanje znaka sa stoga** (čitanjem se znak uzima sa stoga a umjesto njega se zapiše prazni e)
2. niz duljine 1 znaka = **zamjena znakova na vrhu** (ako se stavi isti znak koji je pročitan s vrha stoga – nema promjene, ako se stavi drugi znak – zamjena)
3. niz duljine više znakova = **primjena produkcije** (vrh stoga se zamijeni nizom znakova, nezavršni znak lijeve strane zamijeni se nizom znakova desne strane produkcije)

Rad PA IV

• odluka o prihvaćanju niza:
– prihvatljivim stanjem

• ako pročita sve znakove na ulazu i stane u prihvatljivom stanju
– praznim stogom
• ako čitanjem svih znakove na ulazu stog ostane prazan

Funkcija prijelaza PA
1. prijelaz: na temelju trojke (q,a,Z) PA mijenja stanje u p:
– (q,a,Z)=p, pomiče glavu za 1 mjesto u desno
– zamijeni znak na vrhu stoga Z nizom znakova γ
2. sigma **-prijelaz:** na temelju trojke (q,⍰,Z) PA mijenja stanje u p:
– sigma(q,e,Z)=p, ostavi glavu na istom mjestu
– zamijeni znak na vrhu stoga Z nizom znakova γ – prijelaz se izvrši bez čitanja ulaznog znaka sa trake

Prihvaćanje niza PA
1. prihvatljivo stanje: ulaskom upravljačke jedinke PA u prihvatljivo stanje nakon čitanja svih znakova ulazne trake- jezik koji se prihvaća je **L(M)**
2. prazan stog: kad se čitanjem svih znakova niza isprazni stog- jezik koji se prihvaća je **N(M)**

Formalno prihvaćanje jezika PA
PA M=(Q, suma, Γ, sigma, q0, Z0, F)
1. prihvatljivim stanjem prihvaća jezik **L(M)**
L(M)={w| (q0,w,Z0) $\succ^*$ (p, e ,γ) za stanje p eF i γ e Γ^*}
– stanje p mora biti prihvatljivo, a stog ne mora biti prazan
2. praznim stogom prihvaća jezik **N(M)**
N(M)={w| (q0,w,Z0) $\succ^*$ (p, e,e)za stanje p e Q}
– stanje p ne mora biti prihvatljivo ali se stog mora isprazniti

Deterministički PA
PA M=(Q, suma, Γ, sigma, q0, Z0, F) je deterministički ako i samo ako su ispunjena oba uvjeta
• ako je sigma(q, e, Z) **neprazni** skup onda je sigma(q, a, Z) **prazni** skup za bilo koji ulazni znak a iz suma
– nema izbora između e-prijelaza i prijelaza s ulaznim znakom
• u skupu sigma(q, a, Z) je najviše **jedan** element i to za bilo koje stanje q iz Q, za bilo koji znak stoga Z iz Γ i za bilo koji ulazni znak
– samo jedno stanje nakon prijelaza

Istovjetnost PA
Dva PA su istovjetna ako i samo ako prihvaćaju isti jezik.
1. konstrukcija PA koji prihvaća praznim stogom iz PA koji prihvaća prihvatljivim stanjem
2. konstrukcija PA koji prihvaća prihvatljivim stanjem iz PA koji prihvaća praznim stogom

3. konstrukcija PA koji prihvaća praznim stogom jezik zadan kontekstno neovisnom gramatikom (KNG�PA)
4. konstrukcija kontekstno neovisne gramatike za jezik koji se prihvaća praznim stogom zadanog PA (PA�KNG)

Greibachov normalni oblik produkcija (GNO)
• neka gramatika G=(V, T, P, S) generira kontekstno neodvisni jezik L(G)\{e}. Moguće je izgraditi istovjetnu gramatiku G'=(V', T', P', S') koja ima sve produkcije oblika **A**->a alfa; **a** je završni znak a alfa **niz nezavršnih znakova koji može biti i prazan**
• najprije se gramatika pretvori u CNO
• zatim se izvede algoritam zamjene krajnjeg lijevog nezavršnog znaka i algoritam razrješenja lijeve rekurzije
• izvede se pretvorba u GNO
Istovjetnost: KNG, KNJ i PA
• jezik je kontekstno neovisan ako i samo ako postoji potisni automat koji ga prihvaća
• kontekstno neovisna gramatika (KNG), kontekstno neovisan jezik (KNJ) i potisni automat (PA) su **istovjetni**
– prihvaćaju klasu kontekstno neovisnih jezika
– klasa KNJ je pravi podskup klase svih jezika

Svojstva zatvorenosti jezika
•*Def:* Klasa jezika je zatvorena s obzirom na neku operaciju ako primjenom te operacije na bilo koji jezik iz te klase dobijemo jezik koji je u istoj klasi
–unija
–nadovezivanje (konkatenacija)
–Kleenov operator
–supstitucija
–presjek i
–komplement

Svojstva zatvorenosti kontekstno neovisnih jezika KNJ
•KNJ **su zatvoreni** s obzirom na operacije:
–unije
–nadovezivanja (konkatenacije)
–Kleenovog operatora
–supstitucije
•KNJ **nije zatvoren na** presjek i komplement
–presjek i komplement KNJ nisu nužno KNJ
–presjek KNJ i regularnog jezika je KNJ (pomoću DKA i PA)

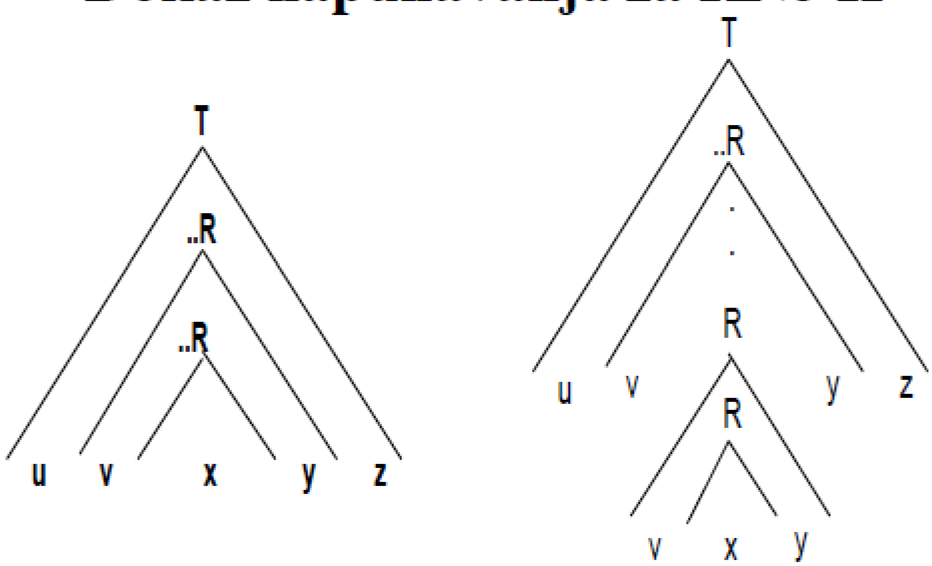
dokaz kontradikcijom
•KNJ nisu zatvoreni na presjek i komplement

Svojstvo napuhavanja KNJ I
•slično svojstvu napuhavanja RJ
•dokazuje se kontekstna neovisnost jezika
•zasniva se na broju čvorova generativnog stabla i broju nezavršnih znakova gramatike
•*Def.:*za dovoljno dugački niz znakova broj unutrašnjih čvorova generativnog stabla veći je od kardinalnog broja skupa nezavršnih znakova gramatike
–znači da je više čvorova označeno istim nezavršnim znakom gramatike

Svojstvo napuhavanja KNJ II
• kaže da uvijek postoje dva kratka podniza koja je oba moguće ponavljati neograničen ali jednak broj puta

Dokaz napuhavanja za KNJ I
• gramatika G generira KNJ A
• moramo pokazati da svaki niz s dovoljne duljine možemo napuhati i da je još uvijek iz jezika A
• s je jako dugi niz iz A i za s je moguće izgraditi generativno drvo
– budući da je s jako dug => drvo je jako visoko => postoji dug put od korijena do listova
– na tom putu postoji simbol R
– R se ponavlja zbog pigeonhole principa
– niz s možemo podijeliti na uvxyz i stablo parsiranja je onda:

Dokaz napuhavanja za KNJ II



TURINGOV STROJ

Umjetna inteligencija

-je potraga za idejama, koje računalima omogućavaju inteligenciju
Cilj: računala učiniti upotrebljivijim i razumjeti principe inteligencije
Inteligentna računala mogu: rješavati teške probleme, pomoći pri istraživanjima i konstruiranjima, pomoći u proizvodnji, razumjeti prirodan jezik, razumjeti slike, naučiti određene primjere i uzorke

Turingov test

pošto su pojmovi misliti i inteligencija neprecizno definirani engleski logičar i matematičar Alan Turing je osmislio empirijski test s kojim se provjerava:
Mogu li strojevi misliti?

• test se provodi kroz igru pitanja u koji sudjeluje ispitivač (osoba) te stroj i osoba koji odgovaraju na pitanja

– njihov identitet je nepoznat i ispitivač samo na osnovu razgovora mora odrediti tko je stroj a tko osoba

• ELIZA – ponaša se kao psihoterapeut

• Loebnerov natječaj

Turingov stroj

TS i rekurzivno prebrojivi jezici

•jezik je rekurzivno prebrojiv ako i samo ako postoji Turingov stroj (TS) koji ga prihvaća

•za bilo koji rekurzivno prebrojiv jezik moguće je izgraditi TS i obratno

•TS ima iste mogućnosti računanja kao bilo koje digitalno računalo

–određene probleme ne možemo riješiti TS-om

–izvan teorijskih granica izračunljivosti

•predstavlja najopćenitiji matematički model računanja

–prihvaćanje niza

–zapis na traci

–računanje cjelobrojnih funkcija

Gramatika neograničenih produkcija i TS

•Gramatika neograničenih produkcija i Turingov stroj su istovjetni

–prihvaćaju klasu rekurzivno prebrojivih jezika

•jezik je rekurzivno prebrojiv neovisan ako i samo ako postoji Turingov stroj koji ga generira

•Type0: Gramatika neograničenih produkcija

Model TS

• upravljačka jedinka nalazi se u jednom od konačnog brojavstanja

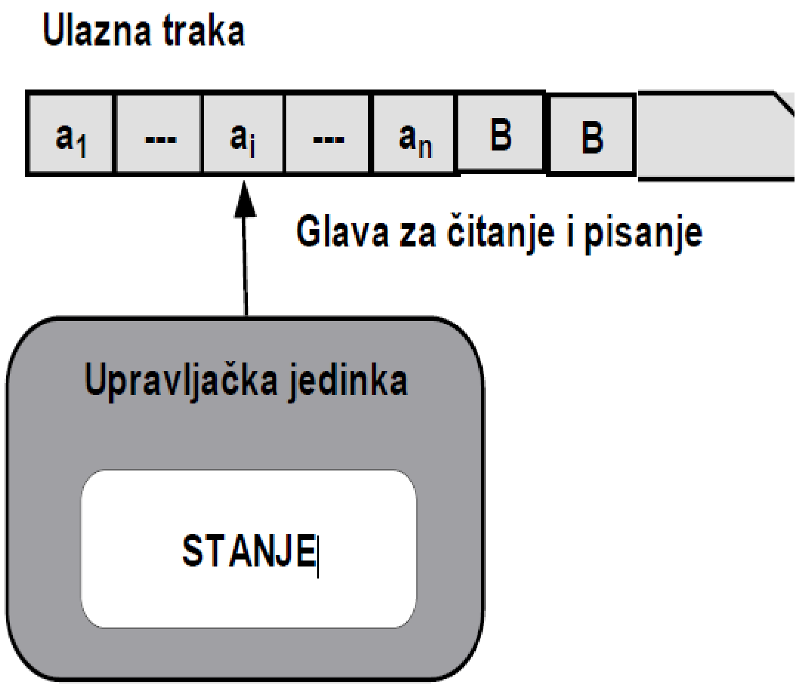
• TS nakon čitanja znaka sa ulazne trake upisuje novi znak za traku

• glava se miče lijevo-desno

• traka ima početak ali ne i završetak – beskonačna

• na početku je u n krajnje lijevih ćelija zapisan niz w

• prazne ćelije: B



Rad TS

• na osnovu:
– stanja i
– pročitanog znaka na traci
• TS odlučuje
– u koje novo stanje prelazi upravljačka jedinica
– koji znak se zapiše na traku umjesto pročitanog znaka
– u koju stranu se miče glava za čitanje i pisanje

Formalna definicija TS

• uređena sedmorka ts=(Q, Σ, Γ, δ, q_0, B, F)
  • Q – konačni skup stanja
  • Γ- konačni skup znakova trake
  • B ϵ Γ – znak za oznaku prazne ćelije
  • Σ⊆(Γ-{B}) - konačni skup ulaznih znakova
  • δ : Q * Γ → Q * Γ *{L, R}; funkcija prijelaza
  • gdje su L i R oznake za pomak glave u lijevo L ili desno R
  • q_0 ϵ Q; početno stanje
  • F ⊆Q, skup prihvatljivih stanja
• Sipser: uređena sedmorka
ts=(Q, Σ , Γ, δ , q_0, F-prihvatljiva st., R-neprihvatljiva stanja)
– ts mora završiti ili u prihvatljivom ili u neprihvatljivom stanju
– ili radi u beskonačnost

Funkcija prijelaza TS

• δ : Q * Γ → Q * Γ *{L, R }
– funkcija prijelaza može biti nedefinirana za određene argumente
• δ(q,V)=(p,Z,W) određuje da TS iz stanja q čitanjem znaka V prelazi u stanje p, na traku zapisuje znak Z (umjesto postojećeg znaka V) te se pomiče u lijevo ili desno ovisno o W

Prihvaćanje niza TS

- prihvaćanje niza: prijelaz među konfiguracijama TS
- konfiguracija TS je trojka: α1 q α2
  - α1 sadržaj ćelija lijevo od glave
  - q stanje upravljačke jedinke
  - α2 sadržaj ćelija desno od glave
- TS na slijedećem koraku čita krajnje lijevi znak iz α2
- TS može na 4 različita načina prelaziti između konfiguracija
  - prijelaz: lijevo ili desno,
  - slijedeći znak je: znak ili prazan znak

Prihvaćanje jezika TS

- TS prihvaćaju rekurzivno prebrojive jezike (Turing-decidable)
  - prebrojivi: moguće je izgraditi TS koji ispisuje (nabraja, broji) sve nizove jezika
  - rekurzivni: TS radi u petlji (ne staje)
- kad TS čita ulazni niz radi akcije
  - prihvaća (accept), odbija (reject) ili radi u petlji (loop) (rekurzija)
  - deciders- TS koji ili prihvaća ili odbija jezik ali uvijek staje s radom (decision to accept or to reject)

Računanje cjelobrojnih funkcija

- cijeli broj se predstavi nizom 0:
  - broj nula = cijeli broj
  - cijeli broj 000..0 i≥0 : 0^i
- ako funkcija ima k argumenata i_1, i_2, ..i_k oni su odvojeni znakom 1: $0^{i1}10^{i2}1..10^{ik}$
- Rad
  - TS ne stane
  - TS se zaustavi, na traci je 0m: vrijednost funkcije je m, bez obzira u kojem stanju se TS na kraju nalazi
- Primjeri
  - prijelaz na n+1 decimalu broja n decimalnog broja (I)
  - pretvaranje unarnog zapisa u decimalni (II)
  - paran i neparan broj jedinica (III)
  - zbrajanje (IV)

Standardni algoritni

- identični prijepis: E(P)=P
- kopiranje: Kop_1(P)=P||P ili Kop_2(P)=P||P ||P
- supstitucija: U^a_αbaba=bαbα
- provjeravanje palindroma
- izdvajanje riječi: Iz(abb||aa||bab)=bab
- preimenovanje svih stanja osim !
- proširenje vanjske abecede...

Parcijalne i potpune rekurzivne funkcije

- broj argumenata funkcije f(i1, i2, ..ik) je k
  - nije nužno da su vrijednosti svih argumenata definirane: parcijalne rekurzivne funkcije
  - ako su svi argumenti definirani: potpuna rekurzivna funkcija
    - primjeri: množenje, n!, 22n
- parcijalno i potpune rekurzivne funkcije su analogne rekurzivnim jezicima:
  - TS koji za bilo koji ulazni niz stane

Složeni TS

- složene oznake ali lakša izrada TS

•definicija TS ostaje ista
•višekomponentna oznaka stanja
  –[q1, q2,...qn]
  –upravljački (upravlja radom TS) i radni dio (spremanje podataka)
  –uvodi se radna memorija!
•višekomponentni znakovi trake
  –više tragova ulazne trake

<u>Višekomponentna oznaka stanja</u>

•komponenta oznake stanja qi: [q1, q2,...qn]
  –u komponente stanja se mogu pohraniti vrijednosti podataka, korisno za pomak znakova na traci u lijevo ili u desno,...
•upravljačke komponente (upravlja radom TS)
•radne komponente (spremanje podataka)
  –u komponentu se sprema znak sa trake
  –pomak znakova na traci u lijevo ili u desno,...

<u>Višekomponentni znakovi trake</u>

• $A^j=[a_1,a_2,..,a_j]$ je složeni znak trake s n komponenti ai
• ako je broj komponenti konačan, i ako je broj vrijednosti koje komponente mogu zauzeti konačan onda su konačni i kardinalni brojevi skupa složenih znakova trake i kardinalni brojevi skupa složenih ulaznih znakova što zadovoljava definiciju TS
• traka je podijeljena na zasebne tragove ulazne trake
  – trag 1, trag 2, .. , trag k
  – broj tragova = broju komponenata

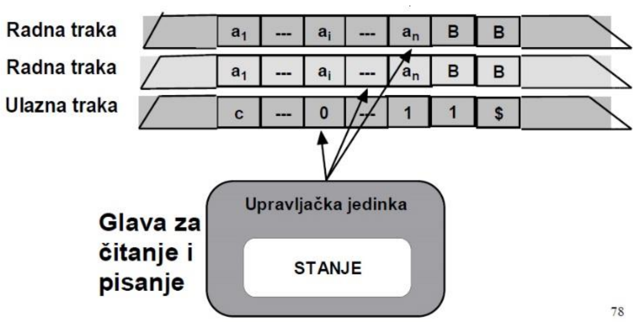
<u>Prošireni model TS</u>

• složeni problemi zahtijevaju korištenje proširenih modela TS:
  – TS s dvostranom beskonačnom trakom
  – TS s višestrukim trakama
• beskonačnim
  – Nedeterministički TS
  – TS s višedimenzionalnim ulaznim poljem
  – TS s više glava za čitanje i pisanje
  – Neizravni TS

<u>TS s dvostranom beskonačnom trakom</u>

TS s dvostranom beskonačnom trakom istovjetan je osnovnom TS
• traka beskonačna
  – na obje strane
• nema krajnje lijevog el.

TS s višestrukim trakama
• višestruke beskonačne trake na obje strane
  – jedna traka je ulazna ostale su radne
  – za svaku traku jedna nezavisna glava



Nedeterministički TS

funkcija δ nije jednoznačna

$\delta : Q * \Gamma \rightarrow \Upsilon(Q * \Gamma * \{L, R\})$

$\delta(q,X)=\{(p_1,Z_1,D_1),(p_2,Z_2,D_2), .., (p_k,Z_k,D_k)\}$

– prijeđe u skup stanja

– u stanju p_i, zapiše znak Z_i i pomakne u smjeru D_i

• nedeterminizam: gradi se stablo koje se grana na svakom koraku

– ako barem jedna slijed (grana) završi u prihvatljivom stanju niz se prihvaća

• nedeterministički TS je istovjetan determinističkom TS

•Decider: nedeterministički TS koji za svaki ulazni niz stane u svakoj od grana

TS s višedimenzionalnim ulaznim poljem

•umjesto trake se koristi k-dimenzionalno polje ćelija

B	B	B	a1	B	B	B
B	B	a2	a3	a4	a5	B
a6	a7	a8	a9	B	a10	B
B	a11	a12	a13	B	a14	a15
B	B	a16	a17	B	B	B

• k-dim polje se sprema na traku kao niz:**BBBa1BBB*BBa2a3
a4a5B*a6a7a8a9Ba10B*Ba11a12a13Ba14a15*BBa16a17BBB**

• glavu je moguće pomaknuti na 2k različita načina

– oko svake k osi u oba smjera

TS s više glava za čitanje i pisanje

•jedna traka i k glava

• glave se nezavisno miču u lijevo ili u desno

– TS donosi odluku na temelju stanja i k pročitanihznakova

• TS s k glava istovjetan je TS-u s jednomglavom

– jedna traka ima k+1 tragova

– prvi trag ima sadržaj

– a k tragova označava položaje k glava

Neizravni TS

• koristi se za dokazivanje prostorne složenosti

– prostorna složenost prihvaćanja jezika

– prostorna složenost računanja cjelobrojnih funkcija

• jedna ulazna (samo se čita) i više radnih traka

– niz na ulaznoj traci je označen graničnicima, glava ne može otići izvan graničnika

• neizravni TS je istovjetan TS-u s višestrukim trakama

Istovjetnost TS

• TS s dvostranom beskonačnom trakom istovjetan je osnovnom TS
• TS s višestrukim trakama istovjetan je osnovnom TS
• nedeterministički TS je istovjetan determinističkom TS
• TS s dvodimenzionalnim poljem istovjetan je TS s jednodimenzionalnom trakom
• TS s k glava istovjetan je TS-u s jednom glavom
• neizravni TS je istovjetan TS-u s višestrukim trakama

Pojednostavljeni model TS

• Pojednostavljeni TS su istovjetni osnovnom TS
• Stogovni stroj
• Stroj s brojilima
• TS s ograničenim brojem stanja i znakova trake
• Univerzalni TS

Stogovni stroj

• deterministički TS s jednom ulaznom trakom i više stogova
  – ulaznu traku je moguće samo čitati
  – stogovi su realizirani kao posebne radne trake s pojednostavljenom funkcijom prijelaza
• ako se glava pomakne u lijevo u ćeliju se obavezno upiše oznaka prazne ćelije
• sve ćelije stoga desno od glave za pisanje su prazne
• TS s jednom trakom moguće je simulirati stogovnim strojem s dva stoga

Stroj s brojilima

•pojednostavi se stogovni stroj
  –umjesto 2 stoga 4 brojila
•skup znakova stoga sadrži:
  –oznaku prazne ćelije B
  –oznaku dna stoga X
•glavu nije moguće pomaknuti lijevo od oznake dna stoga X
•ako se glava pomakne u desno za i ćelija u brojilo se upiše i
•micanjem glave povećava se ili smanjuje vrijednost brojila
•kad glava čita X (dno stoga) u brojilu mora biti 0
•TS s jednom trakom moguće je simulirati strojem s četiri brojila
  –TS s jednom trakom simulira se stogovnim strojem s dva stoga
  –rad jednog stoga simulira se s dva brojila
•količinu brojila moguće je smanjiti na 2
  –pa je TS s jednom trakom moguće simulirati strojem s dva brojila
  –rad četiri brojila simulira s pomoću dva brojila

TS s ograničenim brojem stanja i znakova trake

istodobno se ograniči broj stanja, broj traka i broj znakova trake
  –s time se ograniči broj različitih TS koje je na taj način moguće izgradit
  –ne prihvaća isti skup jezika kao osnovni TS
•ograniči se broj stanja na 3 a broj znakova trake i traka nije ograničen
  –onda je za prihvaćanje bilo kojeg rekurzivno prebrojivog jezika moguće imati jednu traku i 3 stanja (1 prihvatljivo i 2 neprihvatljiva)
•ograniči se broj znakova {0,1,B} i traka a broj stanja nije ograničen

Univerzalni TS

•univerzalni TS M0 može simulirati bilo koji TS s jednom trakom
•ima 3 trake

–prva: funkcije prijelaza TS i ulazni niz w <M,w> (kodiran s oznakam # i ▯ itd.)
–druga: traka za rad
–treća: stanje u kojem se TS nalazi

•prepiše ulazni niz w na drugu traku
•simulira prijelaze TS s druge trake uzme stanje s treće i zapisuje prijelaz u stanje na treću traku
–ako nema prijelaza za znak s 2 i stanje s 3 se rad TS zaustavlja
–ako je stanje na trećoj traci prihvatljivo niz se prihvaća

Generiranje jezika TSom

•TS s višestrukim trakama
–jedna traka je izlazna: kad se znak zapiše na izlaznu traku više ga nije moguće mijenjati ili brisati
–glava se na izlaznoj traci miče isključivo u desno
–izlazna traka koristi se za generiranje jezika G(M)
–izlazni nizovi nad abecedom Σ su podijeljeni s #

•jezik kojeg TS generira G(M) (nije isti kao L(M) kojeg TS prihvaća) je skup nizova wϵΣ*
–ako TS stane je G(M) konačan
–ako TS ne stane je G(M) beskonačan
–isti niz moguće je generirati više puta

•TS generira klasu rekurzivno prebrojivih jezika
–jezik L je rekurzivno prebrojiv ako i samo ako postoji TS M1 koji generira jezik G(M1)=L

Sipser

•Turing recognizable languages- rekurzivno prebrojivi
•Turing decidable languages- rekurzivni, odlučivi

Prihvaćanje jezika generiranog TSom

•za bilo koji TS M2 koji prihvaća jezik L(M2) moguće je izgraditi TS M1 koji generira jezik G(M1)=L(M2)
–TS M1 može biti jednostavan pa generira rekurzivni jezik G(M1)=L(M2)
–TS M1 može biti složen pa generira rekurzivno prebrojiv jezik G(M1)=L(M2)

•TS M1 može biti jednostavan pa generira rekurzivni jezik G(M1)=L(M2)
–određenim redom M1 ispisuje sve nizove w1,w2,w3,.. iz Σ* na radnu traku
–zatim se simulira rad M2
•ako se niz prihvati prepiše ga se na izlaznu traku
–budući da je G(M1)=L(M2) rekurzivan jezik simulacijom prihvaćanja TS M2 uvijek stane s radom (nema beskonačnih petlji)

Prihvaćanje rekurzivno prebrojivog jezika generiranog TSom

•TS M1 može biti složen pa generira rekurzivno prebrojiv jezik G(M1)=L(M2)
–moguće je da postoji niz w_j koji nije u jeziku L(M2) i za koji TS M2 prilikom simulacije prihvaćanja nikad ne stane

•to znači da se nikada ne pređe u ispitivanje prihvaćanja nizova w_{j+1}, w_{j+2},...
•ako je među tim nizovima w_{j+k} niz koji se nalazi u L(M2) i nije na izlaznoj traci a M1 bi ga trebao generirati
•rješenje: M2 ne smije imati neograničen broj prijelaza (broj prijelaza se ograniči)

Stroj s četiri brojila

•k različitih znakova stoga: Z_0, Z_1, Z_2,..,Z_{k-1}
•na stogu su zapisani znakovi: Zi_1, Zi_2,..,Zi_m
•i_1, i_2,..,i_m indeksi znakova su cjelobrojne vrijednosti
•na vrhu stoga je Zim koji ima vrijednost im
•vrijednost znakova stoga je cijeli broj po bazi k
–$j= i_m+k\ i_{m-1}+k_2\ i_{m-2}+k_3\ i_{m-3}+..+k_{m-1}\ i_1$

•osnovne operacije:

–stavljanje znaka na vrh stoga
–uzimanje znaka s vrha stoga
–određivanje znaka na vrhu stoga

•**stavljanje znaka** Zr na vrh stoga
•na stogu su: Zi_1, Zi_2,..,Zi_m, Zi_r
–a njihova vrijednost je jk+r
–s dva brojila A i B se računa vrijednost jk+r
•na početku A: j i B: 0
–jk se izračuna:
•A: j-1 B:k i smanjuje se dok u A: 0 a u B: jk,
•na zadnjem koraku se u B doda r B: jk+r

•**uzimanje znaka** Zi_m s vrha stoga
•na stogu su: Zi_1, Zi_2,..,Zi_m i njihova vrijednost je j
–nakon uzimanja Zim s vrha preostala vrijednost je
–cijeli dio |j/k|
–j/k se izračuna:
•početak A: j B:0
•A: j-k B: +1 (B se poveća za 1)
•A: 0 a u cijeli dio B: j/k

Stroj s dva brojila
•rad četiri brojila simulira s pomoću dva brojila
•i, j, k, l vrijednost 4 brojila u jedno brojilo se može spremiti vrijednost $n=2^i3^j5^k7^l$
–2,3,5,7 su prim brojevi => n jednoznačno određen
•osnovne operacije:
–povećanje/smanjivanje brojila za 1: i, j, k, l se povećaju za 1
•vrijednost brojila se množi s 2,3,5,ili 7
–određivanje da li je vrijednost brojila = 0
•brojilo se dijeli s 2,3,5,7
•s kojim brojem nije djeljivo odgovarajuće brojilo =0 ???

Generiranje jezika kanonskim slijedom
•kraći nizovi su ispred duljih nizova
•redosljed nizova jednake duljine odredi se pomoću numeričke vrijednosti
–baza za izračun je kardinalni broj skupa ulaznih znakova
–binarna abeceda Σ={0,1} baza je 2 a kanonski slijed je : ε, 0,1,00,01,10,11,000,001, ..111, 0000,..
•rekurzivne jezike moguće je generirati kanonskim slijedom

Istovjetnost rekurzivnih jezika i jezika generiranih kanonskim slijedom
•jezik L(M2) je rekurzivan (znači da ga je moguće generirati jednostavnim TSom)
–izlazni nizovi se generiraju na izlaznoj traci istim redoslijedom kao i na radnoj traci
–ako se generiraju kanonskim slijedom na radnu trsku idu kanonskim slijedom i na izlaznu traku
•Jezik G(M1) koji je moguće generirati kanonskim slijedom je rekurzivan jezik

Gramatika neograničenih produkcija (GNP)
•produkcije regularne gramatike su ograničene
–lijevo linearne ili desno linearne
•produkcije kontekstno neovisne gramatike imaju ograničene oblike
–na lijevoj stani jedan znak
•produkcije kontekstno ovisne gramatike

- –na lijevoj stani manje ili jednako znakova desnoj

- gramatika neograničenih produkcija (gramatika tipa 0) nema ograničenja, produkcije oblika
  - α->β, α i β su nizovi završnih i nezavršnih nizova gramatike, α ne smije biti prazan
- gramatika neograničenih produkcija G=(V, T, P, S)
  - za produkciju α-> β je definirana relacija =>: $\gamma\alpha\delta$-> $\gamma\beta\delta$
  - =>* je refleksivno tranzitivno okruženje relacije =>
  - gramatika G generira jezik L(G)
    - L(G)={w | wϵT* i S=>*w}
- GNP generira klasu rekurzivno prebrojivih jezika

<u>Konstrukcija TS za jezik zadan gramatikom neograničenih produkcija</u>

- Ako GNP G=(V, T, P, S) generira jezik L(G) onda je L(G) rekurzivno prebrojiv jezik
  - jezik L(G) je rekurzivno prebrojiv ako postoji TS M koji ga prihvaća L(M)=L(G)
- gradi se nedeterministički TS M s dvije trake koji simulira rad gramatike G:
  - na prvu traku se zapiše ulazni niz w
  - na drugu traku početni nezavršni znak gramatike
  - na drugu traku se tijekom rada zapisuju nizovi α koje generira gramatike
  - TS usporedi α i w na dvije trake (ako su isti prijeđe u prihvatljivo stanje i prihvati)
- TS nedeterministički izabere mjesto i u nizu α s druge trake $1\leq i\leq|\alpha|$
  - nedeterminizam uzrokuje više mogućih simulacija za sve vrijednost i na tom mjestu u produkciji
- TS nedeterministički izabere produkciju β->γ iz G
  - ako je više produkcija β->γ postupak se izvede za sve (nedeterminizam)
  - ako je β na mjestu i onda se β zamijeni s γ
  - niz generiran na drugoj usporedi se s w na prvoj traci
    - ako su jednaki TS zaustavi rad i prihvati niz w
    - ako nisu ponovi od prve točke

<u>Konstrukcija gramatike za jezik zadan TS</u>

- ako TS M prihvaća rekurzivno prebrojiv jezik L(M) onda postoji gramatika G neograničenih produkcija koja generira jezik L(G)=L(M)
- gradi se gramatika G=(V, T, P, S) koja simulira TS M=(Q, Σ, Γ, δ, q_0, B, F):
  - gramatika G generira međunizove koji su konfiguracija TS
  - nezavršni znak q u međunizu predstavlja oznaku stanja TS
  - početna konfiguracija je međuniz a nezavršnim znakovima oblika [ai,ai] gdje je ai ulazni znak iz Σ
  - prva komponenta nezavršnog znaka [ai,ai] čuva znakove a1a2a3,..an dok druga predstavlja znakove trake
- na temelju znakova prve komponente i nezavršnog znaka gramatika G generira niz završnih znakova a1a2a3..an ako i samo ako TS prihvaća taj isti niz znakova
- početna konfiguracija: q0[a1,a1] [a2,a2].. [an,an]
- na kraju je m oznaka praznih ćelija: $[\varepsilon, B]^m$
  - tijekom simulacije nezavršni znakovi q0[a1,a1] [a2,a2].. [an,an] $[\varepsilon, B]^m$
- tijekom simulacije nezavršni znakovi poprimaju oblik[b,X] ulazni znak b$\epsilon\Sigma$ a X znak trake X$\epsilon\Gamma$
- konfiguracija $X_1X_2..X_{r-1}qX_rX_{r+1}..X_s$ predstavlja se $[a_1,X_1]$ $[a_2,X_2]..[a_{r-1},X_{r-1}]q[a_r,X_r]..$ $[a_s,X_s][a_{s+1},X_{s+1}]..[a_{n+m}X_{n+m}]$
  $X_{s+1}..X_{n+m}$ su oznake praznih ćelija
- ako je nezavršni znak q u generiranom međunizu prihvatljivo stanje TS prihvaća w i prihvaća niz a1a2a3..an
- produkcije se dijele u 4 skupine
  - produkcije koje generiraju početnu konfiguraciju
  - produkcije koje dodaju potreban broj praznih ćelija

–produkcije koje simuliraju rad TS-a
–produkcije prihvatljivih stanja

Ograničenje gramatika

•type0: gramatike neograničenih produkcija
•type1: kontekstno ovisne gramatike
•type2: kontekstno neovisne gramatike
•type3: regularne gramatike

Svojstva rekurzivnih i rekurzivno prebrojivih jezika

•zatvorenost rekurzivnih jezika
–unija i komplement
•zatvorenost rekurzivnih prebrojivih jezika
–unija
•izračunljivost
–Church-Turingova hipoteza
•odlučivost
–univerzalni TS

Zatvorenost

•zatvorenost rekurzivnih i rekurzivno prebrojivih jezike se dokazuje TSom
–TS mora uvijek stati, za bilo koji ulazni niz
•zatvorenost rekurzivnih jezika
–unija i komplement
•zatvorenost rekurzivnih prebrojivih jezika
–unija

•pokazano je svojstvo komplementa rekurzivno prebrojivog jezika
–ako je komplement rekurzivno prebrojivog jezika L rekurzivno prebrojiv onda su jezik L i njegov komplement rekurzivni
–ako komplement rekurzivno prebrojivog jezika L nije rekurzivno prebrojiv onda jezik L sigurno nije rekurzivan a moguće nije ni rekurzivno prebrojiv
• **unija** rekurzivnih jezika je rekurzivni jezik
– konstrukcija TS-a M koji prihvaća uniju rekurzivnih jezika
– radi se slijedna simulacija M, M1 i M2
– M1 i M2 prihvaćaju L1 i L2 i uvijek stanu
– ako jedan prihvati niz: niz se prihvaća, inače se ne prihvaća
• unija rekurzivno prebrojivih jezika je rekurzivno prebrojiv jezik
– konstrukcija TS-a M koji prihvaća uniju rekurzivnih jezika
– radi se paralelna simulacija M1 i M2
– M1 i M2 prihvaćaju L1 i L2 i uvijek stanu
– ako jedan prihvati niz: niz se prihvaća, inače se ne prihvaća
• **komplement** rekurzivnog jezika je rekurzivan jezik

Algoritam

• skup jednostavnih naredbi za izvršenje zadatka (procedura, recept,..)
• točna definicija tek u 20.st.
• matematičar Hilbert 1900. identificirao 23 problema
– 10. problem se odnosi na algoritme
– najprije definirati algoritam a zatim dokazati da ne postoji rješenje

Izračunljivost

•intuitivana (“meka”) definicija:

–problem je izračunljiv ako postoji automat koji postupkom korak po korak (mehaničkim) rješava zadani problem

–nema ograničenja:

•broja koraka

•veličine spremnika

•postupak se ne mora zaustaviti

–potrebno samo raščlaniti rješavanje na slijed postepenih koraka

Church - Turingova hipoteza

•izračunljive funkcije se poistovjećuju s klasom parcijalno rekurzivnih funkcija

–točnost hipoteze se ne dokazuje jer nema formalne definicije

–pokazuje se prikladnost i razumnost hipoteze

•parcijalno rekurzivne funkcije su izračunljive jer je za njih moguće izgraditi TS

•TS funkcije računa "mehaničkim" putem: korak po korak

•TS ne mora stati sa svaki ulazni niz

Odlučivost

•rekurzivni jezici su odlučivi

–jer ih prihvaćaju TS koji uvijek stanu i odluče o prihvaćanju ili neprihvaćanju niza

•rekurzivno prebrojivi jezici nisu odlučivi

–ne postoji TS koji uvijek stane

–ako niz nije u jeziku TS neće stati

–ako ne stane ne možemo odrediti da li ga prihvaćamo ili ne

Odlučivost i izračunljivost

•rekurzivni jezici su izračunljivi i odlučivi

•rekurzivno prebrojivi jezici su izračunljivi ali nisu odlučivi

KOPLEKSNOST

Složenost jezika

• strukturna složenost jezika

– složenost automata koji jezik prihvaća

– Chomskyeva hijerahija jezika

• složenost prihvaćanja jezika

– vrijeme i prostor potrebni da se jezik prihvati

– vremenska i prostorna kompleksnost

Strukturna složenost jezika

• ako su A i B dvije klase jezika i ako je A pravi podskup od B onda vrijedi:

– automat koji prihvaća jezike iz klase A jednostavniji je (po strukturi) od automata koji prihvaća jezik iz klase B

– produkcije gramatike koje generira jezik iz klase A su jednostavnije od produkcija gramatike koja generira jezik iz klase B

• jezici iz klase A su jednostavnije strukturne složenosti od klase B

Hijerarhija automata i gramatika I

• zasniva se na istovjetnostima:

– regularnih jezika, konačnih automata i regularnih gramatika
– kontekstno neovisnih jezika, potisnog automata i kontekstno neovisnih gramatika
– kontekstno ovisnih jezika, linearno ograničenog automata i kontekstno ovisnih gramatika
(nismo radili)
• broj znakova desne strane **veći ili jednak** broju znakova lijeve strane produkcije
– rekurzivno prebrojivih jezika, Turingovog stroja i gramatike neograničenih produkcija

Složenost prihvaćanja jezika
• ovisi od veličine trake i vremena da automat prihvati jezik
• zbog hijerarhije jezika i automata TS je osnovni automat za ocjenu složenosti prihvaćanja svih klasa jezika
– **veličina trake**: broj ćelija koje se tijekom rada koriste
– **vrijeme**: broj pomaka glave TS-a
• jedan pomak je jedna jedinica vremena

Prostorna složenost prihvaćanja jezika II
• ulaznu traku se samo čita
• duljina ulaznog niza je n
• k radnih traka je beskonačno na jednu stranu i na njih se čita i piše
• prostorna složenost S(n) određuje se na osnovu **samo jedne radne trake** i to one na kojoj je korišteno **najviše ćelija n**

Vremenska složenost prihvaćanja jezika II

• na sve trake (radne i ulazne) se čita i piše
• vremenska složenost T(n) određuje se pomoću broja pomaka glave za čitanje i pisanje n

Svojstva vremenske i prostorne složenosti
• broj traka ne utječe na prostornu ali utječe na vremensku složenost
• vremenska složenost povećava se povećanjem broja traka

Klase jezika
• ako jezik prihvaća nedeterministički TS jezik je nedeterminističke složenosti
• 4 klase jezika:
– **DSPACE(S(n))** – jezici determinističke prostorne složenosti
– **NSPACE(S(n))** – jezici nedeterminističke prostorne složenosti
– **DTIME(T(n))** – jezici determinističke vremenske složenosti
– **NTIME(T(n))** – jezici nedeterminističke vremenske složenosti

Intractable problems
• to su u principu rješivi problemi ali njihovo rješavanje zahtijeva toliko vremena i prostora da in ne koristimo u praksi

Praksa
•za svaki algoritam određuje se vremenska kompleksnost
–brzina procesora ograničavajući faktor
–**prihvatljivo:** logaritamska ili polinomska kompleksnost algoritma
–**neprihvatljivo**: eksponencijalna kompleksnost algoritma
•rjeđe se određuje i prostorna kompleksnost
–niske cijene memorijskih kapaciteta
–jeftinije kupiti dodatnu brzu ili eksternu memoriju nego optimirati izvođenje algoritma

Koji algoritam odabrati?
• mora biti:
– razumljiv
– jednostavna implementacija
– jednostavno otklanjanje pogreški (debug)
– efikasna iskorištenost računalnih resursa
– brzina vs. prostor...
• jednokratna upotreba (troškovi razvoja)
• učestala upotreba (troškovi korištenja)

Cijena
•ukoliko algoritam radi često i s velikom količinom podataka isplati se potrošiti resurse (vrijeme i rad) na njegovo optimiranje
–isplati se implementirati kompleksniji algoritam koji će raditi efikasnije (vremenski i prostorno)

•potrebno uvesti mjeru kompleksnosti algoritma, koja će ocijeniti njegove vremenske i prostorne potrebe

Vrijeme izvođenja programa
•ovisi od:
–količine i vrste ulaznih podataka u program
–kvalitete koda koju generira compiler
–brzini i performansama računala (sklopovlja)
–vremenskoj zahtjevnosti (kompleksnosti) algoritma
Rješivost (izračunljivost) problema
•ako je problem izračunljiv *(decidable)*
–moguće ga je riješiti (izračunati) - rješiv
•onda u praksi računalni algoritam za rješenje problema zahtijeva
–prostor (memorijske kapacitete)
–vrijeme (potrebno za izvođenje postupaka)
–resurse (računalne resurse)
Vremenska kompleksnost
•određuje kompleksnost algoritma na osnovu potrebnog vremena za rješenje problema
•mjera vremenske kompleksnosti *(time complexity or runing time complexity)*
–je maksimalan broj koraka M u kojima postupak obrađuje ulaz dužine n za rješenje problema
•računa se za
–najgori slučaj *(worst-case, pesimistic)*
–najbolji slučaj *(best-case, optimistic)*
–prosjek
Ocjena vremenske kompleksnosti
•određivanje egzaktnog vremena izvođenja algoritma je kompleksno, pa i teško, zato se u praksi samo **ocjenjuje** vremenska kompleksnost izvođenja
•ocjena se izražava kao **O(n)** gdje n predstavlja gornju asimptotsku ocjenu reda funkcije f(n)
–npr: ako ocijenimo vrijeme izvođenja programa s funkcijom f(n)=6n3+2n2+2n+45 onda je ocjena vremenske kompleksnosti algoritma O(n3)

Dvije ocjene vremenske kompleksnosti
• **veliki O** - gornja asimptotska ocjena reda funkcije
– ocjenjena funkcija nikad nije veća od ocjene O
• **mali o** – donja asimptotska ocjena reda funkcije
– ocjenjena funkcija je veća od ocjene o

Klase
• **P klasa:** polinomska složenost algoritama
– **odlučivi** u polinomskom vremenu na determinističkom TS s 1 trakom
• **NP klasa: ne mogu biti rješeni u** polinomskom vremenu (brut- force)

– možda postoje bolji algoritmi ali ih još nismo pronašli
– u polinomskom vremenu možemo provjeriti (verify) rješenje ali ga ne možemo odrediti (determine)
– **provjerivi** u polinomskom vremenu