Interpretacija programa — predavanja

originalni autor slajdova: Vedran Čačić

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

PMF–Matematički odsjek, Sveučilište u Zagrebu

akademska godina 2022./2023. — ljetni semestar
web arhiva: https://web.math.pmf.unizg.hr/~veky/ip/

O kolegiju

Cilj: uvid u teoriju i praksu interpretacije programskih jezika
Nema formalnih prethodnika, ali znanje (objektnog) programiranja dobro će doći. U prvom dijelu kolegija naglasak je na teoriji, u drugom na programiranju.
Literatura:

- Sipser, *Introduction to the Theory of Computation*, 3rd ed.
- (Stanford), *Compilers*, `http://cs143.stanford.edu`
- Srbljić, *Jezični procesori* 1 & 2

Bodovi: Dvije zadaće ukupno 20; Kolokviji + završni, $25 \cdot 2 + 30$
Pragovi: 30% za završni, 45% za prolaz

Što proučavamo?

Matematički: tri vrste objekata — jezike, gramatike i automate
Softverski: alate za prevođenje (kompilaciju i interpretaciju) jezika

Jezik: skup riječi (nizova znakova neke abecede). Statičan je.
Dinamički aspekti jezika vide se kroz gramatike i automate.
Gramatika *generira* jezik krećući od početnog simbola, kroz pravila.
Automat *prepoznaje* jezik čitajući riječ, kroz konfiguracije.

Analogija: trojno shvaćanje funkcije...

- funkcija kao skup uređenih parova koji prolazi vertikalni test
- dobivanje funkcije kompozicijom iz elementarnih funkcija
- kôd funkcije u (imperativnom) programskom jeziku

Imamo nekoliko *klasâ* jezika, odnosno gramatika i automata ...

Chomskyjeva hijerarhija

. . . jezici	. . . gramatike	. . . automati, Turingovi . . .
regularni (*Reg*) . . . izrazi (RI)	desnolinearne (DLG), lijevolinearne (LLG)	konačni (KA, NKA)
beskontekstni (*BK*, BK_+)	beskontekstne (BKG), Chomskyjeve (ChNF)	potisni (PA, JPA)
kontekstni (*Kon*)	kontekstne (KG), monotone (MG)	ograničeni (OA, JOA)
rekurzivni (*Rek*)	—	odlučitelji (TO)
rekurzivno prebrojivi (*RE*)	opće (OG)	strojevi (TP, NTP), enumeratori (TE)

Do na probleme s pozitivnošću (što ćemo kasnije precizirati), svaka klasa jezika je prava potklasa sljedeće.

Formalne definicije — jezici

Abeceda je (obično fiksan) konačan neprazan skup Σ.
Elemente abecede nazivamo *znakovi*, i obično označavamo s α.
Riječ nad Σ: niz od konačno mnogo (nula ili više) znakova iz Σ.
Jezik: skup (konačan ili beskonačan) riječi nad istom abecedom.

Ako je w riječ, $|w|$ je njena duljina, a $|w|_\alpha$ broj znakova α u njoj.
Praznu riječ (duljine 0) označavamo s ε.
Poistovjećujemo znak i riječ duljine 1, kao i riječ w i jezik $\{w\}$.
Ne poistovjećujemo prazan jezik $\emptyset$ s jezikom $\varepsilon = \{\varepsilon\}$!

Na riječima imamo operaciju *konkatenacije* (nastavljanja): za
$u = \alpha_1 \cdots \alpha_n$ i $v = \beta_1 \cdots \beta_m$, pišemo $uv := \alpha_1 \cdots \alpha_n \beta_1 \cdots \beta_m$.
Možemo konkatenirati i jezike: $LM := \{uv : u \in L \wedge v \in M\}$.

Kleenejeve operacije

Konkatenacija ima multiplikativnu notaciju, i uobičajene oznake za potenciranje u skladu s njom: recimo, $w^2 := ww$, $L^3 := LLL$.
Kleenejev plus jezika L definiramo kao $L^+ := \bigcup_{k \in \mathbb{N}_+} L^k$.
Kleenejeva zvijezda je $L^* := \bigcup_{k \in \mathbb{N}_0} L^k$.

Te operacije smo već vidjeli kod *regex*a na Prog1 ($\cup$ smo pisali $|$).

Zadaci:

- Izrazite L^* pomoću L^+ i obrnuto.
- Ako je Σ abeceda, što bi bilo Σ^*?
- U notaciji *regex*a, definirajte $L^?$ i dokažite $L^* = (L^+)^? = (L^?)^*$.
- Dokažite ili opovrgnite: $(L \cup M)^* = (L^? M^?)^*$.

Regularni izrazi

Neka je Σ abeceda. Skup *RI-regularnih jezika* nad Σ je najmanji skup jezika koji sadrži $\emptyset$ i ε kao elemente, Σ kao podskup, te je zatvoren na konkatenaciju, (konačnu) uniju i Kleenejeve operacije.

Drugim riječima: jezik $L \subseteq \Sigma^*$ je *RI-regularan* ako se može dobiti iz *inicijalnih jezika* pomoću konačno mnogo operacijâ $\cup$, $\cdot$, * i $^+$.
Izraz koji to dobivanje pokazuje zove se *regularan izraz*.
Primjer: $0 \cup 1(0 \cup 1)^*$ je jezik binarnih brojeva (nad $\Sigma = \{0, 1\}$).
Zadatak: napišite regularni izraz za jezik svih riječî nad abecedom $\{\mathtt{a}, \mathtt{b}\}$ koje počinju i završavaju istim slovom.

Pozitivni jezici

Za jezik L kažemo da je *pozitivan* ako ne sadrži praznu riječ: $\varepsilon \notin L$.
To je korisno svojstvo: pozitivni jezici se algebarski ljepše ponašaju.
Konkretno, ako je $w \in LM$ i L je pozitivan, dio riječi w u M je strogo kraći od w, pa možemo provoditi indukciju po $|w|$.

Obično je lako iz definicije provjeriti je li jezik pozitivan. Primjeri:

- α je pozitivan za svaki $\alpha \in \Sigma$. $\emptyset$ je trivijalno pozitivan (!).
- L^* nije pozitivan ni za jedan L. Specijalno, ε nije pozitivan.
- LM je pozitivan ako i samo ako je L pozitivan ili M pozitivan.
- $L \cup M$ je pozitivan ako i samo ako su L i M oba pozitivni.

Konačni automati

Definicija: konačni automat (KA) nad abecedom Σ je idealizirani (matematički) stroj $\mathcal{M} := (Q, \Sigma, \delta, q_0, F)$, koji pored Σ ima:

- **konačan** skup Q čije elemente nazivamo *stanja*,
- istaknuti element $q_0 \in Q$ koji nazivamo *početno* stanje,
- podskup $F \subseteq Q$ čije elemente nazivamo *završna* stanja,
- funkciju *prijelaza* $\delta : Q \times \Sigma \to Q$, često zadanu tablično.

Kažemo da $\mathcal{M}$ *prihvaća* riječ $w = \alpha_1 \cdots \alpha_n \in \Sigma^*$ ako postoji niz stanjâ $r_0, r_1, \ldots, r_n \in Q$ 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 $r_n \in F$.
Kažemo da $\mathcal{M}$ *prepoznaje* jezik $L(\mathcal{M}) := \{w : \mathcal{M} \text{ prihvaća } w\}$.
Kažemo da je jezik *KA-regularan* ako ga prepoznaje neki KA.

Zatvorenost na skupovne operacije

Teorem 1.25: (brojevi su iz SIPSER) Neka je Σ abeceda.
Skup KA-regularnih jezika nad Σ zatvoren je na konačne unije.
Dokaz: ako $\mathcal{M}_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ prepoznaje L_i, $i \in \{1, 2\}$, tada

$$\mathcal{M} := \big(Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), Q_1 \times F_2 \cup F_1 \times Q_2\big),$$
$$\text{gdje je } \delta\big((q, q'), \alpha\big) := \big(\delta_1(q, \alpha), \delta_2(q', \alpha)\big),$$

prepoznaje $L_1 \cup L_2$. Dalje indukcijom po broju jezika u uniji.
Analogno za presjeke: za završna stanja stavimo $F_1 \times F_2$.
Zadatak: dokažite zatvorenost na komplemente (spram Σ^*).

Za zatvorenost na konkatenaciju i Kleenejeve operatore, trebamo jači alat: *nedeterminizam*.

Nedeterministički konačni automati

Konačni automat može ne biti deterministički na dva načina.

Jedan je da umjesto *funkcije* prijelaza ima *relaciju* prijelaza $\Delta \subseteq Q \times \Sigma \times Q$ (Sipser to promatra kao $\delta : Q \times \Sigma \to \mathcal{P}(Q)$).

Drugi omogućava automatu da prelazi između stanja bez čitanja znaka (tzv. ε-prijelazi): kao da proširimo abecedu, $\Sigma_\varepsilon := \Sigma \dot\cup \{\varepsilon\}$.

Zadatak: napišite formalnu definiciju NKA (Sipser 1.37), i njegovog prihvaćanja riječi odnosno prepoznavanja jezika.

Ekvivalentnost KA i NKA

Teorem 1.39: jezik je KA-regularan akko je NKA-regularan.
Ovaj teorem je vrlo koristan jer je često puno lakše konstruirati NKA za neki jezik, nego KA. (Primjeri na idućem slajdu.)

Dokaz: ($\Rightarrow$) je trivijalan (napišite formalno pretvorbu KA u NKA).
Za ($\Leftarrow$), koristimo *partitivnu konstrukciju*:
ako NKA $\mathcal{N} := (Q, \Sigma, \delta, q_0, F)$ prepoznaje L, tada KA

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

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

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

također prepoznaje L.

Zatvorenost na jezične operacije

Ako imamo NKA $\mathcal{N}$ i $\mathcal{N}'$ koji prepoznaju L i L' redom, NKA koji prepoznaje $L \cup L'$ dobijemo tako da dodamo novo početno stanje, i od njega ε-prijelaze prema početnim stanjima od $\mathcal{N}$ i $\mathcal{N}'$.

NKA koji prepoznaje LL' dobijemo tako da dodamo ε-prijelaze između završnih stanja od $\mathcal{N}$ (koja više nisu završna u novom NKA) i početnog stanja od $\mathcal{N}'$.

NKA koji prepoznaje L^+ dobijemo tako da dodamo ε-prijelaze između završnih stanja od $\mathcal{N}$ i početnog stanja od $\mathcal{N}$.

Za L^*, još dodamo novo početno stanje, proglasimo ga završnim, te dodamo ε-prijelaz prema starom početnom stanju.

Formalno: Sipser 1.45, 1.47 i 1.49.

RI-regularni jezici su NKA-regularni

Da bismo mogli svaki regularni izraz pretvoriti u ekvivalentan NKA, preostalo je još napisati NKA za inicijalne jezike (Sipser 1.55):

- $\big(\{1\}, \Sigma, \emptyset, 1, \emptyset\big)$ prepoznaje $\emptyset$.
- $\big(\{1\}, \Sigma, \emptyset, 1, \{1\}\big)$ prepoznaje ε.
- za svako $\alpha \in \Sigma$, $\big(\{1,2\}, \Sigma, \big\{(1, \alpha, 2)\big\}, 1, \{2\}\big)$ prepoznaje α.

Za dokaz drugog smjera, trebat će nam *desnolinearne gramatike*.

Gramatike — općenito

Za razliku od strojeva — koji postaju sve kompleksniji za šire klase jezika — opće gramatike je vrlo jednostavno definirati, a uže klase jezika generiraju gramatike s raznim *restrikcijama* na pravila.

Definicija: *gramatika* nad abecedom Σ je transformacijski sustav $\mathcal{G} = (V, \Sigma, \rightarrow, S)$, koji pored Σ ima još:

- konačan skup *varijabli* V, **disjunktan** sa Σ;
  elemente skupa $Y := \Sigma \dot\cup V$ zovemo jednim imenom *simboli*
- istaknuti element $S \in V$ koji zovemo *početna* varijabla
- konačnu binarnu relaciju na Y^*, čije elemente zovemo *pravila*
  (na lijevoj strani svakog pravila **mora** biti bar jedna varijabla)

Izvodi. Desnolinearne gramatike

Neka je $\mathcal{G}$ gramatika. Za riječi u i v nad Y, kažemo da u *daje* v, i 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 $\mathcal{G}$ je konačan niz riječi $u_0, u_1, \ldots, u_n$ nad Y, takav da je $u_0 = S$, te $u_{i-1} \Rightarrow u_i$ za sve $i \in [1..n]$.
Zovemo ga *izvod za* w ako je $u_n = w$. Kratko pišemo $S \Rightarrow^* w$.
Kažemo da $\mathcal{G}$ *izvodi* riječ w **nad** Σ(!) ako postoji izvod za w u $\mathcal{G}$.
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.
Za jezik kažemo da je DLG-regularan ako ga generira neka DLG.

KA-regularni jezici su DLG-regularni

Moramo pokazati kako pretvoriti KA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ u ekvivalentnu DLG $\mathcal{G}$ (koja generira $L(\mathcal{M})$):

- za svako stanje $q \in Q$ dodamo po jednu varijablu $V_q \notin \Sigma$,
- početno stanje odgovara početnoj varijabli: $S := V_{q_0}$,
- 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 $V_q \to \varepsilon$.

Sada se dokaže da za svako stanje $q \in Q$ vrijedi $L((Q, \Sigma, \delta, q, F)) = L((V, \Sigma, \to, V_q))$, indukcijom

($\subseteq$) po duljini niza stanja $r_0, r_1, \ldots, r_n$ kroz koje automat prolazi;

($\supseteq$) po duljini izvoda riječi u gramatici.

Specijalno, za $q := q_0$, dobijemo $L(\mathcal{M}) = L(\mathcal{G})$.

Gramatički sustavi

Za „zatvaranje kruga" RI→NKA→KA→DLG, koji će pokazati da su sve definicije regularnih jezika ekvivalentne, još treba objasniti kako iz DLG dobiti regularni izraz.

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 $A = \emptyset$.

Sada 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?

Fiksna točka

Lema: Rješenje jezične jednadžbe $X = AX \cup B$ je $X = A^*B$. Ako je A pozitivan, to je jedinstveno rješenje. Inače, to je najmanje rješenje u smislu relacije $\subseteq$.

Dokaz: A^*B zadovoljava jednadžbu:

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

Za jedinstvenost, neka je $X = AX \cup B$. Trebamo dokazati $X \supseteq A^*B$, i da se jednakost postiže za pozitivan A.

($\supseteq$) $w \in A^*B$ znači $w \in A^nB$ za neki $n \in \mathbb{N}_0$. Indukcijom po n dokažemo $w \in AX$ u koraku te $w \in B$ u bazi, dakle $w \in AX \cup B = X$.

($\subseteq$) Ako je $w \in B$, trivijalno je $w \in A^*B$. Inače, totalnom indukcijom po $|w|$ pokažemo da $w \in AX$ povlači $w \in A^*B$. (Ovdje treba pozitivnost od A da indukcija prođe.)

Pretvorba DLG $\to$ RI — primjer

Pretvorimo u ekvivalentni regularni izraz DLG zadanu pravilima:

$$S \to \mathtt{a}A \qquad A \to \mathtt{a}A \qquad A \to \mathtt{b}A \qquad A \to \mathtt{a}N$$
$$S \to \mathtt{b}B \qquad B \to \mathtt{a}B \qquad B \to \mathtt{b}B \qquad B \to \mathtt{b}N \qquad N \to \varepsilon.$$

Gramatički sustav glasi:

$$S = \mathtt{a}A \cup \mathtt{b}B \quad A = \mathtt{a}A \cup \mathtt{b}A \cup \mathtt{a}N \quad B = \mathtt{a}B \cup \mathtt{b}B \cup \mathtt{b}N \quad N = \varepsilon.$$

Uvrštavanjem N dobijemo $A = \mathtt{a}A \cup \mathtt{b}A \cup \mathtt{a}$, što možemo zapisati kao $A = (\mathtt{a} \cup \mathtt{b})A \cup \mathtt{a}$. Po lemi o fiksnoj točki ($\mathtt{a} \cup \mathtt{b}$ je očito pozitivan), $A = (\mathtt{a} \cup \mathtt{b})^*\mathtt{a}$. Analogno $B = (\mathtt{a} \cup \mathtt{b})^*\mathtt{b}$.
Uvrštavanjem u prvu jednadžbu za S dobijemo
$\mathtt{a}\,(\mathtt{a} \cup \mathtt{b})^*\mathtt{a} \cup \mathtt{b}\,(\mathtt{a} \cup \mathtt{b})^*\mathtt{b}$, što je ekvivalentni regularni izraz.

Lijevolinearne gramatike

DLG su „jednostrano" definirane. Prirodno je pitati se: što bi se dogodilo da zamijenimo redoslijed na desnoj strani pravila ($A \to B\alpha$)? Te gramatike zovemo *lijevolinearnima*, a jezike koje one generiraju *LLG-regularnima*, ali, naravno, pokazuje se da se i ta klasa podudara s klasom regularnih jezika.

Najjednostavniji način da se to pokaže je operacija *reverza*.
Reverz riječi $w = \alpha_1\alpha_2\cdots\alpha_n$ definiramo kao $w^{\mathrm{R}} = \alpha_n\alpha_{n-1}\cdots\alpha_1$.
Reverz jezika je $L^{\mathrm{R}} = \{w^{\mathrm{R}} : w \in L\}$.
Reverz klase jezika $\mathcal{C}$ je klasa svih L^{R}, gdje L pripada klasi $\mathcal{C}$.

Očito je reverz klase DLG-regularnih jezika klasa LLG-regularnih jezika. Pokazat ćemo da je reverz klase RI-regularnih jezika ponovo klasa RI-regularnih jezika.

Reverzi regularnih izraza

Lako se može pokazati da za reverze jezika vrijede jednakosti:

- $\emptyset^{\mathrm{R}} = \emptyset$, $\varepsilon^{\mathrm{R}} = \varepsilon$, $\alpha^{\mathrm{R}} = \alpha$ za svaki znak α.
- $(LM)^{\mathrm{R}} = M^{\mathrm{R}} L^{\mathrm{R}}$, $(L \cup M)^{\mathrm{R}} = L^{\mathrm{R}} \cup M^{\mathrm{R}}$, $(L^*)^{\mathrm{R}} = (L^{\mathrm{R}})^*$

Iz njih, indukcijom po izgradnji regularnog izraza, slijedi da je reverz svakog RI-regularnog jezika ponovo RI-regularan.

To znači da imamo sljedeće jednakosti između klasâ:

$$Reg_{\mathrm{LLG}} = Reg^{\mathrm{R}}_{\mathrm{DLG}} = Reg^{\mathrm{R}}_{\mathrm{RI}} = Reg_{\mathrm{RI}} = Reg_{\mathrm{DLG}}.$$

Sažetak dosad napravljenog: dokazali smo jednakost svih klasâ

$$Reg_{\mathrm{KA}} = Reg_{\mathrm{NKA}} = Reg_{\mathrm{RI}} = Reg_{DLG} = Reg_{LLG}.$$

Ubuduće tu klasu jednostavno zovemo *Reg*, klasa *regularnih* jezika.

Ograničenja regularnih jezika

Dva su problema kod dizajniranja konačnog automata:

- „iznenadni kraj”: budući da ne znamo unaprijed kad će doći kraj riječi, moramo u *svakom* trenutku znati je li dotad pročitana riječ u jeziku;
- konačno mnogo stanja: koliko god da smo znakova pročitali, sve što o njima smijemo zapamtiti ulazi u jednu od konačno mnogo mogućnosti.

Intuitivno, ovo drugo je razlog zašto jezik

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

nije regularan: kad počnemo čitati b-ove, moramo znati koliko smo a-ova pročitali, a tih mogućnosti je beskonačno mnogo.

Možemo li to formalno dokazati?

Lema o pumpanju za regularne jezike

Neka je L regularni jezik. To znači da postoji KA $\mathcal{M}$ koji ga prepoznaje. Označimo s $p := \#Q$ broj njegovih stanja. Kad god $\mathcal{M}$ prihvati riječ duljine $|w| \geq p$, pri čitanju prvih p njenih znakova prolazi kroz $p+1$ stanjâ $r_0, r_1, \ldots, r_p$, među kojima se (po Dirichletovom principu) neko stanje mora *ponoviti*.

Označimo s $r_i = r_j$, $i < j$, te dvije pojave, te sa x, y i z dijelove riječi w od prvih i znakova, sljedećih $j - i > 0$ znakova, te preostalih $|w| - j$ znakova. Vidimo da je $j = |xy| \leq p$, te je $y \neq \varepsilon$ jer je $|y| = j - i > 0$. Kako je $\mathcal{M}$ prije i nakon čitanja y u istom stanju, on mora prihvatiti i riječ xz, kao i riječi $xyyz$, $xyyyz$,...

Jezik koji nema to svojstvo „pumpanja" sigurno nije regularan. (Sipser 1.70)

Primjena leme o pumpanju

Dokažimo sada formalno da jezik $L_=$ nije regularan. Pretpostavimo da jest, i neka je p broj stanja nekog KA koji prepoznaje $L_=$.
Riječ $\mathrm{a}^p\mathrm{b}^p$ je u $L_=$, i duljina joj je $2p > p$, pa se prema lemi može zapisati kao $\mathrm{a}^p\mathrm{b}^p = xyz$ tako da je $y \neq \varepsilon$, $|xy| \leq p$, te su sve *varijante* xy^kz također u $L_=$.
Iz $|xy| \leq p = |\mathrm{a}^p|$ slijedi da se x i y sastoje samo od a-ova, recimo $y = \mathrm{a}^m \neq \varepsilon$, pa je $m > 0$. Ali tada varijanta $xz = \mathrm{a}^{p-m}\mathrm{b}^p \notin L_=$ jer je $p - m \neq p$. To je kontradikcija, pa $L_=$ nije regularan.

Domaća zadaća: Sipser 1.74–1.77, i problem 1.29.

Beskontekstni jezici

Vidimo da neki vrlo jednostavni jezici nisu regularni. Ako želimo proučavati i takve jezike, moramo proširiti klasu *Reg*.

Beskontekstno pravilo je pravilo oblika $A \to w$, gdje je $A \in V$ varijabla, a $w \in Y^*$ niz simbola.
Dakle, uvijek zamjenjujemo jedan simbol s nula, jednim ili više njih, i pritom nas nije briga za *kontekst* (znakove oko) varijable A.

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

Primjer: $Reg \subset BK$

Beskontekstna gramatika $S \to \mathtt{a}S\mathtt{b} \mid \mathtt{ab}$ generira jezik $L_=$.
To je lako vidjeti indukcijom: po duljini izvoda za jedan smjer, a po broju a-ova (i b-ova) za drugi.

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

Seminar: $L_=$ je nad dvočlanom abecedom. *Parikhov teorem* kaže da je to nužno: nad jednočlanom abecedom, $Reg_1 = BK_1$.

Zadatak: koji jezik generira gramatika čija pravila su zadana sa

$$S \to 0J \mid 1N \mid \varepsilon, \quad J \to 1S \mid SJ, \quad N \to 0S \mid SN\ ?$$

Je li taj jezik regularan?

Zatvorenost klase BK na $\cup$, $\cdot$ i Kleenejeve operatore

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

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

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 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 Kartezijevom konstrukcijom, za koju nam treba pojam automata koji prepoznaje beskontekstni jezik.

Potisni automati (PA)

KA moramo „osnažiti" ako želimo da može prepoznati beskontekstne jezike. Najjednostavnije je dodati mu *stog*.

Definicija: *Potisni automat* je automat oblika $(Q, \Sigma, \Gamma, \Delta, q_0, F)$ i značenje pojedinih komponenti je 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.

Nedeterminističnost i jednostavnost

Za PA, nedeterminističnost je ključna! Deterministički potisni automati (SIPSER poglavlje 2.4 — seminar...) prepoznaju užu klasu jezika od BK.

Postoji drugi način kako možemo pojednostaviti PA. *Jednostavni potisni automat* (JPA) ima sljedeća tri dodatna svojstva:

- 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* ($|st| = 1$).

Ekvivalentnost PA i JPA

Očito, svaki JPA je PA. Za drugi smjer, trebamo objasniti kako proizvoljni PA možemo transformirati tako da ostvarimo tri potrebna svojstva, a da prepoznaje isti jezik.

Prva dva svojstva postignemo tako da na početku stavimo novi simbol \$ na stog, a iz svakog bivšeg završnog stanja odemo u međustanje koje sve znakove osim \$ skida sa stoga, te odlazi u novo završno stanje $q_\checkmark$ skidanjem \$. Formalno, to zahtijeva povećanje $\#\Gamma$ za 1, $\#Q$ za 3, te $\#\Delta$ za $2 + \#F + \#\Gamma$ (raspišite!).

Treće svojstvo postižemo tako da svaki prijelaz sa $|st| \neq 1$ zamijenimo s dva prijelaza kroz novo međustanje (ako $s \neq \varepsilon \neq t$, prvo skinemo t, pa onda stavimo s; ako $s = \varepsilon = t$, prvo stavimo #, a zatim ga skinemo, gdje je # još jedan novi znak u Γ).

BKG $\rightarrow$ PA

Lema 2.21: ako postoji beskontekstna gramatika koja generira jezik L, tada postoji i PA koji ga prepoznaje.

Algoritam za rad PA ekvivalentnog $\mathcal{G} = (V, \Sigma, \rightarrow, S)$:

push \$; push S; while True:
    $\gamma :=$ top; 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
        case $\gamma =$ \$:
            uđi u „slijepo" završno stanje

Lijevi izvodi

Upravo konstruirani PA koristi nedeterminizam da „pogodi" izvod riječi na ulazu, iz početne varijable od $\mathcal{G}$. Ipak, pristup nije potpuno „stihijski" — kako na ulaznoj traci stoje samo znakovi iz Σ (dok su na stogu znakovi iz Y), izvod koji taj PA nađe ima bitno svojstvo: svaka primjena pravila na riječ w uvijek se obavlja na prvoj slijeva varijabli u w. Takve izvode zovemo *lijevima*.

Nije teško vidjeti da kad god postoji izvod za neku riječ iz BKG $\mathcal{G}$, tada postoji i lijevi izvod: jer nema konteksta, možemo zamijeniti redoslijed primjene pravila i ništa se neće promijeniti.

Ipak, to ne znači da je lijevi izvod nužno jedinstven! Preciznije, lijevih izvoda ima točno onoliko koliko i *stabala parsiranja*.

Stablo parsiranja

Stablo parsiranja iz BKG $\mathcal{G}$ je stablo u čijem korijenu je početna varijabla, a svaki čvor je ili znak bez djece, ili varijabla čija djeca (može ih biti 0) čitana redom predstavljaju desnu stranu jednog pravila za tu varijablu. Stablo parsiranja *za riječ* w je stablo parsiranja čiji znakovi čitani redom su upravo znakovi riječi w.

Očito, $w \in L(\mathcal{G})$ ako i samo ako postoji takvo stablo. Ako ih ima više za istu riječ, takve riječi i gramatike zovemo *višeznačnima*.

- Riječ $w \in L(\mathcal{G})$ je *$\mathcal{G}$-višeznačna* ako postoji više stabala parsiranja iz $\mathcal{G}$ za w.
- BKG $\mathcal{G}$ je *višeznačna* ako postoji $\mathcal{G}$-višeznačna riječ iz $L(\mathcal{G})$.
- $L \in BK$ je *višeznačan* ako je svaka BKG za L višeznačna.
- „Jednoznačna" (riječ, gramatika ili jezik) = „nije višeznačna".

Od ovoga je teško naći samo višeznačni beskontekstni jezik (primjer u Srbljić). [Seminar: višeznačne gramatike i jezici]

JPA $\rightarrow$ BKG (uvod)

Ako neki PA prepoznaje jezik L, već smo vidjeli da ga možemo „pojednostaviti": pretvoriti u JPA $\mathcal{J}$ koji također prepoznaje L. Sada želimo napisati BKG $\mathcal{G}$ za L koristeći $\mathcal{J}$. Ona će imati po jednu varijablu ${}_pV_q$ za svaki par stanja (p, q) od $\mathcal{J}$.

Ideja je da ${}_pV_q \Rightarrow^* w \in \Sigma^*$ znači „$\mathcal{J}$ iz stanja p s praznim stogom može prijeći u stanje q s praznim stogom tako da pročita w". Iz toga odmah slijedi da je ${}_{q_0}V_{q_\checkmark}$ početna varijabla od $\mathcal{G}$.

Pokušajmo sada napisati pravila za općenitu varijablu ${}_pV_q$. Neka je w kao što je gore opisano, i pitamo se što $\mathcal{J}$, između konfiguracijâ (stanje=q, stog=ε) i (stanje=p, stog=ε), čini tako da pročita w.

JPA $\rightarrow$ BKG (prvi slučaj)

Kako je $\mathcal{J}$ jednostavan, svaki njegov prijelaz je ili *push* ili *pop*.
Dakle, prvi prijelaz mora biti *push* (jer je stog prazan na početku), a zadnji mora biti *pop* (jer je stog prazan i na kraju).
Pitamo se je li stog bio prazan i negdje između ta dva trenutka.
(Zbog nedeterminističnosti zapravo je moguć odgovor „i da i ne".
To samo znači da ćemo za *svaku* mogućnost dodati jedno ili više pravila u $\mathcal{G}$.)

Ako jest, označimo sa r neko stanje („između" p i q) kad stog postane prazan. Ako sa u označimo dio riječi w pročitan od stanja p do r, a s v ostatak, tada je $w = uv$, te u prevodi p u r s praznim stogom, a v prevodi r u q s praznim stogom. To znači ${}_pV_r \Rightarrow^* u$ i ${}_rV_q \Rightarrow^* v$, pa u gramatiku dodajemo pravilo ${}_pV_q \rightarrow {}_pV_r\,{}_rV_q$ (po jedno za svaki mogući r), kako bismo omogućili izvod

$$ {}_pV_q \rightarrow {}_pV_r\,{}_rV_q \Rightarrow^* u\,{}_rV_q \Rightarrow^* uv = w. $$

JPA $\rightarrow$ BKG (drugi slučaj)

Ako pak stog nije bio prazan nigdje između stanjâ p i q, to znači da je završni *pop* $(s, \beta, \gamma, q, \varepsilon)$ maknuo sa stoga onaj isti γ kojeg je početni *push* $(p, \alpha, \varepsilon, r, \gamma)$ stavio na stog. Ako sa u označimo dio riječi w pročitan između ta dva prijelaza, to znači $w = \alpha u \beta$, gdje u prevodi r u s, a na dnu stoga miruje γ. No jednostavnost garantira da $\mathcal{J}$ nikako ne može *znati* da je γ na dnu stoga: „provjera" bi se morala realizirati kao zasebni *pop* i *push* γ, a onda bi između ta dva prijelaza stog bio prazan. To pak znači da u također prevodi $\mathcal{J}$ iz r s praznim stogom u s s praznim stogom, te u $\mathcal{G}$ dodajemo pravilo ${}_pV_q \rightarrow \alpha\ {}_rV_s\ \beta$ (po jedno za svaku moguću (r, s, α, β)).

Također, trebaju nam neka „završna" pravila (bez varijabli na desnoj strani). Kako ε očito može ostaviti $\mathcal{J}$ u istom stanju p, dodajemo u $\mathcal{G}$ pravilo ${}_pV_p \rightarrow \varepsilon$ (po jedno za svako stanje p).

JPA $\to$ BKG (zaključak)

Neka je $\mathcal{J} = \big(Q, \Sigma, \Gamma, \Delta, q_0, \{q_\checkmark\}\big)$ JPA, $(p, q) \in Q^2$, $w \in \Sigma^*$, te

$$\mathcal{G} := \big(\{{}_{q_1}V_{q_2} : q_1, q_2 \in Q\}, \Sigma, \to, {}_{q_0}V_{q_\checkmark}\big)$$

upravo konstruirana (očito beskontekstna) gramatika.

Tvrdnja: ${}_pV_q \Rightarrow^* w$ ako i samo ako w prevodi $\mathcal{J}$ iz stanja p s praznim stogom u stanje q s praznim stogom.
Dokaz: ($\Rightarrow$) indukcijom po broju koraka u izvodu w iz ${}_pV_q$.
($\Leftarrow$) indukcijom po broju prijelaza koje $\mathcal{J}$ napravi čitajući w.
(Raspisane indukcije su u SIPSER tvrdnjama 2.30 i 2.31.)

Specijalno, za $p := q_0$ i $q := q_\checkmark$, dobijemo $L(\mathcal{G}) = L(\mathcal{J})$.

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 KA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ 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 2.34: 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?]

Primjer jezika koji nije beskontekstan

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

Kad bi bio, postojao bi broj p takav da se svaka riječ iz L čija je duljina bar p — pa tako specijalno i riječ $\mathtt{a}^p\mathtt{b}^p\mathtt{c}^p$ — može rastaviti na pet dijelova $uvxyz$ koji zadovoljavaju tvrdnju leme o pumpanju.

Kako je $|vxy| \leq p$, a razmak između a-ova i c-ova u riječi je $> p$, zaključujemo da vxy ne može sadržavati i a-ove i c-ove. Bez smanjenja općenitosti pretpostavimo da ne sadrži c-ove. To znači da je svih p c-ova u z, pa tako i u nultoj varijanti uxz.

Po tvrdnji leme o pumpanju, $uxz \in L_{\equiv}$, no to je nemoguće jer je prekratka: zbog $|vy| > 0$ je $|uxz| < |uvxyz| = 3p$, pa uxz ne može imati još p a-ova i još p b-ova, što je nužan uvjet pripadnosti $L_{\equiv}$.

Zadatak: Sipser primjeri 2.37 i 2.38, problemi 2.42–2.45.

Nezatvorenost klase BK na razne skupovne operacije

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 $\triangle$?

Problemi i odlučivost

Intuitivno, kažemo da je *problem* pitanje koje za svaku svoju *instancu* ima jednoznačan da/ne odgovor. Recimo, instance *problema prihvaćanja za konačne automate*, A_{KA}, su parovi $(\mathcal{M}, w)$ gdje je $\mathcal{M}$ konačni automat, a w riječ nad njegovom abecedom. Problem prihvaćanja pita je li $w \in L(\mathcal{M})$.

Problem je *odlučiv* ako postoji *algoritam* za njega, koji prima instancu problema kao ulaz te u konačno mnogo koraka daje „`bool`" izlaz (`True`/`False`) koji predstavlja točan odgovor na pitanje problema.

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

Zasad ćemo se baviti samo problemima prihvaćanja; kasnije ćemo uvesti i neke druge. Također, kasnije ćemo formalno definirati algoritme, probleme i odlučivanje. Puno preciznije o tim pojmovima čut ćete na kolegiju Izračunljivost.

Kompliciraniji izlazi

Matematički, problem predstavlja samo da/ne pitanje (na koje algoritam odgovara), ali u primjenama, često bismo htjeli više. Primjerice, *leksička analiza* programa može se definirati regularnim izrazima koji daju *tokene* za podstringove ulaza koje prepoznaju. Slično, *sintaksna analiza* može se definirati beskontekstnim gramatikama koje proizvode *stabla parsiranja* za validne programe.

Često i takva pitanja u praksi neformalno zovemo „problemi”. Recimo, praktični problem faktorizacije je nadgradnja matematičkog problema provjere prostosti. Praktični problem *parsiranja* je nadgradnja matematičkog problema A_{BKG}.

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**: partitivna konstrukcija u prvoj koordinati, $(\mathcal{M}, w) \mapsto (\mathcal{P}(\mathcal{M}), w)$, svodi problem A_{NKA} na A_{KA}.

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}

Nije veliki problem napisati algoritam koji pojednostavljuje PA, algoritam koji pretvara JPA u BKG, niti algoritam koji pretvara 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}.

Je li taj problem odlučiv? Zasad ne znamo.
[Znamo da je *poluodlučiv* . . . — prije ili kasnije ćemo „nabasati" na izvod ako on postoji i ako dovoljno sistematično tražimo.]
Pokazat ćemo da jest, svođenjem na A_{ChNF}, problem prihvaćanja za beskontekstne gramatike u *Chomskyjevoj normalnoj formi*.

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 dva 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.

Očito su i pravila širenja i pravila čitanja beskontekstna, pa su sve Chomskyjeve gramatike beskontekstne. Također, kako ni pravilo širenja ni pravilo čitanja ne mogu generirati praznu riječ, svi jezici generirani Chomskyjevim gramatikama su pozitivni.

Pretvorba u Chomskyjevu normalnu formu, faza START

Cilj nam je opisati algoritam u 5 faza za pretvorbu proizvoljne BKG $\mathcal{G}$ u Chomskyjevu gramatiku $\mathcal{G}'$ koja joj je „skoro" ekvivalentna: $L(\mathcal{G}') = L(\mathcal{G}) \setminus \{\varepsilon\}$. U procesu ćemo dobiti i odgovor na pitanje je li $\mathcal{G}$ pozitivna — odnosno, jesmo li doista maknuli ε iz $L(\mathcal{G})$.

U prvoj fazi, ako se početna varijabla S pojavljuje na desnoj strani nekog pravila, dodamo novu varijablu S', dodamo novo pravilo $S' \to S$, te proglasimo S' novom početnom varijablom.

Pretvorba BKG $\to$ ChNF, faze TERM i BIN

U drugoj fazi, svakom znaku $\alpha \in \Sigma$ koji se pojavljuje na desnoj strani nekog pravila ali **ne sâm** (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 $k > 0$ pojavâ 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 $A \to \varepsilon$.
Gramatika $\mathcal{G}$ je bila pozitivna ako i samo ako $S \notin np$.

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, osim što DEL miče ε iz jezikâ koji nisu pozitivni (ali prijavi tu okolnost).

Ograničenost duljine izvoda u ChNF

Propozicija: Neka je $\mathcal{G}$ ChNF i $w \in L(\mathcal{G})$. Tada svaki izvod od w u $\mathcal{G}$ ima točno $2|w| - 1$ koraka, i to $|w| - 1$ širenja i $|w|$ čitanja.

Dokaz: Označimo s a broj znakova, a s v broj varijabli, tijekom izvoda w iz $\mathcal{G}$. Na početku je očito $(a, v) = (0, 1)$ (krećemo od S), a na kraju je $(a, v) = (|w|, 0)$ (završavamo s w).

Svaka primjena pravila širenja inkrementira v (zamjenjujemo jednu varijablu dvjema), a svaka primjena pravila čitanja dekrementira v i inkrementira a (zamjenjujemo jednu varijablu znakom). Dakle, ako imamo x prvih i y drugih koraka, oni zadovoljavaju sustav

$$(0, 1) + (0, 1)x + (1, -1)y = (|w|, 0)$$

koji ima jedinstveno rješenje $(x, y) = (|w| - 1, |w|)$.

Posljedica: za fiksnu w imamo konačno mnogo kandidata za izvod.

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

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, $A_{\mathrm{KA}} \in P$.

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 $|t_2| \leq p(|t_1|)$, 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 (do na ε) 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)" svodimo 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 $B, C \in V$.
Kako su sve varijable u $\mathcal{G}$ pozitivne (ne izvode ε), 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, T_1, 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 $(A, (\alpha_i))$ (korijen je A, ispod njega je samo jedan čvor α_i).

Primjeri:
`https://www.cs.bgu.ac.il/~michaluz/seminar/CKY1.pdf`

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 gramatikâ 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 kubne vremenske i kvadratne prostorne složenosti (seminar!).

Šira klasa jezika

Vidjeli smo da još uvijek neke vrlo jednostavne jezike, kao što je $L_{\equiv}$, ne možemo prepoznati potisnim automatima. Problem: znakovi na stogu su „za jednokratnu upotrebu" — možemo pamtiti proizvoljno mnogo informacija, ali jednom kad ih skinemo sa stoga, one su izgubljene (osim ono konačno mnogo bitova koje možemo zapamtiti kroz trenutno stanje).

Jednostavno poboljšanje PAova sastoji se u tome da „preklopimo" ulaznu traku i stog. Time traka postaje promjenjiva (možemo po njoj *pisati*) i možemo se po njoj kretati u oba smjera (kao `push` i `pop`, ali kod pomaka ulijevo, informacije desno od pozicije čitanja ostaju zapisane na traci).

Stog PA je neograničen — ali ako pretvorimo gramatiku u ChNF, izvodi su ograničeni, pa je dovoljno imati stog duljine ulazne riječi.

Ograničeni automat (OA)

Ograničeni automat je matematički stroj formalnog oblika $(Q, \Sigma, \Gamma, \Delta, q_0, q_\checkmark)$, pri čemu je značenje pojedinih komponenti slično kao kod JPA, uz dodatni uvjet $\Sigma \subseteq \Gamma$, i relaciju prijelaza

$$\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 1\}.$$

$(q, \alpha, p, \beta, d) \in \Delta$ znači: ako je automat u stanju q i čita na poziciji m znak α, može prijeći u stanje p, zapisati β na poziciju m na traci, te se pomaknuti na poziciju $m + d$, ali tako da ne izađe s trake (m je restringiran na skup $[0..|w|\rangle$, gdje je w ulaz).

OA su nedeterministični, baš kao i PA. No jesu li ekvivalentni deterističnima, još je uvijek otvoren problem!

Ograničeni automat — konfiguracije i prihvaćanje

Neka je $n \in \mathbb{N}_+$. *Konfiguracija* OA duljine n je uređena trojka $(q, m, t) \in Q \times [0..n\rangle \times \Gamma^n$. q zovemo *stanje*, m *pozicija*, a t *traka*.

Kažemo da konfiguracija $\big(q, k, u\alpha v\big)$, gdje je $\alpha \in \Gamma$ i $|u| = k$, *prelazi* u $(p, c_{|u\alpha v|}(k + d), u\beta v)$, ako je $(q, \alpha, p, \beta, d) \in \Delta$. $c_n(m) := \min\big\{n - 1, \max\{0, m\}\big\}$ je oznaka za tzv. *clip*-funkciju. Pišemo $\vdash$ kao oznaku relacije prijelaza između konfiguracija.

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

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

Primijetimo da ne postoji konfiguracija duljine 0 (skup mogućih pozicija bio bi $[0..0\rangle = \emptyset$). Dakle $L(\mathcal{O})$ je uvijek **pozitivan**.

Odlučivost A_{OA}

Vidimo da za svaki n, za fiksni OA, postoji samo konačno mnogo konfiguracija duljine n (konkretno, ima ih $\#Q \cdot n \cdot (\#\Gamma)^n$). Kako relacija $\vdash$ čuva duljinu konfiguracije, zapravo imamo konačan *graf*

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

takav da se problem prihvaćanja riječi w svodi na problem dostupnosti bilo kojeg vrha iz skupa $\{q_\checkmark\} \times [0..|w|\rangle \times \Gamma^{|w|}$, iz vrha $(q_0, 0, w)$ — a taj pak problem možemo riješiti uobičajenim tehnikama za obilazak grafova: primjerice, DFS.

[Graf je *ogroman*, te algoritam nipošto nije polinoman — ali nije jasno da općenito možemo bolje, jer su OA nedeterministički.]

Moćniji strojevi — napomene

Ograničeni automati su puno moćniji od potisnih. No njihovo „programiranje" zna biti kompliciranо, pa ćemo se uglavnom zadovoljiti pisanjem „pseudokoda".

Također, postoje brojne varijacije u definiciji. Definicija koju smo ovdje naveli je jednostavna za zapamtiti i motivirati, ali nije previše praktična. Navest ćemo nekoliko „poboljšanja" OA, s kojima je lakše raditi, a pokazat ćemo da su ekvivalentna originalnom modelu, u smislu da prepoznaju istu klasu jezika.

U takvim dokazima, jedan smjer je obično trivijalan (jer poboljšani stroj može sve što može i obični OA), dok drugi obično ide metodom *simulacije*: moramo pokazati kako na OA napraviti isti „efekt" kao neki elementarni korak poboljšanog stroja.

Prvo poboljšanje: nemijenjanje trake

Formalno, svaki prijelaz pomiče OA duž trake s ulazom, ostvaruje se samo kad je na traci određeni znak α, i mijenja taj znak u β. Ponekad samo želimo pomak, bez obzira na stanje trake i bez promjene znaka. Možemo smatrati da smo dodali mogućnost prijelaza oblika $(q, *, p, *, d)$ nad proširenom abecedom trake Γ_*.

Takav OA je lako simulirati običnim OA, tako da svaki prijelaz $(q, *, p, *, d)$ zamijenimo s $\#\Gamma$ prijelazâ $(q, \alpha, p, \alpha, d)$, po jedan za svaki $\alpha \in \Gamma$.

Drugo poboljšanje: više tragova

Još jedna jednostavna varijacija je da se traka sastoji od više ($k > 1$) redova (*tragova*), svaki od kojih ima svoju abecedu Γ_i, odnosno svaka ćelija je stupac. OA sada može čitati i pisati po svakom tragu posebno, ali i dalje ima samo jednu poziciju u svakom trenutku. Formalno, relacija prijelaza mu je

$$\Delta_k \subseteq (Q \times \Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_k)^2 \times \{-1, 1\}.$$

Takav OA je lako simulirati običnim OA čija je radna abeceda Kartezijev produkt svih Γ_i. U svrhu reprezentacije ulaza, poistovjećujemo znak $\alpha \in \Sigma \subseteq \Gamma_1$ s k-torkom $(\alpha, \textvisiblespace_2, \ldots, \textvisiblespace_k)$, gdje je $\textvisiblespace_i$ neki istaknuti znak (*praznina*) u Γ_i, za $i \in \langle 1..k]$.

Često korišten specijalni slučaj je *traka s oznakama*: $k = 2$, gdje je $\Gamma_1 = \Gamma$, a $\Gamma_2 = \{\hat{}, \textvisiblespace\}$. Kažemo da je znak $\alpha \in \Gamma_1$ na traci *označen* ako „ispod" njega u drugom tragu piše $\hat{}$.

Treće poboljšanje: detekcija rubova trake

Koristeći traku s oznakama (i nemijenjanje trake), možemo OA obogatiti mogućnošću otkrivanja nalazi li se na (lijevom ili desnom) rubu trake. Ideja simulacije (npr. za lijevi rub) je:

OA u drugi trag zapiše ^, uz pomak lijevo. Nakon toga, ako u drugom redu čita ^, zna da je na lijevom rubu, i tada u drugi trag zapiše ␣, uz „pomak" lijevo.

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

4. i 5. poboljšanje: stajanje na mjestu, više završnih stanja

OAu možemo omogućiti stajanje na mjestu ($d \in \{-1, 0, 1\}$). U simulaciji prijelaza $(q, \alpha, p, \beta, 0)$ će koristiti detekciju ruba:

- 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). Simulacija toga se sastoji u dodavanju novog stanja $q_\checkmark$, i prijelazâ $(q, *, q_\checkmark, *, 0)$ za svaki $q \in F$.

Šesto poboljšanje: konačne varijable

Za proizvoljni **konačni** skup S, OAu možemo omogućiti pristup *varijabli* s koja može poprimati vrijednosti iz S. OA može postaviti s na vrijednost ovisnu o pročitanom znaku i/ili stanju, te ovisno o s može kasnije činiti akcije (svaka transformacija od s čiji je graf konačan može se upisati u relaciju prijelaza Δ).

OA s konačnom varijablom $s \in S$ i skupom stanja Q simuliramo običnim OA sa skupom stanja $Q \times S$, koji vizualiziramo kao 3D model od $\#S$ varijanti početnog OA, jedan iznad drugog. Prijelazi koji ne ovise o s isti su u svim razinama, dok se oni koji ovise o s razlikuju. Stanja koja mogu mijenjati s imaju „vertikalne" prijelaze među razinama.

Sedmo poboljšanje: višetračni ograničeni automat

Kompliciranije za simulaciju, ali vrlo korisno, poboljšanje OA sastoji se u tome da mu se dade $k > 1$ trakâ, svaka sa svojom pozicijom („glavom") za čitanje, koje se pokreću u nezavisnim smjerovima ali istodobno. Formalno, njegova relacija prijelaza je

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

Ulaz se nalazi na prvoj traci ($\Sigma \subseteq \Gamma_1$) — ostale su popunjene s $|w|$ $\sqcup_i$-ova, ali su i dalje **ograničene duljine** $|w|$.

Takav OA_k simuliramo pomoću OA s $2k$ tragova: svaku traku pamtimo kao traku s oznakama, gdje je označen samo onaj znak kojeg trenutno čita glava na toj traci. Sada se „prava" glava OA može kretati trakom: pročitati označene znakove na svakoj traci (pamti ih u konačnim varijablama), odabrati element od Δ_k, zapisati nove znakove na označena mjesta i pomaknuti oznake.

Kontekstne gramatike

Pokazuje se da se jezici prepoznati OAima mogu generirati gramatikama čija pravila i dalje zamjenjuju jednu varijablu nekim (nepraznim) nizom simbola, ali ovaj put u određenom kontekstu (niz simbola „oko" varijable koju zamjenjujemo).

Kontekstno pravilo je oblika $uAv \to ubv$, gdje su $u, v \in Y^*$, $A \in V$, te $b \in Y^+$.
Kontekstna gramatika je gramatika kojoj su sva pravila kontekstna.
Kontekstni jezik je onaj kojeg generira neka kontekstna gramatika.

Jer $b \neq \varepsilon$, svaki kontekstni jezik je pozitivan.
Kontekst uv *može* biti ε, dakle svaki pozitivan beskontekstni jezik je kontekstni (konkretno, generiran je Chomskyjevom gramatikom, a pravila čitanja i širenja su kontekstna). Pišemo $BK_+ \subseteq Kon$.

$BK_+ \subset Kon$

Štoviše, inkluzija je prava: opisujemo OA koji prepoznaje $L_{\equiv}$.

Na početku mora biti a (inače odbijamo riječ). Taj a označimo i čitamo slova a dalje. Prvi znak koji nije a mora biti b (opet, ako nije, odbijamo riječ). Taj b označimo i čitamo slova b dalje. Analogno označimo prvo slovo nakon tih bova (koje mora biti c), i čitamo cove do desnog ruba. Tada se vratimo do najdesnijeg označenog a i ponavljamo postupak, preskačući označena slova.

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) je *monotono* ako je $|u| \leq |v|$.
Gramatika je *monotona* ako su joj sva pravila monotona.

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 dvije faze. Prvu znamo pod imenom TERM iz pretvorbe BKG $\to$ ChNF — jedino što se sad „nesamostalni" znakovi mogu nalaziti i s lijeve strane pravila. Sve takve također zamijenimo novim varijablama.

Druga 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 $C_1, C_2, \ldots, C_n$:

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

Monotona gramatika za $L_{\equiv}$

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

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

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 $n-1$ primjene ②, imamo $\mathtt{a}^{n-1}S(B\mathtt{c})^{n-1}$, te nakon ① imamo $\mathtt{a}^n\mathtt{bc}(B\mathtt{c})^{n-1}$. Sad samo još treba izvesti $\mathtt{b}^n\mathtt{c}^n$ iz $\mathtt{bc}(B\mathtt{c})^{n-1} = \mathtt{b}(\mathtt{c}B)^{n-1}\mathtt{c} \overset{③}{\Rightarrow}{}^{n-1}\ \mathtt{b}(B\mathtt{c})^{n-1}\mathtt{c} \overset{④}{\Rightarrow} \mathtt{bb}(\mathtt{c}B)^{n-2}\mathtt{cc}$, ponavljanjem postupka iz gornjeg reda $n-1$ puta.

Monotona gramatika za $L_{\equiv}$ — 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 dodaje a, b i c. ② dodaje a, B i c. ③ samo mijenja redoslijed simbola, ne njihov broj. ④ 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 ($|w|_B = 0$). Kako se u takvoj w a može pojaviti samo na početku ili odmah iza prisutnog a, a b se ne može pojaviti nakon c, dobivamo $w \in L_{\equiv}$.

Pretvorba monotone gramatike u ograničeni automat

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

$t2_{0:1} := S$ # na početak drugog traga 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 $\rightarrow$ MG: osnovna ideja

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

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

No što je početna varijabla? Očito $q_0 w$, 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 traga. 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 traga (pa onda i s jednim, jer smo samo podebljali Γ, ne i Q): na početku prepišemo ulaz u drugi trag, zatim radimo sve isto kao originalni OA samo u drugom tragu, a umjesto originalnog završnog stanja odemo u „predzavršno" stanje u kojem odemo desno do kraja trake, i obrišemo drugi trag (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: $S \to q_0 A$, te $A \to \alpha \mid \alpha A$ za $\alpha \in \Sigma$. Kako gramatika simulira rad JOA, možemo na kraju dobiti istu riječ (precizno $q_\checkmark w$) ako i samo ako je JOA prihvaća.

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

Imamo još jedan problem iste vrste: početno pravilo zapravo mora biti nešto poput $S \to [q_0 A]$, 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 $(2 + \#Q)$, 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 $\mathtt{b}]$) ima konačno mnogo. To znači da ih možemo „slijepiti" (smatrati istim znakom), i skup Γ ostat će konačan. Sada npr. uklanjanje zatvorene 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.

Ograničenost izvoda u monotonim gramatikama

Prisjetimo se, izvodi u ChNF su fiksne duljine $2|w| - 1$, gdje je w izvedena riječ. 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 u izvodu duljine najviše $|w|$. Takvih riječi 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 $\left(\bigcup_{i=1}^{|w|} Y^i, \Rightarrow\right)$, koji možemo izračunati iz $\mathcal{G}$.

Na tom grafu također možemo ustanoviti (DFS-algoritmom, na primjer) postoji li put između S i w, što znači da je A_{MG} odlučiv problem. Isti algoritam možemo primijeniti na KG, jer je svaka kontekstna gramatika monotona. Kao i prije, imamo ekvivalentne odlučive probleme, te neformalno kažemo da je A_{Kon} odlučiv.

Zatvorenost klase kontekstnih jezika *Kon*

Potpuno istim dokazima kao za klasu BK, možemo vidjeti da je *Kon* zatvorena na uniju, konkatenaciju i Kleenejev plus (zašto ne i na Kleenejevu zvijezdu? ). Ali *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):

jer JOA sačuvaju traku i poziciju ako prihvate riječ, lako ih je „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.

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 jezikâ
- odlučitelji (*deciders*) — TO, za odlučivost problema
- nabrajači (*enumerators*) — TE, za generiranje jezikâ

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, a jezici pozitivni (u ChNF, KG i MG), vraćamo se svijetu koji smo upoznali s regularnim jezicima:

- Turingovi strojevi su podrazumijevano *deterministički*
- nedeterministički TS su ekvivalentni determinističkima
- prazna riječ ε više nema nikakav specijalni status
- gramatike više nisu primarni način zadavanja jezika
- (za klasu rekurzivnih jezika nemamo odgovarajuće gramatike)

Turingov prepoznavač — definicija

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

- ulaznu abecedu Σ
- radnu abecedu $\Gamma \supset \Sigma$ s istaknutim znakom $\textvisiblespace \in \Gamma \setminus \Sigma$
- 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 $(q, n, t) \in Q \times \mathbb{N} \times \Gamma_{\textvisiblespace}^{\mathbb{N}}$, gdje smatramo $0 \in \mathbb{N}$, te $\Gamma_{\textvisiblespace}^{\mathbb{N}} := \{(t : \mathbb{N} \to \Gamma) : t^{-1}[\Gamma \setminus \{\textvisiblespace\}] \text{ je konačan}\}$.

Primijetimo da je skup svih konfiguracija prebrojiv: uvjet da je traka od nekog mjesta nadalje konstanta $\textvisiblespace$, osigurava da trakâ 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 proglasimo q_X „običnim" 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 tragova, 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” trag) 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 trag 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 trag 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” traga (odnosno brišemo ih ako je „backup” trag 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-traga i jedne konačne varijable, $t \in \{-1, 1\}$ inicijalizirane na 1. Znakovi se čitaju iz prvog traga 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 $\Delta \subseteq R := Q \times \Gamma \times Q \times \Gamma \times \{-1, 1\}$. 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 sâmu trenutnu granu, kao niz (iz Δ^*) prijelazâ ($\Gamma_3 := \Delta \dot{\cup} \{\llcorner_3\}$).

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!] — primijetimo samo da trotračna simulacija ovdje ne pomaže, jer kada dobijemo $s = \mathcal{N}.q_X$, nemamo dovoljno podataka da zaključimo znači li to finalno odbijanje, ili treba dalje tražiti.

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 $\#Q \cdot |w| \cdot (\#\Gamma)^{|w|}$ 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) $= \mathrm{I}$:
        prepiši t_1 na t_3; premotaj t_3
        prepiši izlaznu traku na t_4; $s := \mathcal{T}.q_0$
        while read(t_3) $= \mathrm{I}$:
            simuliraj korak od $\mathcal{T}$ (u stanju s) na t_4
            obriši jedan I s kraja t_3
        if $s = \mathcal{T}.q_{\checkmark}$: ispiši (idi u $q_{\hookleftarrow}$)
        obriši jedan I s kraja t_2
        inkrementiraj riječ iz $(\mathcal{T}.\Sigma)^*$ 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 *neprazih* 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

Da bismo dokazali zatvorenost klase RE na jezične operacije, trebaju nam 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

Vrijeme je da se ozbiljnije (formalnije) pozabavimo problemima. Htjeli bismo pomoću Turingovog stroja simulirati neki stroj, ali ne fiksan, već zadan kao ulaz. Na prvi pogled, nema problema: simulirani stroj i njegov ulaz je moguće prikazati kao nizove znakova. Ipak, nije tako jednostavno: 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.

[Priča o *charset*ima; Unicode — praktičnost protiv apstrakcije.]

Pouka: 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 KA 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 $D = \Sigma$ (BSOMP h fiksira $D \cap \mathbb{N}$, a na $D \cap (\Sigma \cup \Gamma)$ djeluje kao Unicode). Jasno je kako prirodno preslikavanje proširujemo na riječi i jezike:

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

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 KA* ($h(\mathcal{M})$ za neki KA $\mathcal{M}$) 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:

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

Primjer: gornji KA 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 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).

Problemi — formalno

Da bismo definirali jezik pomoću kodova, moramo imati prirodne strukture, odnosno specificirati prirodno preslikavanje. No često govorimo o jezicima kojima konkretno prirodno preslikavanje uopće nije bitno. Takve jezike zovemo *problemi*.

Primjer takvog jezika je

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

Ako je $c\big(h_1(\mathcal{M})\big)c\big(h_1(w)\big) \in A_{\mathrm{KA}}$ za neko prirodno preslikavanje h_1, tada je očito i $c\big(h_2(\mathcal{M})\big)c\big(h_2(w)\big) \in A_{\mathrm{KA}}$ za bilo koje prirodno preslikavanje h_2 — naime, drugačije smo kodirali znakove u w, ali smo ih *jednako tako* drugačije kodirali i unutar $\mathcal{M}.\Sigma$ i $\mathcal{M}.\delta$.

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_{KA} 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 KA zadanu riječ?

Dokažimo da je jezik

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

rekurzivan. Prema konvenciji o razvrstavanju,
kôd $\langle \mathcal{M}\rangle$ je na drugoj traci, a kôd $\langle w\rangle$ je na trećoj
[pre-processing nije potreban jer KA 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 KA 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
    $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ KA 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 KA, ali kasnije ćemo ih gledati i za druge sustave). Primjer problema s pseudokodom:

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

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 jezikâ

Još jedan primjer odlučivog problema je

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

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

Tada treba taj kod dati odlučitelju za E_{KA}, i prihvatiti ako on prihvati, a odbiti ako on odbije.

Prepoznajemo to kao *svođenje* problema EQ_{KA} na problem E_{KA}.

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 KA.

Štoviše, kako su svi postupci pretvorbe NKA $\rightarrow$ KA, 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_{KA}, E_{KA} i EQ_{KA}, 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 $A\}$
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!)

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

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

Problem beskontekstne univerzalnosti

Za razliku od klase regularnih jezika, klasa beskontekstnih jezika nije zatvorena na komplement, pa se ima smisla pitati 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 prvo pokaže da je *problem odbijanja* za Turingove prepoznavače

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

neodlučiv i svediv na ALL_{PA}. Neodlučivost $\bar{A}_{\mathsf{TP}}$ ćemo pokazati malo kasnije, a svođenje se dobije tako da za svaki par $(\mathcal{T}, w)$ konstruiramo PA koji prihvaća sve riječi (nad na odgovarajući način proširenom abecedom) koje *nisu* kodovi $\mathcal{T}$-izračunavanja s w. Detaljni opis je u Sipser, Teorem 5.13, stranice 225–226.

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 $\langle P' \rangle$.

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. Ideja je vrlo slična, zapravo još jednostavnija nego za BK: 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

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

Detalji su u Sipser, Teorem 5.10, stranice 223–224.
[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; $\textsc{term}-\textsc{bin}$ konstrukcija 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_{KA}. 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.
- Moramo nekako (fiksno, po dogovoru) kodirati smjerove pomaka (npr. $h(1) := 1$, $h(-1) := 2$) jer $-1 \notin \mathbb{N}$.
- 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 $\mathit{Russell} := \{\langle \mathcal{T} \rangle : \mathcal{T} \text{ je TP nad } \Sigma \wedge \langle \mathcal{T} \rangle \notin L(\mathcal{T})\}$
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 $\langle \mathcal{R} \rangle \in \mathit{Russell} \iff \langle \mathcal{R} \rangle \notin \mathit{Russell}$, 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 $HB \preceq A_{\mathrm{TP}}$ ($t \in HB \iff t() \in A_{\mathrm{TP}}$), no vrijedi i $A_{\mathrm{TP}} \preceq HB$: 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)