Modeli izračunavanja — predavanja

Marko Horvat <mhorvat@math.hr> (A306)
konzultacije: srijedom, 10-12 (uz prethodnu najavu)

PMF–Matematički odsjek, Sveučilište u Zagrebu

akademska godina 2025./2026. — zimski semestar

Kratko o formalnostima

Literatura: Sipser, *Introduction to the Theory of Computation*, 3rd ed.

Bodovi: Kolokvij 25 bodova, pisani ispit 40 bodova, završni usmeni ispit 35 bodova

Pravila ocjenjivanja: zasebni PDF, s odsječkog weba

Formalni jezici

Svaki neprazan konačan skup Σ zovemo *abeceda*. Elemente od Σ nazivamo *znakovima* abecede Σ. Konačan niz w u Σ zovemo *riječ nad* Σ; njenu *duljinu* označavamo sa $|w|$. Riječ duljine 0 označavamo ε i zovemo *prazna riječ*. Broj znakova $\alpha \in \Sigma$ u riječi w nad Σ označavamo sa $|w|_\alpha$.

Bilo koji skup riječi nad Σ zovemo *jezik nad* Σ. Skup svih riječi nad Σ označavamo sa Σ^* (razloga se možda sjećate iz Programiranja 1). Kažemo da je jezik *pozitivan* ako ne sadrži ε kao element.

Isto ćemo pisati znak $\alpha \in \Sigma$ i riječ $\alpha \in \Sigma^*$. Čak ćemo ponekad sa α označavati i jezik $\{\alpha\}$ kad je iz konteksta jasno o čemu se radi. No, važno je razumjeti da se radi o tri različita objekta (u teoriji skupova bismo ih možda precizno definirali redom kao $(\alpha, \Sigma, 0)$, $(\alpha, \Sigma, 1)$, $(\alpha, \Sigma, 2)$).

Što proučavamo?

Klase jezika u *Chomskyjevoj hijerarhiji* i modele izračunavanja koji ih „hvataju”. Originalno ima četiri, no kod nas pet razina:

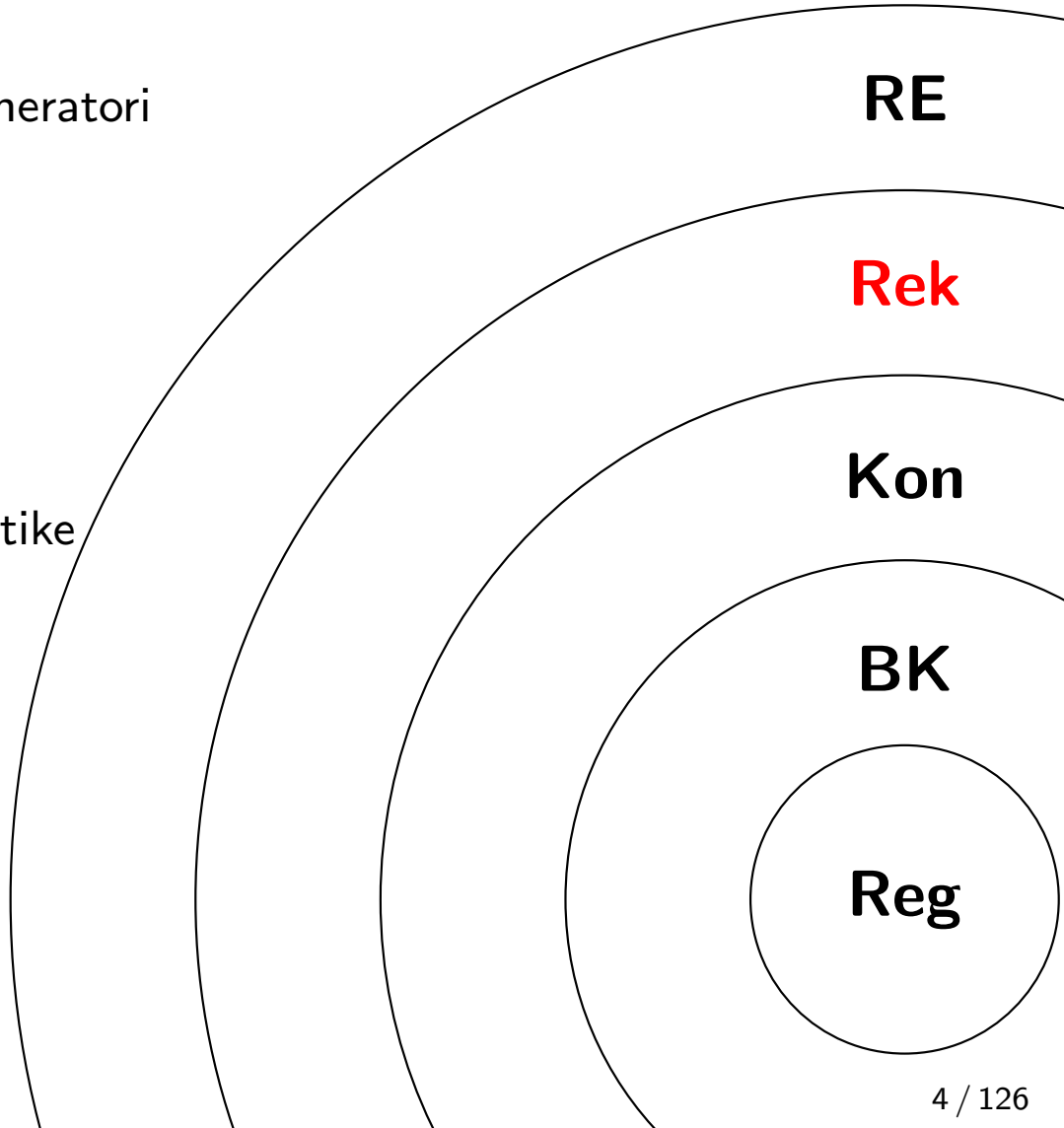
Rekurzivno prebrojivi jezici (RE)
- Turingovi prepoznavači i enumeratori
- opće gramatike

Rekurzivni jezici (Rek)
- Turingovi odlučitelji

Kontekstni jezici (Kon)
- ograničeni automati
- kontekstne i monotone gramatike

Beskontekstni jezici (BK)
- potisni automati
- beskontekstne i Chomskyjeve gramatike

Regularni jezici (Reg)
- regularni izrazi
- konačni automati
- linearne gramatike



Razlozi proučavanja Chomskyjeve hijerarhije

Koje probleme (modelirane jezicima) možemo riješiti algoritamski? Kakvi resursi nam trebaju za izvršavanje tih algoritama?

Preciznije, možemo li danim modelom izračunavanja prepoznati/generirati/reprezentirati dani jezik?

Razmotrimo na trenutak *prepoznavanje* nekog jezika L strojem (=za svaku ulaznu riječ $w \in \Sigma_L^*$, stroj prihvaća w akko je $w \in L$):

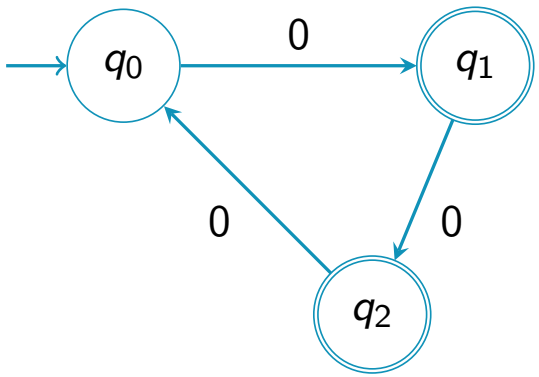
- Možemo li prepoznati jezik L strojem s fiksnom konačnom količinom memorije?
- Ako ne, tj. uvijek postoji riječ za koju stroju treba više memorije za izračunavanje, dostaje li jednokratna upotreba zapamćenog (briše se nakon pristupa, kao skidanje sa stoga)?
- Ako ne, je li barem dovoljno onoliko memorije koliko treba za zapisati samu instancu/ulaz (ili fiksno puta više od toga)?
- Ako ne, je li nužno uvijek dati DA-NE odgovor, ili je dovoljno za pozitivne instance odgovoriti DA, a za negativne instance nikad ne dati odgovor (beskonačna petlja)?

Konačni automati

Pogledajmo prvo kako jednostavan problem možemo modelirati pomoću jezika.
Ako nas zanimaju konačni nizovi nula čija duljina nije djeljiva s tri, razmotrit ćemo abecedu $\Sigma = \{0\}$ i sljedeći jezik nad Σ:

$$L = \{w \in \Sigma^* : |w| \not\equiv 0 \pmod 3\}$$

Iako nismo još precizno definirali pripadnu vrstu automata, gotovo sigurno ćete odmah shvatiti kako donji automat radi i zašto zaista prihvaća točno riječi iz L:



Ovdje je važno primijetiti da je ponašanje ovog automata za svaku pojedinu riječ iz Σ^* *deterministično* (uvijek je isto).

Deterministički konačni automat (DKA)

Definicija: *Deterministički konačni automat (DKA) nad abecedom* Σ je $\mathcal{M} := (Q, \Sigma, \delta, q_0, F)$ koji uz Σ ima **konačan** skup Q (*stanja*), *funkciju prijelaza* $\delta : Q \times \Sigma \to Q$, istaknuti element $q_0 \in Q$ (*početno stanje*) te podskup $F \subseteq Q$ (*završna* stanja).

Kažemo da $\mathcal{M}$ *prihvaća* riječ $w = \alpha_1 \cdots \alpha_n \in \Sigma^*$ ako postoji niz stanja $r_0, r_1, \ldots, r_n \in Q$ takav da je $r_0 = q_0$, $\delta(r_{i-1}, \alpha_i) = r_i$ za sve $i \in [1..n]$ (takav je jedinstven za zadane $\mathcal{M}$ i w), te $r_n \in F$.
$\mathcal{M}$ *prepoznaje* jezik $L(\mathcal{M}) := \{w \in \Sigma^* : \mathcal{M} \text{ prihvaća } w\}$.

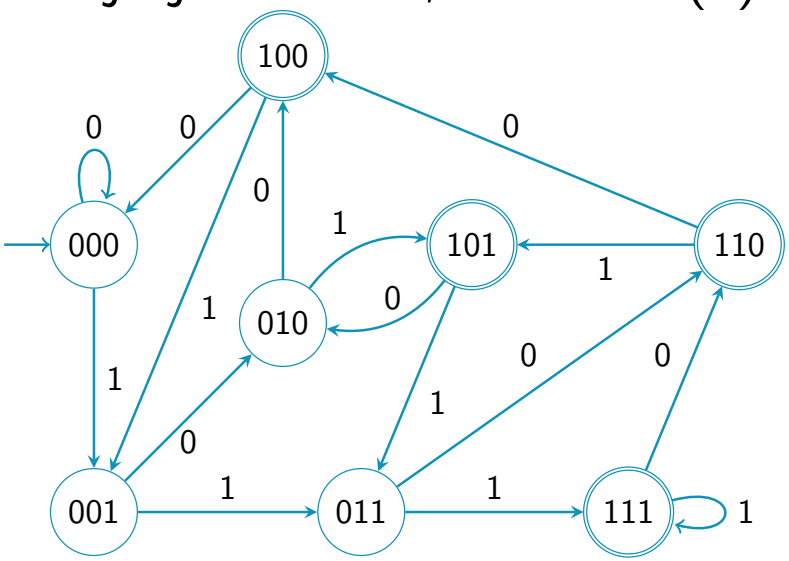
Primjer: Postoji li DKA nad $\Sigma = \{0, 1\}$ koji prepoznaje $L =$ „treći znak od kraja je 1”?

Deterministički konačni automat (DKA)

Definicija: *Deterministički konačni automat (DKA) nad abecedom* Σ je $\mathcal{M} := (Q, \Sigma, \delta, q_0, F)$ koji uz Σ ima **konačan** skup Q (*stanja*), *funkciju prijelaza* $\delta : Q \times \Sigma \to Q$, istaknuti element $q_0 \in Q$ (*početno stanje*) te podskup $F \subseteq Q$ (*završna* stanja).

Kažemo da $\mathcal{M}$ *prihvaća* riječ $w = \alpha_1 \cdots \alpha_n \in \Sigma^*$ ako postoji niz stanja $r_0, r_1, \ldots, r_n \in Q$ takav da je $r_0 = q_0$, $\delta(r_{i-1}, \alpha_i) = r_i$ za sve $i \in [1..n]$ (takav je jedinstven za zadane $\mathcal{M}$ i w), te $r_n \in F$.
$\mathcal{M}$ *prepoznaje* jezik $L(\mathcal{M}) := \{w \in \Sigma^* : \mathcal{M} \text{ prihvaća } w\}$.

Primjer: Postoji li DKA nad $\Sigma = \{0, 1\}$ koji prepoznaje $L =$ „treći znak od kraja je 1”? Da, ali s čak (?) osam stanja!

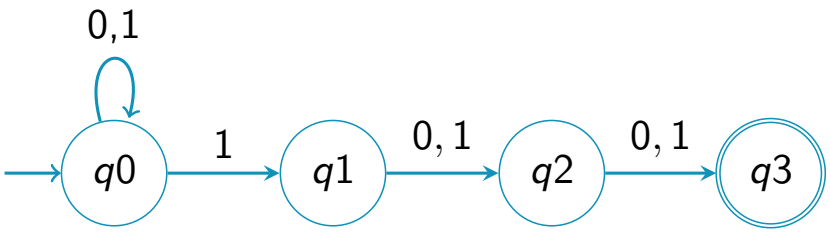


Nedeterministički konačni automat (NKA)

NKA može raditi dvije stvari koje DKA ne može. Prvo, može imati više isto označenih prijelaza iz istog stanja. Ako te prijelaze shvatimo kao „smiješ, ali ne moraš slijediti ovaj prijelaz" te nas samo zanima *postoji* li način da NKA bude na pravom mjestu (završno stanje) u pravo vrijeme (neposredno nakon što pročita zadnji znak ulaza), dobivamo puno manji NKA koji prepoznaje L:

Nedeterministički konačni automat (NKA)

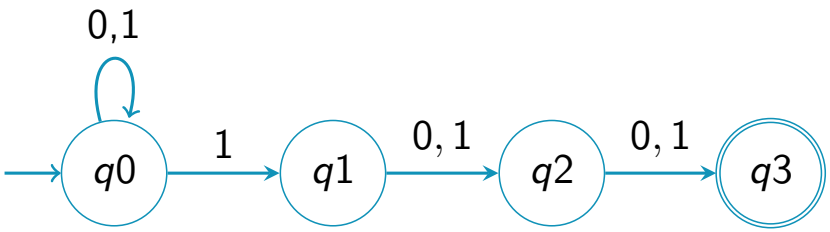
NKA može raditi dvije stvari koje DKA ne može. Prvo, može imati više isto označenih prijelaza iz istog stanja. Ako te prijelaze shvatimo kao „smiješ, ali ne moraš slijediti ovaj prijelaz" te nas samo zanima *postoji* li način da NKA bude na pravom mjestu (završno stanje) u pravo vrijeme (neposredno nakon što pročita zadnji znak ulaza), dobivamo puno manji NKA koji prepoznaje L:



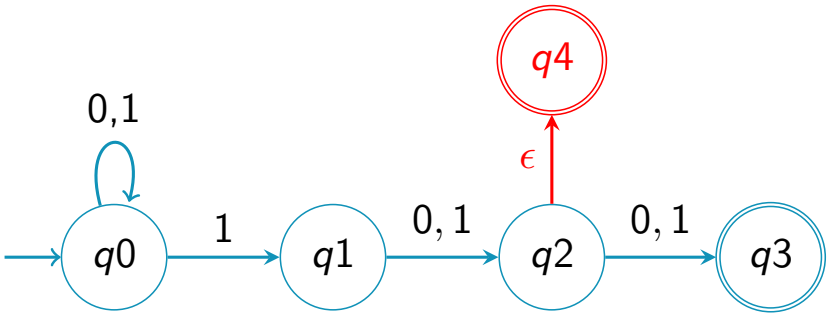
Drugo, može biti korisno da se neki prijelaz *smije* napraviti, ali *bez da se išta čita s ulaza*. Oznaka za takav prijelaz će biti ε na strelici. Primjerice, kako ćemo dodati riječi kojima je 1 drugi znak od kraja?

Nedeterministički konačni automat (NKA)

NKA može raditi dvije stvari koje DKA ne može. Prvo, može imati više isto označenih prijelaza iz istog stanja. Ako te prijelaze shvatimo kao „smiješ, ali ne moraš slijediti ovaj prijelaz" te nas samo zanima *postoji* li način da NKA bude na pravom mjestu (završno stanje) u pravo vrijeme (neposredno nakon što pročita zadnji znak ulaza), dobivamo puno manji NKA koji prepoznaje L:



Drugo, može biti korisno da se neki prijelaz *smije* napraviti, ali *bez da se išta čita s ulaza*. Oznaka za takav prijelaz će biti ε na strelici. Primjerice, kako ćemo dodati riječi kojima je 1 drugi znak od kraja?



Nedeterministički konačni automat (NKA)

Definicija: *Nedeterministički konačni automat (NKA) nad abecedom* Σ je $\mathcal{M} := (Q, \Sigma, \Delta, q_0, F)$ koji uz Σ ima **konačan** skup Q (*stanja*), *relaciju prijelaza* $\Delta \subseteq Q \times \Sigma_\varepsilon \times Q$ gdje je $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$, istaknuti element $q_0 \in Q$ (*početno stanje*) te podskup $F \subseteq Q$ (*završna* stanja).

Označimo $\Delta(q, \alpha) := \{q' \in Q : (q, \alpha, q') \in \Delta\}$. Kažemo da $\mathcal{M}$ *prihvaća* riječ $w \in \Sigma^*$ ako se ona može zapisati kao $w = \alpha_1 \dots \alpha_n$ gdje je svaki $\alpha_i \in \Sigma_\varepsilon$ i postoji niz stanjâ $r_0, r_1, \dots, r_n \in Q$ takav da je $r_0 = q_0$, $r_i \in \Delta(r_{i-1}, \alpha_i)$ za sve $i \in [1..n]$ (takav nije nužno jedinstven za zadane $\mathcal{M}$ i w), te $r_n \in F$.

$\mathcal{M}$ *prepoznaje* jezik $L(\mathcal{M}) := \{w \in \Sigma^* : \mathcal{M} \text{ prihvaća } w\}$.

Svaki DKA je NKA (do na formalne detalje), no prirodno se nameće pitanje: koliko su NKA jači od DKA?

Ekvivalentnost NKA i DKA *(konstrukcija)*

Teorem: Za svaki NKA postoji ekvivalentan DKA.

Dokaz: Za NKA $\mathcal{N} := (Q, \Sigma, \Delta, q_0, F)$, konstruiramo DKA

Ekvivalentnost NKA i DKA *(partitivna konstrukcija)*

Teorem: Za svaki NKA postoji ekvivalentan DKA.

Dokaz: Za NKA $\mathcal{N} := (Q, \Sigma, \Delta, q_0, F)$, konstruiramo DKA

$$\mathcal{P}(\mathcal{N}) := \big(\mathcal{P}(Q), \Sigma, \Delta', [\{q_0\}]_\varepsilon, \{T \subseteq Q : T \cap F \neq \emptyset\}\big),$$

$$\text{gdje je } \Delta'(S, \alpha) := \Big[\bigcup\nolimits_{q \in S} \Delta(q, \alpha)\Big]_\varepsilon, \text{ te}$$

$$[S]_\varepsilon := \bigcup\nolimits_{q \in S} \{q' \in Q : \text{postoji staza } \varepsilon\text{-prijelaza od } q \text{ do } q'\},$$

Tada je $L(\mathcal{N}) = L(\mathcal{P}(\mathcal{N}))$. □

Definicija: Svaki jezik nekog DKA (ekvivalentno: NKA) zovemo *regularni jezik*. Klasu svih regularnih jezika označavamo s *Reg*.

Želimo dokazati da za svaki DKA postoji *desnolinearna gramatika* (DLG) s istim jezikom. Također, za svaki DLG postoji *regularan izraz* (RI) s istim jezikom, a za svaki RI postoji NKA s istim jezikom (DKA→DLG→RI→NKA). Zato su svi ti modeli izračunavanja *ekvivalentni*, odnosno opisuju istu klasu jezika (*Reg*).

Teorem: Neka je Σ abeceda. Unija dvaju regularnih jezika nad Σ je regularan jezik nad Σ.

Dokaz: Ako $\mathcal{M}_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ prepoznaje L_i, $i \in \{1, 2\}$, tada

Zatvorenost na skupovne operacije (produktna konstrukcija)

Teorem: Neka je Σ abeceda. Unija dvaju regularnih jezika nad Σ je regularan jezik nad Σ.

Dokaz: Ako $\mathcal{M}_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ prepoznaje L_i, $i \in \{1, 2\}$, tada

$$\mathcal{M}_1 \times \mathcal{M}_2 := \big(Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), Q_1 \times F_2 \cup F_1 \times Q_2\big),$$

$$\text{gdje je } \delta\big((q, q'), \alpha\big) := \big(\delta_1(q, \alpha), \delta_2(q', \alpha)\big),$$

prepoznaje $L_1 \cup L_2$.

Vrijedi li slična tvrdnja za više od dva skupa? Što je s presjecima?

Zadatak: Dokažite zatvorenost na komplemente (spram Σ^*).

Regularni izrazi

Konkatenaciju riječi $u = \alpha_1 \cdots \alpha_n$ i $v = \beta_1 \cdots \beta_m$ definiramo sa $uv := \alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_m$. Također konkateniramo jezike: $LM := \{uv : u \in L \wedge v \in M\}$. Potenciramo ovako: $w^2 = ww$, $L^3 = LLL$, itd.

Kleenejev plus jezika L definiramo kao $L^+ := \bigcup_{k \in \mathbb{N}_+} L^k$. *Kleenejeva zvijezda* je $L^* := \bigcup_{k \in \mathbb{N}_0} L^k$. Konkatenaciju, Kleenejev plus i zvijezdu zovemo zajedničkim imenom *jezične operacije*. Jezike $\emptyset$, ε te α za sve $\alpha \in \Sigma$ nazivamo *inicijalnim jezicima nad* Σ. Izraz dobiven u konačno mnogo koraka od inicijalnih jezika nad Σ pomoću unija i jezičnih operacija naziva se *regularni izraz nad* Σ.

Zadatak: Neka je R_Σ najmanja klasa jezika nad Σ koja istovremeno sadrži inicijalne jezike nad Σ i zatvorena je na uniju i jezične operacije. Dokažite da je R_Σ upravo klasa (je li čak skup?) svih jezika nad Σ reprezentiranih nekim regularnim izrazom nad Σ.

Želimo dokazati da je R_Σ upravo $Reg \cap \mathcal{P}(\Sigma^*)$. Što ćemo time dokazati za Reg? Hint:

Regularni izrazi

Konkatenaciju riječi $u = \alpha_1 \cdots \alpha_n$ i $v = \beta_1 \cdots \beta_m$ definiramo sa $uv := \alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_m$. Također konkateniramo jezike: $LM := \{uv : u \in L \wedge v \in M\}$. Potenciramo ovako: $w^2 = ww$, $L^3 = LLL$, itd.

Kleenejev plus jezika L definiramo kao $L^+ := \bigcup_{k \in \mathbb{N}_+} L^k$. *Kleenejeva zvijezda* je $L^* := \bigcup_{k \in \mathbb{N}_0} L^k$. Konkatenaciju, Kleenejev plus i zvijezdu zovemo zajedničkim imenom *jezične operacije*. Jezike $\emptyset$, ε te α za sve $\alpha \in \Sigma$ nazivamo *inicijalnim jezicima nad* Σ. Izraz dobiven u konačno mnogo koraka od inicijalnih jezika nad Σ pomoću unija i jezičnih operacija naziva se *regularni izraz nad* Σ.

Zadatak: Neka je R_Σ najmanja klasa jezika nad Σ koja istovremeno sadrži inicijalne jezike nad Σ i zatvorena je na uniju i jezične operacije. Dokažite da je R_Σ upravo klasa (je li čak skup?) svih jezika nad Σ reprezentiranih nekim regularnim izrazom nad Σ.

Želimo dokazati da je R_Σ upravo $Reg \cap \mathcal{P}(\Sigma^*)$. Što ćemo time dokazati za Reg? Hint: Zatvorenost na (nešto staro i) nešto novo.

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom?

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom?
Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde.
Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$?

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint:

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega?

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega? Strogo govoreći, jesmo li zaista pokazali neuspjeh predložene konstrukcije?

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega? Strogo govoreći, jesmo li zaista pokazali neuspjeh predložene konstrukcije? Ne! Naime, nismo navedeni dio konstrukcije razmotrili u kontekstu

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega? Strogo govoreći, jesmo li zaista pokazali neuspjeh predložene konstrukcije? Ne! Naime, nismo navedeni dio konstrukcije razmotrili u kontekstu čitave konstrukcije! Zato ćemo

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega? Strogo govoreći, jesmo li zaista pokazali neuspjeh predložene konstrukcije? Ne! Naime, nismo navedeni dio konstrukcije razmotrili u kontekstu čitave konstrukcije! Zato ćemo krenuti od *prema toj konstrukciji dobivenog* NKA za a^+b i tako uočiti da je konstrukcija pogrešna.

Kako biste popravili konstrukciju za zvijezdu?

RI$\rightarrow$NKA

Za dani RI nad Σ, kako konstruiramo NKA s istim jezikom? Konstruiramo NKA za inicijalne jezike nad Σ, pa slijedimo strukturu RI.

Sve se poklapa s intuicijom dok ne dođemo do Kleenejeve zvijezde. Ako imate NKA $\mathcal{N}$, kako ćete konstruirati NKA koji prepoznaje $L(\mathcal{N})^*$? Možda biste postupili kao za Kleenejev plus i učinili početno stanje završnim?

Hint: Kako izgleda najjednostavniji mogući NKA za jezik a^*b? Kako bi predložena konstrukcija djelovala na njega? Strogo govoreći, jesmo li zaista pokazali neuspjeh predložene konstrukcije? Ne! Naime, nismo navedeni dio konstrukcije razmotrili u kontekstu čitave konstrukcije! Zato ćemo krenuti od *prema toj konstrukciji dobivenog* NKA za a^+b i tako uočiti da je konstrukcija pogrešna.

Kako biste popravili konstrukciju za zvijezdu? Dodamo novo početno stanje (učinimo ga i završnim), ε-prijelaze iz novog u staro početno stanje te iz starih završnih prema starom početnom stanju.

Gramatike, DLG

Definicija: *Gramatika* nad abecedom Σ je $\mathcal{G} = (V, \Sigma, \to, S)$, koja uz Σ ima još konačan skup V (*varijable*) **disjunktan** sa Σ, istaknuti element $S \in V$ (*početna varijabla*) te konačnu binarnu relaciju na $(\Sigma \cup V)^*$, čije elemente zovemo *pravila*. Na lijevoj strani svakog pravila **mora** biti barem jedna varijabla.

Za riječi u i v nad *skupom simbola* $Y := \Sigma \cup V$ pišemo $u \Rightarrow v$, ako postoji pravilo $s \to t$ u $\mathcal{G}$, te riječi a i b nad Y takve da je $u = asb$ i $v = atb$.

Izvod u $\mathcal{G}$ za w je konačan niz riječi $u_0, u_1, \ldots, u_n$ nad Y, takav da je $u_0 = S$, $u_{i-1} \Rightarrow u_i$ za sve $i \in [1..n]$ te $u_n = w$. Pišemo $S \Rightarrow^* w$ i kažemo da $\mathcal{G}$ *izvodi* w.

Kažemo da $\mathcal{G}$ *generira* jezik $L(\mathcal{G}) := \{w \in \Sigma^* : \mathcal{G} \text{ izvodi } w\}$.

Definicija: Gramatika $\mathcal{G}$ je *desnolinearna* (DLG) ako je svako njeno pravilo jednog od sljedeća dva oblika: $A \to \alpha B$ ili $A \to \varepsilon$, gdje su A i B varijable, a α znak.

DKA→DLG

Neka je zadan DKA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$. Definiramo gramatiku $\mathcal{G} = (V, \Sigma, \to, S)$ na sljedeći način: za svaki $q \in Q$ stavimo $V_q \notin \Sigma$ u V, definiramo $S := V_{q_0}$, za sve $q \in Q$ i $\alpha \in \Sigma$ dodamo pravilo $V_q \to \alpha V_{\delta(q,\alpha)}$, za svako završno stanje $q \in F$ dodamo pravilo $V_q \to \varepsilon$.

Za $q \in Q$, označimo $\mathcal{M}_q = (Q, \Sigma, \delta, q, F)$ te $\mathcal{G}_q = (V, \Sigma, \to, V_q)$
Dokažimo da za sve $q \in Q$ vrijedi $L(\mathcal{M}_q) = L(\mathcal{G}_q)$ (je li to dosta?):

($\subseteq$) Neka je $w = \alpha_1 \dots \alpha_n \in L(\mathcal{M}_q)$. Indukcijom po n: za $n = 0$ je $w = \varepsilon$ te $q \in F$, pa je $V_q \to \varepsilon = w$ u $\mathcal{G}_q$ prema konstrukciji. Za korak, neka je $w_{2:} = \alpha_2 \dots \alpha_n$. Tada je $w_{2:} \in L(\mathcal{M}_{\delta(q,\alpha_1)})$, pa je iz $|w_{2:}| < |w|$ i pretpostavke indukcije $w_{2:} \in L(\mathcal{G}_{\delta(q,\alpha_1)})$. Ukupno je $V_q \to \alpha_1 V_{\delta(q,\alpha_1)} \Rightarrow^* \alpha_1 w_{2:} = w$, tj. $w \in L(\mathcal{G}_q)$.

($\supseteq$) Neka je $w \in L(\mathcal{G}_q)$ i $V_q \Rightarrow^m w$. Indukcijom po m: za $m = 1$ je $w = \varepsilon$ (inače w sadrži varijablu $\lightning$). No tada je $q \in F$, tj. $w \in L(\mathcal{M}_q)$. Za korak, $V_q \to \alpha_1 V_{\delta(q,\alpha_1)} \Rightarrow^{m-1} w = \alpha_1 w_{2:}$, pa je $w_{2:} \in L(\mathcal{G}_{\delta(q,\alpha_1)}) \overset{\pi}{\Rightarrow} w_{2:} \in L(\mathcal{M}_{\delta(q,\alpha_1)}) \Rightarrow w \in L(\mathcal{M}_q)$.

DLG→RI

Ideja je pravila gramatike pretvoriti u „sustav jednadžbi”: za svaku varijablu skupimo sva pravila koja je imaju na lijevoj strani:

$$A \to \alpha_1 B_1 \qquad A \to \alpha_2 B_2 \qquad \cdots \qquad (A \to \varepsilon)$$

te dodamo jednadžbu $A = \alpha_1 B_1 \cup \alpha_2 B_2 \cup \cdots (\cup\, \varepsilon)$.

Ako se varijabla A ne pojavljuje na lijevoj strani nijednog pravila, dodamo jednadžbu $A = \emptyset$. Supstitucijom rješavamo taj sustav, primjenjujući distributivnost konkatenacije prema uniji. Tražimo rješenje za S. No što kada nam se pojavi ista varijabla na obje strane jednadžbe?

Lema (Arden, o fiksnoj točki): Jedno rješenje jezične jednadžbe $X = AX \cup B$ je $X = A^*B$; štoviše, to je najmanje rješenje u smislu relacije $\subseteq$. Ako je A pozitivan, to je jedinstveno rješenje.

Dokaz: Najprije pokažimo da A^*B zadovoljava jednadžbu:

$$A(A^*B) \cup B = (AA^*)B \cup B = A^+B \cup \varepsilon B = (A^+ \cup \varepsilon)B = A^*B.$$

Nastavak dokaza Ardenove leme

Lema (Arden, o fiksnoj točki): Jedno rješenje jezične jednadžbe $X = AX \cup B$ je $X = A^*B$; štoviše, to je najmanje rješenje u smislu relacije $\subseteq$. Ako je A pozitivan, to je jedinstveno rješenje.

Dokaz: Za sva rješenja X je $X \supseteq A^*B$ (obrat vrijedi ako $\varepsilon \notin A$):

($\supseteq$) $w \in A^*B$ znači $w \in A^nB$ za neki $n \in \mathbb{N}_0$. Indukcijom po n dokažemo sljedeću tvrdnju (primijetite da smo tada gotovi): za svaki $n \in \mathbb{N}_0$, $A^nB \subseteq X$. Za bazu $n = 0$, $w \in A^0B = B \subseteq AX \cup B = X$. Za korak, ako je $w \in A^{n+1}B$, onda postoje w_1, w_2 takvi da je $w = w_1w_2$, $w_1 \in A$ te $w_2 \in A^nB \overset{\pi}{\subseteq} X$. Drugim riječima, $w \in AX \subseteq AX \cup B = X$.

($\subseteq$) Ako je $w \in B$, trivijalno je $w \in A^*B$. Inače, totalnom indukcijom po $|w|$ pokažemo da $w \in AX$ povlači $w \in A^*B$: neka je $w = w_1w_2$ gdje je $w_1 \in A$ i $w_2 \in X$ (primijetimo: A je pozitivan, pa je $|w_2| < |w|$). Ako je $w_2 \in B$, trivijalno je $w_2 \in A^*B$. Ako je $w_2 \in AX$, prema PI vrijedi $w_2 \in A^*B$. U svakom slučaju je $w_2 \in A^*B$, tj. $w \in AA^*B = A^+B \subseteq A^*B$.

Lijevolinearne gramatike

Kako bi izgledale lijevolinearne gramatike (LLG)? Koju klasu jezika one generiraju?

Lijevolinearne gramatike

Kako bi izgledale lijevolinearne gramatike (LLG)? Koju klasu jezika one generiraju? Prilično je jasno da se radi o Reg^{R} (klasi *reverza* svih regularnih jezika):

$$w = w_1 \cdots w_n$$

$$w^{\mathrm{R}} = w_n \cdots w_1$$

$$L^{\mathrm{R}} = \{w^{\mathrm{R}} : w \in L\}$$

$$\mathcal{C}^{\mathrm{R}} = \{L^{\mathrm{R}} : L \in \mathcal{C}\}$$

Koja je veza klase Reg^{R} s klasom Reg? Kako biste to dokazali? Hint:

Lijevolinearne gramatike

Kako bi izgledale lijevolinearne gramatike (LLG)? Koju klasu jezika one generiraju? Prilično je jasno da se radi o Reg^{R} (klasi *reverza* svih regularnih jezika):

$$w = w_1 \cdots w_n$$

$$w^{\mathrm{R}} = w_n \cdots w_1$$

$$L^{\mathrm{R}} = \{w^{\mathrm{R}} : w \in L\}$$

$$\mathcal{C}^{\mathrm{R}} = \{L^{\mathrm{R}} : L \in \mathcal{C}\}$$

Koja je veza klase Reg^{R} s klasom Reg? Kako biste to dokazali? Hint: Indukcijom po strukturi regularnog izraza.

Primijetite da smo sada dokazali da i sve LLG generiraju točno sve regularne jezike, odnosno imamo peti model izračunavanja koji „hvata" klasu Reg.

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q =$$

$$\delta =$$

$$q_0 =$$

$$F =$$

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q = \Sigma^*/\sim$$

$$\delta =$$

$$q_0 =$$

$$F =$$

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q = \Sigma^*/\sim$$
$$\delta = \{([w], \alpha, [w\alpha]) : w \in \Sigma^*, \alpha \in \Sigma\}$$
$$q_0 =$$
$$F =$$

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q = \Sigma^*/\sim$$
$$\delta = \{([w], \alpha, [w\alpha]) : w \in \Sigma^*, \alpha \in \Sigma\}$$
$$q_0 = [\varepsilon]$$
$$F =$$

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$\begin{aligned} Q &= \Sigma^*/\sim \\ \delta &= \{([w], \alpha, [w\alpha]) : w \in \Sigma^*, \alpha \in \Sigma\} \\ q_0 &= [\varepsilon] \\ F &= \{[w] : w \in L\} \end{aligned}$$

Zadatak: Dokažite da je $\mathcal{M}$ DKA (odnosno da je

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q = \Sigma^*/\sim$$
$$\delta = \{([w], \alpha, [w\alpha]) : w \in \Sigma^*, \alpha \in \Sigma\}$$
$$q_0 = [\varepsilon]$$
$$F = \{[w] : w \in L\}$$

Zadatak: Dokažite da je $\mathcal{M}$ DKA (odnosno da je δ dobro definirana relacija s funkcijskim svojstvom) te da je

Myhill-Nerodeov teorem

Možda će vam zvučati nevjerojatno, ali regularnost jezika možemo karakterizirati bez spominjanja strojeva, gramatika ili izraza.

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te $\sim$ binarna relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Leftarrow}$ Neka je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ gdje su:

$$Q = \Sigma^*/\sim$$
$$\delta = \{([w], \alpha, [w\alpha]) : w \in \Sigma^*, \alpha \in \Sigma\}$$
$$q_0 = [\varepsilon]$$
$$F = \{[w] : w \in L\}$$

Zadatak: Dokažite da je $\mathcal{M}$ DKA (odnosno da je δ dobro definirana relacija s funkcijskim svojstvom) te da je $L(\mathcal{M}) = L$.

Myhill-Nerodeov teorem

Teorem (Myhill-Nerode)

Neka je L jezik nad Σ te neka je $\sim$ relacija na Σ^* zadana sa:

$$x \sim y \iff \text{za sve } z \in \Sigma^*,\ xz, yz \in L \text{ ili } xz, yz \notin L.$$

Tada je L regularan ako i samo ako je skup $\Sigma^*/\sim$ konačan.

Dokaz.

$\boxed{\Rightarrow}$ Pretpostavimo da je L regularan (pa postoji $\mathcal{M}$ takav da je $L(\mathcal{M}) = L$) i $\Sigma^*/\sim$ beskonačan. Uzmimo po jednog reprezentanta iz svake klase (što iz teorije skupova nam to omogućuje?), neka su svi oni $S = \{w_0, w_1, \ldots\}$.

Kad bi čitanje w_i i w_j $(i \neq j)$ dovelo $\mathcal{M}$ u isto stanje, tada bi i $w_i w$ i $w_j w$ za sve w doveli $\mathcal{M}$ u isto stanje, kontradikcija sa $w_i \not\sim w_j$. Prema tome, svaka riječ iz S vodi $\mathcal{M}$ u drugo stanje (kako ovo preciznije izrazimo u teoriji skupova?), kontradikcija s konačnošću od $\mathcal{M}.Q$. □

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti?

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti? Riješimo se *nedostižívih stanja* (onih u koja nije moguće doći iz početnog stanja). Čini se krajnje netrivijalno uočiti daljnja suvišna stanja, barem općenito (*mrtva stanja*, ona iz kojih ne možemo ni do jednog završnog, ne mičemo kako bi nam funkcija prijelaza zaista bila funkcija).

Što ćemo sad?

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti? Riješimo se *nedostižih stanja* (onih u koja nije moguće doći iz početnog stanja). Čini se krajnje netrivijalno uočiti daljnja suvišna stanja, barem općenito (*mrtva stanja*, ona iz kojih ne možemo ni do jednog završnog, ne mičemo kako bi nam funkcija prijelaza zaista bila funkcija).

Što ćemo sad? Stanja koja odgovaraju istoj Myhill-Nerodeovoj klasi bismo mogli nazvati *nerazlučivima* i spojiti ih u jedno stanje. No kako ćemo ih detektirati?

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti? Riješimo se *nedostižih stanja* (onih u koja nije moguće doći iz početnog stanja). Čini se krajnje netrivijalno uočiti daljnja suvišna stanja, barem općenito (*mrtva stanja*, ona iz kojih ne možemo ni do jednog završnog, ne mičemo kako bi nam funkcija prijelaza zaista bila funkcija).

Što ćemo sad? Stanja koja odgovaraju istoj Myhill-Nerodeovoj klasi bismo mogli nazvati *nerazlučivima* i spojiti ih u jedno stanje. No kako ćemo ih detektirati?

Dinamičkim programiranjem: u tablici označimo parove stanja koje razlučuje ε (tj. točno jedno od njih je završno), pa

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti? Riješimo se *nedostižih stanja* (onih u koja nije moguće doći iz početnog stanja). Čini se krajnje netrivijalno uočiti daljnja suvišna stanja, barem općenito (*mrtva stanja*, ona iz kojih ne možemo ni do jednog završnog, ne mičemo kako bi nam funkcija prijelaza zaista bila funkcija).

Što ćemo sad? Stanja koja odgovaraju istoj Myhill-Nerodeovoj klasi bismo mogli nazvati *nerazlučivima* i spojiti ih u jedno stanje. No kako ćemo ih detektirati?

Dinamičkim programiranjem: u tablici označimo parove stanja koje razlučuje ε (tj. točno jedno od njih je završno), pa označimo sve parove stanja koja bilo koji $\alpha \in \Sigma$ vodi u par razlučivih stanja (već označen u tablici), sve dok

Prva važna primjena Myhill-Nerodeovog teorema

Kako možemo iskoristiti Myhill-Nerodeov teorem? Prvo, recimo da imamo DKA $\mathcal{M}$ koji prepoznaje jezik L. Možemo li nekako *smanjiti* $\mathcal{M}$, a da mu jezik ostane isti? Riješimo se *nedostižih stanja* (onih u koja nije moguće doći iz početnog stanja). Čini se krajnje netrivijalno uočiti daljnja suvišna stanja, barem općenito (*mrtva stanja*, ona iz kojih ne možemo ni do jednog završnog, ne mičemo kako bi nam funkcija prijelaza zaista bila funkcija).

Što ćemo sad? Stanja koja odgovaraju istoj Myhill-Nerodeovoj klasi bismo mogli nazvati *nerazlučivima* i spojiti ih u jedno stanje. No kako ćemo ih detektirati?

Dinamičkim programiranjem: u tablici označimo parove stanja koje razlučuje ε (tj. točno jedno od njih je završno), pa označimo sve parove stanja koja bilo koji $\alpha \in \Sigma$ vodi u par razlučivih stanja (već označen u tablici), sve dok više nije moguće označiti novi par razlučivih stanja. Kako ćemo odrediti početno stanje, završna stanja i prijelaze?

Postoje li jezici koji nisu regularni?

Odgovor je

Postoje li jezici koji nisu regularni?

Odgovor je **da**, recimo jezik

Postoje li jezici koji nisu regularni?

Odgovor je **da**, recimo jezik $L_{=} = \{a^n b^n : n \in \mathbb{N}_+\}$. Zašto? Intuitivno, jer

Postoje li jezici koji nisu regularni?

Odgovor je **da**, recimo jezik $L_{=} = \{a^n b^n : n \in \mathbb{N}_+\}$. Zašto? Intuitivno, jer kad DKA krene čitati b-ove, mora znati koliko je a-ova pročitao, a mogućnosti ima beskonačno. Postoji li dovoljno specifično svojstvo regularnih jezika koje taj jezik ne zadovoljava?

Sjetimo se onog našeg DKA s osam stanja, označimo ga $\mathcal{M}$. Ako mu kao ulaz damo riječ s barem 8 znakova, pri čitanju prvih 8 znakova će se barem jedno stanje od $\mathcal{M}$

Postoje li jezici koji nisu regularni?

Odgovor je **da**, recimo jezik $L_= = \{a^n b^n : n \in \mathbb{N}_+\}$. Zašto? Intuitivno, jer kad DKA krene čitati b-ove, mora znati koliko je a-ova pročitao, a mogućnosti ima beskonačno. Postoji li dovoljno specifično svojstvo regularnih jezika koje taj jezik ne zadovoljava?

Sjetimo se onog našeg DKA s osam stanja, označimo ga $\mathcal{M}$. Ako mu kao ulaz damo riječ s barem 8 znakova, pri čitanju prvih 8 znakova će se barem jedno stanje od $\mathcal{M}$ **ponoviti**.

Drugim riječima, ako je $w = xyz$ i y dio koji vodi od prve do druge pojave ponovljenog stanja (možda tih pojava ima i više), znamo da je $y \neq \varepsilon$, $|xy| \leq 8$ i sve „varijante" $w^{(i)} := xy^i z$, za $i \in \mathbb{N}_0$, su također u jeziku L.

Teorem (Lema o pumpanju za regularne jezike):
Za svaki regularan jezik L postoji $p \in \mathbb{N}$ takav da se svaka riječ $w \in L$ duljine barem p može zapisati kao $w = xyz$, gdje je (1) $y \neq \varepsilon$, (2) $|xy| \leq p$, te su (3) za sve $i \in \mathbb{N}_0$, $w^{(i)} := xy^i z \in L$.

Zadatak: Precizno dokažite teorem i primijenite ga na $L_=$.

Druga važna primjena Myhill-Nerodeovog teorema

Predmet naslova je sad postao sasvim očit: i taj teorem možemo koristiti kako bismo dokazali da neki jezik

Druga važna primjena Myhill-Nerodeovog teorema

Predmet naslova je sad postao sasvim očit: i taj teorem možemo koristiti kako bismo dokazali da neki jezik nije regularan!

Kako biste pokazali da $L_=$ nije regularan koristeći M-N teorem?

Druga važna primjena Myhill-Nerodeovog teorema

Predmet naslova je sad postao sasvim očit: i taj teorem možemo koristiti kako bismo dokazali da neki jezik nije regularan!

Kako biste pokazali da $L_=$ nije regularan koristeći M-N teorem?

Pa, treba nam beskonačno mnogo M-N klasa, odnosno po jedan predstavnik iz svake. Prilično je jasno da će dobar odabir biti

Druga važna primjena Myhill-Nerodeovog teorema

Predmet naslova je sad postao sasvim očit: i taj teorem možemo koristiti kako bismo dokazali da neki jezik nije regularan!

Kako biste pokazali da $L_=$ nije regularan koristeći M-N teorem?

Pa, treba nam beskonačno mnogo M-N klasa, odnosno po jedan predstavnik iz svake. Prilično je jasno da će dobar odabir biti

$$\{a^1, a^2, a^3, a^4, a^5, \ldots\}$$

Naime, pogledajmo a^3 i a^5. Zašto su te dvije riječi u različitim M-N klasama?

Druga važna primjena Myhill-Nerodeovog teorema

Predmet naslova je sad postao sasvim očit: i taj teorem možemo koristiti kako bismo dokazali da neki jezik nije regularan!

Kako biste pokazali da $L_=$ nije regularan koristeći M-N teorem?

Pa, treba nam beskonačno mnogo M-N klasa, odnosno po jedan predstavnik iz svake. Prilično je jasno da će dobar odabir biti

$$\{a^1, a^2, a^3, a^4, a^5, \ldots\}$$

Naime, pogledajmo a^3 i a^5. Zašto su te dvije riječi u različitim M-N klasama? Zato što ih recimo a^3 razlučuje (ima li još takvih sufiksa?). Vidimo dakle da se stvarno radi o predstavnicima različitih klasa, odnosno da M-N klasa ima beskonačno, pa $L_=$ nije regularan.

Postoji li i treći način za dokazati neregularnost jezika?

Postoji li i treći način za dokazati neregularnost jezika?

Da! Pomoću zatvorenosti klase *Reg* na skupovne i jezične operacije.

Primjerice, htjeli bismo dokazati da jezik

$$L = \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$$

nad $\Sigma = \{a, b\}$ nije regularan. Što bismo mogli napraviti?

Postoji li i treći način za dokazati neregularnost jezika?

Da! Pomoću zatvorenosti klase *Reg* na skupovne i jezične operacije.

Primjerice, htjeli bismo dokazati da jezik

$$L = \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$$

nad $\Sigma = \{a, b\}$ nije regularan. Što bismo mogli napraviti?

Znamo da je jezik a^*b^* regularan (zašto?). Kad bi i jezik L bio regularan, tada bi prema zatvorenosti klase *Reg* na

Postoji li i treći način za dokazati neregularnost jezika?

Da! Pomoću zatvorenosti klase *Reg* na skupovne i jezične operacije.

Primjerice, htjeli bismo dokazati da jezik

$$L = \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$$

nad $\Sigma = \{a, b\}$ nije regularan. Što bismo mogli napraviti?

Znamo da je jezik a^*b^* regularan (zašto?). Kad bi i jezik L bio regularan, tada bi prema zatvorenosti klase *Reg* na presjeke bio regularan i jezik

Postoji li i treći način za dokazati neregularnost jezika?

Da! Pomoću zatvorenosti klase *Reg* na skupovne i jezične operacije.

Primjerice, htjeli bismo dokazati da jezik

$$L = \{w \in \{a, b\}^+ : |w|_a = |w|_b\}$$

nad $\Sigma = \{a, b\}$ nije regularan. Što bismo mogli napraviti?

Znamo da je jezik a^*b^* regularan (zašto?). Kad bi i jezik L bio regularan, tada bi prema zatvorenosti klase *Reg* na presjeke bio regularan i jezik $L_=$ ↯.

Jesmo li na ovaj način mogli dokazati neregularnost od L prije nego smo dokazali lemu o pumpanju za regularne jezike, odnosno M-N teorem? Zašto?

Postoji li i četvrti način za dokazati neregularnost jezika?

Postoji li i četvrti način za dokazati neregularnost jezika?

Da, pomoću *Kolmogorovljeve složenosti*, odnosno duljine $K(x)$ najkraćeg programa koji vrati/ispiše riječ $x \in \{0,1\}^*$. Zasad zamišljajte da programe pišemo u vašem najdražem (i fiksiranom) Turing-potpunom programskom jeziku te da im duljinu računamo u bitovima. Ako je $K(x) \geq |x|$, kažemo da je x *nesažimljiva*.

Pretpostavimo da je $\mathcal{M}$ DKA takav da je $L(\mathcal{M}) = L_=$. Za svaki $n \in \mathbb{N}$, postoji $q_n \in \mathcal{M}.Q$ u koji $\mathcal{M}$ dođe kad pročita a^n. Tada je $|\langle \mathcal{M} \rangle|$ (duljina binarnog zapisa, *koda*, od $\mathcal{M}$) konstantna (u n), a $|\langle q_n \rangle|$ je omeđena konstantom. Lijevi program ispiše $(n)_2$, pa je:

```
M, q, cnt = ⟨M⟩, ⟨q_n⟩, 0
while q not in final(M):
        q = delta(M, q, b)
        cnt += 1
print((cnt)_2)
```

$$K((n)_2) \leq |\langle \mathcal{M} \rangle| + |\langle q_n \rangle| + O(1) \leq c$$

↯ (za n takav da je $(n)_2$ duljine $c+1$ te nesažimljiva).

O formalnim detaljima ćemo kasnije (tada ćete umjesto *programa* znati oblikovati *Turingov stroj*, *kod* će umjesto binarnog zapisa značiti *zapis prirodnog stroja/stanja u univerzalnoj abecedi*, itd.).

Beskontekstni jezici

Šira klasa jezika od regularnih koju ćemo sad početi proučavati je klasa BK svih *beskontekstnih jezika*.

Beskontekstno pravilo je pravilo oblika $A \to w$, gdje je $A \in V$ varijabla, a $w \in Y^*$ niz simbola. Drugim riječima, zamjenjujemo jedan simbol s nula, jednim ili više njih, *bez* obzira na *kontekst* (znakove oko) varijable A.

Beskontekstna gramatika (BKG) je gramatika čija su sva pravila beskontekstna. Jezik je beskontekstan ako ga generira neka BKG.

Svaka DLG je BKG: pravila dopuštena u DLG, $A \to \alpha B$ i $A \to \varepsilon$, očito su beskontekstna. Dakle svaki regularni jezik je beskontekstan. $L_= \in BK \setminus Reg$ pokazuje da je inkluzija prava.

Kako glasi BKG koja generira jezik $L_=$? **Zadatak:** Dokažite svoju tvrdnju indukcijom.

Zatvorenost klase BK na uniju i jezične operacije

Za BKG $\mathcal{G}_i = (V_i, \Sigma, \to_i, S_i)$, $i \in \{1, 2\}$, (BSOMP $V_1 \cap V_2 = \emptyset$)

$$\mathcal{G}_1\mathcal{G}_2 := \big(V_1 \cup V_2 \cup \{S\}, \Sigma, (\to_1) \cup (\to_2) \cup \{S \to S_1S_2\}, S\big)$$

(BSOMP $S \notin V_1 \cup V_2 \cup \Sigma$) je BKG koja generira $L(\mathcal{G}_1)L(\mathcal{G}_2)$.
Dakle, BK je zatvorena na konkatenaciju.

Zamjenom novododanog pravila sa $S \to S_1 \mid S_2$, dobijemo zatvorenost klase BK na uniju.

Zadatak: Dokažite zatvorenost na Kleenejeve operatore.

Zatvorenost na ostale skupovne operacije (npr. presjek) ne vrijedi općenito, što ćemo dokazati. Ipak, vrijedi *konusno svojstvo*: presjek beskontekstnog i regularnog jezika je beskontekstan.
To se dokazuje produktnom konstrukcijom, za koju nam treba pojam automata koji prepoznaje beskontekstni jezik.

Potisni automati (PA)

Ugrubo, PA će biti NKA sa *stogom*.

Definicija: *Potisni automat* je automat oblika $(Q, \Sigma, \Gamma, \Delta, q_0, F)$ gdje je značenje pojedinih komponenti isto kao kod NKA, samo još ima *abecedu stoga*, koju relacija prijelaza Δ uzima u obzir.

Formalno, $\Delta \subseteq Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \times Q \times \Gamma_\varepsilon$, i $(q, \alpha, t, p, s) \in \Delta$ znači:

„ako automat u stanju q pročita znak α (ili je $\alpha = \varepsilon$), i na vrhu stoga mu je t (ili je $t = \varepsilon$), tada može prijeći u stanje p, skinuti (ako $t \neq \varepsilon$) t sa stoga, i staviti (ako $s \neq \varepsilon$) s na stog".

Izborom odgovarajućih parametara PA može stavljati znakove na stog ($t = \varepsilon \neq s$), skidati znakove sa stoga ($t \neq \varepsilon = s$), provjeravati koji je znak na vrhu stoga ($t = s \neq \varepsilon$) itd. Neovisno o tome, može čitati ($\alpha \neq \varepsilon$) ili ne čitati ($\alpha = \varepsilon$) znakove ulaza.

BKG $\rightarrow$ PA

Želimo dokazati da ako postoji beskontekstna gramatika $\mathcal{G} = (V, \Sigma, \rightarrow, S)$ koja generira jezik L, tada postoji i PA koji ga prepoznaje. Ideja nam je simulirati gramatiku na stogu onako kako bismo to radili olovkom na papiru:

BKG $\rightarrow$ PA

Želimo dokazati da ako postoji beskontekstna gramatika $\mathcal{G} = (V, \Sigma, \rightarrow, S)$ koja generira jezik L, tada postoji i PA koji ga prepoznaje. Ideja nam je simulirati gramatiku na stogu onako kako bismo to radili olovkom na papiru:

push S
while True:
    $\gamma :=$ pop; switch γ:
        case $\gamma \in V$:

BKG $\rightarrow$ PA

Želimo dokazati da ako postoji beskontekstna gramatika $\mathcal{G} = (V, \Sigma, \rightarrow, S)$ koja generira jezik L, tada postoji i PA koji ga prepoznaje. Ideja nam je simulirati gramatiku na stogu onako kako bismo to radili olovkom na papiru:

push S
while True:
    $\gamma :=$ pop; switch γ:
        case $\gamma \in V$:
            nedeterministički odaberi pravilo $\gamma \rightarrow w$
            for α in w^{R}: push α
        case $\gamma \in \Sigma$:

BKG $\rightarrow$ PA

Želimo dokazati da ako postoji beskontekstna gramatika $\mathcal{G} = (V, \Sigma, \rightarrow, S)$ koja generira jezik L, tada postoji i PA koji ga prepoznaje. Ideja nam je simulirati gramatiku na stogu onako kako bismo to radili olovkom na papiru:

push S
while True:
    $\gamma :=$ pop; switch γ:
        case $\gamma \in V$:
            nedeterministički odaberi pravilo $\gamma \rightarrow w$
            for α in w^{R}: push α
        case $\gamma \in \Sigma$:
            $\alpha :=$ pročitaj sljedeći znak
            if $\alpha \neq \gamma$: odbaci ovu granu (kako ovo formalizirati?)

BKG $\rightarrow$ PA

Želimo dokazati da ako postoji beskontekstna gramatika $\mathcal{G} = (V, \Sigma, \rightarrow, S)$ koja generira jezik L, tada postoji i PA koji ga prepoznaje. Ideja nam je simulirati gramatiku na stogu onako kako bismo to radili olovkom na papiru:

push \$
push S
while True:
    $\gamma :=$ pop; switch γ:
        case $\gamma \in V$:
            nedeterministički odaberi pravilo $\gamma \rightarrow w$
            for α in w^R: push α
        case $\gamma \in \Sigma$:
            $\alpha :=$ pročitaj sljedeći znak
            if $\alpha \neq \gamma$: odbaci ovu granu (kako ovo formalizirati?)
        case $\gamma =$ \$:
            uđi u „slijepo" završno stanje

JPA $\rightarrow$ BKG (uvod)

Bit će nam lakše isprogramirati gramatiku prema stroju ako stroj ima jedinstvenu završnu *konfiguraciju* (za PA je konfiguracija

JPA $\rightarrow$ BKG (uvod)

Bit će nam lakše isprogramirati gramatiku prema stroju ako stroj ima jedinstvenu završnu *konfiguraciju* (za PA je konfiguracija uređena trojka iz $Q \times \Sigma^* \times \Gamma^*$, tj. trenutno stanje, ulaz i stog).

Jednostavni potisni automat (JPA) ima tri dodatna svojstva (kako ćemo ih osigurati za dani PA, a da pripadni jezik ostane isti?):

- ima samo jedno završno stanje: $F = \{q_\checkmark\}$;
- ako je automat u stanju $q_\checkmark$, stog mu je prazan;
- svaki prijelaz je (isključivo!) ili *push* ili *pop* ($|st| = 1$).

U nastavku želimo konstruirati ekvivalentnu gramatiku koja ima po jednu varijablu ${}_pV_q$ za svaki par stanja (p, q) našeg JPA.

Cilj nam je postići da ${}_pV_q \Rightarrow^* w \in \Sigma^*$ bude ekvivalentno sa „JPA iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w". Iz toga odmah slijedi da je početna varijabla od $\mathcal{G}$ upravo

JPA $\rightarrow$ BKG (uvod)

Bit će nam lakše isprogramirati gramatiku prema stroju ako stroj ima jedinstvenu završnu *konfiguraciju* (za PA je konfiguracija uređena trojka iz $Q \times \Sigma^* \times \Gamma^*$, tj. trenutno stanje, ulaz i stog).

Jednostavni potisni automat (JPA) ima tri dodatna svojstva (kako ćemo ih osigurati za dani PA, a da pripadni jezik ostane isti?):

- ima samo jedno završno stanje: $F = \{q_\checkmark\}$;
- ako je automat u stanju $q_\checkmark$, stog mu je prazan;
- svaki prijelaz je (isključivo!) ili *push* ili *pop* ($|st| = 1$).

U nastavku želimo konstruirati ekvivalentnu gramatiku koja ima po jednu varijablu ${}_pV_q$ za svaki par stanja (p, q) našeg JPA.

Cilj nam je postići da ${}_pV_q \Rightarrow^* w \in \Sigma^*$ bude ekvivalentno sa „JPA iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w". Iz toga odmah slijedi da je početna varijabla od $\mathcal{G}$ upravo ${}_{q_0}V_{q_\checkmark}$. No što su pravila?

JPA $\rightarrow$ BKG (plan)

Još jednom, cilj nam je postići da ${}_pV_q \Rightarrow^* w$ bude ekvivalentno sa „JPA iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w".

Prolazit ćemo kroz sva izračunavanja JPA i dodavati (stavljati) pravila u gramatiku. Drugim riječima, konstrukcijom ćemo osigurati $\Leftarrow$. Čak štoviše, strpat ćemo značajno više pravila u gramatiku nego što ih opravdamo radom stroja. Zašto?

JPA $\rightarrow$ BKG (plan)

Još jednom, cilj nam je postići da ${}_pV_q \Rightarrow^* w$ bude ekvivalentno sa „JPA iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w".

Prolazit ćemo kroz sva izračunavanja JPA i dodavati (stavljati) pravila u gramatiku. Drugim riječima, konstrukcijom ćemo osigurati $\Leftarrow$. Čak štoviše, strpat ćemo značajno više pravila u gramatiku nego što ih opravdamo radom stroja. Zašto?

Zbog onog „kroz sva izračunavanja"! Naime, u nekom (puno) kasnijem predavanju, bit će nam izuzetno bitno da ova pretvorba ima polinomnu složenost u veličini automata (primjerice, dozvoljavamo ugniježđene petlje koje idu po Q, Γ, Σ_ϵ i Δ). Kad bismo umjesto toga išli po svim izračunavanja našeg JPA, to bi trajalo eksponencijalno u njegovoj veličini.

Koji bi problem mogao nastati s tim dodatnim pravilima?

JPA $\rightarrow$ BKG (plan)

Još jednom, cilj nam je postići da ${}_pV_q \Rightarrow^* w$ bude ekvivalentno sa „JPA iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w".

Prolazit ćemo kroz sva izračunavanja JPA i dodavati (stavljati) pravila u gramatiku. Drugim riječima, konstrukcijom ćemo osigurati $\Leftarrow$. Čak štoviše, strpat ćemo značajno više pravila u gramatiku nego što ih opravdamo radom stroja. Zašto?

Zbog onog „kroz sva izračunavanja"! Naime, u nekom (puno) kasnijem predavanju, bit će nam izuzetno bitno da ova pretvorba ima polinomnu složenost u veličini automata (primjerice, dozvoljavamo ugniježđene petlje koje idu po Q, Γ, Σ_ϵ i Δ). Kad bismo umjesto toga išli po svim izračunavanja našeg JPA, to bi trajalo eksponencijalno u njegovoj veličini.

Koji bi problem mogao nastati s tim dodatnim pravilima? Naravno da nećemo pokvariti $\Leftarrow$, ali bismo mogli ugroziti $\Rightarrow$. Zato ćemo na kraju dokazati da se to nije dogodilo.

JPA $\rightarrow$ BKG (konstrukcija)

Primijetimo najprije da svaki JPA može u bilo kojoj konfiguraciji čitati ε i ostati u istoj konfiguraciji. Zato za sve $p \in Q$ dodajemo pravilo ${}_pV_p \rightarrow \varepsilon$.

Inače, JPA radi prijelaze, i svi oni su ili push ili pop. Prvi mora biti

JPA $\rightarrow$ BKG (konstrukcija)

Primijetimo najprije da svaki JPA može u bilo kojoj konfiguraciji čitati ε i ostati u istoj konfiguraciji. Zato za sve $p \in Q$ dodajemo pravilo ${}_pV_p \rightarrow \varepsilon$.

Inače, JPA radi prijelaze, i svi oni su ili push ili pop. Prvi mora biti push (označimo gurnuti simbol s γ), a zadnji

JPA $\rightarrow$ BKG (konstrukcija)

Primijetimo najprije da svaki JPA može u bilo kojoj konfiguraciji čitati ε i ostati u istoj konfiguraciji. Zato za sve $p \in Q$ dodajemo pravilo ${}_pV_p \rightarrow \varepsilon$.

Inače, JPA radi prijelaze, i svi oni su ili push ili pop. Prvi mora biti push (označimo gurnuti simbol s γ), a zadnji pop jer je na početku i kraju stog

JPA $\rightarrow$ BKG (konstrukcija)

Primijetimo najprije da svaki JPA može u bilo kojoj konfiguraciji čitati ε i ostati u istoj konfiguraciji. Zato za sve $p \in Q$ dodajemo pravilo ${}_pV_p \rightarrow \varepsilon$.

Inače, JPA radi prijelaze, i svi oni su ili push ili pop. Prvi mora biti push (označimo gurnuti simbol s γ), a zadnji pop jer je na početku i kraju stog prazan. Pitanje je hoće li zadnji pop skinuti *taj isti* γ (ne neki drugi γ!).

Ako da, npr. γ je stavljen na stog dok JPA čita $\alpha \in \Sigma_\varepsilon$ i ide iz stanja p u r (pomoću $(r, \gamma) \in \Delta(p, \alpha, \varepsilon)$) te maknut pri čitanju $\beta \in \Sigma_\varepsilon$ dok JPA ide iz s u q (pomoću $(q, \varepsilon) \in \Delta(s, \beta, \gamma)$), dodamo pravilo ${}_pV_q \rightarrow \alpha {}_rV_s \beta$. Štoviše, dodamo ga za sve $p, q, r, s \in Q$, $\alpha, \beta \in \Sigma_\varepsilon$ te $\gamma \in \Gamma$ koji zadovoljavaju navedene uvjete na Δ.

Ako ne, JPA je ranije ušao u neko stanje r u kojem je stog prazan, pa dodamo pravilo ${}_pV_q \rightarrow {}_pV_r {}_rV_q$. Štoviše, dodamo ga za sve $p, q, r \in Q$.

JPA $\rightarrow$ BKG (dokaz ispravnosti konstrukcije)

Tvrdnja: ${}_pV_q \Rightarrow^* w$ akko JPA čitajući w može prijeći iz stanja p s praznim stogom u stanje q s praznim stogom.

Dokaz (indukcijom): ($\Leftarrow$) Po broju n koraka izračunavanja. Za $n = 0$, prema konstrukciji postoji pravilo ${}_pV_p \rightarrow \varepsilon$. Za $n = k + 1$ i granu u kojoj JPA u stanju r isprazni stog prije nego dođe u q s praznim stogom, prema konstrukciji postoji pravilo ${}_pV_q \rightarrow {}_pV_r{}_rV_q$ takvo da je $w = w_1w_2$, do r je pročitan w_1, a dalje w_2. Prema PI, ${}_pV_q \Rightarrow {}_pV_r{}_rV_q \Rightarrow^* w_1w_2 = w$. Za granu gdje se stog ne isprazni prije dolaska do q, prema konstrukciji postoji pravilo ${}_pV_q \rightarrow \alpha_rV_s\beta$, $w = \alpha w_1\beta$ i (prema PI) vrijedi ${}_pV_q \Rightarrow \alpha_rV_s\beta \Rightarrow^* \alpha w_1\beta = w$.

($\Rightarrow$) Po duljini n izvoda. Za $n = 1$, koristi se jedno pravilo koje desno nema varijabli, dakle ${}_pV_p \rightarrow \varepsilon$; jasno, ε vodi JPA iz p s praznim stogom u p s praznim stogom. Za $n = k + 1$, ako je prvo pravilo ${}_pV_q \rightarrow \alpha_rV_s\beta$, onda je $w = \alpha w_1\beta$ te vrijedi ${}_rV_s \Rightarrow^* w_1$ (u najviše k koraka), pa prema PI, JPA vodi iz p s praznim stogom u r u s u q s praznim stogom čitajući w (slično za ${}_pV_q \rightarrow {}_pV_r{}_rV_q$).

Konusno svojstvo za beskontekstne jezike

Sada kada imamo ekvivalenciju PA i BKG, možemo lako dokazati: ako PA $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, p_0, E)$ prepoznaje beskontekstni jezik B, te DKA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ prepoznaje regularni jezik R, tada PA

prepoznaje jezik $B \cap R$, koji je dakle beskontekstan.

Konusno svojstvo za beskontekstne jezike

Sada kada imamo ekvivalenciju PA i BKG, možemo lako dokazati: ako PA $\mathcal{P} = (P, \Sigma, \Gamma, \Delta, p_0, E)$ prepoznaje beskontekstni jezik B, te DKA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ prepoznaje regularni jezik R, tada PA

$$\mathcal{P} \times \mathcal{M} := \big(P \times Q, \Sigma, \Gamma, \Delta', (p_0, q_0), E \times F\big)$$

$$\Delta' := \big\{ \big((p, q), \alpha, t, (p', \underbrace{\delta(q, \alpha)}_{\delta(q,\varepsilon) := q}), s\big) : (p, \alpha, t, p', s) \in \Delta, q \in Q \big\}$$

prepoznaje jezik $B \cap R$, koji je dakle beskontekstan.

No presjek dva beskontekstna jezika ne mora biti beskontekstan! Da bismo to dokazali, trebamo prvo neki alat koji nam omogućuje dokazati da jezik *nije* beskontekstan.

Lema o pumpanju za beskontekstne jezike

Teorem: Za svaki beskontekstni jezik L postoji $p \in \mathbb{N}$ takav da se svaka riječ $w \in L$ duljine barem p može zapisati kao $w = uvxyz$, gdje je (1) $vy \neq \varepsilon$, (2) $|vxy| \leq p$, te su (3) sve „varijante" $w^{(i)} := uv^i xy^i z$, za $i \in \mathbb{N}_0$, također u jeziku L.

Dokaz: Ako $\mathcal{G}$ generira L i b je najveći broj simbola s desne strane nekog pravila u $\mathcal{G}$, stablo parsiranja visine h ima najviše b^h listova. Zato je za $p := b^{|V|+1}$ svako stablo parsiranja za w visine barem $|V| + 1$ (uzmimo ono s najmanje čvorova), pa ima granu duljine barem $|V| + 1$. Unutar zadnjih $|V| + 1$ varijabli te grane se neka varijabla, npr. R, ponavlja: $S \Rightarrow^* uRz \Rightarrow^* uvRyz \Rightarrow^* uvxyz = w$.

(2) $R \Rightarrow^* vxy$ daje stablo visine najviše $|V| + 1$, pa je $|vxy| \leq p$.

(3) Iz $R \Rightarrow^* x$ i $R \Rightarrow^* vRy$ slijedi $S \Rightarrow^* uRz \Rightarrow^* uxz = w^{(0)}$ i $S \Rightarrow^* uRz \Rightarrow^* uvRyz \Rightarrow^* uvvRyyz \Rightarrow^* \cdots \Rightarrow^* uv^i Ry^i z = w^{(i)}$.

(1) Kad bi bilo $vy = \varepsilon$, gornji izvod za $w^{(0)} = w$ bi dao stablo za w s manje čvorova, kontradikcija. [Dokaz vrijedi za $b \geq 2$. A inače?]

Nezatvorenost klase BK na razne skupovne operacije

Zadatak: Dokažite pomoću leme o pumpanju za beskontekstne jezike da sljedeći jezik nije beskontekstan:

$$L_{\equiv} = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \mathbb{N}_+\}.$$

Primijetimo da jezik $L_{\mathtt{a}=\mathtt{b}} := \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^k : n, k \in \mathbb{N}_+\} = L_{=}\mathtt{c}^+$ jest beskontekstan, kao konkatenacija dva beskontekstna jezika (desni je čak regularan). Analogno je $L_{\mathtt{b}=\mathtt{c}} := \{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^k : n, k \in \mathbb{N}_+\}$ beskontekstan. No $L_{\mathtt{a}=\mathtt{b}} \cap L_{\mathtt{b}=\mathtt{c}} = L_{\equiv}$ nije beskontekstan.

Dakle klasa BK nije zatvorena na presjek. Iz de Morganovog pravila sada slijedi da nije zatvorena ni na komplement (dokažite!), pa ni na skupovnu razliku (komplement je specijalni slučaj skupovne razlike, a svaki Σ^* je beskontekstan jer je regularan).

Zadatak: Je li BK zatvorena na simetričnu skupovnu razliku $\triangle$?

Chomskyjeva normalna forma

Definicija: Za gramatiku $\mathcal{G} = (V, \Sigma, \to, S)$ kažemo da je u Chomskyjevoj normalnoj formi (kratko: da je *Chomskyjeva*) ako je svako njeno pravilo jednog od tri oblika:

- $A \to BC$ (širenje), gdje je $\{A, B, C\} \subseteq V$, te $S \notin \{B, C\}$;
- $A \to \alpha$ (čitanje), gdje je $A \in V$ varijabla, a $\alpha \in \Sigma$ znak;
- $S \to \varepsilon$.

Očito, pravilo $S \to \varepsilon$ je u $\mathcal{G}$ ako i samo ako je $\varepsilon \in L(\mathcal{G})$. Također, sva ova pravila su beskontekstna, pa su sve Chomskyjeve gramatike beskontekstne. S druge strane, svaka se BKG može pretvoriti u ekvivalentnu Chomskyjevu gramatiku, pretvorbom u 5 faza.

Pretvorba u ChNF, faze START, TERM, BIN

U prvoj fazi, dodamo novu varijablu S' te novo pravilo $S' \to S$, te proglasimo S' novom početnom varijablom.

U drugoj fazi, svakom znaku $\alpha \in \Sigma$ koji se pojavljuje na desnoj strani nekog pravila ali **ne sam** (dakle ne u pravilu čitanja), dodijelimo novu varijablu N_α, te u gramatiku dodamo pravilo čitanja $N_\alpha \to \alpha$. Zatim sve takve pojave od α zamijenimo s N_α.

U trećoj fazi, za svako pravilo čija desna strana ima $n \geq 3$ simbola (nakon 2. faze oni moraju biti varijable), oblika $A \to V_1 V_2 \cdots V_n$, dodamo nove varijable $A_1, A_2, \ldots, A_{n-2}$, i to pravilo zamijenimo s $n-1$ „ulančanih" pravila širenja

$$A \to V_1 A_1,\ A_1 \to V_2 A_2,\ A_2 \to V_3 A_3,\ \ldots,\ A_{n-2} \to V_{n-1} V_n\,.$$

Pretvorba BKG $\to$ ChNF, faza DEL

U četvrtoj fazi, prvo nađemo sve *nepozitivne varijable*:

$$np := \{A \in V : A \Rightarrow^* \varepsilon\}.$$

$np := \emptyset$
while (np se promijenio u odnosu na prethodnu iteraciju):
    for $A \to T_1 T_2 \cdots T_n$ **in** (pravila gramatike $\mathcal{G}$):
        if $\{T_1, T_2, \ldots, T_n\} \subseteq np$: $np := np \cup \{A\}$

Sada svako pravilo u $\mathcal{G}$, na čijoj desnoj strani postoji $k > 0$ pojava varijabli iz np, zamijenimo s 2^k pravila, po jedno za svaki podskup skupa tih pojava, u kojem se svaka pojava iz tog podskupa briše.

Napomena: jer smo prethodno proveli fazu BIN, jedino je moguće $k = 1$ ili $k = 2$, pa se broj pravila poveća najviše s faktorom 4.

Na kraju obrišemo sva pravila oblika $A \to \varepsilon$ gdje $A \neq S'$.

Pretvorba BKG $\rightarrow$ ChNF, faza UNIT

U posljednjoj fazi, rješavamo se pravila oblika $A \rightarrow B$, koja na desnoj strani imaju samo jednu varijablu. Svako takvo pravilo maknemo, zapamtimo ga u nekom skupu, te za svako pravilo oblika $B \rightarrow u$ (gdje je $u \in Y^*$) dodamo pravilo $A \rightarrow u$, osim ako se ono već nalazi u skupu maknutih pravila.

Može se dokazati da u ovom poretku nijedna faza ne smeta prethodnima, odnosno osigurava neko svojstvo ChNF ne kvareći već postignuta svojstva. Također se može dokazati da sve faze čuvaju jezik generiran gramatikom.

Deterministički potisni automati

Rekli smo da deterministički beskontekstni jezici (klasu označimo DBK) čine tek djelić svih beskontekstnih jezika. Postavlja se pitanje: kako definirati deterministički PA (DPA)?

Prirodno je zahtijevati da u svakom koraku svog izračunavanja, DPA može nastaviti na najviše jedan način.

Prema tome, jedina razlika u definiciji DPA od definicije PA bit će u relaciji prijelaza koja je ovdje *funkcija* δ:

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$$

te zahtjevu da je točno jedna vrijednost od idućih definirana (=različita od $\emptyset$) za sve $q \in Q, \alpha \in \Sigma, \gamma \in \Gamma$:

$$\delta(q, \alpha, \gamma), \delta(q, \alpha, \varepsilon), \delta(q, \varepsilon, \gamma), \delta(q, \varepsilon, \varepsilon)$$

Primijetimo da DPA ne mora uvijek moći doći do kraja ulaza; može *zapeti* (jer je stog prazan, a nastaviti rad može tek kad napravi pop) ili *petljati* (beskonačno radi bez čitanja s ulaza).

Modifikacije DPA

Lema: Za svaki DPA, postoji ekvivalentan DPA koji iz završne konfiguracije prolazi samo kroz završne konfiguracije dok ne učita slovo s ulaza.
Dokaz: Za sve $q, r \in Q$ te $x, y \in \Gamma_\varepsilon$, dodamo završno stanje q_a te ako je $\delta(q, \varepsilon, x) = (r, y)$, definiramo $\delta(q_a, \varepsilon, x) = (r_a, y)$; ako je $q \in F$, redefiniramo $\delta(q, \varepsilon, x) = (r_a, y)$. Za sve $q \in Q$ i $a \in \Gamma$, ako je $\delta(q, a, x) = (r, y)$, definiramo $\delta(q_a, a, x) = (r, y)$.
Primjer: gornja konstrukcija na DPA za $L^+_=$.

Pokažimo da svaki DPA ima ekvivalentan DPA koji čita ulaz do kraja. Dodajmo slijepo završno stanje q_{DA} i mrtvo stanje q_{NE} te:

- Ako DPA zapne u stanju q, dopustimo mu da ipak napravi pop (kako?). Ako je $q \in F$, DPA pritom uđe u q_{DA} (slijepo: ako se pročita još nešto s ulaza, DPA ide u q_{NE}), inače u q_{NE}.
- Ako DPA petlja u stanju q i x je na vrhu stoga te DPA nikad ne čita niti ne pop-a baš taj x, stavimo $\delta(q, \varepsilon, x) = (q_{DA}, \varepsilon)$ ako bi DPA kasnije prihvatio, a $\delta(q, \varepsilon, x) = (q_{NE}, \varepsilon)$ inače.

Zatvorenost klase DBK za komplemente

Teorem: Za svaki $L \in DBK$ vrijedi $L^c \in DBK$.

Skica dokaza: Modificiramo DPA za L kao na prethodnom slajdu (čita ulaz do kraja, iz završnog stanja ide po završnim stanjima dok ne pročita novi znak). Za $q \in Q$ i $a \in \Sigma$, ako je $\delta(q, a, \varepsilon) \neq \emptyset$, reći ćemo da je q *čitajuće* stanje (iz stanja q naš DPA može dalje samo čitanjem nekog znaka; zašto?). Ako pak za $x \in \Gamma$ vrijedi $\delta(q, a, x) = (r, y)$, dodamo (zašto?) stanje q_x (koje će biti čitajuće) i izmijenimo δ ovako: $\delta(q, \varepsilon, x) := (q_x, \varepsilon)$ te $\delta(q_x, a, \varepsilon) = (r, y)$. Proglasimo q_x završnim ako je $q \in F$.

Sad sva završna stanja koja nisu čitajuća uklonimo iz F. Dobili smo ekvivalentan DPA polaznom, samo što za svaki znak ulaza novi DPA ulazi najviše jednom u završno stanje: neposredno prije čitanja idućeg znaka ulaza. Sad možemo zamijeniti završna i nezavršna **čitajuća** stanja kako bismo dobili DPA za L^c. □

Koji važan zaključak možemo izvući primjenom gornjeg teorema (hint:

Zatvorenost klase DBK za komplemente

Teorem: Za svaki $L \in DBK$ vrijedi $L^c \in DBK$.

Skica dokaza: Modificiramo DPA za L kao na prethodnom slajdu (čita ulaz do kraja, iz završnog stanja ide po završnim stanjima dok ne pročita novi znak). Za $q \in Q$ i $a \in \Sigma$, ako je $\delta(q, a, \varepsilon) \neq \emptyset$, reći ćemo da je q *čitajuće* stanje (iz stanja q naš DPA može dalje samo čitanjem nekog znaka; zašto?). Ako pak za $x \in \Gamma$ vrijedi $\delta(q, a, x) = (r, y)$, dodamo (zašto?) stanje q_x (koje će biti čitajuće) i izmijenimo δ ovako: $\delta(q, \varepsilon, x) := (q_x, \varepsilon)$ te $\delta(q_x, a, \varepsilon) = (r, y)$. Proglasimo q_x završnim ako je $q \in F$.

Sad sva završna stanja koja nisu čitajuća uklonimo iz F. Dobili smo ekvivalentan DPA polaznom, samo što za svaki znak ulaza novi DPA ulazi najviše jednom u završno stanje: neposredno prije čitanja idućeg znaka ulaza. Sad možemo zamijeniti završna i nezavršna **čitajuća** stanja kako bismo dobili DPA za L^c. □

Koji važan zaključak možemo izvući primjenom gornjeg teorema (hint: je li BK zatvorena za komplemente)?

Ograničen automat

Definicija: Kažemo da je $(Q, \Sigma, \Gamma, \textvisiblespace, \Delta, q_0, q_\checkmark)$ *ograničen automat* (OA), gdje su komponente kao kod JPA, no uz uvjet $\Sigma \subseteq \Gamma \not\ni \textvisiblespace$ te relaciju prijelaza $\Delta \subseteq Q \times (\Gamma \cup \{\textvisiblespace\}) \times Q \times \Gamma \times \{-1, 1\}$.

Konfiguracija OA je uređena trojka $(q, m, t) \in Q \times \mathbb{N} \times \Gamma_\textvisiblespace^\mathbb{N}$ (komponente nazivamo *stanje*, *pozicija* i *traka*), gdje je

$$\Gamma_\textvisiblespace^\mathbb{N} := \{(t : \mathbb{N} \to \Gamma \cup \{\textvisiblespace\}) : t^{-1}[\Gamma] \text{ je konačan}\}.$$

Kažemo da je (q, m, t) *n-konfiguracija* za $n := \min(t^{-1}[\{\textvisiblespace\}])$.

Ako je $(q, \alpha, p, \beta, d) \in \Delta$ i $(q, k, u\alpha v\textvisiblespace\textvisiblespace\ldots)$ n-konfiguracija gdje je $u, v \in \Gamma^*$ i $|u| = k$, ona *prelazi* ($\vdash$) u $(p, c_n(k + d), u\beta' v\textvisiblespace\textvisiblespace\ldots)$. Pritom je $c_n(m) := \min\{n, \max\{0, m\}\}$ oznaka za *clip*-funkciju te je β' jednak β za $k \neq n$, a $\textvisiblespace$ inače.

OA *prihvaća* $w \in \Sigma^*$ ako postoji konfiguracija c sa stanjem $q_\checkmark$ takva da $(q_0, 0, w\textvisiblespace\textvisiblespace\ldots) \vdash^* c$. OA $\mathcal{O}$ *prepoznaje* jezik

$$L(\mathcal{O}) := \{w \in \Sigma^* : \mathcal{O} \text{ prihvaća } w\}.$$

Poboljšanja OA

Kako bismo s ograničenim automatima lakše radili, napravit ćemo dvije stvari: (1) dat ćemo neka „poboljšanja" OA („poboljšani" strojevi i dalje prepoznaju iste jezike) i (2) koristit ćemo dana poboljšanja dok pišemo *pseudokod* (umjesto često kompliciranih detalja) za ograničene automate.

Kako ćemo dokazati da se ostaje u istoj klasi jezika?

Poboljšanja OA

Kako bismo s ograničenim automatima lakše radili, napravit ćemo dvije stvari: (1) dat ćemo neka „poboljšanja" OA („poboljšani" strojevi i dalje prepoznaju iste jezike) i (2) koristit ćemo dana poboljšanja dok pišemo *pseudokod* (umjesto često kompliciranih detalja) za ograničene automate.

Kako ćemo dokazati da se ostaje u istoj klasi jezika? Pokazivanjem da OA može *simulirati* svaki elementarni korak poboljšanog stroja.

Nemijenjanje trake, više redaka

Ako želimo poboljšanom OA omogućiti samo promjenu stanja i pomak glave, bez obzira na stanje trake i bez promjene znaka, u poboljšanom OA dozvoljavamo prijelaze oblika $(q, *, p, *, d)$ nad proširenom abecedom trake Γ_*. Kako te prijelaze simuliramo običnim OA?

Nemijenjanje trake, više redaka

Ako želimo poboljšanom OA omogućiti samo promjenu stanja i pomak glave, bez obzira na stanje trake i bez promjene znaka, u poboljšanom OA dozvoljavamo prijelaze oblika $(q, *, p, *, d)$ nad proširenom abecedom trake Γ_*. Kako te prijelaze simuliramo običnim OA? Svaki prijelaz $(q, *, p, *, d)$ postaje $\#\Gamma$ prijelazâ $(q, \alpha, p, \alpha, d)$, po jedan za svaki $\alpha \in \Gamma$.

Ako želimo da svaka ćelija na traci ima više redaka, tj. njih $k > 1$ s radnim abecedama $\Gamma_1, \ldots, \Gamma_k$, relacija prijelaza je ovakva:

$$\Delta_k \subseteq Q \times \prod_{i=1}^{k} (\Gamma_i \cup \{\textvisiblespace\}) \times Q \times \prod_{i=1}^{k} \Gamma_i \times \{-1, 1\}.$$

Simulacija?

Nemijenjanje trake, više redaka

Ako želimo poboljšanom OA omogućiti samo promjenu stanja i pomak glave, bez obzira na stanje trake i bez promjene znaka, u poboljšanom OA dozvoljavamo prijelaze oblika $(q, *, p, *, d)$ nad proširenom abecedom trake Γ_*. Kako te prijelaze simuliramo običnim OA? Svaki prijelaz $(q, *, p, *, d)$ postaje $\#\Gamma$ prijelazâ $(q, \alpha, p, \alpha, d)$, po jedan za svaki $\alpha \in \Gamma$.

Ako želimo da svaka ćelija na traci ima više redaka, tj. njih $k > 1$ s radnim abecedama $\Gamma_1, \ldots, \Gamma_k$, relacija prijelaza je ovakva:

$$\Delta_k \subseteq Q \times \prod_{i=1}^{k} (\Gamma_i \cup \{\textvisiblespace\}) \times Q \times \prod_{i=1}^{k} \Gamma_i \times \{-1, 1\}.$$

Simulacija? Obični OA će imati $\Gamma = \prod_i \Gamma_i$. Naravno, ulogu simbola $\textvisiblespace$ preuzima $(\textvisiblespace, \ldots, \textvisiblespace)$. Za reprezentaciju ulaza, poistovjećujemo znak $\alpha \in \Sigma \subseteq \Gamma_1$ i k-torku $(\alpha, \_, \ldots, \_)$ (pretpostavljamo da za $i > 1$, $\_ \in \Gamma_i$). Često koristimo *traku s oznakama* ($k = 2$, $\Gamma_1 = \Gamma$, $\Gamma_2 = \{\hat{}, \_\}$).

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi traku s oznakama:

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi traku s oznakama: u drugi redak zapiše $\hat{}$, i ode lijevo. Nakon toga, ako u drugom retku čita $\hat{}$, zna da je na lijevom rubu, i tada u drugi redak zapiše $\_$, i ode lijevo.

U suprotnom, zna da nije na lijevom rubu, ali još treba obrisati $\hat{}$ koji je upravo zapisao: pomakne se desno, zatim lijevo zapisavši $\_$, i ponovo desno (desni pomaci su bez mijenjanja trake).

Ovako omogućimo stajanje na mjestu, tj. prijelaz $(q, \alpha, p, \beta, 0)$:

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi traku s oznakama: u drugi redak zapiše $\hat{}$, i ode lijevo. Nakon toga, ako u drugom retku čita $\hat{}$, zna da je na lijevom rubu, i tada u drugi redak zapiše $\_$, i ode lijevo.

U suprotnom, zna da nije na lijevom rubu, ali još treba obrisati $\hat{}$ koji je upravo zapisao: pomakne se desno, zatim lijevo zapisavši $\_$, i ponovo desno (desni pomaci su bez mijenjanja trake).

Ovako omogućimo stajanje na mjestu, tj. prijelaz $(q, \alpha, p, \beta, 0)$:

- ako je na lijevom rubu, pomakne se „ulijevo": $(q, \alpha, p, \beta, -1)$.

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi traku s oznakama: u drugi redak zapiše $\hat{}$, i ode lijevo. Nakon toga, ako u drugom retku čita $\hat{}$, zna da je na lijevom rubu, i tada u drugi redak zapiše $\_$, i ode lijevo.

U suprotnom, zna da nije na lijevom rubu, ali još treba obrisati $\hat{}$ koji je upravo zapisao: pomakne se desno, zatim lijevo zapisavši $\_$, i ponovo desno (desni pomaci su bez mijenjanja trake).

Ovako omogućimo stajanje na mjestu, tj. prijelaz $(q, \alpha, p, \beta, 0)$:

- ako je na lijevom rubu, pomakne se „ulijevo": $(q, \alpha, p, \beta, -1)$.
- inače, pomakne se ulijevo pa udesno:
  $(q, \alpha, p', \beta, -1)$, $(p', *, p, *, 1)$, gdje je p' novo stanje.

Također, možemo omogućiti *skup* F završnih stanja (kao kod PA). Simuliramo ih pomoću

Detekcija rubova, stajanje na mjestu, više završnih stanja

OA može detektirati je li na desnom rubu: ako čita $\sqcup$ (i ode desno), zna da je na desnom rubu. Ako jest, i ode lijevo bez mijenjanja trake, pa pročita $\sqcup$ (i ode desno), onda je i na lijevom rubu. Ako nije bio na desnom rubu, koristi traku s oznakama: u drugi redak zapiše $\hat{}$, i ode lijevo. Nakon toga, ako u drugom retku čita $\hat{}$, zna da je na lijevom rubu, i tada u drugi redak zapiše $\_$, i ode lijevo.

U suprotnom, zna da nije na lijevom rubu, ali još treba obrisati $\hat{}$ koji je upravo zapisao: pomakne se desno, zatim lijevo zapisavši $\_$, i ponovo desno (desni pomaci su bez mijenjanja trake).

Ovako omogućimo stajanje na mjestu, tj. prijelaz $(q, \alpha, p, \beta, 0)$:

- ako je na lijevom rubu, pomakne se „ulijevo": $(q, \alpha, p, \beta, -1)$.
- inače, pomakne se ulijevo pa udesno:
  $(q, \alpha, p', \beta, -1)$, $(p', *, p, *, 1)$, gdje je p' novo stanje.

Također, možemo omogućiti *skup* F završnih stanja (kao kod PA). Simuliramo ih pomoću prijelaza $(q, *, q_\checkmark, *, 0)$ za svaki $q \in F$.

Konačne varijable, više traka

OAu možemo omogućiti pristup *varijabli s* koja poprima vrijednosti iz **konačnog** skupa S; on postavlja s ovisno o pročitanom znaku i/ili stanju, odnosno ovisno o s (ne) može činiti prijelaze. Nije ga teško simulirati:

Konačne varijable, više traka

OAu možemo omogućiti pristup *varijabli* s koja poprima vrijednosti iz **konačnog** skupa S; on postavlja s ovisno o pročitanom znaku i/ili stanju, odnosno ovisno o s (ne) može činiti prijelaze. Nije ga teško simulirati: skup stanja je $Q \times S$, a ostale promjene su jasne. Kako biste nacrtali OA s binarnom konačnom varijablom?

Ako želimo omogućiti $k > 1$ traka, svaku sa svojom glavom koje se **istodobno** pokreću u nezavisnim smjerovima, relacija prijelaza je

Konačne varijable, više traka

OAu možemo omogućiti pristup *varijabli* s koja poprima vrijednosti iz **konačnog** skupa S; on postavlja s ovisno o pročitanom znaku i/ili stanju, odnosno ovisno o s (ne) može činiti prijelaze. Nije ga teško simulirati: skup stanja je $Q \times S$, a ostale promjene su jasne. Kako biste nacrtali OA s binarnom konačnom varijablom?

Ako želimo omogućiti $k > 1$ traka, svaku sa svojom glavom koje se **istodobno** pokreću u nezavisnim smjerovima, relacija prijelaza je

$$\Delta_k \subseteq Q \times \prod_{i=1}^{k} (\Gamma_i \cup \{\textvisiblespace\}) \times Q \times \prod_{i=1}^{k} \big(\Gamma_i \times \{-1, 0, 1\}\big).$$

Ulaz je na prvoj traci ($\Sigma \subseteq \Gamma_1$) — ostale su popunjene sa $\textvisiblespace$.

Simulacija:

Konačne varijable, više traka

OAu možemo omogućiti pristup *varijabli* s koja poprima vrijednosti iz **konačnog** skupa S; on postavlja s ovisno o pročitanom znaku i/ili stanju, odnosno ovisno o s (ne) može činiti prijelaze. Nije ga teško simulirati: skup stanja je $Q \times S$, a ostale promjene su jasne. Kako biste nacrtali OA s binarnom konačnom varijablom?

Ako želimo omogućiti $k > 1$ traka, svaku sa svojom glavom koje se **istodobno** pokreću u nezavisnim smjerovima, relacija prijelaza je

$$\Delta_k \subseteq Q \times \prod_{i=1}^{k} (\Gamma_i \cup \{\textvisiblespace\}) \times Q \times \prod_{i=1}^{k} \big(\Gamma_i \times \{-1, 0, 1\}\big).$$

Ulaz je na prvoj traci ($\Sigma \subseteq \Gamma_1$) — ostale su popunjene sa $\textvisiblespace$.

Simulacija: Obični OA s $2k$ redaka. Svaku traku pamtimo kao traku s oznakama, gdje je označen samo onaj znak na kojem je „glava" te trake. Glava običnog OA sad može pročitati označene znakove na svim trakama (kako ih pamti?), odabrati element od Δ_k, prijeći u novo stanje, zapisati nove znakove na označena mjesta i pomaknuti oznake.

Kontekstne gramatike

Kontekstno pravilo je pravilo oblika $uAv \to ubv$, gdje su $u, v \in (Y \setminus \{S\})^*$, $A \in V$, te $b \in (Y \setminus \{S\})^+$.

Kontekstna gramatika je gramatika kojoj je svako pravilo kontekstno ili $S \to \varepsilon$ (gdje je S početna varijabla).

Kontekstni jezik je onaj koji generira neka kontekstna gramatika.

Klasu svih kontekstnih jezika označavamo sa *Kon*.

Kontekst *uv može* biti ε, dakle svaki beskontekstni jezik je kontekstni (konkretno, generiran je Chomskyjevom gramatikom, a pravila čitanja i širenja su kontekstna). Štoviše, inkluzija je prava.

Zadatak: Opišite OA koji prepoznaje $L_{\equiv}$.

Ipak, da bismo iz toga zaključili $L_{\equiv} \in Kon$, treba nam (težak) teorem ekvivalencije KG i OA. Napisati KG za $L_{\equiv}$ također nije jednostavno. Možemo li se nekako izvući?

Monotone gramatike

MG predstavljaju generalizaciju KG, a također generiraju kontekstne jezike, uz vrlo „mehanički" proces pretvorbe u KG.

Pravilo $u \to v$ (gdje u sadrži barem jednu varijablu i S se ne pojavljuje u v) je *monotono* ako je $|u| \leq |v|$.

Gramatika je *monotona* ako joj je svako pravilo monotono ili $S \to \varepsilon$ gdje je S početna varijabla.

Zbog $|b| \geq 1 = |A|$, svako kontekstno pravilo je monotono, pa je svaka KG ujedno i MG. Dakle, svaki kontekstni jezik generiran je nekom MG. No vrijedi i obrat: ako je $\mathcal{G}$ monotona, $L(\mathcal{G})$ je kontekstan. To se dokazuje pretvorbom MG $\to$ KG.

Pretvorba MG $\to$ KG

Pretvorba ima četiri faze. Prve dvije znamo redom pod imenima START i TERM iz pretvorbe BKG $\to$ ChNF — jedino što se u TERM „nesamostalni" znakovi mogu nalaziti i s lijeve strane pravila. Sve takve također zamijenimo novim varijablama.

Treća faza je generalizacija faze BIN, i odnosi se na pravila čija *lijeva* strana (pa onda i desna) ima bar 2 simbola (sada varijabli). Svako takvo pravilo je oblika $A_1 A_2 \cdots A_n \to B_1 B_2 \cdots B_m$ gdje je $m \geq n \geq 2$, te ga zamijenimo s $2n$ kontekstnih pravila, uvodeći n novih varijabli $C_1, C_2, \ldots, C_n$:

$$C_1 \cdots C_{i-1} A_i A_{i+1} \cdots A_n \to C_1 \cdots C_{i-1} C_i A_{i+1} \cdots A_n, \quad i \in [1..n]$$
$$B_1 \cdots B_{i-1} C_i C_{i+1} \cdots C_n \to B_1 \cdots B_{i-1} B_i C_{i+1} \cdots C_n, \quad i \in [1..n\rangle$$
$$B_1 \cdots B_{n-1} C_n \to B_1 \cdots B_{n-1} B_n \cdots B_m$$

Četvrta faza je poznat nam DEL: primijetite da imamo $S' \to S$ i možda imamo $S \to \varepsilon$, pa ova faza miče $S \to \varepsilon$ i dodaje $S' \to \varepsilon$  .

Monotona gramatika za $L_{\equiv}$

$$\mathcal{G} := \left(\{S, B, U\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\overset{①}{\to}, \overset{②}{\to}, \overset{③}{\to}, \overset{④}{\to}, \overset{⑤}{\to}\}, S\right)$$

$$S \overset{①}{\to} U \qquad U \overset{②}{\to} \mathtt{abc} \qquad U \overset{③}{\to} \mathtt{a}UB\mathtt{c} \qquad \mathtt{c}B \overset{④}{\to} B\mathtt{c} \qquad \mathtt{b}B \overset{⑤}{\to} \mathtt{bb}$$

je primjer monotone gramatike za $L_{\equiv}$ (dokaz u nastavku).

Iz toga i upravo dokazanog slijedi $L_{\equiv} \in \mathit{Kon}$.

Za dokaz inkluzije $L_{\equiv} \subseteq L(\mathcal{G})$, uzmimo proizvoljnu riječ $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \in L_{\equiv}$ i napišimo izvod za nju.

Nakon ① i $n-1$ primjene ③, imamo $\mathtt{a}^{n-1}U(B\mathtt{c})^{n-1}$, te nakon ② imamo $\mathtt{a}^n\mathtt{bc}(B\mathtt{c})^{n-1}$. Još treba izvesti $\mathtt{b}^n\mathtt{c}^n$ iz $\mathtt{bc}(B\mathtt{c})^{n-1} = \mathtt{b}(\mathtt{c}B)^{n-1}\mathtt{c} \overset{④}{\Rightarrow}{}^{n-1} \mathtt{b}(B\mathtt{c})^{n-1}\mathtt{c} \overset{⑤}{\Rightarrow} \mathtt{bb}(\mathtt{c}B)^{n-2}\mathtt{cc}$, postupanjem kao u gornjem redu $n-1$ puta.

Monotona gramatika za $L_{=}$ — dokaz druge inkluzije

Riječ $w \in Y^*$ je *balansirana* ako je $|w|_{\mathtt{a}} = |w|_{\mathtt{b}} + |w|_B = |w|_{\mathtt{c}}$.

Lema: Ako $u \Rightarrow v$ i u je balansirana, tada je i v balansirana.
Dokaz: po slučajevima, ovisno o upotrijebljenom pravilu.

① ne utječe na relevantne simbole. ② dodaje a, b i c. ③ dodaje a, B i c. ④ samo mijenja redoslijed simbola, ne njihov broj. ⑤ pretvara B u b.

Korolar: Ako $S \Rightarrow^* w$, tada je w balansirana.

Dokaz: indukcijom po broju koraka u izvodu.
Baza: S je balansirana. Korak: koristimo gornju lemu.

Dakle, ako je $w \in L(\mathcal{G})$, tada je $w \in \Sigma^*$, pa činjenica da je w balansirana znači da ima jednak broj svakog slova ($|w|_B = 0$).

Prema tome, druga inkluzija bi bila dokazana kad bismo još imali

Monotona gramatika za $L_{=}$ — dokaz druge inkluzije

Riječ $w \in Y^*$ je *balansirana* ako je $|w|_{\mathtt{a}} = |w|_{\mathtt{b}} + |w|_B = |w|_{\mathtt{c}}$.

Lema: Ako $u \Rightarrow v$ i u je balansirana, tada je i v balansirana.
Dokaz: po slučajevima, ovisno o upotrijebljenom pravilu.

① ne utječe na relevantne simbole. ② dodaje a, b i c. ③ dodaje a, B i c. ④ samo mijenja redoslijed simbola, ne njihov broj. ⑤ pretvara B u b.

Korolar: Ako $S \Rightarrow^* w$, tada je w balansirana.

Dokaz: indukcijom po broju koraka u izvodu.
Baza: S je balansirana. Korak: koristimo gornju lemu.

Dakle, ako je $w \in L(\mathcal{G})$, tada je $w \in \Sigma^*$, pa činjenica da je w balansirana znači da ima jednak broj svakog slova ($|w|_B = 0$).

Prema tome, druga inkluzija bi bila dokazana kad bismo još imali $L(\mathcal{G}) \subseteq a^+b^+c^+$.

Monotona gramatika za $L_{=}$ — dokaz druge inkluzije

Lema: $L(\mathcal{G}) \subseteq \mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$.

Dokaz: Pojednostavimo gramatiku (zašto?) tako da $L(\mathcal{G}) = L(\mathcal{G}')$:

$$\mathcal{G}' := \left(\{S, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\overset{①}{\to}, \overset{②}{\to}, \overset{③}{\to}, \overset{④}{\to}\}, S\right)$$

$$S \overset{①}{\to} \mathtt{abc} \qquad S \overset{②}{\to} \mathtt{a}S B\mathtt{c} \qquad \mathtt{c}B \overset{③}{\to} B\mathtt{c} \qquad \mathtt{b}B \overset{④}{\to} \mathtt{bb}$$

Dokaz provodimo totalnom indukcijom po

Monotona gramatika za $L_{\equiv}$ — dokaz druge inkluzije

Lema: $L(\mathcal{G}) \subseteq \mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$.

Dokaz: Pojednostavimo gramatiku (zašto?) tako da $L(\mathcal{G}) = L(\mathcal{G}')$:

$$\mathcal{G}' := \left(\{S, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\overset{①}{\to}, \overset{②}{\to}, \overset{③}{\to}, \overset{④}{\to}\}, S\right)$$

$$S \overset{①}{\to} \mathtt{abc} \qquad S \overset{②}{\to} \mathtt{a}S B\mathtt{c} \qquad \mathtt{c}B \overset{③}{\to} B\mathtt{c} \qquad \mathtt{b}B \overset{④}{\to} \mathtt{bb}$$

Dokaz provodimo totalnom indukcijom po duljini izvoda $w \in L(\mathcal{G}')$.
Izvod mora početi sa

Monotona gramatika za $L_{=}$ — dokaz druge inkluzije

Lema: $L(\mathcal{G}) \subseteq \mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$.

Dokaz: Pojednostavimo gramatiku (zašto?) tako da $L(\mathcal{G}) = L(\mathcal{G}')$:

$$\mathcal{G}' := \left(\{S, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\overset{①}{\to}, \overset{②}{\to}, \overset{③}{\to}, \overset{④}{\to}\}, S\right)$$

$$S \overset{①}{\to} \mathtt{abc} \qquad S \overset{②}{\to} \mathtt{a}S B\mathtt{c} \qquad \mathtt{c}B \overset{③}{\to} B\mathtt{c} \qquad \mathtt{b}B \overset{④}{\to} \mathtt{bb}$$

Dokaz provodimo totalnom indukcijom po duljini izvoda $w \in L(\mathcal{G}')$. Izvod mora početi sa $S \Rightarrow \mathtt{abc}$ (pa smo gotovi), ili sa $S \Rightarrow \mathtt{a}SB\mathtt{c}$. BSOMP da je izvod lijevi (inače ga preuredimo). Tada on nastavlja do riječi $\mathtt{a}w_S B\mathtt{c}$ gdje je $w_S \in L(\mathcal{G}')$, pa po PI (primijetite da je w_S dobiven iz S u strogo manje koraka nego w) slijedi $w_S \in \mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$.

No, $aw_S Bc$ krajem izvoda postaje $w \in \Sigma^*$, pa B mora u međuvremenu nestati. Jedini način za to je primjena pravila ④, a to je tek moguće nakon iterirane primjene pravila ③. Drugim riječima, izvod za w mora izgledati ovako (za neke $k, l, m \in \mathbb{N}_+$):

$$S \Rightarrow^* \mathtt{a}w_S B\mathtt{c} = \mathtt{a}\mathtt{a}^k\mathtt{b}^l\mathtt{c}^m B\mathtt{c} \overset{③}{\Rightarrow}{}^* \mathtt{a}^{k+1}\mathtt{b}^l B\mathtt{c}^{m+1} \overset{④}{\Rightarrow} \mathtt{a}^{k+1}\mathtt{b}^{l+1}\mathtt{c}^{m+1}. \qquad \square$$

Pretvorba monotone gramatike u ograničeni automat

Za MG $\mathcal{G}$, pseudokod ekvivalentnog OA je (ulaz w u 1. retku):

$t2_{0:1} := S$ # na početak drugog retka stavi početnu varijablu
forever:
    odaberi pravilo $u \to v$ gramatike $\mathcal{G}$
    odaberi $i \in [0..|w| - |v|\rangle$
    provjeri $t2_{i:i+|u|} = u$ (ako nije, odbaci)
    $t2_{i+|v|:|w|} := t2_{i+|u|:|w|-|v|+|u|}$
    $t2_{i:i+|v|} := v$
    if $t1 = t2$: prihvati

Ukratko, koristimo nedeterminizam da bismo „pogodili" izvod w.

Ključno je da, jer su pravila monotona, ako $t2$ ikad postane „dulji" od w, možemo odbaciti tu granu jer sigurno neće završiti s w.

Pretvorba OA $\to$ MG: osnovna ideja

Neka je zadan OA $\mathcal{O}$. BSOMP $Q \cap \Gamma = \emptyset$ (inače preimenujemo stanja). Zato svaku n-konfiguraciju $(q, m, ab␣␣\ldots)$ gdje je $|a| = m$ i $|ab| = n$ možemo kodirati kao $aqb␣ \in (Q \dot\cup \Gamma \dot\cup \{␣\})^*$. Cilj je opisati relaciju $\vdash$ na takvim kodovima pomoću pravila gramatike. U tu svrhu, stavimo $Y = Q \dot\cup \Gamma \dot\cup \{S, ␣\}$ (BSOMP $S \notin Q \cup \Gamma$), odnosno preciznije $V := Q \dot\cup \Gamma \dot\cup \{S, ␣\} \setminus \Sigma$.

Za svaki prijelaz $(q, \alpha, p, \beta, 1) \in \Delta$ dodajemo pravilo $q\alpha \to \beta p$ (s malo iznimkom: ako je $\alpha = ␣$, ostaje isti; rubove ćemo riješiti na jednom od idućih slajdova).

Za svaki pak prijelaz $(q, \alpha, p, \beta, -1) \in \Delta$, dodajemo $\#\Gamma$ pravila $\gamma q\alpha \to p\gamma\beta$, po jedno za svaki $\gamma \in \Gamma$ (uz istu napomenu kao gore). Očito su sva ta pravila monotona (obje strane su im duljine 2 odnosno 3), te imaju bar jednu varijablu (q) slijeva.

Samo... Od čega ćemo početi? Htjeli bismo od $q_0 w$, ali kako znati w unaprijed?

Jednostavni ograničeni automat

Ukratko, problem je u tome što automat i gramatika imaju suprotne načine rada: automat počinje od w i mijenja konfiguraciju ne bi li došao do fiksnog stanja $q_\checkmark$, dok gramatika počinje od fiksne varijable S i mijenja riječ nad Y ne bi li došla do w.

U jednom smjeru to smo riješili korištenjem 2 retka. Možemo i u ovom — prvo trebamo pojednostaviti $\mathcal{O}$, tako da završna konfiguracija bude jedinstvena: *JOA* prihvaća w *samo* u konfiguraciji $(q_\checkmark, 0, w)$. [Primijetite da je dobar dio definicije JPA također osiguravanje jedinstvenosti završne konfiguracije.]

Za svaki OA postoji ekvivalentni JOA s dva retka (pa onda i s jednim, jer smo samo podebljali Γ, ne i Q): na početku prepišemo ulaz u drugi redak, zatim radimo sve isto kao originalni OA samo u drugom retku, a umjesto originalnog završnog stanja odemo u „predzavršno" stanje u kojem odemo desno do kraja trake, i obrišemo drugi redak (vratimo znakove iz $\Sigma \times \Gamma$ u Σ) nalijevo.

Pretvorba JOA $\to$ MG — skoro pa rješenje

Sada na kraju računanja imamo netaknutu riječ w, ali kako je pogoditi na početku? Jednostavno: dodamo pravila koja mogu na početku generirati proizvoljnu riječ: $S \to q_0 \textvisiblespace \mid q_0 A \textvisiblespace$, te $A \to \alpha \mid \alpha A$ za $\alpha \in \Sigma$. Kako gramatika simulira rad JOA, možemo na kraju dobiti istu riječ (precizno $q_\checkmark w \textvisiblespace$) ako i samo ako je JOA prihvaća.

Sada je još samo potrebno ukloniti $q_\checkmark$ i $\textvisiblespace$ — ali to se čini nepremostivim problemom, jer pravila moraju biti monotona!

Imamo još jedan problem iste vrste: gornja pravila zapravo moraju biti nešto poput $S \to [q_0 \textvisiblespace] \mid [q_0 A \textvisiblespace]$, da bismo u pravilima mogli detektirati rubove trake (npr. $[q\alpha \to [p\beta$ za $(q, \alpha, p, \beta, -1) \in \Delta$). Te „znakove" također na kraju treba ukloniti.

Trik ujedinjavanja znakova

Kako uklonjivih znakova ima konačno mnogo $(3 + \#Q)$, i svaki se može pojaviti najviše jednom u jednom koraku izvoda, svih *kombinacija* uklonjivih znakova zajedno s jednim „pravim" znakom (poput $[q_0\mathtt{a}$, ili $\mathtt{b}\sqcup]$) ima konačno mnogo. To znači da ih možemo „slijepiti" (smatrati istim znakom), i skup Γ ostat će konačan. Sada npr. uklanjanje otvorene uglate zagrade postaje monotono pravilo poput $\mathtt{a}[\to \mathtt{a}$.

[Ovaj trik se koristi i kod Turingovih strojeva pri proučavanju složenosti, primjerice kod dokaza teorema o linearnom ubrzanju. Sjetite ga se na kolegiju Složenost algoritama!]

Dokazali smo: jezik je kontekstan akko ga prepoznaje neki OA.

Zatvorenost klase kontekstnih jezika *Kon*

Istim dokazima kao za klasu *BK*, možemo vidjeti da je *Kon* zatvorena na uniju i jezične operacije. No, *Kon* se ponaša i bolje:

Teorem (Immerman–Szelepcsényi): Komplement kontekstnog jezika je kontekstan. [**Dokaz** (težak) na kolegiju SA.]

Iz toga (i zatvorenosti na unije) odmah slijedi da je *Kon* zatvorena na sve skupovne operacije. No i bez gornjeg teorema, možemo dokazati zatvorenost na presjeke (a time i konusno svojstvo):

JOA sačuvaju traku i poziciju ako prihvate riječ, pa ih je lako „spojiti serijski" tako da završno stanje prvog postane početno stanje drugog, i tako dobiveni JOA prihvaća riječ ako i samo ako je svaki pojedini JOA u seriji prihvaća: drugim riječima, prepoznaje presjek njihovih jezika.

Problemi i odlučivost

Intuitivno, *problem* je pitanje koje za svaku svoju *instancu* ima jednoznačan da/ne odgovor. Recimo, instance *problema prihvaćanja za DKA*, A_{DKA}, su parovi $(\mathcal{M}, w)$ gdje je $\mathcal{M}$ DKA i $w \in \mathcal{M}.\Sigma^*$. Problem prihvaćanja pita je li $w \in L(\mathcal{M})$.

Problem je *odlučiv* ako postoji *algoritam* koji prima instancu problema te u konačno mnogo koraka daje ispravan odgovor.

Zadatak: Napišite algoritam koji pokazuje da je A_{DKA} odlučiv.

Ponekad nas zanima više od da/ne: faktorizacija umjesto provjere prostosti, niz tokena umjesto A_{DKA}, stablo parsiranja umjesto A_{BKG}, itd. No, zasad ćemo se baviti samo problemima prihvaćanja.

Kasnije ćemo promatrati i druge probleme te formalno definirati algoritme, probleme i odlučivanje. Puno preciznije o tim pojmovima čut ćete na kolegiju Izračunljivost.

Svođenje (redukcija)

Svođenje problema Q_1 na problem Q_2 je algoritam koji prima instance od Q_1 i daje instance od Q_2, tako da točan odgovor (da/ne) ostane isti.

Primjer: algoritam za partitivnu konstrukciju u prvoj koordinati, $(\mathcal{M}, w) \mapsto (\mathcal{P}(\mathcal{M}), w)$, svodi problem A_{NKA} na A_{DKA}.

Ako postoji takav algoritam, kažemo da je Q_1 *svediv* na Q_2. Ako je pritom Q_2 odlučiv, tada je i Q_1 odlučiv (*komponiramo* algoritme). Dakle, iz gornjeg primjera slijedi: A_{NKA} je odlučiv.

Nedeterminističke konstrukcije (SIPSER 1.45, 1.47, 1.49 i 1.55) svode A_{RI} na A_{NKA}, pa zaključujemo da je i A_{RI} odlučiv.

Zadatak: Kako biste dokazali da je i A_{DLG} odlučiv?

Neformalno kažemo da je „problem prihvaćanja za regularne jezike" A_{Reg} odlučiv.

Problem A_{BK}

Slično, moguće je algoritamski pojednostaviti PA, pretvoriti JPA u BKG, i BKG u PA. Sve te pretvorbe čuvaju jezik, pa su problemi A_{JPA}, A_{PA} i A_{BKG} svedivi jedni na druge. Dakle ima smisla govoriti o „problemu prihvaćanja za beskontekstne jezike" A_{BK}.

Znamo da je A_{BK} *poluodlučiv*: jednom ćemo „nabasati" na izvod ako on postoji i ako ga sistematično tražimo. No, je li odlučiv?

Propozicija: Neka je $\mathcal{G}$ ChNF i $w \in L(\mathcal{G})$. Ako je $w = \varepsilon$, jedini izvod za w iz $\mathcal{G}$ je $S \Rightarrow \varepsilon$. Inače, svaki izvod od w u $\mathcal{G}$ ima točno $2|w| - 1$ koraka, i to $|w| - 1$ širenja i $|w|$ čitanja.

Dokaz: Za netrivijalan slučaj $w \neq \varepsilon$, označimo s a broj znakova i s v broj varijabli tijekom izvoda w iz $\mathcal{G}$. U početku je $(a, v) = (0, 1)$, a na kraju je $(a, v) = (|w|, 0)$. Svaka primjena pravila širenja inkrementira v, a primjena pravila čitanja dekrementira v i inkrementira a. Dakle, za x prvih i y drugih koraka, x i y zadovoljavaju sustav $(0, 1) + (0, 1)x + (1, -1)y = (|w|, 0)$ koji ima jedinstveno rješenje $(x, y) = (|w| - 1, |w|)$. □

Korolar: A_{BK} je odlučiv

Algoritam za odluku A_{BKG}:

input: $(\mathcal{G}, w)$ takvi da je $\mathcal{G} = (V, \Sigma, P, S)$ BKG, te $w \in \Sigma^*$
if $w = \varepsilon$: pretvaraj $\mathcal{G}$ u ChNF do faze DEL; **return** bool$(S \in np)$
dovrši pretvorbu $\mathcal{G}$ u ChNF $\mathcal{G}' =: (V', \Sigma, P', S')$
for $(p_1, \ldots, p_n)$ **in** $(P')^{n:=2|w|-1}$:
    $r := S'$
    for $p_i = (A_i \to u_i)$ **in** $(p_1, \ldots, p_n)$:
        if (prva varijabla u r) $= A_i$: zamijeni je s u_i
        else: **break**
    if $r = w$: **return True**
return False

Problem A_{Kon}

Je li problem A_{OA} odlučiv?

Problem A_{Kon}

Je li problem A_{OA} odlučiv? Da! Iz početne konfiguracije OA, postoji samo konačno mnogo *dostižívih* konfiguracija (konkretno, ima ih najviše $\#Q \cdot (|w|+1) \cdot (\#\Gamma)^{|w|}$). Drugim riječima, graf

$$(Q \times [0..|w|] \times \Gamma^{|w|}, \vdash)$$

je konačan, a problem prihvaćanja riječi w svodi se na problem dostupnosti bilo kojeg vrha iz skupa $\{q_\checkmark\} \times [0..|w|] \times \Gamma^{|w|}$, iz vrha $(q_0, 0, w\sqcup\sqcup\ldots)$. Kako biste ga riješili? Složenost?

Znamo da je izvod za w u ChNF fiksne duljine $\max\{2|w|-1, 1\}$. U MG ne možemo biti toliko precizni, ali ipak možemo ograničiti izvode. Naime, ako je $\mathcal{G}$ monotona i $w \in L(\mathcal{G})$, tada je svaka riječ (nad Y) u izvodu duljine najviše $|w|$. Njih ima konačno mnogo ($\frac{t^{|w|+1}-1}{t-1}$, gdje je $t := \#Y = \#V + \#\Sigma \geq 2$), pa kao i u slučaju OA imamo konačan graf $\left(\bigcup_{i=1}^{|w|} Y^i, \Rightarrow\right)$; možemo ga izračunati iz $\mathcal{G}$.

Kao gore, A_{MG} je odlučiv. Isti algoritam je primjenjiv na KG (svaka KG je MG). Prema tome, zaključujemo da je A_{Kon} odlučiv.

Polinomni algoritmi

Algoritam s prethodnog slajda je „pristup grubom silom": znamo da mogućih kandidata ima konačno mnogo, pa ih ispitamo sve.

Za algoritam kažemo da je *polinoman* ako postoji polinom $p(x) \in \mathbb{N}[x]$ takav da za svaki ulaz t tog algoritma, algoritam stane nakon najviše $p(|t|)$ koraka, gdje je $|t|$ *duljina* ulaza t (broj bitova potrebnih za njegov „razumni" prikaz).

Algoritam partitivne konstrukcije nije polinoman: 2^n (broj stanja koje treba konstruirati) raste brže od svakog polinoma od n.

Za problem kažemo da je *polinomno rješiv* (ili da je *u klasi* P) ako postoji polinomni algoritam koji ga rješava. Recimo, $A_{\text{DKA}} \in P$.

Napomena: 2002. godine je dokazano da je provjera prostosti polinomno rješiva (AKS-algoritam). Za faktorizaciju još ne znamo.

Polinomno svođenje i nedeterminizam

Kažemo da se problem Q_1 *polinomno svodi* na Q_2 ako postoji polinomni algoritam $t_1 \mapsto t_2$ koji svodi Q_1 na Q_2. Tada se lako vidi da je $|t_2| \leq p(|t_1|)$ za neki polinom p, dakle ako se Q_1 polinomno svodi na polinomno odlučiv Q_2, tada je i Q_1 polinomno odlučiv.

Nedeterminizam, kao i gruba sila, obično vode do eksponencijalnog povećanja broja koraka, te napuštaju klasu polinomnih algoritama.

[Kažemo da je problem u klasi NP ako postoji polinomni „nedeterministički algoritam" koji ga rješava.

Nemamo dokaz da je $P \neq NP$, ali većina računaraca u to vjeruje.]

Algoritam za svođenje A_{BKG} na A_{ChNF} je polinoman. Takvi su i algoritmi za svođenje A_{JPA} na A_{BKG} i A_{PA} na A_{JPA}.

Ali algoritam za rješavanje A_{ChNF} koji smo prezentirali koristi grubu silu, te nije polinoman. Možemo li bolje?

Cocke–Younger–Kasami (CYK) algoritam — osnovna ideja

Neka je zadana Chomskyjeva gramatika $\mathcal{G} = (V, \Sigma, \to, S)$, te riječ $w = \alpha_0\alpha_1 \cdots \alpha_{n-1} \in \Sigma^*$. Problem „izvodi li S riječ w (i kako)" razdvajamo (osim za trivijalni slučaj $w = \varepsilon$ ekvivalentan pitanju je li $S \to \varepsilon$ u $\mathcal{G}$) na manje probleme, općenitog tipa „izvodi li A podriječ $w_{i:j} = \alpha_i\alpha_{i+1} \cdots \alpha_{j-1}$ od w (i kako)", skraćenog zapisa $A?w_{i:j}$ (gdje je $A \in V$, te $0 \leq i < j \leq n = |w|$).

Takvih problema ima konačno mnogo, konkretno $\binom{n+1}{2} \cdot \#V$, i možemo ih rješavati redom, poredano rastuće po $|w_{i:j}| = j - i$. Njihova rješenja pišemo u tablicu (dinamičko programiranje).

Za početak, elementarni problem $A?w_{i:i+1} = \alpha_i$ riješimo gledajući postoji li u $\mathcal{G}$ pravilo čitanja $A \to \alpha_i$. U tablicu zapisujemo samo `bool` vrijednost, `True` ili `False`.

CYK algoritam — korak

Zamislimo sad da smo već riješili sve probleme oblika $X?w_{\ell:m}$ za $m - \ell < n$, i promotrimo problem $A?w_{i:j}$, gdje je $j - i = n$. Očito, svako rješenje tog problema mora početi pravilom širenja $A \to BC$ za neke $B, C \in V \setminus \{S\}$.

Budući da B i C nisu S, pozitivne su (ne izvode ε), pa znamo da podriječi od $w_{i:j}$ koje B i C izvode moraju biti strogo kraće od n. To znači da postoji $k \in \langle i..j \rangle$ takav da $B \Rightarrow^* w_{i:k}$ i $C \Rightarrow^* w_{k:j}$.

Primijetimo da smo (jer su $k - i$ i $j - k$ manji od n) probleme $B?w_{i:k}$ i $C?w_{k:j}$ već riješili. Ako takvi izvodi postoje, zapišemo u tablicu B, C i k (može biti više mogućnosti).

Na kraju, $w \in L(\mathcal{G})$ ako i samo ako problem $S?w_{0:n}$ ima rješenje, što jednostavno pogledamo u tablici. Algoritam CYK je polinoman (kojeg stupnja?), pa je A_{BK} polinomno odlučiv.

CYK algoritam — konstrukcija stabla parsiranja

Kao što rekosmo, često nam treba više od same informacije je li $w \in L(\mathcal{G})$. Srećom, dodatne informacije koje smo pisali u tablicu omogućuju i rekonstrukciju čitavog stabla parsiranja za w iz $\mathcal{G}$.

Dakle, pretpostavimo da je riječ u jeziku, odnosno da na mjestu $(S, 0, n)$ tablice postoji bar jedna trojka (B, C, k). Pogledamo u tablici mjesta $(B, 0, k)$ i (C, k, n) (ona moraju postojati), te ih rekurzivnim pozivom pretvorimo u stabla T_1 i T_2. Vratimo stablo s korijenom S i redom podstablima T_1 i T_2.

Kad dođemo do faze da u tablici gledamo rješenja od $A?w_{i:i+1}$, znamo da tamo mora pisati `True`, te vratimo stablo koje se sastoji od korijena A i njegovog djeteta α_i.

CYK algoritam — kritički pogled

Najveći nedostatak je što zapravo nismo dobili stablo parsiranja za *početnu* gramatiku, osim ako je ona već bila u ChNF. Pretvaranje u ChNF može dosta promijeniti gramatiku, odnosno uvesti mnoga „irelevantna" pravila, tako da je na kraju teško ustanoviti što u stablu parsiranja predstavlja pravu informaciju o izvodu iz početne gramatike, a što je artefakt njenog pretvaranja u ChNF.

Također, nedostatak je što kod jako višeznačnih gramatika možemo imati eksponencijalno mnogo različitih izvoda za jednu riječ, čime se bitno povećava potrošnja memorije. Ipak, za jednoznačne gramatike (mogu biti i višeznačne, ali da nas zanima samo jedan izvod), CYK je kvadratne prostorne i kubne vremenske složenosti.

Formalizacija algoritama

Vidjeli smo da su problemi prihvaćanja za klase regularnih, beskontekstnih i kontekstnih jezika odlučivi, tako što smo napisali (neformalne) algoritme za njih. Možemo li generalizirati naše automate u *računala* koja mogu izvršavati takve algoritme?

[Zašto bismo to radili? Prvo, ne ovisimo o konkretnom programskom jeziku, niti o intuitivnom shvaćanju pseudokoda — razmišljamo svevremenski. Drugo i važnije, dobivamo jednostavnu refleksiju (samoreferiranje) koje omogućuje dijagonalizaciju i time dokaze **nepostojanja algoritma** za neke probleme.]

Pokazuje se da je dovoljno OAu „otvoriti" traku s jedne strane.

Moćniji strojevi — vrste

Za razliku od dosadašnjih primjera automata koji su uglavnom bili samo s jednom određenom svrhom, *Turingovi strojevi* (TS) su univerzalna računala i koriste se za razne svrhe:

- prepoznavači (*recognizers*) — TP, za prepoznavanje jezika
- odlučitelji (*deciders*) — TO, za odlučivost problema
- nabrajači (*enumerators*) — TE, za generiranje jezika

Svi imaju sličnu definiciju i rad, razlikuju se samo u završnim odnosno istaknutim stanjima.

Precizno ćemo definirati TP, a za ostale ćemo samo navesti razlike.

Napomena: TP se mogu koristiti i za *računanje jezičnih funkcija*. Na ovom kolegiju reći ćemo samo nešto malo o tome, a puno više detalja čut ćete na kolegiju IZRAČUNLJIVOST.

Determinizam i nepozitivnost

Nakon izleta u svijet (bes)kontekstnih jezika gdje su automati bili isključivo nedeterministički, vraćamo se svijetu koji smo upoznali s regularnim jezicima:

- Turingovi strojevi su podrazumijevano *deterministički*
- nedeterministički TS su ekvivalentni deterističkima
- gramatike više nisu primarni način zadavanja jezika
  (za klasu rekurzivnih jezika nemamo odgovarajuće gramatike)

Turingov prepoznavač — definicija

Turingov prepoznavač $\mathcal{T} = (Q, \Sigma, \Gamma, \sqcup, \delta, q_0, q_\checkmark)$ ima:

- ulaznu abecedu Σ
- radnu abecedu $\Gamma \supset \Sigma$ s istaknutim znakom $\sqcup \in \Gamma \setminus \Sigma$
- skup stanja Q, s istaknutim početnim i završnim stanjem
- funkciju prijelaza $\delta : (Q \setminus \{q_\checkmark\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$

Konfiguracija od $\mathcal{T}$ je uređena trojka $(q, n, t) \in Q \times \mathbb{N} \times \Gamma_\sqcup^\mathbb{N}$, gdje smatramo $0 \in \mathbb{N}$, te $\Gamma_\sqcup^\mathbb{N} := \{(t : \mathbb{N} \to \Gamma) : t^{-1}[\Gamma \setminus \{\sqcup\}] \text{ je konačan}\}$.

Uvjet da je traka od nekog mjesta nadalje konstanta $\sqcup$, osigurava da traka ima

Turingov prepoznavač — definicija

Turingov prepoznavač $\mathcal{T} = (Q, \Sigma, \Gamma, \sqcup, \delta, q_0, q_\checkmark)$ ima:

- ulaznu abecedu Σ
- radnu abecedu $\Gamma \supset \Sigma$ s istaknutim znakom $\sqcup \in \Gamma \setminus \Sigma$
- skup stanja Q, s istaknutim početnim i završnim stanjem
- funkciju prijelaza $\delta : (Q \setminus \{q_\checkmark\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$

Konfiguracija od $\mathcal{T}$ je uređena trojka $(q, n, t) \in Q \times \mathbb{N} \times \Gamma_\sqcup^\mathbb{N}$, gdje smatramo $0 \in \mathbb{N}$, te $\Gamma_\sqcup^\mathbb{N} := \{(t : \mathbb{N} \to \Gamma) : t^{-1}[\Gamma \setminus \{\sqcup\}] \text{ je konačan}\}$.

Uvjet da je traka od nekog mjesta nadalje konstanta $\sqcup$, osigurava da traka ima prebrojivo mnogo. Stoga konfiguracija ima

Turingov prepoznavač — definicija

Turingov prepoznavač $\mathcal{T} = (Q, \Sigma, \Gamma, \sqcup, \delta, q_0, q_\checkmark)$ ima:

- ulaznu abecedu Σ
- radnu abecedu $\Gamma \supset \Sigma$ s istaknutim znakom $\sqcup \in \Gamma \setminus \Sigma$
- skup stanja Q, s istaknutim početnim i završnim stanjem
- funkciju prijelaza $\delta : (Q \setminus \{q_\checkmark\}) \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$

Konfiguracija od $\mathcal{T}$ je uređena trojka $(q, n, t) \in Q \times \mathbb{N} \times \Gamma_\sqcup^{\mathbb{N}}$, gdje smatramo $0 \in \mathbb{N}$, te $\Gamma_\sqcup^{\mathbb{N}} := \{(t : \mathbb{N} \to \Gamma) : t^{-1}[\Gamma \setminus \{\sqcup\}] \text{ je konačan}\}$.

Uvjet da je traka od nekog mjesta nadalje konstanta $\sqcup$, osigurava da traka ima prebrojivo mnogo. Stoga konfiguracija ima prebrojivo mnogo.

Prihvaćanje i prepoznavanje

Kažemo da konfiguracija (p, m, s) prelazi u konfiguraciju (q, n, t) od $\mathcal{T}$ ako uz oznake $(q, \gamma, d) := \delta(p, s(m))$ vrijedi $n = \max\{m + d, 0\}$, $t(m) = \gamma$, a $t(k) = s(k)$ za sve $k \neq m$.

Ako je $w \in \Sigma^*$, $\mathcal{T}$-izračunavanje s w je konačan ili beskonačan niz konfiguracija $c_0 := (q_0, 0, w\sqcup\sqcup\dots), c_1, c_2, \dots$, takvih da za svaki i, ili je c_i *završna* (stanje joj je $q_\checkmark$), ili c_i prelazi u c_{i+1}.

Za svaku $w \in \Sigma^*$ postoji jedinstveno $\mathcal{T}$-izračunavanje s w (to upravo znači da je $\mathcal{T}$ deterministički). Kažemo da $\mathcal{T}$ *prihvaća* w ako to izračunavanje *stane*, odnosno ako je gore opisani niz $c_0, c_1, \dots, c_n$ konačan i c_n je završna konfiguracija.

Kažemo da $\mathcal{T}$ *prepoznaje* jezik $L(\mathcal{T}) := \{w \in \Sigma^* : \mathcal{T} \text{ prihvaća } w\}$. Kažemo da je jezik *rekurzivno prebrojiv* (ili da je u klasi RE) ako postoji Turingov prepoznavač koji ga prepoznaje.

Turingov odlučitelj

Pored svih komponenti Turingovog prepoznavača, *odlučitelj* $\mathcal{D}$ ima još jedno završno stanje q_X (konfiguracija sa stanjem q_X također završava izračunavanje, a $\delta(q_X, \alpha)$ nije definirano ni za koji α), te svojstvo da **za svaku** $w \in \Sigma^*$, $\mathcal{D}$-izračunavanje s w stane.

Ako je u zadnjoj konfiguraciji stanje $q_\checkmark$, kažemo (kao i prije) da $\mathcal{D}$ *prihvaća* w, a ako je q_X, kažemo da $\mathcal{D}$ *odbija* w.

Kažemo da je jezik *rekurzivan* (ili da je u klasi *Rek*) ako postoji Turingov odlučitelj koji ga prepoznaje.

$Rek \subseteq RE$, jer svaki odlučitelj možemo lako pretvoriti u prepoznavač tako da

Turingov odlučitelj

Pored svih komponenti Turingovog prepoznavača, *odlučitelj* $\mathcal{D}$ ima još jedno završno stanje q_X (konfiguracija sa stanjem q_X također završava izračunavanje, a $\delta(q_X, \alpha)$ nije definirano ni za koji α), te svojstvo da **za svaku** $w \in \Sigma^*$, $\mathcal{D}$-izračunavanje s w stane.

Ako je u zadnjoj konfiguraciji stanje $q_\checkmark$, kažemo (kao i prije) da $\mathcal{D}$ *prihvaća* w, a ako je q_X, kažemo da $\mathcal{D}$ *odbija* w.

Kažemo da je jezik *rekurzivan* (ili da je u klasi *Rek*) ako postoji Turingov odlučitelj koji ga prepoznaje.

Rek $\subseteq$ *RE*, jer svaki odlučitelj možemo lako pretvoriti u prepoznavač tako da proglasimo q_X nezavršnim stanjem, i dodefiniramo $\delta(q_X, \alpha) := (q_X, \alpha, 1)$. Na taj način, kad prepoznavač uđe u stanje q_X, zauvijek će ostati u tom stanju, te će efekt biti isti kao i u slučaju odlučitelja (iako q_X više nije završno): neće prihvatiti riječ.

Poboljšanja Turingovih strojeva naslijeđena od OA

Kao i u slučaju OA, postoje brojne varijacije u definiciji Turingovih strojeva. Kako su to uglavnom „poboljšanja" odnosno dodavanja novih mogućnosti, a Turingovi strojevi su univerzalna računala (mogu simulirati proizvoljna računala, pa tako i poboljšane strojeve), prirodno je očekivati da će svi ti modeli biti ekvivalentni.

Sva poboljšanja OA koja smo upoznali (nemijenjanje trake, više redaka, detekcija *lijevog* ruba, stajanje na mjestu, više završnih stanja, konačne varijable, više traka) mogu se jednako primijeniti na TS. Sada ćemo vidjeti i neka druga.

Detekcija zadnje posjećene ćelije

Kod JOA, koristili smo detekciju desnog ruba (i dodatni „backup" redak) da bismo mu objasnili kako vratiti traku u prvotno stanje. TS nema desni rub trake, ali sličnu ulogu ima zadnja posjećena ćelija.

O njoj možemo voditi računa tako da uvedemo još jedan redak s oznakama, te svaki prijelaz (p, α, q, β, d) postaje dva prijelaza $\left(p, \underset{m}{\alpha}, q, \underset{\hat{}}{\beta}, d\right)$, gdje je $m \in \{\hat{}, \sqcup\}$. Lako je vidjeti da je u svakom izračunavanju taj redak s oznakama uvijek oblika $\hat{} \cdots \hat{} \sqcup \cdots$, te da su jedino označene ćelije mogle biti promijenjene. Dakle, kad trebamo vratiti traku u prvobitno stanje, odemo desno do prvog neoznačenog znaka, te se vratimo skroz ulijevo, vraćajući znakove preko kojih prelazimo iz „backup" retka (odnosno brišemo ih ako je „backup" redak prazan).

Obostrano neograničena traka

Još jedno poboljšanje osnovnog modela TSa je omogućavanje da čita odnosno piše i lijevo od početnog položaja. Formalno, njegovo stanje trake je element od $\Gamma_{\sqcup}^{\mathbb{Z}}$ (i dalje zahtijevamo da je samo konačno mnogo znakova na traci različito od $\sqcup$).

To se jednostavno simulira pomoću dva Γ-retka i jedne konačne varijable, $t \in \{-1, 1\}$ inicijalizirane na 1. Znakovi se čitaju iz prvog retka ako je $t = 1$ a iz drugog ako je $t = -1$. Pomak udesno kad je $t = 1$ je obični pomak udesno. Pomak ulijevo kad je $t = -1$ je također obični pomak *udesno*.

Preostala dva tipa pomaka realiziraju se tako da stroj prvo detektira je li na lijevom rubu, te ako jest, postavlja $t := -t$ i ostaje na mjestu. Ako nije, normalno se pomiče lijevo.

Nedeterministički Turingov stroj (prepoznavač — NTP)

Lako je definirati NTP: samo umjesto funkcije prijelaza zahtijevamo relaciju $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 1\}$. Kao i obično, NTP $\mathcal{N}$ *prihvaća* riječ w ako *postoji* $\mathcal{N}$-izračunavanje s w koje stane.

Ono što je teško, je dokazati da svaki NTP ima ekvivalentan TP. Ideja dokaza: simuliramo NTP pomoću trotračnog TP.

Na prvoj traci nalazi se ulazna riječ, i ta traka se nikada ne mijenja. Ona služi samo kao „backup" iz kojeg se druga traka vraća na početno stanje prilikom svakog prebacivanja „s grane na granu" izračunavanja. Dakle, druga traka je ona na kojoj se doista odvija simulacija (jedne grane) izračunavanja. Na trećoj držimo samu trenutnu granu, kao niz (iz Δ^*) prijelaza ($\Gamma_3 := \Delta \dot{\cup} \{\textvisiblespace_3\}$).

Simulacija NTP pomoću TP — pseudokod

nekako (leksikografski,...) uredi $\mathcal{N}.\Delta =: (\delta_1, \delta_2, \ldots, \delta_n)$
while True:
    isprazni t_2; prepiši t_1 na t_2; premotaj t_3 na početak
    $s := \mathcal{N}.q_0$
    while $((p, \alpha, q, \beta, d) :=$ read$(t_3)) \neq \sqcup$:
        if read$(t_2) = \alpha \wedge s = p$:
            $s := q$; write(t_2, β); move(t_2, d); move$(t_3, 1)$
    if $s = \mathcal{N}.q_\checkmark$: prihvati
    premotaj t_3 na početak
    while read$(t_3) = \delta_n$: write(t_3, δ_1); move$(t_3, 1)$
    if read$(t_3) = \sqcup$: write(t_3, δ_1)
    else: transformiraj read$(t_3) = \delta_i$ za $i \in [1..n\rangle$ u δ_{i+1}
        write(t_3, δ_{i+1})

Nedeterministički Turingov odlučitelj — NTO

NTO je nedeterministički stroj (ima relaciju prijelaza) koji je i odlučitelj: ima stanje q_X, te svojstvo da **svako** njegovo izračunavanje stane ili u $q_\checkmark$ ili u q_X. Ovdje doista mislimo *svako*: iako nedeterministički stroj „zna" napraviti pravi odabir u svakoj situaciji, NTO se mora zaustaviti nakon konačno mnogo koraka čak i onda kad ne napravi pravi odabir.

Takva restriktivna definicija služi da bismo imali ekvivalenciju s deterministčkim TO. Puno je teže to dokazati [seminar Nore Berdalović] — primijetimo samo da trotračna simulacija nije dovoljna, jer kada dobijemo $s = \mathcal{N}.q_X$, ne znamo je li to finalno odbijanje, ili treba dalje tražiti. Potrebno je voditi računa o svim granama (kako?) i odbiti akko sve završe u q_X.

Uniformno ograničeni NTO — $Kon \subseteq Rek$

Ipak, ako imamo *uniformnu* ogradu na broj koraka koje NTO napravi po svim granama od početne konfiguracije do završne (prihvaćanja ili odbijanja), ekvivalenciju s TO možemo dokazati puno lakše: simulacija samo treba *brojati korake* na zasebnoj traci, i kada taj broj prestigne unaprijed definiranu granicu, odbiti riječ. [Tehnički detalj: lakše je brojati unatrag.]

Takva granica, vidjeli smo, postoji u slučaju kontekstnih jezika: za OA $\mathcal{O} = (Q, \Sigma, \Gamma, \Delta, q_0, q_\checkmark)$ i $w \in \Sigma^*$, ako $\mathcal{O}$ ne može prihvatiti w u najviše $\#Q \cdot (|w| + 1) \cdot (\#\Gamma)^{|w|}$ koraka, tada $w \notin L(\mathcal{O})$, pa je simulacija također može odbiti.

Dakle, svaki OA se može simulirati *uniformno ograničenim* NTO, a ovaj pak običnim TO, što znači da je svaki kontekstni jezik rekurzivan.

Turingovi nabrajači (enumeratori — TE)

Turingovi strojevi mogu služiti i za generiranje riječi (slično kao gramatike). Turingov enumerator $\mathcal{E}$ pored radne trake ima još jednu, *izlaznu* traku. Nema ulaza (trake su na početku prazne). Umjesto završnog stanja, $\mathcal{E}$ ima *izlazno* stanje $q_{\hookleftarrow}$.

Kako nema završno stanje, njegov rad

$$c_0 := (q_0, 0, \sqcup\sqcup \ldots, 0, \sqcup\sqcup \ldots), c_1, c_2, \ldots$$

je uvijek beskonačan, te kažemo da $\mathcal{E}$ *izvodi* $w \in \Sigma^*$ ako postoji $i \in \mathbb{N}$ takav da je $c_i = (q_{\hookleftarrow}, \ldots, w\sqcup\sqcup \ldots)$. Kažemo da $\mathcal{E}$ *generira* jezik $L(\mathcal{E}) := \{w \in \Sigma^* : \mathcal{E} \text{ izvodi } w\}$.

Bili smo, na neki način, već konstruirali primjer enumeratora (za skup Δ^*), kod simulacije NTP.

Zadatak: konstruirajte enumerator za I^*, gdje je I neki fiksni znak.

Simulacija rada fiksnog Turingovog stroja

Važan i vrlo netrivijalan **teorem** enumeracije: jezik je *RE* ako i samo ako postoji TE koji ga generira. (Povijesno, tako su uvedeni *RE* jezici: engleski *recursively enumerable* dolazi od enumeratora.)

Za dokaz tog teorema, treba nam pojam *simulacije* nekog TS.

Primjer smo također već vidjeli kod priče o NTP: simulator drži stanje simuliranog stroja u konačnoj varijabli, a njegove prijelaze (konačno mnogo njih) u svojoj funkciji prijelaza, te njegovo stanje trake na svojoj zasebnoj traci. Drugim riječima, u pseudokodu za algoritam koji obavlja neki k-tračni Turingov stroj, možemo koristiti i naredbu „simuliraj jedan korak od $\mathcal{T}$ na i-toj traci", gdje je $\mathcal{T}$ unaprijed zadani (jednotračni) TS, te $i \in [1..k]$. (Podrazumijevamo da ta naredba ne radi ništa ako je $\mathcal{T}$ već u završnom stanju.)

Dokaz teorema enumeracije, smjer $\Leftarrow$

Neka je $\mathcal{E}$ proizvoljni TE. Pišemo pseudokod za trotračni TP koji prepoznaje $L(\mathcal{E})$, simulirajući $\mathcal{E}$.

$s := \mathcal{E}.q_0$
while True:
    # simuliraj korak od $\mathcal{E}$ na t_2 kao radnoj i t_3 kao izlaznoj
    transformiraj $(s, z, d, z', d') := \mathcal{E}.\delta(s, read(t_2), read(t_3))$
    write(t_2, z); write(t_3, z'); move(t_2, d); move(t_3, d')

    if $s = \mathcal{E}.q_{\hookleftarrow}$:
        usporedi t_1 (do prve praznine) sa t_3
        ako nema razlike: prihvati

Dokaz teorema enumeracije, smjer $\Rightarrow$

Neka je $\mathcal{T}$ TP za jezik L. Dajemo skicu pseudokoda za višetračni enumerator za isti jezik.

- Na prvoj traci brojimo redom, koristeći enumerator za I^*. Broj znakova I na t_1 označimo s n.
- Na drugoj traci, za svaki n, brojimo moguće ulaze za $\mathcal{T}$ od n do 0, inkrementirajući izlaznu traku svaki put.
- Na trećoj traci brojimo korake od n do 0, svaki put simulirajući na četvrtoj traci po jedan korak od $\mathcal{T}$ na svakom od n ulaza
- Kad god $\mathcal{T}$ prihvati ulaz, idemo u stanje $q_{\hookleftarrow}$.

Dokaz teorema enumeracije, smjer $\Rightarrow$ — pseudokod

```
while True:
    prepiši t_1 na t_2; premotaj t_2; isprazni izlaznu traku
    while read(t_2) = I :
        prepiši t_1 na t_3; premotaj t_3
        prepiši izlaznu traku na t_4; s := T.q_0
        while read(t_3) = I :
            simuliraj korak od T (u stanju s) na t_4
            obriši jedan I s kraja t_3
        if s = T.q_✓: ispiši (idi u q_↩)
        obriši jedan I s kraja t_2
        inkrementiraj riječ iz (T.Σ)* na izlaznoj traci
    dodaj jedan I na kraj t_1
```

Potpuna simulacija rada Turingovog stroja

Uvodimo naredbu „simuliraj rad od $\mathcal{T}$ na sadržaju i-te trake" kao pokratu za sljedeći pseudokod, gdje je k indeks neke trake koja se ne pojavljuje nigdje drugdje u kodu:

isprazni t_k; prepiši t_i na t_k (do prve praznine)
$s := \mathcal{T}.q_0$
while $s \neq \mathcal{T}.q_\checkmark$ ($\wedge\, s \neq \mathcal{T}.q_X$ ako je $\mathcal{T}$ odlučitelj) :
    simuliraj korak od $\mathcal{T}$ (u stanju s) na t_k

Za razliku od simulacije jednog koraka, potpuna simulacija TP može nikad ne završiti — jasno, potpuna simulacija TO mora uvijek završiti.

Pomoću potpune simulacije možemo vrlo jednostavno dokazati zatvorenost klase *Rek* na skupovne operacije.

Klasa rekurzivnih jezika je zatvorena na $\cap$, $\cup$, ^{c}, $\setminus$, $\triangle$, ...

Za sve skupovne operacije dokazuje se tako da odsimuliramo rad oba odlučitelja na sadržaju iste trake (operaciju smo definirali tako da sačuva sadržaj trake, jer se prepiše na novu traku), i na kraju konzultiramo tablicu istinitosti odgovarajućeg logičkog veznika.

Primjera radi, dokažimo zatvorenost na skupovnu razliku. Neka su L_1 i L_2 rekurzivni jezici. To znači da postoje TO $\mathcal{T}_1$ i $\mathcal{T}_2$ koji prepoznaju L_1 i L_2 redom. Tada TO s pseudokodom

$s_1 := \mathcal{T}_1.q_0$; $s_2 := \mathcal{T}_2.q_0$
simuliraj rad od $\mathcal{T}_1$ (koristeći stanje s_1) na sadržaju od t_1
simuliraj rad od $\mathcal{T}_2$ (koristeći stanje s_2) na sadržaju od t_1 (!)
if $s_1 = \mathcal{T}_1.q_\checkmark \wedge s_2 = \mathcal{T}_2.q_X$: prihvati; else: odbij

očito uvijek stane, te prepoznaje upravo jezik $L_1 \setminus L_2$.

Zatvorenost klase *Rek* na jezične operacije

Primjera radi, pokažimo zatvorenost na Kleenejevu zvijezdu. Neka je $\mathcal{T}$ TO za L, te w riječ za koju moramo utvrditi je li iz L^*.

Ako je $w = \varepsilon$, odmah je prihvaćamo. Inače, w se sigurno može rastaviti na jednu ili više *neprazznih* podriječi. Takvih rastava ima konačno mnogo, i sve ih možemo isprobati: samo trebamo za svaki $k \in [1..|w|\rangle$ odlučiti hoćemo li rastaviti riječ na tom mjestu. Ako postoji rastav gdje su sve pripadne podriječi u L, prihvatimo w, inače odbacimo w.

Za zatvorenost na konkatenaciju, samo trebamo rastav na dvije riječi (na jednom mjestu $k \in [0..|w|]$), a takvih očito ima konačno. Kleenejev plus možemo izraziti pomoću zvijezde i konkatenacije. Kleenejev upitnik je unija s jezikom ε, koji je očito rekurzivan. (**Zadatak**: napišite TO za ε.)

Zatvorenost *Rek* na Kleenejevu zvijezdu — pseudokod

if read(t_1) $= \sqcup$: prihvati // $w = \varepsilon$
for $p \in \mathcal{P}([1..|w|\rangle)$:
    $k := \#p$; $(a_1, \ldots, a_k) :=$ sort(p); $a_0 := 0$; $a_{k+1} := |w|$
    svi := True
    for $i \in [0..k]$:
        prepiši t_1 (pozicije $\in [a_i, a_{i+1}\rangle$) na t_2
        $s := \mathcal{T}.q_0$; simuliraj rad od $\mathcal{T}$ (stanje s) na sadržaju od t_2
        if $s = \mathcal{T}.q_X$: svi := False
    if svi: prihvati
odbij

Zatvorenost klase RE na uniju i presjek

Neka su $L, M \in RE$. To znači da postoje TP $\mathcal{S}$ i $\mathcal{T}$ koji prepoznaju L i M redom. Tada TP s pseudokodom

> simuliraj $\mathcal{S}$ na sadržaju od t_1; simuliraj $\mathcal{T}$ na sadržaju od t_1
> prihvati

prepoznaje $L \cap M$ (jer za riječi koje nisu u L prvi korak će biti beskonačna petlja, a za riječi koje nisu u M to će biti drugi korak).

Za $L \cup M$, moramo koristiti simulaciju korak po korak (zašto?):

> prepiši t_1 na t_2; prepiši t_1 na t_3; $s := \mathcal{S}.q_0$; $s' := \mathcal{T}.q_0$
> while True:
> simuliraj korak od $\mathcal{S}$ (stanje s) na t_2
> simuliraj korak od $\mathcal{T}$ (stanje s') na t_3
> if $s = \mathcal{S}.q_\checkmark \vee s' = \mathcal{T}.q_\checkmark$: prihvati

Postov teorem

Klasa RE nije zatvorena na komplement, ali da bismo to dokazali treba nam neki postupak kojim ćemo dokazati da neki jezik nije RE (lema o pumpanju ovdje ne postoji). Zasad možemo dokazati

Teorem: L je rekurzivan ako i samo ako su L i L^c RE.

- $\Rightarrow$ Po zatvorenosti Rek na skupovne operacije, ako je L rekurzivan, tada je i L^c takav, a dokazali smo $Rek \subseteq RE$.
- $\Leftarrow$ Neka je $\mathcal{S}$ TP za L, a $\mathcal{T}$ TP za L^c. Pseudokod TO za L je:

  prepiši t_1 na t_2; prepiši t_1 na t_3; $s := \mathcal{S}.q_0$; $s' := \mathcal{T}.q_0$
  while True:
    simuliraj korak od $\mathcal{S}$ (stanje s) na t_2
    simuliraj korak od $\mathcal{T}$ (stanje s') na t_3
    if $s = \mathcal{S}.q_\checkmark$: prihvati
    else if $s' = \mathcal{T}.q_\checkmark$: odbij

Opće gramatike

Zatvorenost klase *RE* na jezične operacije najlakše ćemo dokazati koristeći gramatike koje generiraju rekurzivno prebrojive jezike. Jedini uvjet na pravila u općoj gramatici je: na lijevoj strani mora biti bar jedna varijabla. Primjerice, $AB \to AB$, $\mathtt{a}B \to \mathtt{b}$ i $\mathtt{a}B\mathtt{c} \to \varepsilon$ su dozvoljena pravila općih gramatika, dok $\mathtt{a} \to \mathtt{b}$ i $\varepsilon \to B$ nisu.

Jednakim sredstvima kao u slučaju OA $\leftrightarrow$ MG, možemo dokazati:
Teorem: Jezik je *RE* ako i samo ako ga generira neka OG.

[$\Rightarrow$: pišemo pravila gramatike koja pretvaraju konfiguracije (jednostavnog) TPa; ne moramo koristiti objedinjavanje znakova; $\Leftarrow$: koristimo NTP da „pogodimo" izvod, te iskoristimo ekvivalenciju TP i NTP]

Sada kombiniranje gramatika (sasvim jednako kao za BKG) daje zatvorenost *RE* na konkatenaciju i Kleenejeve operatore.

Ograničenost abecede

Već smo razmatrali probleme, ali neformalno (intuitivno). Formalno, htjeli bismo pomoću Turingovog stroja simulirati neki stroj, ali ne fiksan, već zadan kao ulaz. Na prvi pogled samo treba simulirani stroj i njegov ulaz prikazati kao nizove znakova. No, abeceda simulatora mora biti unaprijed fiksirana i konačna. Tada nije moguće direktno reprezentirati ulaz *simuliranog* stroja čim njegova abeceda ima znak koji abeceda simulatora nema.

Prema tome, ne možemo reprezentirati određene objekte direktno, moramo ih *kodirati*. Kodirat ćemo samo strojeve i riječi jer ćemo samo njih davati kao ulaze u ostatku predavanja.

Prvi korak kodiranja strojeva i riječi: prirodno preslikavanje

U prvom koraku, sva stanja i znakove (ulaznih i/ili radnih) abeceda reprezentiramo prirodnim brojevima pomoću *prirodnog preslikavanja*, injekcije oblika $h : D \to \mathbb{N}$. Ovdje D ovisi o tome što kodiramo; za DKA i NKA je $D = Q \cup \Sigma$, za (J)PA, (J)OA i TP/TO/TE je $D = Q \cup \Sigma \cup \Gamma$, a za riječi nad Σ je $D = \Sigma$.

Jasno je kako prirodno preslikavanje proširujemo na riječi i jezike:

$$h(\alpha_1 \cdots \alpha_n) := h(\alpha_1) \cdots h(\alpha_n) \in h(\Sigma)^*$$
$$h(L) := \{h(w) : w \in L\} \subseteq h(\Sigma)^*$$

No, kako ga proširujemo na strojeve, tj. od $\mathcal{M}$ dobijemo $h(\mathcal{M})$? Skup stanja postaje $h(Q)$, (ulazna) abeceda $h(\Sigma)$, a radna abeceda (ako ju $\mathcal{M}$ ima) $h(\Gamma)$. Svi ostali dijelovi stroja se na odgovarajući način dobivaju iz njih (recimo, $(p, \alpha, t, \beta, s) \in \Delta$ postaje $(h(p), h(\alpha), h(t), h(\beta), h(s))$). Slično bismo postupali i kod drugih konačnih matematičkih struktura poput gramatika, grafova, itd.

Drugi korak kodiranja strojeva i riječi: univerzalna abeceda

Primijetite da za svaki stroj $\mathcal{M}$, $h(\mathcal{M})$ je stroj takav da vrijedi: $\mathcal{M}$ prepoznaje jezik L ako i samo ako $h(\mathcal{M})$ prepoznaje $h(L)$.

Primjer: jedan *prirodni DKA* ($h(\mathcal{M})$ za neki DKA $\mathcal{M}$) je
$\big(\{0,1,2\},\{0,1\},$
$\{(0,0,0),(0,1,0),(1,0,1),(1,1,0),(2,0,1),(2,1,0)\},2,\{1\}\big)$.

Univerzalna abeceda je $\mathbb{\Sigma} := \{\texttt{.}, \texttt{I}, \texttt{(}, \texttt{)}\}$. Definiramo *kodiranje* c:

$$c(n) := \texttt{I.}^n, \text{ za } n \in \mathbb{N}$$
$$c((t_1,\ldots,t_k)) := \texttt{(}\, c(t_1)\cdots c(t_k)\,\texttt{)}$$
$$c(\{t_1,\ldots,t_k\}) := \texttt{(}\, c(t_1)\cdots c(t_k)\,\texttt{)}, \text{ za } t_1 < \cdots < t_k \text{ leksikografski}$$

Primjer: gornji DKA kodira se u `((II.I..)(II.)`
`((III)(II.I)(I.II.)(I.I.I)(I..II.)(I..I.I))I..(I.))`.

Konačan niz od k objekata (strojeva i/ili riječi) kodiramo riječju $\langle o_1,\ldots,o_k\rangle := c(h(o_1))\cdots c(h(o_k))$ nad $\mathbb{\Sigma}$ (za neki fiksiran h).

[U praksi možemo uzeti: $h :=$ Unicode, $c :=$ UTF-8, $\mathbb{\Sigma} := \{0,1\}$.]

Problemi — formalno

Jezike nad univerzalnom abecedom koji sadrže kodove konačnih nizova strojeva i/ili riječi nazivamo *problemima*. Uočimo sljedeće: kako bismo zadali konkretan problem, u principu je potrebno odabrati prirodno preslikavanje. No, taj odabir uopće nije bitan zahvaljujući našoj definiciji proširenja prirodnog preslikavanja.

Gornja tvrdnja bit će odmah jasna iz primjera jednog problema:

$$A_{\text{DKA}} := \{\langle \mathcal{M}, w\rangle : \mathcal{M} \text{ je DKA} \wedge w \in L(\mathcal{M})\}.$$

Za $(\mathcal{M}, w)$ kao gore te sva prirodna preslikavanja h_1 i h_2, očito je

$$c\big(h_1(\mathcal{M})\big)c\big(h_1(w)\big) \in A_{\text{DKA}} \iff c\big(h_2(\mathcal{M})\big)c\big(h_2(w)\big) \in A_{\text{DKA}}.$$

Naime, drugačije smo kodirali znakove u w, ali smo ih *jednako tako* drugačije kodirali i unutar $\mathcal{M}$.

Za problem koji je (kao jezik) rekurzivan još kažemo da je *odlučiv*, a za problem koji je (kao jezik) rekurzivno prebrojiv još kažemo da je *poluodlučiv*. Uskoro ćemo formalno dokazati da je A_{DKA} odlučiv.

Konvencija o razvrstavanju

Za svaki k, moguće je napisati pseudo-potprogram za $(k+2)$-tračni Turingov stroj, koji očekuje k kodiranih objekata (riječ $\langle o_1, \ldots, o_k \rangle$) na prvoj traci (odbija riječ ako nije tog oblika), te ih raspoređuje na sljedećih k trakâ.

Granice je lako prepoznati, jer svaki kodirani objekt:

- ili počinje s `I` — tada čitamo `.` dok ne dođemo do `I` ili `(`
- ili počinje s `(` — tada koristimo posljednju (pomoćnu) traku kao stog, da vidimo kad smo došli do odgovarajuće `)`.

Ubuduće, kad god nam je jezik zadan kao

$$L := \{\langle o_1, \ldots, o_k \rangle : \text{neko svojstvo od } o_1, \ldots, o_k\},$$

podrazumijevamo da je na početku već izvršen takav potprogram.

Beskonačne varijable

Znamo da Turingovom stroju možemo dati mogućnost korištenja varijable čija je vrijednost u svakom trenutku element nekog unaprijed zadanog konačnog skupa. Tipični primjer je stanje nekog unaprijed zadanog automata $\mathcal{M}$ koji simuliramo.

Ali ako nam je $\mathcal{M}$ zadan kao ulaz, tada iako je njegov skup stanja konačan, on nije unaprijed zadan i poznat prilikom konstrukcije našeg Turingovog stroja. To znači da ga ne možemo pamtiti u stanju (kao što smo dosad činili s konačnim varijablama) — ali možemo ga pamtiti na zasebnoj traci! Sve što trebamo je mogućnost prepisivanja neke riječi na tu traku, te uspoređivanje neke riječi sa sadržajem te trake, a to je oboje očito izvedivo na običnom višetračnom Turingovom stroju.

Rad s dugačkim znakovima

Analogni pristup (beskonačna varijabla čuvana na zasebnoj traci, koja podržava prepisivanje i uspoređivanje) možemo primijeniti na čitane znakove iz $\mathcal{M}.\Gamma$. Ipak, kod *pisanja* trebamo biti oprezniji. Kako najčešće na takvoj traci nećemo držati samo jedan znak nego čitavu traku simuliranog stroja, a nisu svi znakovi iste duljine, prepisivanjem jednog znaka preko drugog možemo uništiti susjedne znakove.

Zato imamo još jednu posebnu traku na kojoj u unarnom zapisu $\mathtt{I}.^k$ držimo najveću duljinu k znaka iz $\mathcal{M}.\Gamma$. Na početku simulacije napravimo „pre-processing" u kojem odredimo k (kako?) i zapišemo ga na tu traku. Tada svaka „ćelija" na traci gdje simuliramo rad od $\mathcal{M}$ može biti dugačka *točno* k, te će znak $\gamma \in \mathcal{M}.\Gamma$ biti zapisan na traku kao $c(h(\gamma))\textvisiblespace^{k-|c(h(\gamma))|}$.

Primjer: prihvaća li zadani DKA zadanu riječ?

Dokažimo da je jezik

$$A_{\text{DKA}} := \{\langle \mathcal{M}, w\rangle : \mathcal{M} \text{ je DKA} \wedge w \in L(\mathcal{M})\}$$

rekurzivan. Prema konvenciji o razvrstavanju,
kod $\langle \mathcal{M}\rangle$ je na drugoj traci, a kod $\langle w\rangle$ je na trećoj
[pre-processing nije potreban jer DKA ne može pisati po traci].
Na četvrtoj traci držimo beskonačnu varijablu iz $\mathcal{M}.Q$
koja predstavlja trenutno stanje od $\mathcal{M}$ (inicijaliziranu na $\mathcal{M}.q_0$).
Na petoj držimo trenutno pročitani „dugački znak" s treće trake.

Dok god čitani znak na trećoj traci nije) (dok nismo došli do kraja riječi), prepišemo sljedeći dugački znak s treće trake na petu; konkateniramo (, četvrtu i petu traku na šestu traku; te unutar $\mathcal{M}.\delta$ nađemo podriječ koja počinje sadržajem šeste trake. Ostatak te podriječi (do)) zapišemo na četvrtu traku.

Primjer: prihvaća li zadani DKA zadanu riječ — pseudokod

Na kraju (kad pročitamo `)` na trećoj traci) pokušamo unutar $\mathcal{M}.F$ na drugoj traci (slijeva nadesno) naći sadržaj četvrte trake. Ako ga nema, odbijemo riječ. Ako ga nađemo, sljedeći znak na drugoj traci mora biti `.` (odbijemo riječ), `I` ili `)` (prihvatimo riječ).

Vidimo da je čak i tako jednostavan primjer prilično kompliciran za detaljno opisivanje. Naši pseudokodovi su puno više razine:

input: $\langle \mathcal{M}, w \rangle$, gdje je
    $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ DKA i $w \in \Sigma^*$ (inače odbij)
$q := q_0$
while (ima još znakova u w):
    $\alpha :=$ (sljedeći znak u w)
    $q := \delta(q, \alpha)$
if $q \in F$: prihvati
else: odbij

Problem praznosti jezika

Pogledajmo još neke probleme (vezane uz DKA, ali kasnije ćemo ih gledati i za druge sustave). Primjer problema s pseudokodom:

$$E_{\text{DKA}} := \{\langle \mathcal{M} \rangle : \mathcal{M} \text{ je DKA} \wedge L(\mathcal{M}) = \emptyset\}$$

na posebnu traku s oznakama prepiši $\mathcal{M}.Q$; označi $\mathcal{M}.q_0$
$m :=$ True
while m:
    $m :=$ False
    for $(p, \alpha, q) \in \mathcal{M}.\delta$:
        if p označen $\wedge \neg(q$ označen): označi q; $m :=$ True
for $p \in \mathcal{M}.F$:
    if p označen: odbij
prihvati

Problem jednakosti jezika

Još jedan primjer odlučivog problema je

$$EQ_{\text{DKA}} := \{\langle \mathcal{M}_1, \mathcal{M}_2 \rangle : \mathcal{M}_1 \text{ je DKA} \wedge \mathcal{M}_2 \text{ je DKA} \wedge \\ \wedge \mathcal{M}_1.\Sigma = \mathcal{M}_2.\Sigma \wedge L(\mathcal{M}_1) = L(\mathcal{M}_2)\}\,.$$

Odlučitelj treba, koristeći Kartezijevu konstrukciju, na posebnoj traci konstruirati kod prirodnog DKA koji prepoznaje jezik $L(\mathcal{M}_1) \triangle L(\mathcal{M}_2)$.

Tada treba taj kod dati odlučitelju za E_{DKA}, i prihvatiti ako on prihvati, a odbiti ako on odbije.
Prepoznajemo to kao *svođenje* problema EQ_{DKA} na problem E_{DKA}.
Sada pokušajmo formalizirati ideju svođenja.

Izračunljivost jezičnih funkcija; svođenje

Neka je Σ abeceda. *Jezična funkcija nad* Σ je parcijalna funkcija $\varphi : \Sigma^* \rightharpoonup \Sigma^*$ (domena $\mathcal{D}_\varphi$ je podskup od Σ^*, dakle jezik nad Σ).

Kažemo da prepoznavač $\mathcal{T}$ *računa* φ ako mu je ulazna abeceda Σ, vrijedi $L(\mathcal{T}) = \mathcal{D}_\varphi$, te za svaku riječ w iz tog jezika, traka završne konfiguracije $\mathcal{T}$-izračunavanja s w je $\varphi(w)\,\sqcup\sqcup\sqcup\ldots$.

Kažemo da je φ *Turing-izračunljiva* ako postoji TP koji je računa. [Više na kolegiju IZRAČUNLJIVOST!]

Neka su L i M jezici nad istom abecedom Σ (ili problemi, nad $\mathbb{\Sigma}$). Kažemo da se L *svodi* na M, i pišemo $L \preceq M$, ako postoji *totalna* ($\mathcal{D}_\varphi = \Sigma^*$) Turing-izračunljiva funkcija φ takva da za svaku riječ $w \in \Sigma^*$ vrijedi: $w \in L \iff \varphi(w) \in M$.

Svojstva relacije svedivosti $\preceq$

„Serijskim spajanjem" dva TP lako vidimo da je kompozicija Turing-izračunljivih funkcija ponovo Turing-izračunljiva.
Zato je relacija $\preceq$ tranzitivna (**zadatak**: dokažite da je refleksivna, ali nije parcijalni uređaj). Na sličan način dobivamo sljedeći

Teorem: Neka su L i M problemi takvi da vrijedi $L \preceq M$.

- Ako je M (polu)odlučiv, tada je L (polu)odlučiv.
- Ako L nije (polu)odlučiv, tada M nije (polu)odlučiv.

Skica dokaza: Za prvu tvrdnju, ako imamo TP $\mathcal{T}$ koji računa totalnu φ koja svodi L na M, te TP (ili TO) $\mathcal{S}$ koji prepoznaje M, tada TP (ili TO) za L dobijemo tako da „spojimo serijski" $\mathcal{T}$ i $\mathcal{S}$ (disjunktificiramo $\mathcal{T}.Q$ i $\mathcal{S}.Q$, i između $\mathcal{T}.q_\checkmark$ i $\mathcal{S}.q_0$ premotamo traku na početak). Druga tvrdnja je samo kontrapozicija prve.

Problemi za regularne jezike

Vidjeli smo da su sva tri problema: prihvaćanja (A), praznosti (E) i jednakosti (EQ) jezika, odlučiva za regularne jezike zadane pomoću DKA.

Štoviše, kako su svi postupci pretvorbe NKA $\rightarrow$ DKA, RI $\rightarrow$ NKA i DLG $\rightarrow$ RI vrlo mehanički, za svakog od njih postoji TP koji ga provodi (računa odgovarajuću tzv. *prateću* funkciju na kodovima). **Zadatak**: kako biste prirodno kodirali regularne izraze?

To znači da se ta tri problema za regularne jezike zadane pomoću NKA, RI i DLG, svode na probleme A_{DKA}, E_{DKA} i EQ_{DKA}, pa su također odlučivi.

Sada je vrijeme da pogledamo te probleme za šire klase jezika.

Problem beskontekstne praznosti

Za zadanu Chomskyjevu gramatiku $\mathcal{G} = (V, \Sigma, \to, S)$, algoritam

prod $:= \{A \in V : \mathcal{G}$ ima pravilo čitanja za varijablu A ili ima pravilo $A \to \varepsilon\}$
while (prod se promijenio u odnosu na prethodnu iteraciju):
    for $A \to BC$ in (sva pravila širenja u $\mathcal{G}$):
        if $\{B, C\} \subseteq$ prod: prod $:=$ prod $\cup \{A\}$
return bool $(S \notin$ prod$)$

određuje skup *produktivnih* varijabli (dokažite indukcijom!)

$$prod := \{A \in V : (\exists w \in \Sigma^*)(A \Rightarrow^* w)\}$$

te vrijedi $L(\mathcal{G}) \neq \emptyset \iff S \in prod$. Drugim riječima, taj algoritam odlučuje problem E_{ChNF}. Sada slično kao prije (svođenjem) možemo riješiti i probleme E_{BKG}, E_{JPA} i E_{PA}. Dakle E_{BK} je odlučiv.

Problem beskontekstne univerzalnosti

Klasa BK nije zatvorena na komplement, pa se pitamo je li problem

$$ALL_{\text{PA}} := \{\langle \mathcal{P} \rangle : \mathcal{P} \text{ je PA} \wedge L(\mathcal{P}) = (\mathcal{P}.\Sigma)^*\}$$

odlučiv. Pokazuje se da

Problem beskontekstne univerzalnosti

Klasa BK nije zatvorena na komplement, pa se pitamo je li problem

$$ALL_{\mathsf{PA}} := \{\langle \mathcal{P} \rangle : \mathcal{P} \text{ je PA} \wedge L(\mathcal{P}) = (\mathcal{P}.\Sigma)^*\}$$

odlučiv. Pokazuje se da *nije*, tako da se na ALL_{PA} svede neodlučiv (dokazujemo kasnije) *problem odbijanja* za Turingove prepoznavače

$$\bar{A}_{\mathsf{TP}} := \{\langle \mathcal{T}, w \rangle : \mathcal{T} \text{ je TP} \wedge w \in (\mathcal{T}.\Sigma)^* \setminus L(\mathcal{T})\}.$$

Svođenje svakom $\langle \mathcal{T}, w \rangle \in \bar{A}_{\mathsf{TP}}$ pridruži kod $\langle \mathcal{P}_{\langle \mathcal{T}, w \rangle} \rangle$ potisnog automata koji prihvaća riječi

Problem beskontekstne univerzalnosti

Klasa BK nije zatvorena na komplement, pa se pitamo je li problem

$$ALL_{\mathrm{PA}} := \{\langle \mathcal{P} \rangle : \mathcal{P} \text{ je PA} \wedge L(\mathcal{P}) = (\mathcal{P}.\Sigma)^*\}$$

odlučiv. Pokazuje se da *nije*, tako da se na ALL_{PA} svede neodlučiv (dokazujemo kasnije) *problem odbijanja* za Turingove prepoznavače

$$\bar{A}_{\mathrm{TP}} := \{\langle \mathcal{T}, w \rangle : \mathcal{T} \text{ je TP} \wedge w \in (\mathcal{T}.\Sigma)^* \setminus L(\mathcal{T})\}.$$

Svođenje svakom $\langle \mathcal{T}, w \rangle \in \bar{A}_{\mathrm{TP}}$ pridruži kod $\langle \mathcal{P}_{\langle \mathcal{T}, w \rangle} \rangle$ potisnog automata koji prihvaća riječi $\#C_1\#C_2\#\ldots\#C_n\#$ koje *nisu* kodovi $\mathcal{T}$-izračunavanja s w (svođenje ostalim riječima iz Σ^* pridruži ε). Pomoću PA nije teško provjeriti je li istina da:

- stanje u C_1 *nije* početno
- stanje u C_n *nije* završno, ili
- postoji i takav da se u C_{i+1} ne može u jednom koraku iz C_i.

Detalj: ipak reverz na parnim mjestima (C_2^R umjesto C_2) (zašto?)

Problem beskontekstne ekvivalentnosti

Za proizvoljni PA $\mathcal{P}$ s ulaznom abecedom Σ, lako je konstruirati PA koji prepoznaje Σ^*: recimo za $\Delta := \{(1, \alpha, \varepsilon, 1, \varepsilon) : \alpha \in \Sigma\}$, $\mathcal{P}' := (\{1\}, \Sigma, \emptyset, \Delta, 1, \{1\})$ je takav. Sada očita ekvivalencija

$$\langle \mathcal{P} \rangle \in ALL_{\mathsf{PA}} \iff \langle \mathcal{P}, \mathcal{P}' \rangle \in EQ_{\mathsf{PA}}$$

pokazuje $ALL_{\mathsf{PA}} \preceq EQ_{\mathsf{PA}}$, jer nije preteško napraviti Turingov stroj koji na kod $\langle \mathcal{P} \rangle$ dodaje $\langle \mathcal{P}' \rangle$.

Sveukupno imamo $\bar{A}_{\mathsf{TP}} \preceq ALL_{\mathsf{PA}} \preceq EQ_{\mathsf{PA}}$, te iz toga slijedi da su ALL_{PA} i EQ_{PA} neodlučivi (kad dokažemo neodlučivost $\bar{A}_{\mathsf{TP}}$).

Lako je sada standardnim algoritmima pretvorbe dokazati $EQ_{\mathsf{PA}} \preceq EQ_{\mathsf{JPA}} \preceq EQ_{\mathsf{BKG}} \preceq EQ_{\mathsf{ChNF}}$ (i analogno za ALL), te su svi oni neodlučivi. Skraćeno kažemo da su ALL_{BK} i EQ_{BK} neodlučivi.

Problem kontekstne praznosti

Kako idemo dalje prema sve moćnijim strojevima odnosno sve većim klasama jezika, sve više svojstava postaje neodlučivo.

Konkretno, za kontekstne jezike, problem praznosti više nije odlučiv, jer se problem prihvaćanja za Turingove prepoznavače A_{TP} može svesti na njegov komplement. Ideju smo već vidjeli: za zadani TP $\mathcal{T}$ i riječ w nad njegovom ulaznom abecedom, konstruiramo OA $\mathcal{O}$ koji prihvaća sve riječi koje *jesu* kodovi $\mathcal{T}$-izračunavanja s w. Tada očito

$$\langle \mathcal{T}, w \rangle \in A_{\mathsf{TP}} \iff w \in L(\mathcal{T}) \iff L(\mathcal{O}) \neq \emptyset \iff \langle \mathcal{O} \rangle \in E^{\mathsf{c}}_{\mathsf{OA}}.$$

Svođenje je još lakše nego kod ALL_{PA} jer ne trebamo reverze. Pri usporedbi C_i i C_{i+1} možemo koristiti traku s oznakama.

[Izračunljivost: Kleenejeva relacija T je primitivno rekurzivna.]

Ostali neodlučivi problemi

Naravno, problem praznosti je neodlučiv i za kontekstne jezike zadane preko (kontekstnih ili monotonih) gramatika, jer algoritmi pretvorbe (kodiranje konfiguracija riječima uz objedinjavanje znakova za OA $\rightarrow$ MG; konstrukcija u četiri faze za MG $\rightarrow$ KG) pokazuju $E_{\mathrm{OA}} \preceq E_{\mathrm{MG}} \preceq E_{\mathrm{KG}}$. Kratko: E_{Kon} je neodlučiv.

Jednostavno je konstruirati OA koji prepoznaje $\emptyset$, i pomoću njegovog koda svesti E_{OA} na EQ_{OA}, čime (kao gore) dobijemo i da su EQ_{OA}, EQ_{MG} i EQ_{KG} neodlučivi. Kratko: EQ_{Kon} je neodlučiv.

Zadatak: Što je s problemom ALL_{Kon}? Što s klasom Rek?

$Kon \subset Rek$

Dosta je teško smisliti rekurzivan ali ne kontekstan jezik.
Evo jednog primjera:

Promotrimo regularne izraze nad abecedom $\{0, 1\}$ kao jezik RI2 nad abecedom $\{0, 1, (,), \cup, ^+, ^*, ^?, ^2\}$, gdje je semantika znaka ^{2} zadana sa $r^2 := rr$.
Tada je jezik EQ_{RI2} rekurzivan, jer ga (kao problem) očito možemo svesti na odlučiv problem EQ_{RI} tako da „raspišemo" sve kvadrate pomoću konkatenacija.

No jezik EQ_{RI2} nije kontekstan [**seminar**!] — ugrubo, jer za gornje raspisivanje treba potencijalno eksponencijalno mnogo prostora (npr. kod $1^{22\cdots 2} \rightsquigarrow 1^{2^n}$), a to ne stane na traku OA.

Univerzalni Turingov stroj

Teorem: Jezik $A_{\mathsf{TP}} := \{\langle \mathcal{T}, w\rangle : \mathcal{T} \text{ je TP} \wedge w \in L(\mathcal{T})\}$ je RE.

Skica dokaza: Vrlo slično kao za A_{DKA}. Nekoliko bitnih razlika:

- Na početku moramo preprocesirati treću traku (na kojoj po konvenciji o razvrstavanju završi w).
- Kod praznine $\sqcup$ (četvrtu komponentu od $\mathcal{T}$) moramo dodati na treću traku čim njena glava izađe izvan preprocesiranog dijela.
- Jedini uvjet zaustavljanja nam je da stanje simuliranog stroja (koje držimo kao beskonačnu varijablu na zasebnoj traci) postane $\mathcal{T}.q_{\checkmark}$ — ne znamo hoće li se i kada to doista dogoditi!

Turingov prepoznavač koji smo konstruirali zovemo još *univerzalni* Turingov stroj, jer (očito) ima sposobnost simulacije svakog Turingovog stroja. [Priča o Harvard vs. von Neumann arhitekturi.]

Dijagonalizacija

Da bismo pokazali da neki jezik nije
rekurzivan / rekurzivno prebrojiv, treba nam nova tehnika.

Lema o pumpanju ovdje ne postoji, ali možemo dijagonalizirati!

Teorem: Jezik *Russell* $:= \{\langle \mathcal{T} \rangle : \mathcal{T}$ je TP nad $\Sigma \wedge \langle \mathcal{T} \rangle \notin L(\mathcal{T})\}$
nije RE.

(Problem „da li TP ne prihvaća vlastiti kod" nije poluodlučiv.)

Dokaz: pretpostavimo suprotno, i neka je $\mathcal{R}$ neki TP za *Russell*.
Tada po definiciji $\langle \mathcal{R} \rangle \in$ *Russell* $\iff \langle \mathcal{R} \rangle \notin$ *Russell*, kontradikcija.

Naravno, jezik *Russell* nije naročito koristan.
Ali njegovom redukcijom na druge probleme možemo sada
dokazivati da ni oni nisu (polu)odlučivi.

Neodlučivost problema prihvaćanja za Turingove strojeve

Lema: Jezik $\bar{A}_{\mathsf{TP}}$ nije rekurzivno prebrojiv.

Skica dokaza: treba primijetiti $w \in \mathit{Russell} \iff w\langle w \rangle \in \bar{A}_{\mathsf{TP}}$ (BSOMP $\mathbb{\Sigma} \subseteq D = \mathcal{D}_h$), odnosno $\mathit{Russell} \preceq \bar{A}_{TP}$.

Teorem: Jezik A_{TP} nije rekurzivan.

Skica dokaza: „Lako" (dosadno) je vidjeti da je jezik $\mathit{Valid} := \{\langle \mathcal{T}, w \rangle : \mathcal{T} \text{ je TP} \wedge w \in (\mathcal{T}.\Sigma)^*\}$ rekurzivan: treba provjeriti desetak mehaničkih uvjeta.

Sada, kad bi A_{TP} bio rekurzivan, tada bi i $\bar{A}_{\mathsf{TP}} = \mathit{Valid} \setminus A_{\mathsf{TP}}$ bio rekurzivan (Rek je zatvorena na skupovne razlike), što je kontradikcija s gornjom lemom.

Prave inkluzije i kontraprimjeri

Prisjetimo se dosadašnjih jezika koji su pokazivali da je Chomskyjeva hijerarhija prava:

- $L_{=} \in BK \setminus Reg$
- $L_{\equiv} \in Kon \setminus BK$
- $EQ_{\mathrm{RI2}} \in Rek \setminus Kon$
- $A_{\mathrm{TP}} \in RE \setminus Rek$

Iz Postovog teorema ovo zadnje odmah povlači $A^{\mathrm{c}}_{\mathrm{TP}} \notin RE$.

(Naime, kad bi $A^{\mathrm{c}}_{\mathrm{TP}}$ i A_{TP} bili rekurzivno prebrojivi, tada bi A_{TP} bio rekurzivan, što je kontradikcija s prethodnim slajdom.)

To znači da klasa RE nije zatvorena na komplement.

Problem zaustavljanja (*Halting problem*)

Budući da smo Turingove prepoznavače definirali tako da stanu ako i samo ako prihvate riječ, problem prihvaćanja je ekvivalentan problemu stajanja: poluodlučiv je, ali ne i odlučiv.

Ponekad se promatra specijalni slučaj HB, „stane li zadani TP na praznoj traci". Očito je $HB \preceq A_{\mathsf{TP}}$ ($t \in HB \iff t() \in A_{\mathsf{TP}}$), no vrijedi i $A_{\mathsf{TP}} \preceq HB$: za proizvoljni par $(\mathcal{T}, w)$ možemo konstruirati TP koji stane na praznoj traci (prihvaća ε) ako i samo ako $\mathcal{T}$ prihvaća w. Sve što trebamo je dodati na početak (prije $\mathcal{T}.q_0$) „lanac" od $2\,|w|$ novih stanja, čija jedina svrha je da na praznu traku napišu sve znakove od w redom, i vrate glavu na početak.

[Veza s izračunljivošću: teorem o parametru (*s*-*m*-*n* teorem).]

Zadatak: što je s problemima E_{TP} i EQ_{TP}?

(Sipser, Teorem 5.2, stranice 217–218; Teorem 5.4, stranica 220)

Uvod u praktičnu interpretaciju programa

- Napisali smo program (*source code*) u apstraktnom jeziku
- Kod treba interpretirati ili kompajlirati u jezik čije naredbe procesor na dostupnoj platformi može (efikasno) izvesti
- Faze intepretacije/kompajliranja programa:
  1. Leksička analiza
  2. Sintaksna analiza
  3. Semantička analiza
  4. Optimizacija
  5. Generiranje koda (*object code*)
- Prve tri faze obično obavlja *front end* kompajlera, a zadnje dvije *back end*

Leksička analiza

- Ulaz: Programski kod (string)
- Izlaz: Niz *tokena* — podstringova s pridruženim značenjem
- Primjer: leksička analiza aritmetičkih izraza

  `2+-3*5.1`
- Kod rastavljamo na tokene — ovdje, brojeve i operatore:

  `BROJ'2' PLUS'+' MINUS'-' BROJ'3' PUTA'*'`
  `BROJ'5.1'`
- Primjeri koji nisu aritmetički izrazi:
  - `25?3+2` (Lekser: „`neočekivan treći znak`")
  - `1.23-/2` (Lekser se ne žali!)
- Primijetimo: nismo još razlučili unarne operatore od binarnih
- Gornji niz tokena je ulaz za iduću etapu interpretacije programa: sintaksnu analizu

Sintaksna analiza

- Ulaz: Niz tokena
- Izlaz: Stablo parsiranja (modernije: AST)
- Rezultat leksičke analize za izraz `2+-3*5.1` bio je:

```
BROJ'2' PLUS'+' MINUS'-' BROJ'3' PUTA'*'
                         BROJ'5.1'
```

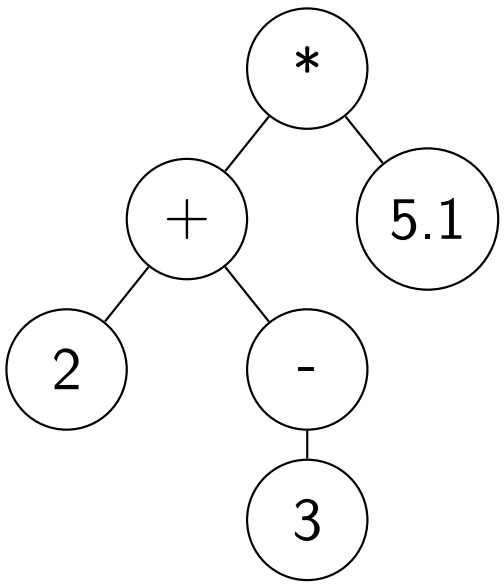
- U ovoj fazi se tokeni organiziraju u stabla (obično prema prioretima i asociranosti operatora)
- Koje je uobičajeno značenje danog niza tokena?

Sintaksna analiza

- Ulaz: Niz tokena
- Izlaz: Stablo parsiranja (modernije: AST)
- Rezultat leksičke analize za izraz `2+-3*5.1` bio je:

```
BROJ'2' PLUS'+' MINUS'-' BROJ'3' PUTA'*'
                         BROJ'5.1'
```

- U ovoj fazi se tokeni organiziraju u stabla (obično prema prioretima i asociranosti operatora)
- Koje je uobičajeno značenje danog niza tokena?

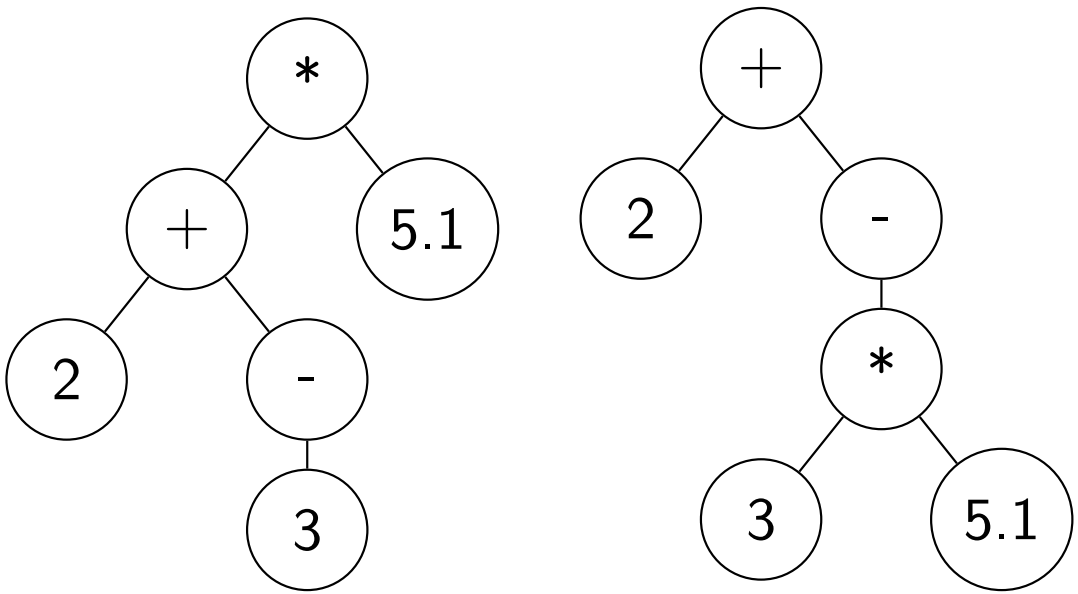


Sintaksna analiza

- Ulaz: Niz tokena
- Izlaz: Stablo parsiranja (modernije: AST)
- Rezultat leksičke analize za izraz `2+-3*5.1` bio je:

```
BROJ'2' PLUS'+' MINUS'-' BROJ'3' PUTA'*'
                         BROJ'5.1'
```

- U ovoj fazi se tokeni organiziraju u stabla (obično prema prioretima i asociranosti operatora)
- Koje je uobičajeno značenje danog niza tokena?

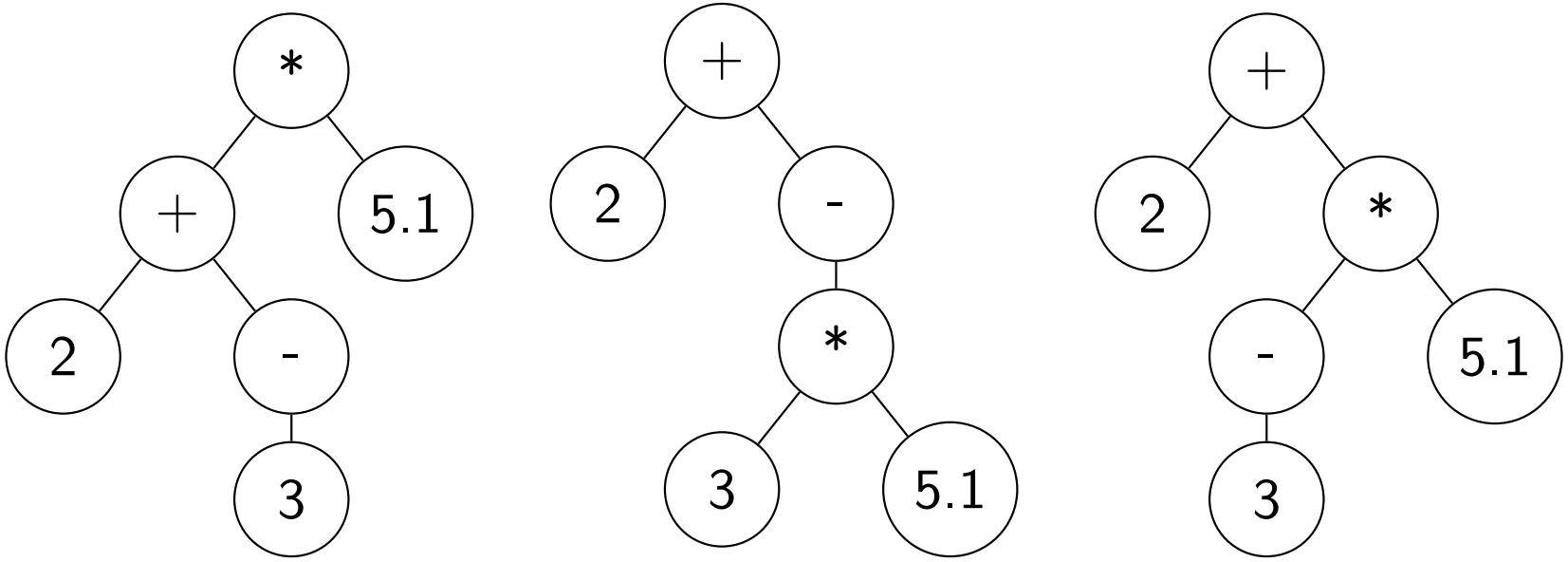


Sintaksna analiza

- Ulaz: Niz tokena
- Izlaz: Stablo parsiranja (modernije: AST)
- Rezultat leksičke analize za izraz `2+-3*5.1` bio je:

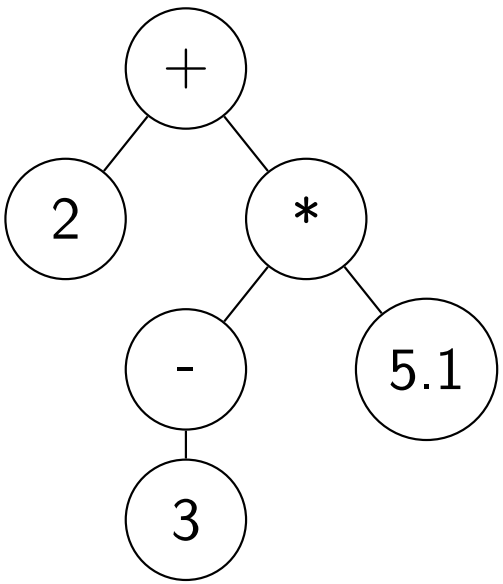
```
BROJ'2' PLUS'+' MINUS'-' BROJ'3' PUTA'*'
                         BROJ'5.1'
```

- U ovoj fazi se tokeni organiziraju u stabla (obično prema prioretima i asociranosti operatora)
- Koje je uobičajeno značenje danog niza tokena?



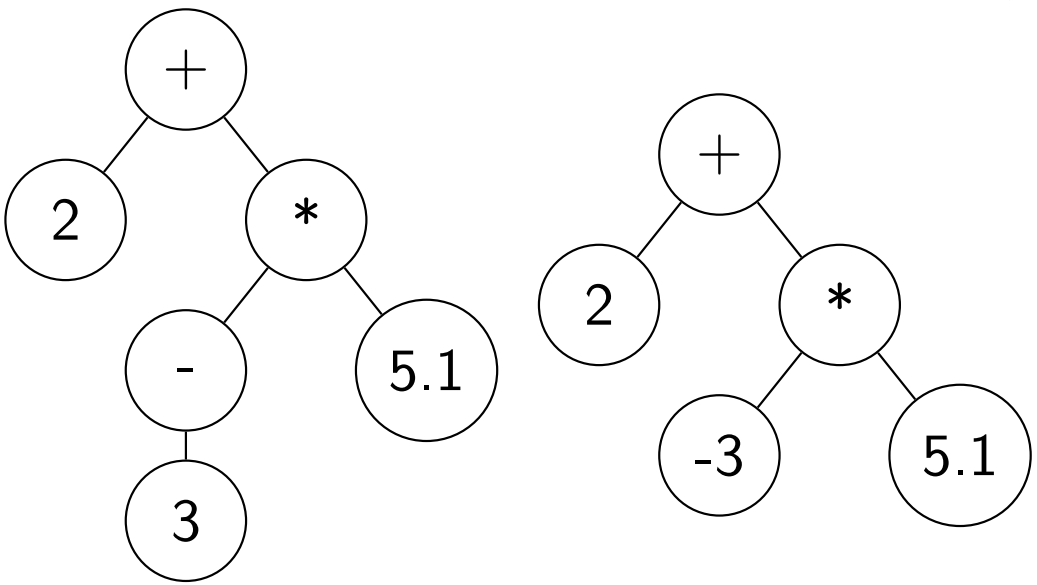
Semantička analiza. Back end

- U semantičkoj analizi se obavlja provjera tipova, jesu li varijable deklarirane prije upotrebe, itd.
- Ulaz: AST
- Izlaz: Međuprikaz (*Intermediate Representation*, IR)
- Što back end radi s IRom? Ovisi o cilju!
  - Optimiziraj: izlaz je jednostavniji IR
  - Evaluiraj/izvrši/interpretiraj: preslikavanje u „stvarni svijet”
  - Prevedi/kompajliraj: izlaz je kod (često u strojnom jeziku)
- U primjeru, sve! — IR je AST, optimiziramo 0/1-podizraze, pa rekurzivno evaluiramo te kompajliramo u RPN (stog-jezik)



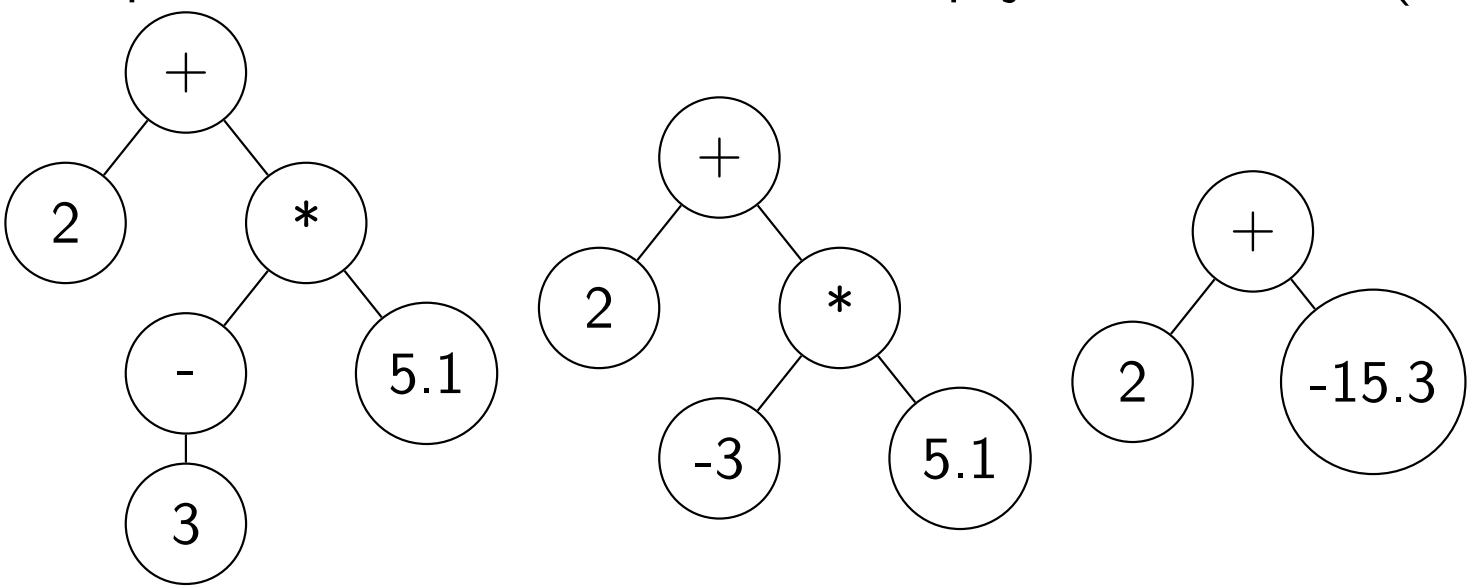
Semantička analiza. Back end

- U semantičkoj analizi se obavlja provjera tipova, jesu li varijable deklarirane prije upotrebe, itd.
- Ulaz: AST
- Izlaz: Međuprikaz (*Intermediate Representation*, IR)
- Što back end radi s IRom? Ovisi o cilju!
  - Optimiziraj: izlaz je jednostavniji IR
  - Evaluiraj/izvrši/interpretiraj: preslikavanje u „stvarni svijet”
  - Prevedi/kompajliraj: izlaz je kod (često u strojnom jeziku)
- U primjeru, sve! — IR je AST, optimiziramo 0/1-podizraze, pa rekurzivno evaluiramo te kompajliramo u RPN (stog-jezik)



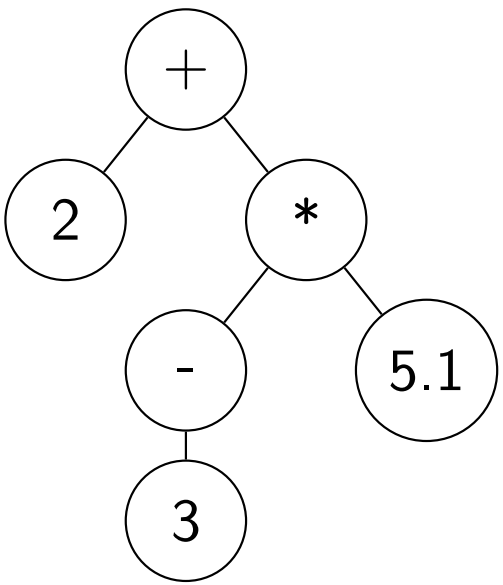
Semantička analiza. Back end

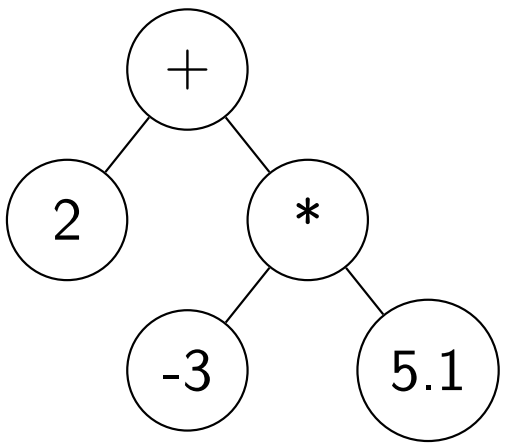
- U semantičkoj analizi se obavlja provjera tipova, jesu li varijable deklarirane prije upotrebe, itd.
- Ulaz: AST
- Izlaz: Međuprikaz (*Intermediate Representation*, IR)
- Što back end radi s IRom? Ovisi o cilju!
  - Optimiziraj: izlaz je jednostavniji IR
  - Evaluiraj/izvrši/interpretiraj: preslikavanje u „stvarni svijet”
  - Prevedi/kompajliraj: izlaz je kod (često u strojnom jeziku)
- U primjeru, sve! — IR je AST, optimiziramo 0/1-podizraze, pa rekurzivno evaluiramo te kompajliramo u RPN (stog-jezik)

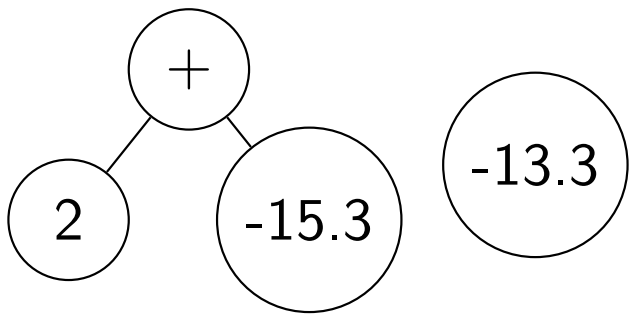


Semantička analiza. Back end

- U semantičkoj analizi se obavlja provjera tipova, jesu li varijable deklarirane prije upotrebe, itd.
- Ulaz: AST
- Izlaz: Međuprikaz (*Intermediate Representation*, IR)
- Što back end radi s IRom? Ovisi o cilju!
  - Optimiziraj: izlaz je jednostavniji IR
  - Evaluiraj/izvrši/interpretiraj: preslikavanje u „stvarni svijet”
  - Prevedi/kompajliraj: izlaz je kod (često u strojnom jeziku)
- U primjeru, sve! — IR je AST, optimiziramo 0/1-podizraze, pa rekurzivno evaluiramo te kompajliramo u RPN (stog-jezik)







Prvi primjer: Logika sudova

Definicija (Formula logike sudova): Neka je $\Sigma = \{P_n : n \in \mathbb{N}_0\} \cup \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)\}$. Svaka propozicijska varijabla P_n je formula. Ako su $\varphi, \psi \in \Sigma^*$ formule, tada su i sljedeće riječi formule:

$$\neg\varphi \qquad (\varphi \wedge \psi) \qquad (\varphi \vee \psi) \qquad (\varphi \rightarrow \psi) \qquad (\varphi \leftrightarrow \psi).$$

Riječ iz Σ^* je formula ako je nastala primjenom konačno mnogo gornjih pravila.

Tokeni:

```
PVAR'P\d+' NEG'!' KONJ'&' DISJ'|' KOND'->'
BIKOND'<->' OTV'(' ZATV')'
```

Gramatika:

form → PVAR | NEG *form* | OTV *form binvez form* ZATV

binvez → KONJ | DISJ | KOND | BIKOND