UNIVERZA EDVARDA KARDELJA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

JOHN E. HOPCROFT, JEFFREY D. ULLMAN

UVOD V TEORIJO AVTOMATOV, JEZIKOV, IN IZRAČUNOV

LJUBLJANA 1986

Naslov izvirnika:
John E. Hopcroft, Jeffrey D. Ullman:
INTRODUCTION TO AUTOMATA, THEORY,
LANGUAGES AND COMPUTATION

John E. Hopcroft in Jeffrey D. Ullman
UVOD V TEORIJO AVTOMATOV, JEZIKOV
IN IZRAČUNOV

Delen in prirejen prevod

Poglavja izbral in z dovoljenjem avtorjev ter založbe
prevedel dr. Boštjan Vilfan, dipl. ing., izredni profesor
na Fakulteti za elektrotehniko v Ljubljani

Izdala in založila Fakulteta za elektrotehniko
v Ljubljani
Pripravlja Komisija za tisk — urednik Božidar Magajna
Tisk »F. Tori«, Ljubljana
Naklada 200 izvodov

VAŽNO OPOZORILO

Pričujoča knjiga je bila natiskana z navadnim računalniškim vrstičnim tiskalnikom. Zaradi tega imajo nekateri simboli nenavaden videz. Bralcu priporočamo, naj si ogleda seznam simbolov, ki je na koncu knjige.

KAZALO

1. POGLAVJE

UVOD

1.1 Nekaj besed o snovi

Področje teorije jezikov, avtomatov in izračunov, ali bolje, njen cilj, je poiskati natančne oblike za vprašanja, ki se postavljajo v zvezi z računanjem -- in seveda odgovore na njih. Da bi dosegla ta cilj, preučuje najrazličnejše matematične modele računalnikov z namenom ugotoviti njihove prednosti in pomanjkljivosti.

Primera vprašanj v zvezi z računanjem sta:

1. Ali je neko funkcijo f možno izračunati z določeno vrsto računalnika?
2. Ali je neko funkcijo f teže izračunati kot drugo funkcijo g?

Če hočemo najti odgovor na ta in podobna vprašanja, moramo pojme, s katerimi so ta vprašanja postavljena, in ki imajo tu le nekakšno neprecizno, ad hoc obliko, določiti natančno. Potem pa jih lahko matematično analiziramo. Seveda je najvažnejsi pojem

v zvezi z računanjem računalnik in je potrebno predvsem ta pojem razčistiti in podati njegovo "matematično definicijo".

Primeri matematičnih modelov računalnikov so:

1. Končni avtomat.
2. Skladovni avtomat.
3. Turingovi stroji.

V zvezi z vsako teorijo se seveda postavlja vprašanje njene uporabnosti. V kakšnem pomenu in v kolikšni meri je teorija avtomatov in izračunov uporabna? Odgovor je, da nam ta teorija predvsem daje vpogled v zmožnosti in omejitve posameznih vrst "računalnikov". V nekaterih maloštevilnih primerih pa nam daje tudi konkretne napotke, kako se lotiti določenih praktičnih problemov (n.pr. v zvezi z izdelavo jezikovnih prevajalnikov).

Na tem mestu naj podamo kratko vsebino razvoja teorije od njenega nastanka do danes. Prikazali bomo razvoj osnovnih pojmov teorije s stališča njihovega mesta v hierarhiji računalniskih modelov (torej na prvem mestu so preprostejši modeli), ne pa kronološko.

Najpreprostejši model računalnika je končni avtomat. Definirala sta ga McCulloch in Pitts leta 1943, ko sta iskala primeren teoretičen model za živčno celico (nevron). Kleene je leta 1959 uporabil za opis jezikov, ki jih sprejemajo končni avtomati, mehanizem, ki mu pravimo regularni izrazi. Na modelu končnega avtomata so veliko delali Huffman, Moore in Mealy,

kakor tudi Trahtenbrot, dokončno obliko pa sta mu dala Rabin in Scott l. 1959. Pri končnih avtomatih lahko opozorimo na zelo važno "dualnost" med nekim avtomatom in sintaktičnim opisom jezika, ki ga avtomat sprejema. Dualnost je v tem, da neki razred jezikov lahko definiramo bodisi z vrsto avtomatov, ki ga razpoznavajo oz. sprejemajo, ali pa z nekim razredom sintaks, ki natančno opisujejo zgradbo jezikov v razredu. N. pr. jezike, ki jih sprejemajo končni avtomati, lahko prav tako opišemo na sintaktičen način z regularnimi izrazi. To velja tudi za druge vrste računalnikov: vsaka vrsta računalnikov je povezana z nekakšnim razredom jezikov, katerega je sposobna razpoznavati ali sprejemati, isto vrsto jezikov pa je po navadi možno opisati tudi z ustreznimi sintaksami ali gramatikami.

Leta 1956 je Chomsky opisal model kontekstno neodvisnih jezikov, ker se je izkazal za primernega pri jezikoslovnih raziskavah. Isto vrsto jezikov sta uporabljala Backus in Naur leta 1959 in 1960 pri opisu programskega jezika Algol, prav tako pa Schutzenberger in Floyd l. 1962. Skladovni avtomat, ki sprejema kontekstno neodvisne jezike, so opisali Oetinger l. 1961, Schutzenberger l. 1963 ter Chomsky l. 1962.

Najsplošnejši (najmočnejši) model računalnika je Turingov stroj, ki ga je definiral Turing l. 1936. Enakovredne modele so definirali Kleene, Church, Post in Markov. V zvezi s Turingovim strojem se postavlja vprašanje izračunljivosti oz. kaj je ta stroj sposoben izračunati (seveda se isto vprašanje postavlja v zvezi s poljubno vrsto računalnika). S tem

vprašanjem se je najprej ukvarjal Turing; za njim pa cela vrsta raziskovalcev.

Končno so se v 60-ih letih pojavile prve študije o zahtevnosti računanja (kompleksnosti) -- torej kolikšna je (primerno definirana) cena, ki jo moramo plačati (n. pr.) za izračun neke funkcije. Raziskovalci, ki so prispevali k tem raziskavam so Hartmanis, Stearns l. 1965, Blum l. 1967 ter kasneje McCreight, Meyer l. 1969 in vrsta drugih.

1.2 Matematično predznanje

Za matematično predznanje, ki je potrebno za ta predmet, ugotavljamo, da je "majhno po obsegu, nezanemarljivo pa po globini". Potrebno je predvsem znati skrbno ravnati z osnovnimi pojmi v matematiki.

Tisto, kar predvsem predpostavljamo, je poznavanje osnovnega jezika matematične logike, oziroma bolj natančno predikatnega računa prvega reda. Pri tem mislimo predvsem na to, da je potrebno poznati pomen osnovnih logičnih veznikov

or , and , ==> , <==> ter kvantifikatorjev $\forall$ in $\exists$. Na tej podlagi se pa gradijo vsebinski matematični pojmi, med katerimi je na prvem mestu pojem množice. Množice pogosto definiramo z nekakšnimi lastnostmi: $A = \{x \mid P(x)\}$, kar predstavlja množico vseh elementov x z lastnostjo, da je P(x) resnično. V zvezi z množicami imamo osnovno primitivno relacijo pripadnosti, $x \in X$, ki pomeni "x pripada množici X" (seveda

poleg osnovne relacije enakosti). S to relacijo lahko definiramo nove relacije, kot so n. pr. $\subseteq$ (relacija vsebovanosti): $A \subseteq B \iff \forall x[x \in A \Rightarrow x \in B]$.

Če sta podani mnozici A in B, poznamo vrsto operacij, s katerimi tvorimo nove množice:

unija: $A \cup B = \{x \mid (x \in A) \text{ or } (x \in B)\}$

presek: $A \cap B = \{x \mid (x \in A) \text{ and } (x \in B)\}$

kartezični produkt: $A \times B = \{\langle x,y\rangle \mid (x \in A) \text{ and } (y \in B)\}$ [1]

potenčna množica: $POT(A) = \{S \mid S \subseteq A\}$

Množice določenih oblik so tako pomembne, da jim dajemo posebna imena, čeprav bi jih lahko opisali s prej omenjenimi operacijami.

Korespondenca med množicama S in T je element POT(S x T) (gl. sl. 1.1).

---

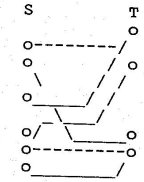
1 Pozorni bralec bo takoj ugotovil, da nekaj ni v redu: vpeljali smo simbol za urejeni par <,>, ne da bi ga definirali s primitivnima pojmoma množice in pripadnosti. Vendar je to možno in si bralec lahko poišče ustrezno izpeljavo v delih Prijatelj [PN] ali Halmos [HPR].

sl. 1.1 Korespondenca med S in T.

Funkcija ali preslikava med S in T, kar simbolično zapišemo v obliki f: S --> T, je korespondenca med S in T z dodatno lastnostjo

$$\langle x_1, y_1\rangle \in f \text{ and } \langle x_1, y_2\rangle \in f \Rightarrow y_1 = y_2$$

(gl. sl. 1.2). Če velja $\langle x,y\rangle \in f$, po navadi to zapišemo kot $y = f(x)$



sl. 1.2 Funkcija f med S in T.

Funkcija je tako pomemben pojem, da prepoznamo vrsto posebnih primerov in izpeljanih pojmov, ki so povezani z njo. Domena ali področje definiranosti neke funkcije f: S --> T je $\{x \mid \exists y[f(x) = y]\}$. Domeno f označujemo z DOM(f). Če velja DOM(f) = S pravimo, da je funkcija totalna sicer je parcialna.

Včasih se omejimo samo na totalne funkcije. Kodomena ali zaloga vrednosti f: S --> T je CODOM(f) = {y | ∃x[f(x) = y]}. Za neko funkcijo f: S --> T pravimo, da je injekcija, če velja x ≠ y ==> f(x) ≠ f(y), ter surjekcija v primeru CODOM(f) = T. Če je neka funkcija oboje, injekcija in surjekcija, pravimo, da je bijekcija. Če je podano f: S --> T ter U |⁻ S, potem pravimo funkciji g: U --> T, ki sovpada z f na U omejitev ali restrikcija f na U. Nasprotno pa je f razširitev g z U na S. Očitno je, da medtem, ko ima vsaka funkcija le eno omejitev na poljubno podmnožico svoje domene, ima lahko več razširitev na neko mnozico, ki vsebuje njeno domeno (če velja A |⁻ B, je A podmnožica B, B pa je nadmnožica A). Če sta podani f: S --> T in g: T --> U, potem pravimo funkciji h: S --> U kompozicija ali kompozitum f in g v primeru, ko velja h(x) = g(f(x)) in pišemo h = gf.

Če je podana množica A, potem pogosto elementu R ∈ POT(A x A) pravimo relacija in v primeru <x,y> ∈ R pišemo x R y. Relacije z določenimi lastnostmi so prav posebno pomembne. Tako na primer relaciji, ki ima lastnosti

1. Refleksivnost. ∀a[a R a];
2. Simetrija. ∀a,b[a R b ==> b R a];
3. Tranzitivnost. ∀a,b,c[a R b and b R c ==> a R c];

pravimo ekvivalenčna relacija. Pomembno je, da poljubna ekvivalenčna relacija E na A določa porazdelitev množice A na

medsebojno tuje podmnožice, ki jim pravimo ekvivalenčni razredi E. Velja torej:

1. $\underline{a} = \{x \mid x \; E \; a\}$
2. $A = \bigcup_{a \in A} \underline{a}$
3. $\forall a,b[\underline{a} = \underline{b} \text{ or } \underline{a} \cap \underline{b} = \emptyset]$

Relaciji R, za katero ne velja refleksivnost, ne velja simetrija, velja pa tranzitivnost, pravimo relacija urejenosti in jo pogosto označujemo z <.

Gotovo najpomembnejši matematični objekt je množica naravnih števil NN = {0, 1, 2, ... } skupaj z operacijo naslednika $a^+ = a + 1$. V navadi je, da lastnosti matematičnih objektov opisujemo s sistemi aksiomov. Zelo znan aksiomatičen opis naravnih števil je podal Peano, čigar aksiomi so naslednji:

P1. $\forall a[a^+ \neq 0]$
P2. a^+ je injektivna funkcija.
P3. (Aksiom indukcije)

$$((S \mid\!\!\!- NN) \text{ and } (0 \in S) \text{ and } (\forall a[(a \in S) ==> (a^+ \in S)])) ==> (S = NN)$$

P3 je osnova za dokaze na podlagi prvega načela indukcije. V tem primeru imamo neki stavek P(n), ki je definiran pri vseh n. Želimo dokazati, da je pri vseh n P(n) resnično. Naj bo S množica naravnih števil, pri katerih je P(n) resničen. Želeno trditev bomo dokazali, če ugotovimo, da je S enako NN. Za to je

potrebno le dokazati, da 0 pripada množici S in da iz P(n) sledi P(n+1), nato pa želena trditev sledi iz P3. Za vajo naj bralec izpelje opisani postopek za naslednji stavek:

$$\text{VSOTA } i = \frac{1}{2}n(n+1) \quad (i = 0 \text{ do } n)$$

Naravna števila imajo še druge znane lastnosti (ki pa jih seveda lahko izpeljemo iz P1 - P3). Na primer veljajo naslednje trditve:

PP1. $\forall x,y[(x \geq y) \text{ and } (y \geq x) \Rightarrow (x = y)]$
PP2. $\forall x,y,z[(x \geq y) \text{ and } (y \geq z) \Rightarrow (x \geq z)]$
PP3. $\forall x,y[(x \geq y) \text{ or } (y \geq x)]$

zato pravimo, da je NN <u>popolnoma urejeno</u>. Velja tudi

PP4. $(S \subseteq NN) \text{ and } (S \neq \emptyset) \Rightarrow \exists 1 \in S[x \in S \Rightarrow 1 \leq x]$

zaradi česar pravimo, da je NN <u>dobro urejeno</u>. Kako na primer izpeljemo PP4 iz P1 - P3? Naj bo M množica naravnih števil m z lastnostjo $\forall s \in S[m \leq S]$. Potem imamo $0 \in M$, če pa $s \in S$, potem velja $s^+ \notin M$. Torej velja $M \neq NN$ in na podlagi P3 dobimo, da eksistira $1 \in M$ z lastnostjo $1^+ \notin M$. Tedaj je 1 iskano število, kajti velja $1 \leq s$ za vse $s \in S$ (definicija M) in tudi $1 \in S$, saj bi sicer imeli $1 < s$ pri vseh $s \in S$ in torej $1^+ \leq s$, kar je pa v protislovju z $1^+ \notin M$. |_|

PP4 je osnova za dokaze na podlagi <u>drugega načela indukcije</u>: naj imamo za vse $n \in NN$ stavek E(n). Recimo, da lahko iz resničnosti E(s) pri vseh $s < r$ izpeljemo resničnost E(r). Potem je E(r) resnično za vse elemente NN. Kako izpeljemo to načelo iz PP4? Naj bo S množica na kateri je E(r) neresnično in recimo, da velja $S \neq \emptyset$. Potem ima ta množica najmanjši element (na podlagi PP4). Naj bo ta element r. Ker pa je E(s) resnično za vse $s < r$, dobimo protislovje z izhodiščno predpostavko.

<u>Primer dokaza na podlagi drugega načela indukcije</u>. Za naravno število $p \neq 0, 1$ pravimo, da je <u>praštevilo</u>, če velja $p \neq qr$ pri vseh $q,r < p$. Potem velja:

VSAKO NARAVNO ŠTEVILO > 1 SE DA IZRAZITI KOT PRODUKT PRAŠTEVIL.

<u>Dokaz</u> Naj velja trditev za vsa naravna števila < n. Dokazali bomo, da velja tudi za n. Resnično: n je bodisi praštevilo ali pa ni. V prvem primeru je trditev dokazana, v drugem primeru pa velja $n = st$ pri $s,t < n$. Na podlagi induktivne predpostavke pa se dasta oba, s in t, tako predstaviti, iz česar sledi, da podobno velja tudi za n. |_|

<u>Opomba</u> E(n) je stavek "n = 0 ali n = 1 ali n se da zapisati kot produkt praštevil".

Naslednji pojem, ki ga bomo tu na kratko predstavili, je pojem jezika. Po navadi imamo opravka z neko končno množico elementov, ki jim pravimo <u>znaki</u> ali <u>simboli</u>, množici v tem primeru pa pravimo <u>abeceda</u>.

Primera SIGMA = {0, 1}, GAMA = {A, B,..., Z}

Beseda dolžine n nad abecedo SIGMA je preprosto element

SIGMA = SIGMA x SIGMA x ... x SIGMA (n krat).

Pišemo jo preprosto takole: $w = sgm_1 sgm_2 ... sgm_n$, pri čemer velja $sgm \in SIGMA$, $1 \leq i \leq n$. Beseda pa je beseda dolžine n pri nekem končnem n. Nad besedami je definirana binarna operacija, ki ji pravimo stik ali konkatenacija, in katere pomen je preprosto stikanje ali združitev dveh besed v eno samo. Naj bosta x in y besedi, $x = x_1 x_2 ... x_m$ in $y = y_1 y_2 ... y_n$. Potem je njun stik $x \circ y = x_1 x_2 ... x_m y_1 y_2 ... y_n$. Zelo koristno je, če imamo neko besedo, ki nam lahko rabi kot enota pri operaciji stika. To besedo označujemo z ϵ in velja torej: $x\epsilon = \epsilon x = x$ pri vseh x. V takem primeru postane množica besed monoid. Besedni enoti pravimo tudi prazna beseda, njena dolžina pa je seveda 0.

Množico besed nad abecedo SIGMA označujemo s $SIGMA^*$, velja pa

$$SIGMA^* = \bigcup_{n=0}^{\infty} SIGMA^n$$

pri čemer je $SIGMA^0 = \{\epsilon\}$, kjer je ϵ prazna beseda.

Sedaj pa lahko definiramo pojem formalnega jezika: to je pri neki izbrani abecedi SIGMA preprosto neka podmnožica $SIGMA^*$. Najbolj preprosti primeri so n.pr. $\emptyset$, $\{\epsilon\}$, $SIGMA^*$.

Še en pojem, ki ga je koristno tukaj omeniti, je pojem grafa. Graf oz. točneje usmerjen graf je par $\langle V, E \rangle$, kjer je V neka množica, ki ji pravimo množica vozlišč, $E \subseteq V \times V$ pa je množica povezav. Zaporedju vozlišč $v_1, v_2, ..., v_k$ z lastnostjo $\langle v_i, v_{i+1} \rangle \in E$, $1 \leq i \leq k - 1$, pravimo pot (dolžine k). Če velja $k > 1$ in $v_1 = v_k$, imamo opravka s ciklom. Vhodna stopnja nekega vozlišča v je število vozlišč x z lastnostjo $\langle x,v \rangle \in E$. Podobno je izhodna stopnja vozlišča v število vozlišč x z lastnostjo $\langle v,x \rangle \in E$. E seveda lahko pojmujemo kot relacijo nad V in če je le-ta simetrična, potem za graf rečemo, da je neusmerjen.

Poseben primer usmerjenega grafa je drevo, ki ima naslednje lastnosti:

1. Eksistira posebno vozlišče, ki mu pravimo koren, ki nima predhodnikov in od katerega izhaja pot do vsakega vozlišča.
2. Vsako vozlišče, razen korena, ima vhodno stopnjo 1.

Nekateri pomembni pojmi pri drevesih so naslednji: stopnja drevesa je maksimalna izhodna stopnja nekega vozlišča v drevesu. Končno vozlišče je vozlišče z izhodno stopnjo, ki je manjša od stopnje drevesa (v nekaterih primerih definiramo končno vozlišče kot vozlišče z izhodno stopnjo enako nič). Ostala vozlišča so

notranja. Veja v drevesu je pot od korena do nekega končnega vozlišča. Višina drevesa je maksimalna dolžina veje, višina vozlišča pa dolžina poti od korena do vozlišča. Dvojiško drevo je drevo stopnje 2. Iz logične doslednosti je koristno vpeljati tudi pojem praznega drevesa, ki ima 0 vozlišč.

Na koncu uvoda bomo raziskali logične zveze med kriteriji popolnega ravnotežja pri dvojiških drevesih.

Pri različnih aplikacijah se srečujemo s popolnoma uravnoteženimi dvojiškimi drevesi. Popolno ravnotežje definiramo na več načinov, med katerimi se največ uporabljajo naslednji:

1. Dvojiško drevo je popolnoma uravnoteženo, če je pri vsakem vozlišču razlika med številom vozlišč v levem in desnem poddrevesu ≤ 1.

2. Dvojiško drevo je popolnoma uravnoteženo, če je razlika v dolžini med poljubnima dvema vejama ≤ 1.

2'. Dvojiško drevo je popolnoma uravnoteženo, če je pri vsakem vozlišču razlika v višinah med levim in desnim poddrevesom ≤ 1.

Kakšna je relacija med 1, 2 in 2'?

(a) 2 ==> 2' in 2' ==> 2, oziroma 2 <==> 2'. To lahko izpeljemo iz dejstva, da se poljubni dve veji nekje srečata (če ne drugje pa pri korenu).

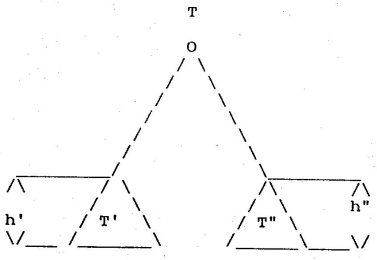
(b) 2 ==> $2^{h-1} \leq n \leq 2^h - 1$ pri $h \geq 1$, kjer je h višina drevesa, n pa število vozlišč. To trditev dokažemo takole: Če so vse veje dolžine h, potem drevo vsebuje $2^h - 1$ vozlišč (dokaz

z indukcijo). Če pa ima samo ena veja dolžino h, ostale pa h - 1, potem vsebuje 2^{h-1} vozlišč (sledi iz prejšnega dejstva). Torej je trditev resnična. Dve relaciji iz trditve lahko tudi zapišemo v obliki $h = \lfloor \log_2 n \rfloor + 1$ pri $n \geq 1$.

(c) 2 =/=> 1. To ugotovimo s protiprimerom, ki je prikazan na sl. 1.3.

sl. 1.3 Protiprimer za 2 ==> 1

(d) 1 ==> 2. Dokaz izpeljemo z indukcijo po višini drevesa. Pri h = 1 je trditev očitna. Sedaj pa naj bo h > 1 in naj trditev velja za drevesa višin h - 1 in h - 2. Oglejmo si diagram drevesa na sl. 1.4.



sl. 1.4 Drevo v dokazu 1 ==> 2

Prvotno dvojiško drevo je označeno s T, levo in desno poddrevo pa z T' oziroma T". Na podlagi definicij imamo h = max(h',h") + 1, kjer h' in h" predstavljata višini T' in T", po vrsti, in torej velja, da sta h' in h" bodisi h - 1 ali h - 2, kar pomeni, da induktivna predpostavka velja za T' in T". Sedaj pa na podlagi predpostavke 1 nastopata dva primera:

primer 1: n' = n". Tedaj imamo h' = h" = $\lfloor \log_2 n' \rfloor$ + 1. Vse veje v T' in T" so dolžine h' ali h' - 1, torej so vse veje v T dolžine h' ali h' + 1 in 2 je resnično.

primer 2: n' = n" + 1 (primer n" = n' + 1 obravnavamo podobno). Sedaj pa lahko imamo (a) n' = 2^k - 1 pri nekem k $\geq$ 1. V tem primeru so vse veje T' dolžine h' = k. Za T" pa velja n" = 2^k in vse veje T" so dolžine k razen ene, ki je dolžine h" = k + 1 (gl. dokaz (b) zgoraj). Torej 2 je resnično. Sicer imamo (b) n' $\neq$ 2^k - 1 in velja h' = $\lfloor \log_2 n' \rfloor$ + 1 = $\lfloor \log_2 n" \rfloor$ + 1 =

h" in zopet imamo, da so vse veje v T' in T" dolžine h' ali h' - 1. Torej 2 je zopet resnično. Na koncu lahko še omenimo, da smo (d) dokazali s posebnim primerom drugega načela indukcije. Namreč nismo potrebovali resničnost induktivne predpostavke za vse r < n temveč le pri r = n - 1 in n - 2.

Na koncu uvoda naj podamo napotek za nadaljnji študij snovi, ki jo obravnava uvod. Danes obstaja izredno mnogo učbenikov za to snov: skoraj vsak učbenik algebre ali diskretnih struktur na visokošolski stopnji obravnava osnovne pojme logike in teorije množic (primera podobnih učbenikov sta Prijatelj [PN] in Jacobson [JNI]), novejši učbeniki pa tudi obravnavajo jezike in grafe.

1.3 Naloge

1.1 Naj bo S = {1,2,...}. Poiščite primer dveh funkcij alf, bta iz S v S tako, da velja alf o bta = 1_S, vendar bta o alf $\neq$ 1_S. Ali lahko pride do tega, če je alf bijektivna?

1.2 Pokažite, da je alf: S --> T injektivna natanko, ko obstaja funkcija bta:T --> S ter bta o alf=1_S, ter surjektivna natanko, ko obstaja bta:T --> S in alf o bta=1_T. V obeh primerih preučite trditev: Če je bta enolično določena, potem je alf bijektivna.

1.3 Koliko različnih binarnih relacij obstaja na neki množici dveh elementov? Kaj pa treh elementov, ali v splošnem n elementov? Koliko od teh relacij je ekvivalenčnih relacij?

1.4 Za naravna števila a, b, c, d dokažite, da iz $a \geq b$ in $c \geq d$ sledi $a+c \geq b+d$ in $ac \geq bd$.

2. POGLAVJE

KONČNI AVTOMATI IN REGULARNI IZRAZI

2.1 Sistemi s končnim številom stanj

Končni avtomat je abstrakcija različnih mehanizmov, ki imajo diskretne vhode, diskretne izhode ter končno število notranjih "stanj".

Primeri

1. Mehanizem za nadzor nad dvigalom. V tem primeru so vhodi klici od zunaj in ukazi znotraj dvigala, izhod pa sta podatka, ali se dvigalo giba, ter njegova smer; stanje mehanizma pa je določeno s položajem dvigala ter z "nedokončanimi" klici in ukazi.

2. Zaporedni seštevalnik dveh dvojiških števil. Vhod predstavlja par bitov, izhod je en bit vsote, stanje pa prisotnost ali odsotnost prenosa v naslednji višje mesto.

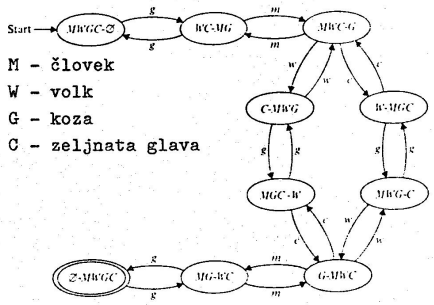
3. Telefonska centrala. Vhodi so pozivi, izhodi prevezave (vzpostavitev zvez), stanje trenutne povezave.

4. Pregledovalnik pravajalnika. Vhodi so znaki, izhodi prepoznani simboli, stanja pa posamezne faze pri odločanju o naslednjem simbolu.

i.t.d.

i.t.d.

Oglejmo si podrobneje pojem končnega avtomata na primeru naslednje znane uganke: na rečnem bregu so človek, volk, koza in zeljnata glava. Vsi se morajo prepeljati do nasprotnega brega s čolnom, ki drži le dva na enkrat. Seveda pa volk nikdar ne sme ostati sam s kozo niti koza z zeljnato glavo. Kako naj stvar izpeljejo?

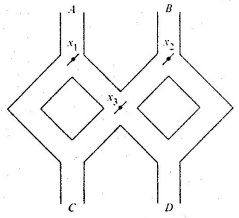


sl. 2.1 Uganka o človeku, volku, kozi in zeljnati glavi

Primeren način za predstavljanje končnega avtomata je graf, kjer vozlišča predstavljajo stanja, povezave pa prikazujejo možne prehode iz stanja v stanje. Identiteta vhoda, ki uresničuje neki prehod je zapisana ob povezavi. V primeru na

sl. 2.1 predstavljata stanja množici elementov, ki so na obeh bregovih reke, vhodi pa nakazujejo, kaj se je (poleg človeka) peljalo čez reko. Pri končnih avtomatih je najpogosteje eno stanje "začetno", pri njem se namreč postopek začne, nekaj stanj pa je "končnih" in pri njih se lahko postopek uspešno konča. V tipičnem primeru nas zanima zaporedje vhodov, ki nas pripelje iz začetnega stanja v neko končno stanje. V zgornjem primeru je začetno stanje {MWGC}-Ø, končno je stanje Ø-{MWGC}, primer zaporedja, ki nas pripelje iz začetnega v končno stanje pa je gmcgwmg. V splošnem bomo rekli množici zaporedij (besed), ki nas pripeljejo do nekega končnega stanja "jezik, ki ga sprejema podani končni avtomat".

Predno podamo formalno definicijo vseh teh pojmov si oglejmo še en primer



sl. 2.2 Igra s kroglicami

Obstaja igračka, ki je sestavljena iz nekakšne škatle z dvema luknjama na zgornji strani in dvema na spodnji. V zgornje luknje spuščamo kroglice, ki včasih priletijo ven skozi eno od spodnjih lukenj, včasih pa skozi drugo. Cilj je uganiti, skozi katero luknjo bo kroglica priletela ven. Seveda vsebuje škatla nekakšen mehanizem, zaradi katerega zveza med vhodom in izhodom ni prav enostavna. Mehanizem temelji na razporeditvi nihalnih delov, ki spreminjajo položaj, ko se kroglica zakotali čez njih (gl. sl. 2.2). V tem primeru vidimo, da imamo opravka z dvema možnima vhodoma in dvemi izidi. Vendar je v tem primeru pomembno, da izhoda ne moremo neposredno povezati s stanjem, kot je to bilo možno prej, temveč je funkcija obeh, stanja in vhoda. To naj bo prvo opozorilo, da se končni avtomati pojavljajo v mnogih različicah, za katere pa se izkaže (kot bomo to videli) da so vse v nekem smislu enakovredne.

2.2 Osnovne definicije

Začeli bomo z osnovno različico, ki ustreza prvemu primeru v uvodu, kasneje pa bomo opisali tudi druge.

Torej

2.1. DEFINICIJA. Deterministični končni avtomat (DKA) M je peterka $\langle Q, SIGMA, dlt, q_0, F\rangle$ kjer je Q neka končna množica stanj, SIGMA je neka končna abeceda, dlt je neka totalna funkcija Q x SIGMA --> Q (funkcija prehodov), $q_0 \in Q$ je neko

posebno začetno stanje in $F \subseteq Q$ je neka množica končnih stanj.

Sedaj pa moramo definirati, kaj je to "jezik nekega determinističnega avtomata". Pred tem pa potrebujemo točno definicijo pojma, da nas "neka beseda pripelje iz stanja q v stanje r". Zato bomo razširili funkcijo. dlt tako, da bo definirana na besedah in ne le na simbolih:

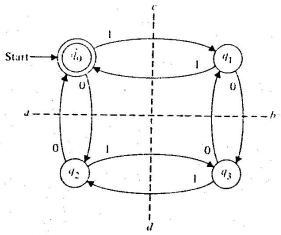
2.2. DEFINICIJA. Naj bo dana dlt: Q x SIGMA --> Q. Potem definiramo $\hat{dlt}$: Q x $SIGMA^*$ --> Q takole

1. $\forall q \in Q[\hat{dlt}(q,\epsilon) = q]$,
2. $\forall q \in Q, w \in SIGMA^*, a \in SIGMA[\hat{dlt}(q,wa) = dlt(\hat{dlt}(q,w),a)]$

Torej če velja $\hat{dlt}(q,w) = r$, potem to ustreza stavku "vhodna beseda w nas pripelje iz stanja q v stanje r". Odslej bomo pogosto za $\hat{dlt}$ uporabljali isto oznako kot za dlt. Razlika bo le v tem, da je v prvem primeru drugi argument beseda in ne simbol. In končno:

2.3. DEFINICIJA. Jezik nekega determinističnega končnega avtomata M, kar zapišemo kot L(M), je množica besed $L(M) = \{w \mid \hat{dlt}(q_0,w) \in F\}$, (z besedami: množica besed, ki nas pripeljejo iz začetnega stanja q_0 v neko končno stanje).

2.4. PRIMER.



sl. 2.3 Deterministični končni avtomat

$Q = \{q_0, q_1, q_2, q_3\}$

SIGMA = {0,1} $F = \{q_0\}$

tabela 2.1 Vrednosti funkcije prehodov dlt avtomata na sl. 2.3

	Simbol	
Stanje	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

In kaj je jezik, ki ga sprejema M s sl. 2.3? Z nekoliko

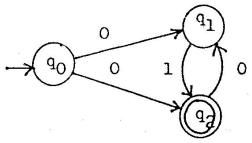
premisleka uganemo, da so to vse besede, ki imajo sodo število ničel in enic.

2.3 Nedeterministični končni avtomat

Sedaj pa bomo začeli osnovni model končnega avtomata spreminjati in raziskovati, kako te spremembe vplivajo na "moč" avtomata. Z drugimi besedami, ali se bo množica jezikov, ki jih je avtomat sposoben sprejeti povečala ali zmanjšala. Prva različica, ki nas zanima je nedeterministični avtomat.

2.5. DEFINICIJA. Nedeterminističen končni avtomat (NKA) M je peterka $\langle Q, SIGMA, dlt, q_0, F\rangle$, kjer je vse kot v definiciji 2.1 razen dlt, ki je v tem primeru neka korespondenca med Q x SIGMA in Q.

2.6. PRIMER.



sl. 2.4 nedeterministični avtomat

$Q = \{q_0, q_1, q_2\}$,

SIGMA = {0, 1},

$q_0 = q_0$,

$F = \{q_2\}$.

tabela 2.2 Vrednosti korespondence prehodov avtomata na sl. 2.4

	Simbol	
Stanje	0	1
q_0	q_1, q_2	-
q_1	-	q_2
q_2	q_1	-

Kaj pa je sedaj jezik avtomata? Zopet naj bo to množica besed, ki nas pripelje iz začetnega stanja v neko končno stanje. Vendar sedaj nas v splošnem neka beseda lahko pripelje v končno stanje po več poteh. Zato, da pridemo do primerne definicije

moramo podobno kot prej razširiti dlt na korespondenco med $Q \times SIGMA^*$ in Q.

2.7. DEFINICIJA. Naj je dlt korespondenca

$Q \times SIGMA \dashrightarrow Q$, kjer sta Q in dlt kot v definiciji 2.5. Potem definiramo $\hat{dlt}$ takole:

1. $\forall q \in Q[\langle q, \epsilon, q\rangle \in \hat{dlt}]$
2. $\forall q \in Q, w \in SIGMA^*, a \in SIGMA[\langle q, wa, r\rangle \in \hat{dlt}$
   $\Longleftrightarrow \exists s \in Q[(\langle q, w, s\rangle \in \hat{dlt})$ and $(\langle s, a, r\rangle \in dlt)]$

V tem primeru $\langle q, w, r\rangle \in \hat{dlt}$ pomeni "eksistira pot od q do r, ki je zaznamovana s simboli w". Sedaj pa lahko zapišemo definicijo, ki ustreza definiciji 2.3:

2.8. DEFINICIJA. Jezik L(M) nekega nedeterminističnga končnega avtomata M je množica besed

$$L(M) = \{w \mid \exists r[(\langle q_0, w, r\rangle \in \hat{dlt})\ \text{and}\ (r \in F)]\}$$

Z besedami (podobno kot prej): jezik nedeterminističnega končnega avtomata je množica besed, s katerimi so označene poti od stanja q_0 do nekega končnega stanja.

Ko preučujemo različne razrede avtomatov, prav pogosto srečujemo situacijo, ko je neki razred "šibkejši", "enako močan" ali "močnejši" od drugega razreda. Kaj to pomeni? "Razred avtomatov" v grobem pomeni neko množico avtomatov. Torej imamo dve množici avtomatov MM_1 in MM_2, $L(MM_i)$, $i = 1,2$, pa naj predstavlja množico jezikov, ki jih sprejemajo elementi MM_i.

Potem pravimo v primeru $L(MM_1) \not\subseteq L(MM_2)$, da je MM_1 <u>šibkejši</u> od MM_2 in nasprotno, da je MM_2 <u>močnejši</u> od MM_1. Če je MM_1 šibkejši in obenem močnejši od MM_2, pravimo, da sta enako močna oziroma <u>ekvivalentna</u>. Enakovreden zapis za trditev, da je MM_1 šibkejši od MM_2 je

$$\forall L \in L(MM_1) \not\exists M \in MM_2 [L = L(M)].$$

Predno nadaljujemo je umestno, da spregovorimo o razmerju pojmov "množica" in "razred". V aksiomatični teoriji množic obstajajo določene omejitve na uporabo pojma "množica", ki zagotavljajo, da ne zaidemo v protislovje. Konkretno velja, da so nekatere "množice" takšne narave, da ne morejo biti elementi drugih množic. Takšnim množicam pravimo razredi[1] V vsakdanji uporabi pa se razred nanaša na množico, ki je definirana zgolj z

---

[1] Podrobna obravnava te problematike presega okvir sedanje knjige, bralec pa lahko najde napotke za nadaljnji študij v delih Jacobson [JNII], str. 6, in MacLane, Birkhoff [MLS], str. 510.

nekakšnimi lastnostmi, brez navedbe univerzuma, čigar elemente preverjamo, ali imajo podano lastnost. Na primer razred deterministični ali nedeterministični končnih avtomatov postane množica takoj, ko izberemo konkretna univerzuma (množici), katerih končne podmnožice so množice stanj ter vhodne abecede konkretnih avtomatov.

Sedaj pa dokažimo naslednji:

2.9. IZREK. Razreda determinističnih končnih avtomatov in nedeterminističnih končnih avtomatov sta ekvivalentna.

<u>Dokaz</u>. Če hočemo dokazati izrek moramo ugotoviti, da (a) če je podan neki determinističen avtomat, potem eksistira tudi nedeterminističen avtomat (z isto vhodno abecedo), ki sprejema isti jezik, in (b), če je podan neki nedeterminističen končni avtomat, potem eksistira tudi neki determinističen končni avtomat, ki sprejema isti jezik.

Vprašajmo se kakšna je razlika med determinističnim in nedeterminističnim končnim avtomatom? Le v tem, da so pri prvem prehodi definirani s funkcijo, pri drugem pa s korespondenco. Na podlagi tega je (a) očitno, saj je vsaka funkcija obenem tudi korespondenca. Glede (b) pa lahko najprej ugotovimo, da, če je podana korespondenca $dlt_0 : Q \times SIGMA \to Q$, potem lahko na naraven način dobimo funkcijo $dlt_1 : Q \times SIGMA \to POT(Q)$, če postavimo $dlt_1(q,a) = \{r \mid \langle q,a,r\rangle \in dlt_0\}$. Iz dlt_1 lahko potem dobimo funkcijo $dlt_2 : POT(Q) \times SIGMA \to POT(Q)$, če postavimo

$$dlt_2(A,a) = \bigcup_{p \in A} dlt_1(p,a)$$

Sedaj pa recimo, da je podan nedeterministični končen avtomat M_0 = $\langle Q,SIGMA,dlt_0,q_0,F\rangle$. Trdimo, da deterministični končen avtomat $M_2 = \langle POT(Q),SIGMA,dlt_2,\{q_0\},\{A \mid A \in POT(A), A \cap F \neq \emptyset\}\rangle$ sprejema isti jezik. Najprej se prepričajmo, da je M_2 deterministični končni avtomat. Ker je Q končno je POT(Q) tudi končno, SIGMA je isto kot pri nedeterminističnem avtomatu, dlt_2 je funkcija POT(Q) x SIGMA --> POT(Q), ki smo jo pridobili iz dlt_0 na pravkar opisan način, $\{q_0\}$ in $\{A \mid A \in POT(A), A \cap F \neq \emptyset\}$ pa tudi ustrezata pogojem definicije. Če pa sedaj pogledamo definicijo $L(M_0)$ (definicija 2.8), kaj pa je $\{r \mid \langle q_0,w,r\rangle \in \hat{dlt}_0\}$? To je natanko $\hat{dlt}_2(\{q_0\},w)$. Potem imamo

$w \in L(M_0) \iff \hat{dlt}_2(\{q_0\},w) \cap F \neq \emptyset$, oziroma

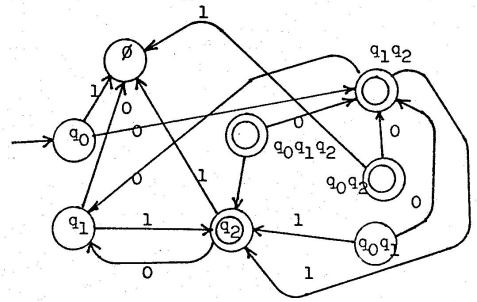
$w \in L(M_0) \iff \hat{dlt}_2(\{q_0\},w) \in \{A \mid A \in POT(A), A \cap F \neq \emptyset\}$, kar pa ustreza definiciji jezika deterministističnega avtomata M_2, ki smo ga pridobili iz nedeterminističnega avtomata M_0. $\square$

2.10. PRIMER. Vzemimo avtomat iz primera 2.6 in poiščimo ustrezen deterministični avtomat.

$POT(Q) = \{\emptyset, q_0, q_1, q_2, q_0q_1, q_0q_2, q_1q_2, q_0q_1q_2\}$

$F = \{q_2, q_0q_2, q_1q_2, q_0q_1q_2\}$

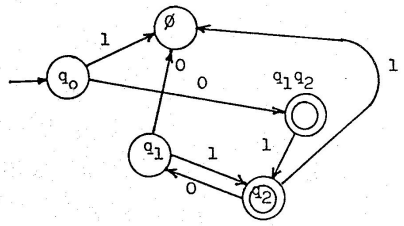
Avtomat je prikazan na sl. 2.5. Opomba: da bi se izognili pretiranemu številu oklepajev, smo množice predstavili kar z zaporedji elementov.



sl. 2.5 Deterministični avtomat, ki ustreza nedeterminističnemu s sl. 2.4

Če ima prvotni nedeterministični avtomat n stanj, potem opisani postopek daje avtomat z 2^n stanji. Vendar lahko takoj odpravimo mnoga od teh stanj, če izločimo nedosegljiva stanja.

(Dosegljiva so stanja iz množice $\bigcup_{w \in SIGMA^*} \hat{dlt}(q_0,w)$). Očitno je, da, če iz množice stanj avtomata izločimo nedosegljiva stanja, dobimo avtomat, ki je ekvivalenten prvotnemu (gl. sl. 2.6)



sl. 2.6 Avtomat s sl. 2.5 potem, ko smo odpravili nedosegljiva stanja

Sedaj pa se postavlja vprašanje, čemu rabi nedeterminističen avtomat? Namreč glede na nedeterminizem ne moremo reči, da ustreza kakšnemu realnemu mehanizmu. Odgovor na vprašanje je, da nam pogosto olajšuje razmišljanje o končnih avtomatih, o čemer se bomo prepričali prav kmalu. Za enkrat pa si oglejmo naslednji primer.

2.11. PRIMER. (Primer uporabnosti pojma nedeterminističnega končnega avtomata). Naj bo SIGMA neka abeceda in $x = x_1 x_2 \ldots x_n \in SIGMA^*$. Vpeljali bomo zapis x^R s pomenom $x^R = x_n x_{n-1} \ldots x_1$, $x_i \in SIGMA$ Če pa je L neki jezik nad SIGMA, potem ima L^R pomen $L^R = \{w \mid w^R \in L\}$. Sedaj pa vprašanje: podan je končni avtomat $M = \langle Q,SIGMA,dlt,q_0,F\rangle$ (deterministični ali nedeterministični), ki sprejema jezik L. Ali je tudi L^R jezik nekega končnega avtomata? Odgovora ni težko poiskati, če uporabimo pojem nedeterminističnega končnega avtomata. Poskušali bomo sestaviti nedeterministični končni avtomat $M' = \langle Q,SIGMA,dlt',q_0',F'\rangle$ za jezik L^R. Najprej definirajmo $dlt' = \{\langle q,a,r\rangle \mid q,r \in Q,\ a \in SIGMA, \langle r,a,q\rangle \in dlt\}$ (torej "obrnemo puščice"). Če obenem postavimo $F' = \{q_0\}$, dobimo skoraj nedeterminističen končni avtomat. Naskladje se pojavi le v tem, da bi morali imeti več začetnih stanj, da bi naš avtomat sprejemal vse besede iz L^R. Namreč za začetna stanja bi morali proglasiti vsa končna stanja. Čeprav bi lahko nedeterministični avtomat tako definirali, vendar to ni nujno, ker se bomo prav kmalu seznanili z načinom, ki nam omogoča, da imamo tudi v tem primeru le eno začetno stanje. V sedanjem

primeru pa lahko (začasno) uberemo nakazano pot (torej, da nekoliko spremenimo definicijo 2.5). To nalogo naj bralec opravi za vajo. Tedaj je množica začetnih stanj torej F.

2.4 Nedeterministični končni avtomat z Є-prehodi

Težavo, ki smo jo ugotovili v primeru 2.11 (nedeterministični avtomat, ki smo ga dobili pri reševanju problema, ali obstaja končen avtomat, ki sprejema L^R, ni bil povsem pravilno definiran) lahko odpravimo, če našo definicijo končenga avtomata še nekoliko posplošimo.

Najprej potrebujemo neki pomožen pojem. Omenili smo že, da je množica besed nad poljubno abecedo SIGMA monoid glede na operacijo stika. Prazna beseda Є igra vlogo enote. Če imamo dva monoida M_1 in M_2, potem je preslikava f: M_1 --> M_2 homomorfizem (monoidov), če velja (a) $f(e_1) = e_2$ in (b) $f(x_1 o x_2) = f(x_1)*f(x_2)$, kjer sta e_1 in e_2 enoti v M_1 in M_2 (po vrsti), o in * pa sta množenji v M_1 in M_2. Če M_1 generira neka končna množica G, z drugimi besedami, če vsak element M_1 lahko predstavimo kot produkt elementov G, potem je homomorfizem določen s podobami elementov G. Sedaj pa naj bosta GAMA in SIGMA = GAMA |_| {a} pri a ∉ GAMA dve končni abecedi. Potem homomorfizmu $SIGMA^*$ --> $GAMA^*$, ki ga določa pravilo x |--> x,

x Є GAMA, in a |--> Є (prazna beseda) pravimo a-izbris

Primer SIGMA = {0,1,2}, GAMA= {0,1}

f je definiran kot 2-izbris

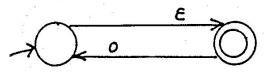
f(0012210) = 00110

2.12. DEFINICIJA (Nedeterministični) končen avtomat z Є-prehodi (Є-NKA) nad abecedo SIGMA je navaden nedeterministični avtomat nad abecedo SIGMA |_| {(Є)}, kjer je (Є) neki simbol, ki ni v SIGMA. Naj bo M tak avtomat in naj L(M) označuje jezik M, če M pojmujemo kot navaden končen avtomat nad abecedo SIGMA |_| {(Є)}. Če M pojmujemo kot končen avtomat z Є-prehodi (nad abecedo SIGMA), potem je jezik M, ki ga označujemo z $L_Є(M)$ naslednja množica

$$L_Є(M) = \{x \mid x Є f(L(M))\}$$

kjer je f (Є)-izbris). (Seveda, zapis f(S), kjer je S neka množica, pomeni {y | y = f(x), x Є S}.)

Primer. SIGMA = {0}



sl. 2.7 Primer avtomata z Є-prehodi

Bralec naj se prepriča, da je jezik avtomata na sl. 2.7 enak $SIGMA^*$.

Sedaj bomo dokazali

2.13. IZREK. Razreda (nedeterminističnih) končnih avtomatov z in brez ϵ-prehodov sta ekvivalentna.

Dokaz Moramo pokazati, da, če je podan neki končen avtomat brez ϵ-prehodov, potem eksistira avtomat z ϵ-prehodi, ki sprejema isti jezik in, nasprotno, če je podan avtomat z ϵ-prehodi, potem eksistira avtomat brez ϵ-prehodov, ki sprejema isti jezik.

Prvi del je seveda očiten, saj je avtomat brez ϵ-prehodov le poseben primer avtomata z ϵ-prehodi. Drugi del pa dokažemo takole. Naj bo $M_1 = \langle Q_1, SIGMA \mid\_\mid \{(\epsilon)\}, dlt_1, q_{01}, F_1 \rangle$. Naloga je definirati tak avtomat $M_2 = \langle Q_2, SIGMA, dlt_2, q_{02}, F_2 \rangle$, ki sprejema natanko besede oblike $x = x_1 x_2 \ldots x_n$, ko velja

$$e_0 x_1 e_1 x_2 e_2 \ldots x_n e_n \in L(M_1)$$

pri poljubnih $e_i \in \{(\epsilon)\}^*$, $0 \leq i \leq n$, $x_i \in SIGMA^*$, $1 \leq i \leq n$. Ideja je v tem da iz M_1 odstranimo ϵ-prehode, dodamo pa določene druge prehode, ki bodo oponašali delovanje M_1 na besedah e_i. Zato najprej poiščemo $\hat{dlt}_1$ (gl. definicija 2.7) in definiramo $\epsilon\text{-CL}(A) = \hat{dlt}_1(A, (\epsilon)^*)$. Torej ϵ-CL(A) je množica vseh stanj, ki so dosegljiva iz množice stanj A z besedami, ki so sestavljene iz samih simbolov (ϵ). (ϵ-CL izgovarjamo kot "epsilon ovojnica", znak pa izvira iz angleške

besede ϵ-closure.) Sedaj pa postavimo: $Q_2 = Q_1$; $dlt_2 = dlt_{20} \mid\_\mid dlt_{21}$, pri čemer

$$dlt_{20}(q,a) = \epsilon\text{-CL}(dlt_1(q,a)),\ q \in Q \text{ in } a \in SIGMA,$$

ter

$$dlt_{21}(q,a) = \epsilon\text{-CL}(dlt_1(\epsilon\text{-CL}(q),a))$$

pri $q = q_{01}$ in $\emptyset$ sicer. Nadalje postavimo $q_{02} = q_{01}$ in $F_2 = F_1$ v primeru, ko velja $\epsilon\text{-CL}(q_0) \bar{\mid} F_1 = \emptyset$ in $F_2 = F_1 \mid\_\mid \{q_{01}\}$ sicer.

Sedaj se pa prepričajmo, da velja $L(M_1) = L(M_2)$. Pri dolžini besede $n = 0$ trditev drži, saj smo F_2 definirali tako, da vsebuje q_{01} v primeru, ko je neko stanje iz F_1 dosegljivo iz q_{01} z ϵ-prehodi. Pri $n \geq 1$ pa trditev sledi iz resničnosti stavka

$$(n \geq 1) \text{ and } (|x| = n) \Longrightarrow \forall q \in Q[\hat{dlt}_2(q,x) = \mid\_\mid_{y \in f^{-1}(x)} \hat{dlt}_1(q,y)] \tag{2.1}$$

kjer je f ϵ-izbris[2]. Stavek (2.1) ni težko dokazati z

indukcijo. Pri n = 1 je resničen na podlagi definicije dlt .
21

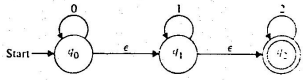
Pri n > 1 pa najprej ugotovimo, da velja

$$\forall x \in SIGMA^{n-1}, a \in SIGMA[w \in f^{-1}(xa) \Rightarrow$$
$$\exists y \in f^{-1}(x), e \in (\epsilon)^*$$
$$[w = yae],$$

nato pa iz resničnosti stavka (2.1) za x (induktivna predpostavka) in definicije dlt sledi resničnost stavka za xa.
20

□

2.14. PRIMER



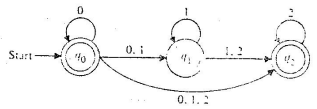
sl. 2.8 Končni avtomat z ϵ-prehodi

---

2 Spomnimo se, da, če je podana neka preslikava oziroma funkcija med množicama A in B, potem lahko definiramo inverzno preslikavo f^{-1} s pravilom $f^{-1}(y) = \{x \mid f(x)=y\}$. Seveda pa f^{-1} ni nujno, da je preslikava med B in A.

tabela 2.3 Množice ϵ-CL(A) za avtomat na sl. 2.8.

A	ϵ-CL(A)
∅	∅
q_0	q_0, q_1, q_2
q_1	q_1, q_2
q_2	q_2
q_0, q_1	q_0, q_1, q_2
q_0, q_2	q_0, q_1, q_2
q_1, q_2	q_1, q_2
q_0, q_1, q_2	q_0, q_1, q_2



sl. 2.9 Avtomat brez ϵ-prehodov, ki ustreza avtomatu na sl. 2.8.

Sedaj pa bomo podali nekaj primerov uporabnosti končnega avtomata z ϵ-prehodi.

2.15. PRIMER. Vzamemo primer 2.11. Sedaj je možno definirati avtomat za L^R, ki popolnoma ustreza sprejeti definiciji nedeterminističnega končnega avtomata. Namreč obstoječemu avtomatu za L dodamo eno novo stanje q_{00} nato "obrnemo puščice" ter končno dodamo ϵ-prehode iz q_{00} do končnih stanj avtomata za L. V tako pridobljenem avtomatu je začetno stanje q_{00}, množica končnih stanj pa $\{q_0\}$ kjer je q_0 začetno stanje avtomata za L.

Zadnji primer se naslanja na naslednjo trditev:

2.16. STAVEK. Končni avtomati z več začetnimi stanji so ekvivalentni avtomatom z enim začetnim stanjem.

Dokaz. Ekvivalenten avtomat z enim začetnim stanjem dobimo z uporabo ϵ-prehodov. |_|

Prav tako lahko ugotovimo

2.17. STAVEK. Končni avtomati z enim končnim stanjem so ekvivalentni avtomatom z več končnimi stanji.

Dokaz. Ekvivalenten avtomat z enim končnim stanjem dobimo tako, da dodamo novo končno stanje in uporabimo ϵ-prehode. |_|

2.5 Regularni izrazi

Doslej smo za opisovanje nekega jezika uporabljali teoretičen model, ki sloni na predstavi stroja (avtomata), ki sproti preverja, ali je neka beseda v jeziku. Vendar so jezike že od nekdaj opisovali tudi z gramatikami ali sintaksami. Pri tem gre za pravila, ki nakazujejo, kako so zgrajene previlne jezikovne oblike. V tem poglavju podajamo opis nekega razreda sintaks, ki opisujejo (kot bomo videli) jezike, ki jih sprejemajo končni avtomati.

2.18. DEFINICIJA. SIGMA je neka končna abeceda, ki ne vsebuje simbolov $\underline{\emptyset}, \underline{\epsilon}, \underline{o}, \underline{+}, \underline{*}, \underline{(}, \underline{)}$. SIGMA naj bo prav tako končna abeceda, pri čemer velja SIGMA = $\{\underline{a} \mid a \in \text{SIGMA}\}$. Regularni izrazi nad SIGMA je jezik, ki ga sestavljajo besede nad abecedo SIGMA |_| $\{\underline{\emptyset}, \underline{\epsilon}, \underline{o}, \underline{+}, \underline{*}, \underline{(}, \underline{)}\}$, in ki so zgrajene na podlagi naslednje induktivne definicije:

1. $\underline{a}$, $a \in$ SIGMA, $\underline{\emptyset}$ in $\underline{\epsilon}$ so regularni izrazi
2. Naj sta E in F regularna izraza. Potem so (E+F), (EoF) in $(E)^*$ prav tako regularni izrazi.
3. Ni drugačnih regularnih izrazov kot takih, ki zadoščajo pogojema 1 in 2.

Primer: SIGMA = {0,1}

$E_1 = (((\underline{0})^* o \underline{1})o(\underline{0})^*)$, $E_2 = ((\underline{0} + \underline{1})^* o \underline{1})$.

Vsakemu regularnemu izrazu nad SIGMA pripisujemo neki pomen -- predstavlja namreč določen jezik nad SIGMA. Da bi natanko opisali ta pomen, definirajmo najprej neke operacije nad jeziki

oziroma množicami besed. Naj sta L_1 in L_2 jezika nad abecedo SIGMA. Potem je

$$L_1 oL_2 = \{x \mid x = x_1 ox_2 , x_1 \in L_1 , x_2 \in L_2 \}$$

(z besedami: stik L_1 in L_2).

$$L_1^i = L_1 oL_1 o...oL_1 \quad (i \text{ faktorjev})$$

in

$$L_1^* = \bigcup_{i=0}^{\infty} L_1^i$$

(z besedami: iteracija L_1). Oznaka L_1^0 ima pomen $\{\epsilon\}$,

Sedaj pa lahko pomen nekega regularnega izraza zopet definiramo induktivno:

2.19. DEFINICIJA. Regularen izraz E predstavlja jezik L(E), ki je določen na naslednji način:

1. V primeru E = a, a $\in$ SIGMA, velja L(E) = {a}, v primeru E = $\emptyset$ velja L(E) = $\emptyset$ in v primeru E = ϵ velja L(E) = $\{\epsilon\}$.
2. Naj sta E_1 in E_2 regularna izraza ki jima že znamo pripisati jezika $L(E_1)$ in $L(E_2)$. Potem velja

$$L((E_1 + E_2)) = L(E_1) \cup L(E_2),$$
$$L((E_1 oE_2)) = L(E_1)oL(E_2)$$

in

$$L((E_1)^*) = L(E_1)^* .$$

Potemtakem nam prvi izraz v zgornjem primeru predstavlja jezik, ki ga sestavljajo besede iz poljubnega števila ničel, ki jim sledi enka, in zopet poljubno število ničel. Drugi izraz predstavlja jezik {01,11}.

Da bi dosegli večjo zgoščenost zapisa, po navadi vpeljemo naslednje dogovore: namesto posebnih simbolov a, a $\in$ SIGMA, in ϵ ter $\emptyset$, uporabljamo kar a, ϵ in $\emptyset$ in si pri tem mislimo, da gre za poseben simbol. Poleg tega večino oklepajev lahko odpravimo z uporabo naslednjega dogovora o prednosti operatorjev: * ima prednost pred o in slednji pred +. Na primer $(a^* + b)^* o c$ pomeni $((a)^* + (b)^*) o c$, $a o b^*$ pa pomeni $(a) o ((b)^*)$.

Osnovni namen tega poglavja je dokazati ekvivalentnost regularnih izrazov in končnih avtomatov. Torej vsak regularen izraz predstavlja jezik, ki ga sprejema neki končen avtomat, in nasprotno, vsak končen avtomat sprejema jezik, ki ga je možno predstaviti z nekim regularnim izrazom. Najprej bomo dokazali prvi del in nato še drugi del zapisane trditve.

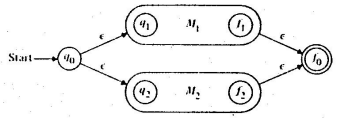
2.20. IZREK. Naj bo E regularen izraz nad SIGMA. Potem obstaja končen avtomat M $\in$ ϵ-NKA, ki sprejema L(E).

Dokaz. Dokaz izpeljemo z indukcijo po številu operatorjev, ki jih vsebuje izraz E, in ima zgradbo, ki zrcali definicijo 2.18:

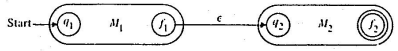
(1) Naj bo E regularen izraz nad SIGMA oblike a, a ∈ SIGMA, ali Ø oziroma ϵ. Potem jezike L(E) sprejemajo avtomati, ki so prikazani na sl. 2.10.

sl. 2.10 Končni avtomati za regularne izraze brez operatorjev

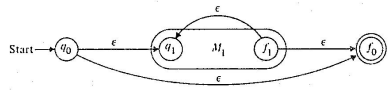
(2) Naj bosta E in F regularna izraza nad SIGMA in naj bosta L(E) in L(F) ustrezna jezika. Nadalje naj bosta M_E = $\langle Q_E, SIGMA, dlt_E, q_{0E}, \{1E\}\rangle$ in $M_F = \langle Q_F, SIGMA, dlt_F, q_{0F}, \{1F\}\rangle$ končna avtomata ki sprejemata L(E) in L(F), po vrsti, pri čemer sta Q_E in Q_F medsebojno tuji množici. Na podlagi stavka 2.17 lahko predpostavljamo da imata M_E in M_F le eno končno stanje. Potem je avtomat, ki sprejema L(E + F), prikazan na sl. 2.11, avtomat, ki sprejema L(E o F), je prikazan na sl. 2.12, in končno je avtomat, ki sprejema $L(E^*)$, prikazan na sl. 2.13. Da opisani avtomati resnično sprejemajo ustrezne jezike sledi iz definicije operatorjev +, o in *. Podrobno izpeljavo lahko bralec naredi sam.



sl. 2.11 Končni avtomat za L(E + F)

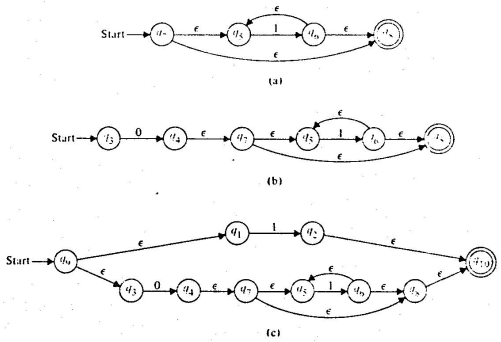


sl. 2.12 Avtomat za L(E o F)



sl. 2.13 Avtomat za $L(E^*)$

2.21. PRIMER. Sl. 2.14 prikazuje avtomat za jezik L(E), ki ga predstavlja izraz $E = (0(1^*)) + 1$.



sl. 2.14 Sestavljanje NKA iz regularnega izraza: (a) za $r_4 = 1^*$; (b) za $r_1 =)1^*$; ter (c) za $r = 01^* +1$

Sedaj pa dokažimo

2.22. IZREK. Naj bo M = <Q,SIGMA,dlt,q_1,F> ∈ DKA končni avtomat. Potem obstaja regularen izraz E nad SIGMA z lastnostjo L(E) = L(M).

Opomba: Glede na ekvivalenco vseh različic končnih avtomatov nismo prav nič izgubili na splošnosti, ko smo omejili M na razred DKA.

Dokaz: Dokaz je zgrajen tako, da postopno sestavljamo čedalje boljše približke jezika L(M). Za vsak tak približek dokažemo, da ga lahko predstavimo z regularnim izrazom. Množico

stanj Q oštevilčimo tako, da imamo Q = {q_1, q_2, ... , q_n}, pri čemer je q_1 začetno stanje. Nato definiramo

$$R^k_{ij} = \{x \mid (dlt(q_i,x) = q_j) \text{ and } [(x = yz) \text{ and } (y \neq \epsilon) \text{ and } (y \neq x) \text{ and } (dlt(q_i,y) = q_1) \Longrightarrow 1 \leq k]\}$$

pri čemer velja $1 \leq i, j \leq n$ in $0 \leq k \leq n$. Tolmačenje R^k_{ij} je, da ga sestavljajo vse besede, s katerimi so zaznamovane poti od q_i do q_j, ki imajo še lastnost, da peljejo le skozi stanja q_l, $l \leq k$. Pomembno je, da se omejitev q_l, $l \leq k$, ne nanaša na začetno stanje q_i in na končno stanje q_j. Sedaj pa definirajmo

$$L^k = \bigcup_{q_i \in F} R^k_{1i} \qquad (2.2)$$

Potem zgoraj zapisana trditev o postopni gradnji čedalje boljših približkov jezika L(M) sledi iz

$$L^0 \subseteq L^1 \subseteq \ldots \subseteq L^n = L(M), \qquad (2.3)$$

ki pa sledi iz $R^k_{ij} \subseteq R^{k+1}_{ij}$ (bralec naj se na lastno pest

prepriča v resničnost $R_{ij}^{k} \subseteq R_{ij}^{k+1}$, iz česar prej zapisano sledi neposredno).

Množice R_{ij}^{k} lahko definiramo induktivno: pri k = 0 velja

$$R_{ij}^{0} = \begin{cases} \{a \mid dlt(q_i,a) = q_j\} & \text{pri } i \neq j \\ \{a \mid dlt(q_i,a) = q_j\} \cup \{\epsilon\} & \text{sicer} \end{cases}$$

za ostale vrednosti k pa imamo

$$R_{ij}^{k} = R_{ij}^{k-1} \cup R_{ik}^{k-1} o (R_{kk}^{k-1})^{*} o R_{kj}^{k-1} \qquad (2.4)$$

z besedami: R_{ij}^{0} je sestavljen iz vseh besed, ki nas neposredno pripeljejo iz q_i do q_j, brez vmesnega pristanka v kakšnem drugem stanju. Pri k>0 pa premišljujemo takole: naj beseda x pripada R_{ij}^{k}; potem bodisi pripada obenem množici R_{ij}^{k-1}, ali pa jo lahko razdelimo na naslednji način:

$$x = x_1 x_2 \ldots x_m,$$

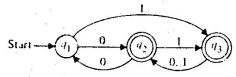
kjer je m največje možno število z lastnostjo

$dlt(q_i, x_1 x_2 \ldots x_h) = q_k$ pri vseh $1 \leq h < m$. Za vsako drugačno

razdelitev $x = yz$, $\forall 1 \leq h < m [y \neq x_1 x_2 \ldots x_h]$ pa imamo $dlt(q_i, y) \neq q_k$ (torej smo poiskali prav tiste simbole v x, ki nas zapeljejo v q_k). Potem očitno $x_1 \in R_{ik}^{k-1}$, $x_i \in R_{kk}^{k-1}$, $1 < i < m$, in $x_m \in R_{kj}^{k-1}$. Torej smo (2.4) preverili.

Sedaj pa opazimo, da je možno R_{ij}^{k} pri vseh i,j ter $0 \leq k \leq n$ predstaviti z regularnim izrazom, saj je to za R_{ij}^{0} neposredno razvidno, pri ostalih vrednostih k pa ugotovimo, da smo pri definiciji R_{ij}^{k} z R_{--}^{k-1} uporabljali le operatorje, ki so dovoljeni pri sestavljanju regularnih izrazov ($\cup$ oziroma +, o in *). Potem pa iz (2.2) in (2.3) sledi, da lahko tudi L(M) izrazimo z regularnim izrazom. S tem je dokaz končan. $\square$

2.23. PRIMER. Naj bo podan naslednji avtomat



sl. 2.15 Primer končnega avtomata

Regularni izrazi r_{ij}^k za R_{ij}^k so prikazani v tabeli 2.4.

tabela 2.4 Regularni izrazi r_{ij}^k v primeru 2.23

	$k=0$	$k=1$	$k=2$
r_{11}^k	ϵ	ϵ	$(00)^*$
r_{12}^k	0	0	$0(00)^*$
r_{13}^k	1	1	0^*1
r_{21}^k	0	0	$0(00)^*$
r_{22}^k	ϵ	$\epsilon + 00$	$(00)^*$
r_{23}^k	1	$1 + 01$	0^*1
r_{31}^k	$\varnothing$	$\varnothing$	$(0 + 1)(00)^*0$
r_{32}^k	$0 + 1$	$0 + 1$	$(0 + 1)(00)^*$
r_{33}^k	ϵ	ϵ	$\epsilon + (0 + 1)0^*1$

Pri sestavljanju tebele 2.4 smo uporabili določene ekvivalence med regularnimi izrazi kot so n.pr. $(r + s)t = rt + st$ in $(\epsilon + r)^* = r^*$. Na primer, po definiciji dobimo

$$r_{22}^1 = r_{22}^0 + r_{21}^0 (r_{11}^0)^* r_{12}^0 = \epsilon + 0(\epsilon)^* 0,$$

drugi člen na desni pa je enak 00. Na podoben način imamo

$$r_{13}^2 = r_{13}^1 + r_{12}^1 (r_{22}^1)^* r_{23}^1 = 1 + 0(\epsilon + 00)^* (1 + 01).$$

Če upoštevamo $(\epsilon + 00)^* = (00)^*$ in $1 + 01 = (\epsilon + 0)1$ dobimo

$$r_{13}^2 = 0(00)^* (\epsilon + 0)1 + 1$$

Prav tako ugotovimo, da je $(00)^* (\epsilon + 0)^* = 0^*$. Torej je izraz $0(00)^* (\epsilon + 0)1 + 1$ ekvivalenten izrazu $00^* 1 + 1$ in naprej izrazu $0^* 1$.

Sestavljanje regularnega izraza $r_{12}^3 + r_{13}^3$ za L(M) nadaljujemo takole:

$$r_{12}^3 = r_{12}^2 + r_{13}^2 (r_{33}^2)^* r_{32}^2$$

$$= 0^* 1(\epsilon + (0 + 1)0^* 1)^* (0 + 1)(00)^* + 0(00)^*$$

$$= 0^* 1((0 + 1)0^* 1)^* (0 + 1)(00)^* + 0(00)^*$$

in

$$r_{13}^3 = r_{13}^2 + r_{13}^2 (r_{33}^2)^* r_{33}^2$$

$$= 0\ 1(\epsilon + (0 + 1)0\ 1)^*\ (\epsilon + (0 + 1)0\ 1)^* + 0\ 1$$

$$= 0\ 1((0 + 1)0\ 1)^*$$

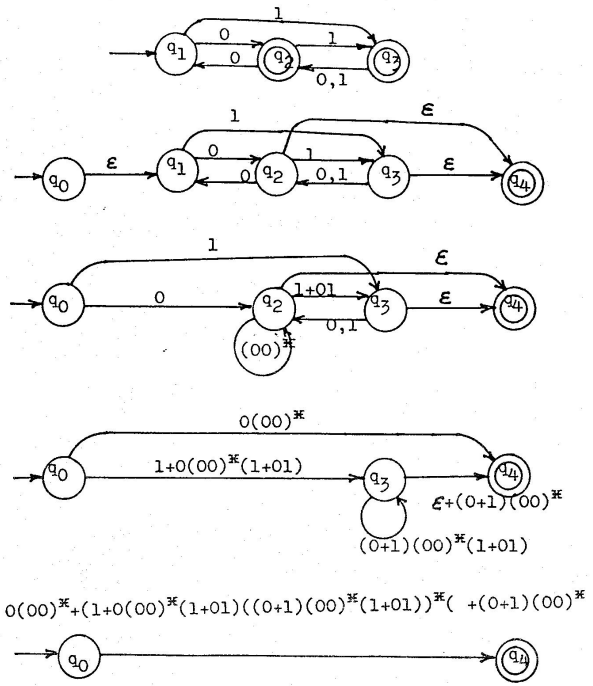
In končno

$$r_{12}^3 + r_{13}^3 = 0\ 1((0 + 1)0\ 1)^*\ (\epsilon + (0 + 1)(00)^*)+0(00)^*$$

Zaradi veljavnosti izrekov 2.20 in 2.22 bomo jezikom, ki jih sprejemajo končni avtomati, rekli kar regularni jeziki.

Na koncu naj opišemo neki učinkovit grafičen postopek za pridobivanje regularnega izraza, ki ustreza nekemu podanemu končnemu avtomatu. Podan je avtomat M = $\langle\{q_1,q_2,\ldots,q_n\},SIGMA,dlt,\{q_1\},F\}$, ki si ga predstavimo v obliki diagrama (gl. na primer sl. 2.15). V prvem koraku postopka spremenimo M v avtomat M', ki ima eno samo začetno in eno samo končno stanje, tako, da vpeljemo novi stanji q_s in q_f ter ju povežemo z M z ϵ prehodi, kot je to že bilo pojasnjeno (gl. stavka 2.16 in 2.17). Sedaj so povezave v diagramu, ki predstavlja M', zaznamovane z regularnimi izrazi r_{ij}^0, $i,j \in \{1,2,\ldots,n,s,f\}$. Nato pa začnemo postopno odpravljati stanja q_1-q_n. Ko odpravimo q_k so oznake povezav r_{ij}^k, $i,j \in \{k+1,k+2,,\ldots,n,s,f\}$. Na koncu, ko odpravimo q_n, ostane le ena povezava med q_s in q_f, ki predstavlja jezik L(M). Na sl.

2.16 je prikazan postopek za primer s sl. 2.15.



sl. 2.16 Grafična metoda za pridobivanje regularnega izraza za podani avtomat

2.6 Dvosmerni končni avtomat

Doslej smo opisali poglavitne različice končnih avtomatov, ki v literaturi najpogosteje nastopajo. Vendar včasih srečujemo še nadaljnje različice in kot zanimivost bomo opisali dve izmed njih. Najprej opišemo dvosmerni končni avtomat, ki si ga predstavljamo kot strojček, ki se premika levo in desno po nekakšnem traku, na katerem so zapisani simboli. Na podlagi prebranega simbola prehaja v novo stanje in (morebiti) menja smer gibanja. Zapomnimo pa si, da v tem primeru avtomat ne more spreminjati simbolov na traku. Na prvi pogled bi kazalo, da je taka naprava močnejša od končnega avtomata, saj ji dvosmerno gibanje omogoča večjo izbiro pri delu. Vendar izkaže se, da to ni res. Razred jezikov, ki jih je tak avtomat sposoben prepoznati, je isti kot razred, ki jih sprejemajo že opisani končni avtomati. Vendar zato, da bomo to vprašanja lahko natančno obravnavali je potrebno najprej pojem enolično definirati.

2.24. DEFINICIJA. Dvosmerni (deterministični) končni avtomat (2-DKA) je peterka $M=\langle Q,SIGMA,dlt,q_0,F\rangle$ pri čemer so $Q,SIGMA,q_0$ in F definirani podobno kot v definiciji 2.1, dlt pa je preslikava Q x SIGMA --> Q x {L,D}

Pomen komponente iz množice {L,D} v vrednosti preslikave dlt je, da rabi kot navodilo za smer gibanja bralne glave: L označuje levi premik, D pa desni.

Sedaj ne moremo več definirati jezika avtomata preprosto z razširitvijo funkcije dlt, saj lahko naprava obišče en in isti simbol nekajkrat. Zato bomo s tem namenom uporabili neki mehanizem, ki nam bo tudi kasneje pogosto koristil.

Naj bo podan neki dvosmerni deterministično končni avtomat M. Potem je množica trenutnih opisov avtomata M, ki jo označujemo s TO_M, kartezični produkt $SIGMA^* \times Q \times SIGMA^*$. Če je M = <u,q,v> trenutni opis, je njegov pomen naslednji: u predstavlja del vhoda, ki leži levo od bralne glave, q predstavlja trenutno stanje nadzorne enote, v pa del vhoda pod bralno glavo in desno od nje (prvi simbol v, če obstaja, je tisti simbol, ki ga avtomat trenutno bere). V prieru v = ϵ predstavlja <u,q,v> trenutek, ko bralna glava prekorači desni rob vhodne besede. Sedaj pa na množici trenutnih opisov definiramo relacijo $\vdash_M$ ("neposredno povzroči"), katere pomen je, da v primeru resničnosti $u \vdash_M v$, avtomat preide v enem karaku (po enem prehodu stanja) iz trenutnega opisa u v opis v. Na podlagi relacije $\vdash_M$ definiramo njeno "tranzitivno ovojnico" $\vdash_M^*$ ("povzroči") takole:

$$u \overset{*}{\underset{M}{\vdash}} v \iff \exists (k \in NN) \exists (u_0, u_1, \ldots, u_k \in TO$$

$$[(u=u_0) \text{ and } (v=u_k) \text{ and } (u_0 \overset{}{\underset{M}{\vdash}} u_1) \text{ and } (u_1 \overset{}{\underset{M}{\vdash}} u_2) \text{ and } \ldots$$

$$\text{and } (u_{k-1} \overset{}{\underset{M}{\vdash}} u_k)]$$

(Na podlagi te definicije vedno velja $u \overset{*}{\underset{M}{\vdash}} u$.) Če vemo za kateri stroj gre, pišemo preprosto $\vdash$ oziroma $\overset{*}{\vdash}$. Pomen $\overset{*}{\vdash}$ je, da je v primeru resničnosti $u \overset{*}{\vdash} v$ možno preiti iz trenutnega opisa u v trenutni opis v v končnem številu korakov.

Sedaj pa opišimo $\vdash$:

(1) $a_1 a_2 \ldots a_{i-1} q a_i \ldots a_n \vdash a_1 a_2 \ldots a_{i-1} a_i p a_{i+1} \ldots a_n$

v primeru $dlt(q,a_i) = (p,R)$ in

(2) $a_1 a_2 \ldots a_{i-2} a_{i-1} q a_i \ldots a_n \vdash a_1 a_2 \ldots a_{i-2} p a_{i-1} a_i \ldots a_n$

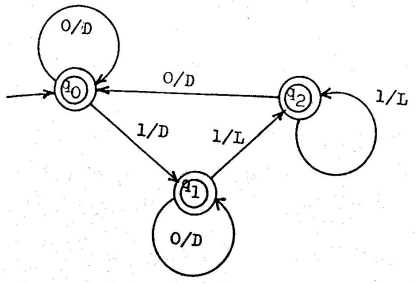
v primeru $dlt(q,a_i) = (p,L)$ in $i > 1$. Pogoj $i > 1$ v (2) pomeni, da trenutni opis delovanja stroja ni definiran, ko glava prekorači levi rob levi rob besede na traku. Sedaj pa:

2.25. DEFINICIJA. Naj bo M dvosmerni končni avtomat. Jezik M definiramo kot

$$L(M) = \{x \mid (\epsilon, q_0, x) \overset{*}{\vdash} (x, p, \epsilon),\ p \in F\}$$

Torej jezik M tvorijo besede, na katerih avtomat zapusti desni rob v končnem stanju. Mimogrede povedano, dozdevne omejitve pri delovanju stroja na robovih (ne more se popeljati preko robov) niso bistvene, saj v abecedo lahko vključimo dva posebna simbola, ki označujeta levi in desni rob -- z njimi pa dosežemo, da se stroj lahko "zave", da se približuje robu.

2.26. PRIMER. Opisali bomo 2-DKA M, ki deluje takole: začenši v stanju q_0 M ponavlja cikel korakov, ko se glava premika v desno dokler ne pride do dveh zaporednih enk, nato v levo do prve ničle, nakar M ponovno pride v stanje q_0 in se cikel ponovi. M ima tri stanja, ki so vsa končna.



sl. 2.17 **Dvosmerni končni avtomat. Puščica iz q_i v q_j z oznako (a,d) ima pomen $dlt(q_i,a) = (q_j,d)$**

Opisani avtomat deluje na vhodu 101001 takole:

$$q_0 101001 \vdash 1q_1 01001 \vdash 10q_1 1001 \vdash 1q_2 01001 \vdash 10q_0 1001 \vdash$$

$$101q_1 001 \vdash 1010q_1 01 \vdash 10100q_1 1 \vdash 1010q_2 01 \vdash 10100q_0 1 \vdash$$

$$101001q_1$$

(2.5)

Osnovni rezultat, ki ga želimo dokazati je, da je dvosmerni končni avtomat ekvivalenten enosmernemu. S tem namenom vpeljemo pojem mejnega zaporedja. Naj bo M 2-DKA in $x = a_1 a_2 \ldots a_n$ neki vhod. Zaporedju stanj $(q_1, q_2, \ldots, q_r)$ (ki je morebiti prazno) bomo rekli i-to <u>mejno zaporedje izračuna</u> M <u>na</u> x, $1 \leq i \leq n$, če velja (1) q_{2j-1} je stanje, v katerem avtomat j-tič prestopi mejo med i-tim in i+1-im simbolom v smeri od leve proti desni in (2) q_{2j} je stanje v katerem avtomat j-tič prestopi isto mejo od desne proti levi. (1) pomeni, da v zaporedju trenutnih opisov $\langle \epsilon, q_0, x\rangle \vdash \ldots \vdash \langle u,q,v\rangle \vdash \ldots$ velja $q_{2j-1} = r$ pri j-tem opisu po vrsti z lastnostjo

$$[(u,q,v) = (a_1 a_2 \ldots a_{i-1}\, q\, a_i \ldots a_n)] \text{ <u>and</u> } [dlt(q,a_i) = (r,D)].$$

(2) pa seveda pomeni, da velja $r = q_{2j}$ za j-ti opis z lastnostjo

$$[(u,q,v) = (a_1 a_2 \ldots a_i\, q\, a_{i+1} \ldots a_n)] \text{ <u>and</u> } [dlt(q,a_{i+1}) = (r,L)]$$

Kot ničelno mejno zaporedje definiramo zaporedje (q_0), kjer je q_0 začetno stanje avtomata. Pogoja (1) in (2) nam omogočata, da poiščemo pri nekem podanem vhodu x dolžine n, i-to mejno zaporedje za vse $0 \leq i \leq n$.

2.27. PRIMER. i-to mejno zaporedje, pri i = 1, avtomata

iz primera 2.26 pri izračunu (2.5) je (q_1, q_2, q_0).

Sedaj bomo opisali potreben pogoj, ki ga morata izpolnjevati dve zaporedji stanj $\underline{q}$ in $\underline{r}$ zato, da sta i-1 in i-to mejno zaporedje M na vhodu $x \in L(M)$ z i-tim znakom a. Pogoj bo odvisen le od a. Ko je ta pogoj izpolnjen bomo rekli, da sta $\underline{q}$ in $\underline{r}$ desno združljivi zaporedji pri vhodu a. Simbolično bomo to zapisali kot $\underline{q} \xrightarrow{a} \underline{r}$. Kot pomoč pri definiciji desne združljivosti bomo definirali še relacijo leve združljivosti pri vhodu a, oziroma simbolično $\underline{q} \xleftarrow{a} \underline{r}$.

Obe relaciji sta definirani s konjunkcijo dveh stavkov. Pri desni združljivosti je prvi stavek "$\underline{q}$ in $\underline{r}$ sta lihe dolžine", pri levi pa "$\underline{q}$ in $\underline{r}$ sta sode dolžine". Ta dva stavka sta pogoja, da je čitalna glava na koncu desno oziroma levo od simbola a.

Drugi stavek pa induktivno opisuje zgradbo pravilnih parov zaporedij:

(1) $\forall a \in SIGMA[\epsilon \xleftarrow{a} \epsilon]$

(2a) Naj velja $\underline{q} \xleftarrow{a} \underline{r}$. Naj za $q \in Q$ velja

$dlt(q,a) = (r,L)$. Potem velja $\underline{q} \circ q \circ r \xleftarrow{a} \underline{r}$, pri čemer $\circ$ označuje operacijo podaljševanja zaporedja.

(2b) Naj za $q \in Q$ velja $dlt(q,a) = (r,D)$. potem velja

$\underline{q} \circ q \xrightarrow{a} \underline{r} \circ r$.

(3a) Naj velja $\underline{q} \xrightarrow{a} \underline{r}$. Naj za $q \in Q$ velja $dlt(q,a) = (r,L)$.

Potem velja $\underline{q} \circ r \xleftarrow{a} \underline{r} \circ q$.

(3b) Če pa imamo

$dlt(q,a) = (r,D)$ velja $\underline{q} \xrightarrow{a} \underline{r} \circ q \circ r$.

(4a) Edini pari $\underline{q}$, $\underline{r}$, za katere velja $\underline{q} \xrightarrow{a} \underline{r}$ so pari, ki izpolnjujejo pogoja 2b in 3b.

(4b) Edini pari $\underline{q}$, $\underline{r}$, za katere velja $\underline{q} \xleftarrow{a} \underline{r}$ so pari, ki izpolnjujejo pogoje 1, 2a in 3a.

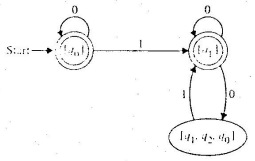
Sedaj najprej ugotovimo, da pri podanem vhodu $x = a_1 a_2 \ldots a_n$ avtomat M sprejme besedo x natanko takrat, ko eksistira zaporedje $\underline{q}_0 \xrightarrow{a(1)} \underline{q}_1 \xrightarrow{a(2)} \ldots \xrightarrow{a(n)} \underline{q}_n$, pri čemer velja $\underline{q}_0 = (q_0)$ in $\underline{q}_n = (q)$ pri $q \in F$. Nadalje ugotovimo, da, če takšno zaporedje obstaja, potem je dolžina vseh $\underline{q}_i$ omejena. Namreč, če je dolžina $\underline{q}_i$ večja od $2N$, kjer je N število stanj avtomata, potem se v zaporedju $\underline{q}_i$ neko stanje, recimo r, ponovi na mestih, ki imajo isto parnost. To pa pomeni, da se avtomat zapelje skozi mejo v istem stanju in v isti smeri. Iz tega pa sledi da ne more zapustiti desnega roba besede. Torej se lahko omejimo

na zaporedja stanj lihe dolžine, katerih dolžina ne presega 2N. Sedaj pa lahko definiramo nedeterminističen končni avtomat, ki sprejema isti jezik kot 2DKA M in katerega množica stanj je množica zaporedij stanj $(q_1, q_2, \ldots, q_r)$, $q_i \in Q$ in $r < 2N$ in r je liho, $dlt(\underline{q},a) = \{\underline{r} \mid \underline{q} \xrightarrow{a} \underline{r}\}$. Začetno stanje je (q_0), končna pa (q) pri $q \in F$. Očitno je, da opisani avtomat sprejme $x = a_1 a_2 \ldots a_n$ natanko takrat, ko ga sprejme 2-DKA M.

Torej smo dokazali naslednji

2.28. IZREK. Razreda enosmernih nedeterminističnih in dvosmernih determinističnih končnih avtomatov sta ekvivalentna.

2.26. PRIMER (nadaljevanje). V splošnem bi morali raziskati vsa zaporedja stanj, katerih dolžina ne presega $2*3 = 10$, vendar, ker le štiri zaporedja nastopajo kot mejna zaporedja dejanskih izračunov, nam je naloga olajšana.



sl. 2.18 Navaden nedeterminističen avtomat, ki je ekvivalenten avtomatu na sl. 2.17.

Čeprav se nam zdi opisana konstrukcija preprosta, je pa potrebno opozoriti na to, da ustrezne konstrukcije za končne avtomate, ki bi se sprehajali po več dimenzionalnih prostorih niso znane.

2.7 Končni avtomati z izhodi

V dosedanjih primerih so končni avtomati imeli le dvojiške izhode zaradi našega pretežnega zanimanja za njihovo vlogo kot sprejemnikov jezikov. Vendar so znani tudi modeli, ko ima končni avtomat splošen izhod, ki priprada neki izhodni abecedi DELTA. Najpomembnejša tovrstna modela sta avtomat, pri katerem je izhod povezan s stanjem (ti. Mooreov avtomat), in avtomat, pri katerem je izhod povezan s prehodom (ti. Mealyjev avtomat).

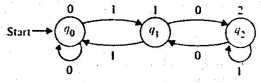
2.29. DEFINICIJA. Mooreov avtomat je šesterka $M = \langle Q, SIGMA, DELTA, dlt, lmd, q_0 \rangle$ pri čemer so Q, SIGMA, dlt in q_0

definirani enako kot v definiciji 2.1, DELTA je izhodna abeceda medtem, ko je lmd preslikava med Q in DELTA, ki določa izhod, ki priprada nekemu stanju.

Izhod Mooreovega avtomata M pri vhodu $x = a_1 a_2 \ldots a_n$ je

$lmd(q_0), lmd(q_1), \ldots, lmd(q_n)$, kjer je $q_i = dlt(q_0, a_1 a_2 \ldots a_i)$.

2.30. PRIMER



sl. 2.19 Mooreov avtomat za ostanke po modulu 3

Vhod za avtomat na 2.19 je dvojiško število (bit z večjo težo stoji pred bitom z manjšo težo), izhod pa je zaporedje ostankov po deljenju s 3.

2.31. DEFINICIJA. Mealyev avtomat je prav tako šesterka $M=<Q,SIGMA,DELTA,dlt,lmd,q_0>$ s to razliko, da pomeni sedaj lmd neko funkcijo Q x SIGMA --> DELTA. Izhod M pri vhodu $x = a_1 a_2 \ldots a_n$ je definiran tako, da je i-ti izhodni znak

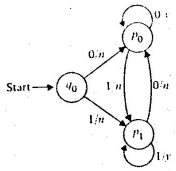
$lmd(dlt(q_0, a_1 a_2 \ldots a_{i-1}), a_i))$.

Moramo se zavedati, da obstaja neka malenkostna razlika med Mooreovim in Mealyevim avtomatom. Namreč pri prvem je izhod za prazno besedo definiran medtem, ko pri drugem ni.

2.32. PRIMER. Kot zanimivost lahko omenimo naslednje: Tudi v primeru, ko ima izhodna abeceda le dva simbola, nam lahko Mealyjev avtomat prihrani na številu stanj v primerjavi s končnim avtomatom. Vzemimo jezik $(0+1)^*(00+11)$, ki ga tvorijo vsi nizi, sestavljeni iz 0 in 1, z dvema enakima zadnjima simboloma. V naslednjem poglavju bomo razvili orodja za dokaz, da ta jezik ne sprejema noben DKA z manj kot petimi stanji. Po drugi strani pa lahko definiramo Mealyjev avtomat s tremi stanji, ki uporablja svoja stanja zato, da si zapomni zadnji prebrani znak, in na izhod zapiše y, ko je trenutni vhod enak prejšnjemu in n v nasprotnem primeru. Zaporedje znakov y in n na izhodu Mealyjevega avtomata ustreza zaporedju končnih in nekončnih stanj DKA pri istem vhodu. Vendar Mealyjev avtomat zapiše izhodni znak šele po tem, ko sprejme naslednji vhodni znak medtem, ko DKA nima neodvisni "službi" spomina in generiranja izhoda.

Mealyjev avtomat $M = <\{q_0, p_0, p_1\}, \{0,1\}, \{y,n\}, dlt, lmd, q_0>$ je prikazan na sl. 2.20. Na sliki uporabljamo oznako a/b ob povezavi med stanjema p in q v primeru dlt(p,a) = q ter lmd(p,a) = b. Odgovor M na vhod 01100 je nnyny, pri čemer je zaporedje stanj $q_0, p_0, p_1, p_1, p_0, p_0$. Stanje p_0 si zapomni 0, stanje p_1 pa

si zapomni 1. Stanje q_0 je začetno in ustreza temu, da še ni bilo vhodnih znakov.



sl. 2.20 Mealyjev avtomat za preverjanje enakosti zadnjih dveh znakov

---

Pravkar smo omenili, da obstaja neka malenkostna razlika med Mooreovim avtomatom in Mealuevim avtomatom. Naj $T_M(x)$ predstavlja izhodno zaporedje pri vhodnem zaporedju x. V primeru Mooreovega avtomata M je $T_M(\epsilon)$ definirano, dolžina $T_M(x)$ pa je $|x|+1$ medtem, ko pri Mealyevem avtomatu $T_M(\epsilon)$ ni definirano in je dolžina $T_M(x)$ enaka $|x|$. Vendar kljub temu lahko definiramo ekvivalenco med Mooreovim avtomatom M in Mealyevim avtomatom M' takole: M in M' sta ekvivalentna, če za vse vhode $x \neq \epsilon$ velja

$$T_M(x) = T_M(\epsilon)T_{M'}(x). \qquad (2.6)$$

Potem imamo

2.33. IZREK. Naj bo $M_1 = \langle Q,SIGMA,DELTA,dlt,lmd,q_0 \rangle$ Mooreov avtomat. Potem obstaja Mealyev avtomat M_2, ki je ekvivalenten M_1.

Dokaz. Naj bo $M_2 = \langle Q,SIGMA,DELTA,dlt,lmd',q_0 \rangle$, pri čemer velja $lmd'(q,a) = lmd(dlt(q,a))$ (torej Mealyev izhod pri nekem stanju q in vhodnem simbolu a je Mooreov izhod pri naslednjem stanju). V tem primeru ni težko preveriti (2.6). |_|

Prav tako pa velja

2.34. IZREK. Naj bo $M_1 = \langle Q,SIGMA,DELTA,dlt,lmd,q_0 \rangle$ Mealyjev avtomat. Potem obstaja Mooreov avtomat M_2, ki je ekvivalenten M_1.

Dokaz. Naj bo $M_2 =$ $\langle Q \times DELTA,SIGMA,DELTA,dlt',lmd',(q_0,b_0)\rangle$, kjer je b_0 neki poljuben znak izhodne abecede DELTA. Bistvo konstrukcije je v tem, da si izhodni znak Mealyjevega avtomata zapomnimo s stanjem Mooreovega avtomata. Nato postavimo $dlt'(\langle q,b\rangle,a) = \langle dlt(q,a),lmd(q,a)\rangle$ in . $lmd'(\langle q,b\rangle) = b$. Tudi v tem primeru ni težko preveriti (2.6). |_|

2.8 Naloge

2.1 Poiščite deterministične končne avtomate nad abecedo {0,1}, ki sprejemajo naslednje jezike.

1. nize, ki se končujejo z 00;
2. nize, ki vsebujejo tri zaporedne 0;
3. nize, pri katerih vsakih pet sosednih znakov vsebuje vsaj dve 0;
4. nize, ki se začenjajo z 1 in če jih pojmujemo kot dvojiška števila so kongruentna z 0 po modulu 5;
5. nize, pri katerih je deseti simbol z desne strani 1.

2.2 Poiščite nedeterministične končne avtomate, ki sprejemajo naslednje jezike.

1. nize iz množice $(0+1)^*$, pri katerih sta vsaj dve ničli narazen za 4i znakov, pri nekem $i \geq 0$.
2. nize nad abecedo {a,b,c}, ki imajo isto vrednost pri evaluaciji od leve proti desni kot od desne proti levi na podlagi množenja, ki je prikazano na sl. 2.21

2.3 Zapišite regularne izraze za vsakega od naslednjih jezikov nad abecedo {0,1}. Podajte tudi utemeljitev svoje rešitve.

1. množica nizov, ki vsebujejo največ en par zaporednih ničel in največ en par zaporednih enk;
2. množica nizov, v katerih vsi pari zaporednih ničel stojijo pred vsemi pari zaporednih enk;
3. množica nizov, ki nimajo podniza 101;
4. množica nizov, ki vsebujejo enako število ničel in enk in katerih vsi začetki imajo lastnost, da je razlika med številom ničel in enk manjša ali enaka dva.

2.4 Sestavi NKA, ki je enakovreden 2DKA $M = \langle \{q_0, \ldots, q_5\}, \{0,1\}, dlt, q_0, \{q_2\} \rangle$, pri čemer je dlt prikazana na sl. 2.21.

	0	1
q_0	(q_0, R)	(q_1, R)
q_1	(q_1, R)	(q_2, R)
q_2	(q_2, R)	(q_3, L)
q_3	(q_3, L)	(q_3, L)
q_4	(q_0, R)	(q_4, L)

sl. 2.21 Funkcija prehodov za neki 2DKA

2.5 Dvosmerni nedeterministični končni avtomat (2NKA) je definiran podobno kot 2DKA le, da ima 2NKA množico možnih prehodov, ki ustreza nekemu stanju in vhodnemu simbolu. Dokaži, da je jezik 2NKA regularen. (Napotek: sedaj ne velja več, da se v mejnem zaporedju stanje ne more ponoviti pri isti parnosti. Vendar lahko obravnavamo najkrajši izračun, ki nas pripelje do sprejemanja in izkoristimo njegove lastnosti za zahtevani dokaz.)

3. POGLAVJE

LASTNOSTI REGULARNIH JEZIKOV

Obstaja vrsta vprašanj, ki jih lahko postavimo v zvezi z regularnimi jeziki. Na primer, podan je neki jezik (na karšenkoli način) in je potrebno ugotoviti, ali je regularen. Nadalje, pri podanih dveh regularnih izrazih se lahko vprašujemo, ali predstavljata isti regularen jezik, pri podanem končnem avtomatu nas lahko zanima, koliko stanj ima avtomat z minimalnim številom stanj, ki sprejema jezik podanega avtomata, i.t.n

V tem poglavju bomo preučevali orodja za podobna vprašanja v zvezi z regularnimi jeziki. Opisali bomo t.im. "lemo o napihovanju", ki nam omogoča dokazati, da nekateri jeziki niso regularni. Nadalje bomo opisali nekaj operacij, ki ohranjajo regularne jezike. Vprašanja o regularnosti in neregularnosti pa lahko obravnavamo tudi z izrekom Myhill-Nerode iz razdelka 3.4. Končno obravnavamo tudi nekaj algoritmov, ki odgovarjajo na določena vprašanja o regularnih jezikih, na primer, ali je neki regularen jezik končen ali neskončen.

3.1 Lema o napihovanju za regularne jezike

Doslej smo obravnavali številne zglede in različice regularnih jezikov in končnih avtomatov tako, da bi nekdo morebiti odnesel vtis, da drugačnih jezikov sploh ni. Vendar temu ni tako. Torej obrnimo se k vprašanju, kako dokazati, da neki jezik ni regularen, oz. da zanj ne eksistira končen avtomat. V takem primeru je potrebno izbrati neko karakteristično lastnost regularnih jezikov in pokazati, da jezik, ki nas zanima, te lastnosti nima. Lastnost, ki največkrat pride v poštev za to, je dejstvo, da se deterministični končni avtomat na dovolj dolgi vhodni besedi "zacikla" oziroma, da pride do ponovitve stanja.

3.1. LEMA (o napihovanju za regularne jezike). Naj bo L regularen jezik. Potem obstaja neka konstanta n z lastnostjo, da za $x \in L$ velja $|x| \geq n$, potem eksistirajo u, v, w $\in$ SIGMA* z lastnostjo $x = uvw$, pri čemer imamo $|uv| \leq n$, $|v| \geq 1$ in za vse vrednosti $i \geq 0$ velja $uv^i w \in L$.

Dokaz. Naj bo M DKA in L = L(M). Naj ima M n stanj. Potem lahko uporabljamo n kot konstanto iz leme. Naj $x = a_1 a_2 \ldots a_N$ izpolnjuje pogoje iz leme. Potem zaporedje q_0, $dlt(q_0, a_1)$, $dlt(q_0, a_1 a_2)$, ..., $dlt(q_0, a_1 a_2 \ldots a_N)$ vsebuje vsaj dva enaka elementa (dolžina zaporedja je najmanj n + 1, različnih stanj pa je n). Naj sta to i-ti in k-ti element. Potem lahko

postavimo $u = a_1 a_2 \ldots a_i$ (v primeru i = 0 velja $u = \epsilon$) in $v = a_{i+1} a_{i+2} \ldots a_k$ ter $w = a_{k+1} \ldots a_N$. Očitno lahko v "napihnemo", saj s tem ne vplivamo na stanje $dlt(q_0, x)$, kajti napihovani del je cikel. Trditve o dolžinah uv ter v lahko brez težav preverimo. Dokaz je končan. |_|

Lemo 3.1 po navadi uporabljamo takole: podan je jezik L, za katerega želimo pokazati, da ni regularen. Dokazujemo od nasprotnega: predpostavimo, da je L regularen; torej velja lema 3.1 in obstaja konstanta n z opisanimi lastnostmi. Sedaj pa na podlagi opisa jezika L pokažemo, da bi lastnost napihovanja v jezik L vključila določene besede, ki po definiciji L ne bi smele biti v njemu.

3.2. PRIMER. Dokažimo, da $L = \{a^k b^k \mid k \geq 0\}$ ni regularen. Denimo, da je regularen. Potem eksistira konstanta n iz leme 3.1. Naj bo $x = a^n b^n$. Iz posledice $|uv| \leq n$ v lemi sledi, da je v sestavljeno iz samih simbolov a. Z napihovanjem nato dobimo neko besedo $x' = a^m b^n$ pri $m \neq n$, ki bi po lemi 3.1 morala pripadati L, vendar na podlagi definicije L ne sme. Prišli smo do protislovja in sklepamo da L ni regularen. |_|

3.2 Operacije, ki ohranjajo regularne jezike

Če imamo opravka z neko vrsto matematičnih objektov, je povsem naravno vprašati se, kakšne je operacije možno definirati na objektih te vrste, da so tudi rezultati operacije objekti iste vrste. V našem primeru se torej sprašujemo, kakšne operacije ohranjajo regularne jezike.

3.3. IZREK Operacije unija, stik in iteracija ohranjajo regularne jezike.

Dokaz. Sledi takoj iz definicije regulanih izrazov (s temi operacijami so regularni izrazi namreč definirani). |_|

3.4. IZREK. Operacija komplementiranja ohranja regularne jezike.

Dokaz. Naj bo podan regularen jezik L. Potrebno je le opisati končen avtomat, ki sprejema $\overline{L}$. Lahko predpostavimo, da je L podan z deterministčnim končnim avtomatom $M = (Q, SIGMA, dlt, q_0, F)$, pri čemer velja $L(M) = L$. Bralec se bo zlahka prepričal, da avtomat $M' = (Q, SIGMA, dlt, q_0, Q-F)$ sprejema jezik $\overline{L}$. |_|

3.5. IZREK. Operacija presek ohranja regularne izraze.

Dokaz. Podana sta regularna jesika L_1 in L_2. Dokazati moramo, da je $L = L_1 \,|\overline{\ }|\, L_2$ regularen. To pa sledi iz izrekov 3.3

---

in 3.4 ter identitete $L_1 \,|\overline{\ }|\, L_2 = \overline{L}_1 \,|\_|\, \overline{L}_2 \,.\, |\overline{\ }|$

Za unijo in presek obstaja tudi neposredna konstrukcija, ki nam zagotavlja regularnost. Torej podana sta regularna L_1 in L_2 ter ustrezna DKA M_1 in M_2. Naj bo $M_1 = \langle Q_1, SIGMA, dlt_1, q_{1_1}, F_1 \rangle$ in $M_2 = \langle Q_2, SIGMA, dlt_2, q_{2_2}, F_2 \rangle$. Sedaj definirajmo "produktna avtomata" M_U in M_P takole:

$$M_i = \langle Q_1 \times Q_2, SIGMA, dlt, \langle q_{1_1}, q_{2_2} \rangle, F_i \rangle, \; i \in \{U,P\},$$

pri čemer je dlt definirano kot $dlt(\langle q,r \rangle, a) = \langle dlt_1(q,a), dlt_2(r,a) \rangle$

in

$$F_U = \{\langle q,r \rangle \mid (q \in F_1) \text{ or } (r \in F_2)\}$$

oziroma

$$F_P = \{\langle q,r \rangle \mid (q \in F_1) \text{ and } (r \in F_2)\}.$$

Dokaz, da M_U in M_P sprejemata $L_1 \,|\_|\, L_2$ in $L_1 \,|\overline{\ }|\, L_2$, po vrsti, prepuščamo za vajo bralcu.

Naslednja operacija, ki nas zanima, je substitucija. Naj bo SIGMA neka abeceda, f pa preslikava, ki priredi vsakemu simbulu $a \in$ SIGMA neki regularen jezik R_a (morebiti nad neko drugo abecedo DELTA). Potem lahko f razširimo na besede takole:

1. $f(\epsilon) = \epsilon$
2. $f(x \circ a) = f(x) \circ R_a$

Potemtakem f vsaki besedi x priredi neki regularen jezik R_x. f lahko naprej razširimo na jezike:

$$f(L) = \; |\_|_{x \in L} \, R_x$$

Opisani preslikavi $f: POT(SIGMA^*) \to POT(DELTA^*)$ pravimo substitucija.

3.6. IZREK. Substitucija ohranja regularne jezike.

Dokaz. Podan je regularen jezik L in substitucija f. Dokazati moramo, da je f(L) regularen. Z drugimi besedami, če je podan končen avtomat $M = \langle Q, SIGMA, dlt, q_0, F \rangle$ za L, je potrebno definirati končen avtomat M' za f(L). M' zlahka dobimo tako, da v diagramu prehodov avtomata M zamenjamo vsako povezavo, ki je zaznamovana z $a \in$ SIGMA, z avtomatom M_a, ki sprejema jezik R_a. Ker za M_a lahko predpostavimo, da ima le eno začetno in eno končno stanje (gl. stavka 2.16 in 2.17), je taka zamenjava možna. Bralcu prepuščamo za vajo, da preveri, da tako opisan

avtomat resnično sprejema f(L). Na podlagi opisane konstrukcije je M' prav tako končen avtomat in je torej f(L) regularen. |_|

Poseben primer substitucije je homomorfizem monoidov, ki smo ga že definirali v odstavku 2.4. Očitno je homomorfizem taka substitucija, pri kateri velja |f(a)| = 1. Torej vsak simbol, pa tudi besedo, preslikamo v eno samo besedo. Če je f homomarfizem, pravimo korespondenci f^{-1} inverzni homomorfizem (gl. str. 36).

3.7. IZREK. Homomorfizmi in inverzni homomorfizmi ohranjajo regularnost.

Dokaz. Trditev velja za homomorfizme na podlagi izreka 6 in zgornje pripombe, da je homomorfizem poseben primer substitucije. Glede inverznega homomorfizma pa postopamo takole: naj bo f homomorfizem, L pa regularen jezik. Želimo pokazati, da je $f^{-1}(L)$ regularen. Začnimo z definicijo: $f^{-1}(L) = \{x \mid f(x) \in L\}$. Sedaj pa lahko sestavimo avtomat M' za $f^{-1}(L)$ tako, da uporabljamo stroj M za L. Vsakič ko M' sprejme simbol a, pošlje na vhod stroja M (za L) besedo f(a). Če po sprejemu besede x M' ugotovi, da je M v končnem stanju, potem tudi M' sprejme besedo x (se postavi v končno stanje), sicer pa ne. Podrobno formalno definicijo stroja M' prepuščamo bralcu za vajo. |_|

3.3 Odločitveni algoritmi za regularne jezike

Tehniko iz dokaza leme 3.1 lahko uporabimo tudi za sestavljanje algoritmov, ki dajejo odgovor na neka karakteristična vprašanja v zvezi z jeziki. Naj je podan regularen jezik L z determinističnim končnim avtomatom M. Zanima nas, ali eksistira računalniški algoritem[1], katerega vhod je primerno zakodiran opis stroja M, izhod pa je odgovor v obliki "da" ali "ne" na vprašanje (1) ali je L(M) = Ø in (2) ali je L(M) končen.

Algoritem za L(M) = Ø sloni na naslednjem rezultatu.

3.8. LEMA. Naj bo L regularen in naj bo L = L(M), kjer je M DKA z n stanji. Če velja L ≠ Ø, potem vsebuje L neko besedo x z lastnostjo $|x| \leq n-1$.

Dokaz. Naj velja L ≠ Ø in naj bo x ena med najkrajšimi besedami v jeziku L. Naj velja $|x| \geq n$. Potem v zaporedju iz dokaza leme 3.1 zasledimo ponavljanje stanj, iz česar sledi, da lahko en del besede x izrežemo in tako dobimo $x' \in L$ ter $|x'| < |x|$. Torej x ne more biti med najkrajšimi besedami v jeziku L in smo prišli v protislovje. Sklepamo torej, da je bila predpostavka $|x| \geq n$ napačna. |_|

---

[1] S pojmom algoritma se bomo podrobneje seznanili v poglavjih 7 in 8, za enkrat se zanašamo na bralčevo intuicijo.

Algoritem za L(M) = Ø potem sestavimo tako, da preverimo vse besede dolžine do vključno n - 1, ali pripadajo jeziku L(M). Če nobena ne pripada, je odgovor "DA" sicer "NE". Seveda pa je možno sestaviti mnogo učinkovitejši algoritem za L = Ø na podlagi algoritmov za iskanje poti v grafih, vendar, ker to presega okvir tega dela, se v takšne algoritme ne bomo spuščali.

Algoritem za |L(M)| je neskončno pa sloni na naslednjem rezultatu.

3.9. LEMA. Naj bo L regularen in naj bo L = L(M), kjer je M DKA z n stanji. |L(M)| je neskončno natanko takrat, ko L vsebuje neko besedo x z lastnostjo $n \leq |x| \leq 2n - 1$.

Dokaz. Naj obstaja x z opisano lastnostjo. Potem izpolnjuje n pogoje za konstanto n iz leme 3.1, torej x lahko napihnemo in dobimo neskončno elementov L. Torej je L(M) neskončno. Sedaj pa denimo da je L(M) neskončno. Potem v L obstajajo besede, ki so daljše od poljubnega vnaprej danega števila. Vzemimo tako dolgo besedo x, da se v zaporedju stanj pojavi ponavljanje, podobno kot v dokazu leme 3.1. Dolžina najkrajšega cikla (zaporedja stanj do ponovitve je $\leq n$. Sedaj pa iz x izrežemo tak cikel ter dobimo $x' \in L$. Za x' velja bodisi $|x'| \geq n$ ali $|x'| \leq n-1$. V prvem primeru lahko postopek nadaljujemo, v drugem pa vrnemo pravkar izrezani del in pridemo z dolžino x znotraj intervala, ki ga zahteva lema. |_|

Algoritem za vprašanje, ali je |L(M)| neskončno, dobimo tako, da preverimo vse besede, katerih dolžina je v intervalu [n, 2n - 1], ali pripadajo L. Če nobena ne pripada jeziku L, je odgovor "NE", sicer je "DA".

3.4 Izrek Myhill-Nerode

Sedaj bomo pa opisali neko lastnost regularnih jezikov, ki nam omogoča, da poiščemo determinictični končni avtomat z minimalnim številom stanj za neki vnaprej podani regularni jezik L. Najprej nekaj definicij.

Za neko ekvivalenčno relacijo E, ki je definirana na nekem monoidu (denimo $SIGMA^*$) pravimo, da je desno invariantna, če velja

$$\forall x,y[(xEy) \Rightarrow \forall z[xzEyz]]$$

Pojem indeksa neke ekvivalenčne relacije nam je tudi že znan in pomeni število ekvivalenčnih razredov. Sedaj pa k naslednjemu rezultatu.

3.10. IZREK (Myhill-Merode). Naslednje trditve so logično ekvivalentne:

1. L je jezik nekega končnega avtomata (regularen jezik).
2. L je unija določenega stevila ekvivalenčnih razredov neke desno invariantne ekvivalenčne relacije R končnega indeksa.
3. Desno invariantna ekvivalenčna relacija R_L, ima končen indeks, kjer je R_L definirano kot

$$xR_L y \Leftrightarrow \forall z[(xz \in L) \Leftrightarrow (yz \in L)] \qquad (3.1)$$

(z besedami : x in y sta ekvivalentna takrat, ko za vse

podaljške z velja, da je xz ∈ L natanko takrat, ko je yz ∈ L) Dejstvo, da je R_L desno invariantna ekvivalenčna relacija naj bralec preveri sam.

Dokaz je sestavljen po shemi 1 ==> 2 ==> 3 ==> 1.

(1 ==> 2). Denimo, da velja 1. Potem eksistira DKA M = $\langle Q,SIGMA,dlt,q_0,F\rangle$ z lastnostjo L = L(M). Definirajmo C_q = {x | $x \in SIGMA^*$, $dlt(q_0,x) = q$}. Zlahka se prepričamo, da velja (a) množice C_q so medsebojno tuje, (b)

$x,y \in C_q \Longrightarrow \forall z \in SIGMA^* \exists r \in Q[xz,yz \in C_r]$ in seveda (c)

$|\_| = SIGMA^*$ ter (d) število množic C_q je končno. Iz (a) - $q \in Q$

(c) sledi da C_q, $q \in Q$, predstavljajo ekvivalenčne razrede neke desno invarantne ekvivalenčne relacije R_M, (d) pa pomeni da je njen indeks končen. Ker velja $L = \bigcup_{q \in F} C_q$, smo dokazali, da velja 2.

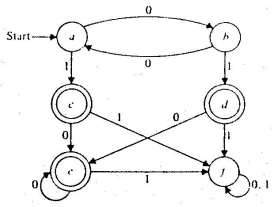
(2 ==> 3). Denimo, da velja 2. Zadostovalo bo da pokažemo, da velja

$$\text{indeks}(R_L) \leq \text{indeks}(R); \tag{3.2}$$

slednje pa bo sledilo iz xRy ==> $xR_L y$. Torej dokažimo pravkar zapisano implikacijo. Naj velja xRy. Ker je R desno invariantno, velja tudi xzRyz za vse z. Torej xz in yz pripadata istemu ekvivalenčnemu razredu relacije R. Ker pa je L unija določenega števila ekvivalenčnih razredov R, velja, da sta xz in yz istočasno v jeziku ali izven njega. Torej smo dokazali (3.2) in s tem tudi 3.

(3 ==> 1) Denimo, da velja 3. Sestavili bomo končnen avtomat M tako, da bo veljalo L = L(M). Naj [x] predstavlja ekvivalenčni razred besede x glede na R_L. M definiramo kot M = $\langle Q,SIGMA,dlt,q_0,F\rangle$, kjer je Q = {[x] | x ∈ SIGMA}, dlt([x],a) = [xa], q_0 = [ϵ] in F = {[x] | x ∈ L}. Q je seveda končno na podlagi končnega indeksa R_L, dlt pa je dobro definirano na podlagi desne invariance. S tem je dokaz izreka končan. |‾|

Pojme iz izreka 3.10 lahko prikažemo na naslednjem primeru:



sl. 3.1 Primer determinističnega končnega avtomata.

3.11. PRIMER. Podan je avtomat na sl. 3.1. Ni se težko prepričati v naslednje:

$$C_a = (00)^*,\ C_b = (00)^*0,\ C_c = (00)^*1,\ C_d = (00)^*01,$$

$$C_e = (C_c + C_d)00 = 0\,100^*\ \text{in}\ C_f = (C_c + C_d + C_e)1(0+1)^*$$

$$= 0\,10^*1(0+1)^*$$

sk 1.1 0 Za jezik avtomata ugotovimo

$$L = C_c + C_d + C_e = 0\,10^*$$

(3.3)

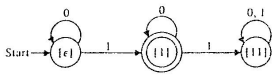
Kako pa poiščemo R_L? Zapisali bomo kar trditev, da so ekvivalenčni razredi R_L $C_0 = 0^*$, $C_1 = 0\,10^*$ in $C_{11} = (0+1)^* - 0^*(\epsilon + 10^*) = 0\,10^*1(0+1)^*$, in jo dokazali. Z besedami lahko

2

opišemo C_0 kot množico vseh besed, ki ne vsebujejo simbola 1, C_1 kot množico besed, ki vsebujejo eno samo enko in C_{11} kot množico besed, ki vsebujejo vsaj dve enki. Če definiramo R_L kot ekvivalenčno relacijo z ekvivalenčnimi razredi C_0, C_1 in C_{11}, potem se na podlagi relacij

$$C_0 = C_a + C_b,\ C_1 = C_c + C_d + C_e,\ C_{11} = C_f,$$

definicije (3.1) in sl. 3.1 ni težko prepričati v $xRy \Rightarrow xR_L y$. Potrebno je namreč le preveriti, da sta podaljška dveh besed iz istega ekvivalenčnega razreda R_L, ki pa pripadata različnim ekvivalenčnim razredom R_M s sl. 3.1, vedno istočasno v jeziku ali izven. Torej velja, da je R_L kvečjemu bolj groba relacija. Vendar prav tako ugotovimo, da po združevanju katerih koli dveh ekvivalenčnih razredov R_L pogoj (3.1) ni več izpolnjen. Torej velje $R_L = R_L$. Sklepamo torej, da je ekvivalenčna relacija avtomata na sl. 3.1 bolj fina kot R_L in bi lahko L sprejemal avtomat na sl. 3.2.



sl. 3.2 Poenostavljeni avtomat za jezik avtomata s sl. 3.1

Nadaljnje zmanjševanje števila stanj avtomata za L ni možno, ker smo v dokazu izreka 3.10 ugotovili, da je indeks R_L najmanjši možni indeks ekvivalenčne relacije ki nastopa v 2. trditvi izreka 3.10.

Izrek 3.10 predstavlja osnovo za algoritem ki poišče DKA z minimalnim številom stanj za neki regularen jezik L, če je podan kakršenkoli DKA M = <Q,SIGMA,dlt,q_0,F> z lastnostjo L = L(M). V dokazu izreka smo videli, da indeks(R_L) $\leq$ indeks(R_M), kjer je indeks(R_M) pravzaprav število stanj avtomata M za L. Nadalje smo ugotovili, da obstaja avtomat za L, katerega stanja so ekevivalenčni razredi R_L in končno, da ekvivalenčne razrede R_L (oz. stanja ustreznega avtomata za L) dobimo z združevanjem ekvivalenčnih razredov R_L (oz. ustreznih stanj avtomata M). Definirali bomo:

$$qEr \iff C_q \; R_L \; C_r \;, q,r \in Q \tag{3.4}$$

(kjer seveda $C_q \; R_L \; C_r$ pomeni, da sta poljubni dve besedi, če eno

vzamemo iz C_a, drugo pa iz C_r, R_L ekvivalentni). Torej, če za L želimo poiskati avtomat z minimalnim številom stanj, je potrebno le poiskati skupine ekvivalentnih stanj Q in vsako skupino združiti v eno samo stanje.

Algoritem deluje tako, da poišče neekvivalentna stanja. To pa je enakovredno določanju ekvivalentnih stanj, saj če poznamo vse pare neekvivalentnih stanj, vemo, da so preostali pari ekvivalentni. Prepišimo sedaj (3.4) v drugačno obliko:

$$qEr \iff C_q \; R_L \; C_r \;,\; q,r \in Q, \iff \forall x \in C_q, y \in C_r, z \in SIGMA^* \; [(xz \in L) \iff (yz \in L)] \tag{3.5}$$

Izraz v oglatih oklepajih v (3.5) je seveda enakovreden izrazu

$$(xz \in L) \text{ and } (yz \in L) \text{ or } (xz \notin L) \text{ and } (yz \notin L) \tag{3.6}$$

Sedaj pa negirajmo (3.5) upoštevajoč (3.6):

$$qEr \iff \text{not}(C_q \; R_L \; C_r) \iff \exists x \in C_q, y \in C_r, z \in SIGMA^* \; [(xz \in L) \text{ and } (yz \notin L) \text{ or } (xz \notin L) \text{ and } (yz \in L)] \tag{3.7}$$

Vendar če smo našli en par z lastnostjo (3.7), velja to za vse pare <x,y>, $x \in C_q$, $y \in C_r$, (to sledi iz R_M ekvivalence besed, ki pripadajo C_q oziroma C_r). Torej lahko zapišemo

$$qEr \iff \forall x \in C_q, y \in C_r \; \exists z \in SIGMA^* \; [(xz \in L) \text{ and } (yz \notin L) \text{ or } (xz \notin L) \text{ and } (yz \in L)] \tag{3.8}$$

Sedaj pa denimo, da velja qĘr. Potem na podlagi (3.8) sledi, da eksistira ustrezen z, ki nam neekvivalenco pravzaprav dokazuje. Sedaj pa izberimo enega izmed najkrajših z z lastnostjo (3.8). Če velja |z| = i, potem bomo rekli, da sta q in r i-neekvivalentna. Seveda vsi pari, ki niso i-neekvivalentni so i-ekvivalentni. Naj bo $z = a_1 a_2 \ldots a_i$.

Oglejmo si zaporedje

$$q, dlt(q,a_1), dlt(q,a_1 a_2), \ldots, dlt(q,a_1 a_2 \ldots a_i) = q' \qquad (3.9)$$

in

$$r, dlt(r,a_1), dlt(r,a_1 a_2), \ldots, dlt(r,a_1 a_2 \ldots a_i) = r' \qquad (3.10)$$

Na podlagi (3.8) ugotovimo, da velja

(q' Є F) and (r' Ę F) or (q' Ę F) and (r' Є F)

iz česar sledi, da sta q' in r' 0-neekvivalentna. Na splošno lahko trdimo, da sta v zaporedjih (3.9) in (3.10) $dlt(q,q_1 a_2 \ldots a_k)$ in $dlt(r,a_1 a_2 \ldots a_k)$ (i-k)-neekvivalentna pri k = 0,1,...,i. Torej velja tudi

q in r sta i-neekvivalentna natanko takrat, ko eksistira tak a Є SIGMA, da sta dlt(q,a) in dlt(r,a) (i-1)-neekvivalentna. (3.11)

Sedaj pa še edino vprašanje, ki ga moramo razčistiti predno opišemo algoritem za minizacijo. Kako velik je lahko i? Iz (3.8) sledi

$$qĘr \Rightarrow \bar{=}|i[qĘ_i r],$$

kjer $Ę_i$ pomeni i-ekvivalenco. Velikost i ocenomo iz zaporedij (3.9) in (3.10). Če se med pari istoleženih stanj neki par ponovi, pomeni, da lahko zaporedje skrajšamo, saj beseda z, ki ji manjka del, ki ustreza parom med ponovitvijo, prav tako dokazuje neekvivalentnost q in r. Prav tako velja, da med pari v zaporedjih (3.9) in (3.10) ne moreta dve komponenti para biti enaki, saj v tem primeru beseda ne bi mogla ugotoviti neekvivalentnost q in r. Torej je i manjše ali enako številu parov različnih stanj, ki pa je enako

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

kjer je n = |Q|.

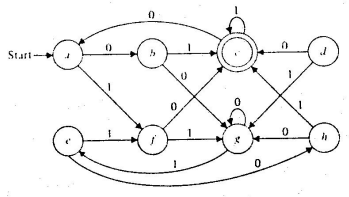
Sedaj pa

3.12. ALGORITEM (za minimizacijo števila stanj deterministićnega končnega avtomata za jezik L).

Vhod: avtomat M = <Q,SIGMA,dlt,q_0,F> z lastnostjo L = L(M)

```
Algoritem:
    BEGIN
      FOR <q,r> ∈ Q x Q, q ≠ r DO
      BEGIN
        neekv[q,r] :=
        (q ∈ F) and (r ∉ F) or (q ∉ F) and (r ∈ F) ;
        {0-neekvivalenca}
        go := true ;
        WHILE go DO
        BEGIN go := false ;
          FOR <q,r> ∈ Q x Q, q ≠ r, not neekv[q,r] DO
            BEGIN
              neekv[q,r] :=
              ¬|a ∈ SIGMA[neekv[dlt(q,a),dlt(r,a)]];
              go := neekv[q,r]
            END
        END
      END
    END
```

Delovanje algoritma 3.12 na avtomatu s sl. 3.3 prikazuje sl. 3.4.

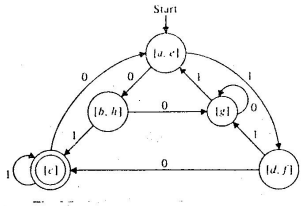


sl. 3.3 Vhod za algoritem 3.12

b	1						
c	0	0					
d	1	1	0				
e		1	0	1			
f	1	1	0		1		
g	2	1	0	1	2	1	
h	1		0	1	1	1	1
	a	b	c	d	e	f	g

sl. 3.4 Delovanje algoritma 3.12

Elementi tabele neekv, ki ustreezajo paru <q,r> vsebujejo bodisi vrednost i ≥ 0 ali pa posebno vrednost "prazno". Nezaznamovani pari stanj (vrednost je "prazno") so ekvivalentni, pri zaznamovanih parih pa nam število i pove, v katerem izvajanju jedra WHILE stavka je bil par zaznamovan (i = 0 pomeni, da je bil par zaznamovan pred vstopom v WHILE zanko). Avtomat, ki je ekvivalenten avtomatu s sl. 3.3, in čigar opis smo pravkar pridobili z algoritmom 3.12, je prikazan na sl. 3.5.



sl. 3.5 Avtomat z minimalnim številom stanj

3.5 Naloge

3.1 Kateri izmed naslednjih jezikov so regularne množice? Dokaži svojo trditev.

1. $\{0^{2n} \mid n \geq 1\}$
2. $\{0^m 1^n 0^{m+n} \mid m \geq 1 \text{ and } n \geq 1\}$
3. $\{0^n \mid n \text{ je praštevilo}\}$
4. Množica nizov, ki ne vsebujejo tri zaporedne 0;
5. Množica nizov, z istim številom 0 in 1;
6. $\{x \mid x \in (0+1)^* \text{ and } x^R = x\}$
7. $\{xwx^R \mid x,w \in (0+1)^+\}$
8. $\{xx^R w \mid x,w \in (0+1)^*\}$

3.2 Dokaži naslednjo razširitev leme o napihovanju za regularne množice. Naj bo L regularna množica. Potem obstaja konstanta n tako, da pri vseh z_1, z_2, z_3, kjer je $z_1 z_2 z_3 \in L$ in $|z_2| = n$, z_2 lahko predstavimo kot $z_2 = uvw$ in $|v| \geq 1$ ter pri vseh $i \geq 0$,

$uv^i wz_3 \in L$.

3.3 Uporabi nalogo 3.2 v dokazu, da $\{0^{i^m} \mid i \geq 1, m \geq 1\}$ ni regularen.

3.4 Naj bo L regularna množica. Kateri izmed naslednjih jezikov so regularni? Podaj utemeljitev svoje trditve.

1. $\{a_1 a_3 a_5 \ldots a_{2n-1} \mid a_1 a_2 a_3 a_4 \ldots a_{2n} \in L\}$
2. $\{a_2 a_1 a_4 a_3 \ldots a_{2n} a_{2n-1} \mid a_1 a_2 a_3 a_4 \ldots a_{2n} \in L\}$
3. $CYCLE(L) = \{x_1 x_2 \mid x_2 x_1 \in L, x \text{ in } x \text{ poljubno}\}$
4. $MAX(L) = \{x \mid x \in L \text{ and } y \neq \epsilon \Rightarrow xy \notin L\}$
5. $MIN(L) = \{x \mid x \in L \text{ and noben začetek x ni v L}\}$
6. $INIT(L) = \{x \mid \text{eksistira } y^R \neq \epsilon \text{ tako, da } xy \in L\}$
7. $\{x \mid xx^R \in L\}$

3.5 Naj bo val(x) rezultat množenja simbolov, ki sestavljajo x, od leve proti desni na podlagi tabele na sl. 3.6.

1. Ali je $L = \{xy \mid |x| = |y| \text{ and } val(x) = val(y)\}$ regularen?
2. Ali je $L = \{xy \mid val(x) = val(y)\}$ regularen?

Podajte utemeljitev svojih trditev.

\	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

sl. 3.6 Neasociativno množenje

4. POGLAVJE

KONTEKSTNO NEODVISNE GRAMATIKE

4.1 Osnovni pojmi

Primer 3.2 je primer jezika, ki ni regularen. Že iz tega primera smo lahko sklepali, da bomo potrebovali močnejša orodja za opisovanje in definicijo jezikov. Razen tega pa potreba po takih orodjih izvira iz potreb računalniške in druge prakse.

Za razredom regularnih jezikov je naravno obravnavati razred kontekstno neodvisnih jezikov, ki jih generirajo kontekstno neodvisne gramatike. Kontekstno neodvisna gramatika je sestavljena iz določenega števila zamenjevalnih pravil, ali kot jim pravimo, produkcij. Vsaka produkcija opisuje, kako lahko neko spremenljivko ali z drugimi besedami sintaktično kategorijo zamenjamo z neko besedo, ki je sestavljena iz spremenljivk in končnih simbolov abecede, nad katero je definiran jezik. Postopek generiranja neke pravilne besede iz jezika poteka tako, da začnemo z neko predpisano začetno spremenljivko in uporabljamo produkcije (opravljamo zamenjave) tako dolgo, dokler ne dobimo neke besede, ki ne vsebuje

spremenljivk. Ker so zamenjave neke spremenljivke možne ne glede na simbole, ki jo obkrožajo, oziroma ne glede na spremenljivkino okolico ali kontekst, pravimo tem gramatikam kontekstno neodvisne.

Študij kontekstno neodvisnih gramatik se je začel najprej na področju jezikoslovja, ker lahko z njimi opišemo nekatere vidike naravnih jezikov. Na primer: v slovenščini lahko rečemo "tisti je šel", kjer se "tisti" nanaša na neko osebo. Lahko pa tudi rečemo "moj prijatelj je šel" ali "moj najboljši prijatelj je šel". Torej vidimo, da lahko pridobivamo čedalje bolj zapletene stavke iz nekega preprostega stavka z zamenjavo nekega elementa. Kot primer vzemimo naslednje produkcije

```
<stavek> --> <osebek><povedek>
<osebek> --> <samostalnik>
<povedek> --> <glagol>
<samostalnik> --> maček
<samostalnik> --> pes
<glagol> --> je
<glagol> --> pije
```

Spremenljivke so obkrožene z znakoma < in > medtem, ko so končni simbol jezika besede brez oklepajev. Iz začetnega simbola <stavek> lahko na primer razvijemo

```
<stavek> ==> <osebek><povedek> ==> <samostalnik><povedek>
 ==> maček<povedek> ==> maček<glagol> ==> maček je
```

(dvojna puščica ==> pomeni "neposredno razvijemo v" ali "neposredno daje").

Pomen npr. prve produkcije <stavek> --> <osebek><povedek> je, da je stavek sestavljen iz osebka, ki mu sledi povedek. Ob zgornjem primeru je dobro, da si določene stvari zapomnimo. Ko smo vpeljali pojem jezika, smo njegovim elementom dali ime "beseda" vendar pri kontekstno neodvisnih jezikih je bolj v navadi, da jim pravimo "stavek". Torej si moramo zapomniti, da je stavek sinonim za besedo. Prav tako v tem konkretnem primeru utegne zavajati dejstvo, da so končni simboli jezika sestavljeni iz več tipografskih znakov. Vendar si moramo npr. besedo "pije" predstavljati kot en sam znak.

Iz različnih razlogov kontekstno neodvisne gramatike ne morejo popolnoma adekvatno opisovati naravnih jezikov. V bistvu gre za to, da zamenjav le ni možno delati neodvisno od konteksta.

Približno istočasno z jezikoslovci so računalniški strokovnjaki za opisovanje programskih jezikov začeli uporabljati določen zapis, ki so mu rekli Backus-Naurova oblika (BNF). BNF notacija se je izkazala za uspešno pri opisovanju "rekurzivne" zgradbe nekaterih programskih jezikov. Primer take zgradbe je npr. možnost gnezdenja blokov, ki so razmejeni z BEGIN in END simboloma, ali izrazov, kjer lahko oklepaje prav tako gnezdimo. BNF notacija je pravzaprav enakovredna kontekstno neodvisnim gramatikam. Kot primer navajamo BNF zapis gramatike aritmetičnih izrazov:

```
<izraz> --> <izraz> + <izraz>
<izraz> --> <izraz> * <izraz>                    (4.1)
<izraz> --> (<izraz>)
```



```
<izraz> --> ime
```

Sedaj bomo uporabili pravkar zapisano gramatiko za generiranje nekega izraza

```
<izraz> ==> <izraz> * <izraz>
        ==> (<izraz>) * <izraz>
        ==> (<izraz>) * ime
        ==> (<izraz> + <izraz>) * ime
        ==> (<izraz>) + ime) * ime
        ==> (ime + ime) * ime
```

Podobno kot pri jezikoslovju lahko ugotovimo, da kontekstno neodvisne gramatike ne omogočajo popolnega opisa programskih jezikov, vendar so se izkazale kot izredno koristne pri podajanju preciznih definicij programskih jezikov, kakor tudi pri sestavljanju algoritmov za analizo programov, napisanih v programskem jeziku.

4.2 Kontekstno neodvisne gramatike

Sedaj pa je čas, da podamo formalno definicijo:

4.1. DEFINICIJA. Kontekstno-neodvisna gramatiko (KNG) je četverka $G = \langle V,T,P,S \rangle$, kjer sta V in T množici spremenljivk (ali nekončnih simbolov) in končnih simbolov, po vrsti. Predpostavljali bomo, da sta V in T medsebojno tuji množici. $P \subseteq V \times (V \cup T)^*$ je množica parov ali produkcij, katerih prva komponenta je spremenljivka, druga pa beseda sestavljena iz spremenljivk in končnih simbolov (v skrajnem primeru je druga

komponenta prazna beseda). Komponente neke produkcije, kot smo videli, ločimo s puščico --> , da nakažemo, da leva stran porodi desno. Končno je S ∈ V neki predpisani začetni simbol ali aksiom.

Na primer gramatiko (4.1) lahko bolj shematično predstavimo kot G = <{E}, {+,*,*,(,),ime},P,E>, pri čemer P sestavljajo produkcije

```
E --> E + E
E --> E * E

E --> (E)                                   (4.2)
E --> ime
```

Pri razpravah o kontekstno neodvisnih jezikih in gramatikah se bomo držali naslednjih dogovorov:

1. Velike črka A,B,C,D,E ter S predstavljajo spremenljivke; S je začetni simbol (razen če ni izrecno zapisano drugače);
2. Male črke a,b,c,d,e, cifre ter podčrtane besede so končni simboli;
3. X,Y,Z so simboli, ki so lahko spremenljivke ali končni simboli;
4. Male črke u,v,w,x,y predstavljajo besede, sestavljene iz končnih simbolov;
5. Male grške črke alf, bta in gma predstavljajo besede, ki so sestavljene iz končnih in nekončnih simbolov.

Ti dogovori nam omogočajo, da se izognemo odvečnim pojasnilom pri opisovanju neke gramatike. Če imamo več produkcij z isto levo stranjo npr. A --> ..., A --> ..., ... , A --> ..., potem jih bomo skrajšano zapisali kot A --> ...|...|...|...; torej s pokončno črto med alternativnimi desnimi stranmi. Na primer gramatiko (4.2) bi lahko zapisali v obliki


```
E --> E + E | E * E | (E) | ime
```

Sedaj pa je čas, da pojem jezika, ki ga generira neka KNG G = <V,T,P,S>, definiramo formalno. Najprej bomo definirali na množici $(V \cup T)^*$ relacijo ==> , ki smo jo že prej poimenovali z "neposredno razvijemo v" ali "neposredno daje". Definiramo pa jo takole: alf ==> bta natanko, ko velja alf = alf_1 Aalf$_2$, bta = alf_1 gmaalf$_2$ in eksistira produkcija A --> gma. Torej je pomen alf ==> bta, da lahko bta dobimo iz alf z enkratno zamenjavo nekega nekončnega simbola z desno stranjo ene izmed produkcij, ki imajo ta nekončni simbol na levi strani. Z relacijo ==> lahko definiramo njeno tranzitivno ovojnico $\overset{*}{\Longrightarrow}$; torej velja alf $\overset{*}{\Longrightarrow}$ bta natanko, ko eksistira zaporedje alf = alf_1 ==> alf_2 ==> ... ==> alf_k = bta, pri nekem končnem k $\geq$ 0.

alf $\overset{*}{\Longrightarrow}$ bta beremo kot "alf razvijemo v bta". (Poudarjamo, da alf $\overset{*}{\Longrightarrow}$ alf vedno velja.)

4.2. DEFINICIJA. Jezik neke kontekstno neodvisne gramatike G = <V,T,P,S> je množica

$$L(G) = \{x \mid x \in T^*, S \overset{*}{\Longrightarrow} x\}.$$

Torej je to množica besed, ki so sestavljene iz samih končnih simbolov, in ki jih je možno izpeljati iz začetnega simbola S.

4.3. PRIMER. Jezik $\{a^n b^n \mid n \geq 1\}$, (gl. tudi primer 3.2) lahko generiramo z gramatiko s produkcijami S --> aSb | ab.

Dokaz. Očitno je, da vsako besedo $a^n b^n$, $n \geq 1$, lahko izpeljemo iz S z natanko n koraki (z n-1 uporabami produkcije S --> aSb in eno uporabo pro- dukcije S --> ab). Torej velja $L \subseteq L(G)$. Za dokaz dejstva $L = L(G)$ pa je potrebno le ugotoviti, da velja $L(G) - L = \emptyset$. To pa gotovo velja saj v tem primeru pri uporabi produkcij nimamo nobene alternative: takoj ko uporabimo produkcijo S --> ab smo izpeljavo zaključili. Torej vsaka izpeljava, ki vsebuje n korakov, uporablja S --> aSb pri prvih n -1 korakih in S --> ab na zadnjem koraku. $\square$

4.4. PRIMER. Naj bo T = {a,b} in V = {S,A,B}. Potem gramatika G s produkcijami S --> ϵ | aB | bA, A --> a | aS | bAA, B --> b | bS | aBB generira jezik L = {x | število simbolov a v x je enako številu simbolov b}.

Dokaz. Dokazali bomo nekoliko močnejši rezultat. Vpeljimo oznako $L_C = \{x \mid x \in T^*, C \Longrightarrow^* x\}$, $C \in V$ in zapišimo naslednje trditve:

1. $T_S : L_S = L$,

2. $T_A : L_A$ = {x | število simbolov a v x je za ena večje od števila simbolov b} in
3. $T_B : L_B$ = {x | število simbolov a v x je za ena manjša od števila simbolov b}.

Torej imamo tri trditve (o jezikih L_S, L_A in L_B), ki jih zaznamujemo z T_S, T_A in T_B. Sedaj vpeljemo trditve $T_C(n)$, $C \in V$, ki jih dobimo iz T_C, če naredimo presek leve in desne strani ustrezne enakosti z T^n, kjer je T množica končnih simbolov gramatike. Potem je T_C enakovredno

$$\forall n[T_C(n)]. \tag{4.3}$$

Dokažimo sedaj (4.3) z indukcijo po n. Pri n = 0 ugotovimo, da lahko iz S izpeljemo ϵ, iz A in B pa ne moremo izpeljati nobene besede dolžine 0. Ker drugih besed dolžine 0 razen ϵ ni, ugotavljamo da so $T_S(0)$, $T_A(0)$ in $T_B(0)$ resnične. Sedaj pa denimo, da so resnične trditve $T_S(n)$, $T_A(n)$ in $T_B(n)$. Dokažimo, da veljajo $T_S(n+1)$, $T_A(n+1)$ in $T_B(n+1)$. Potrebno je preveriti (1), da vse besede dolžine n + 1 jezikov L_S, L_A in L_B lahko izpeljemo iz ustreznih spremenljivk in (2), da vse besede dolžine n + 1, ki jih lahko izpeljemo iz spremenljivk S, A in B, pripadajo jezikom L_S, L_A in L_B (po vrsti).

Naj bo $|x| = n + 1$ in $x \in L_S$. Potem velja $x = ay$ ali $x = by$, $y \in T^*$ (x ne more biti enako ϵ, ker je $n \neq 0$). V prvem primeru mora biti $y \in L_B$, v drugem pa $y \in L_A$. Vsekakor pa velja $|y| = n$. Na podalgi induktivne predpostavke velja v prvem primeru $B \Rightarrow^* y$ in $A \Rightarrow^* y$ v drugem; torej lahko izpeljemo x iz S če v prvem primeru uporabimo najprej produkcijo $S \rightarrow aB$, v drugem primeru pa produkcijo $S \rightarrow bA$. Torej velja $S \Rightarrow^* x$ v obeh primerih. Podobno lahko sklepamo v primerih $|x| = n+1$, $x \in L_A$ oziroma $x \in L_B$.

Sedaj pa denimo, da velja $S \Rightarrow^* x$. Naj bo prva produkcija $S \rightarrow aB$ (podobno premišljujemo v primeru $S \rightarrow bA$). Torej je $x = ay$ in $|y| = n$. Na podlagi induktivne predpostavke imamo $y \in L_B$ in ima y za 1 več b-jev kot a-jev. Iz tega sklepamo $x \in L$. Sedaj bi morali podobno dokazati v primerih $A \Rightarrow^* x$ ali $B \Rightarrow^* x$. Podroben dokaz prepuščamo bralcu s pripombo, da edino produkciji $A \rightarrow bAA$ oziroma $B \rightarrow aBB$ zahtevata nekoliko skrbnejšo obravnavo. $|\bar{}|$

4.3 Drevesa izpeljav

V zvezi s kontekstno neodvisnimi gramatikami in jeziki je zelo koristno vpeljati pojem drevesa izpeljav. Namreč izpeljava neke besede ni popolnoma enolično določena, ker lahko s spremembo vrstnega reda uporabe produkcij, dobimo različne izpeljave, kar pa ni zaželeno.

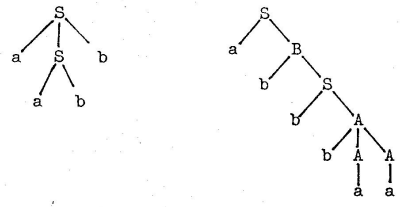
4.5. DEFINICIJA. Naj bo dana KNG $G = \langle V,T,P,S \rangle$. Potem bomo nekemu drevesu z vozlišči, zaznamovanimi z elementi $V \cup T$, rekli, da je <u>drevo izpeljav na podlagi</u> G, natanko ko velja naslednje

1. Vozlišča so zaznamovana z elementi $V \cup T$.
2. Če je neko notranje vozlišče zaznamovano z A in če so neposredni nasledniki zaznamovani z $X_1, X_2, \ldots, X_k$ (od leve proti desni), potem obstaja produkcija $A \rightarrow X_1 X_2 \ldots X_k$.
3. Koren je zaznamovan z S.

V primeru ko točko 3 spremenimo (ostali elementi definicije ostanejo nespremenjeni) v "Koren je zaznamovan z A" imamo opravka z A-<u>drevesom</u>. Besedo, ki jo dobimo tako, da odčitamo od leve proti desni oznake končnih vozlišč drevesa izpeljav (oz. A-drevesa) T poimenujemo <u>rezultat</u> T.

<u>Opomba</u>. Eno samo vozlišče, zaznamovano z A je drevo, ki ustreza definiciji A-drevesa.

Na sl. 4.1 vidimo dve drevesi izpeljav, ki pripadata gramatikam iz primerov 4.3 in 4.1.



sl. 4.1 Drevesi izpeljav za gramatiko iz primerov 4.3 in 4.1

Sedaj pa lahko zapišemo

4.6. IZREK. Naj bo dana KNG G = <V,T,P,S>. Potem velja $A \overset{*}{\Longrightarrow} alf$, $A \in V$, $alf \in \{V \mid\_\mid T\}^*$, natanko ko eksistira A-drevo z rezultatom alf.

Opomba. V primeru A = S in $alf = x \in T^*$ dobimo "$x \in L(G)$, natanko ko eksistira drevo izpeljav z rezultatom x".

Dokaz. Razpade na dva dela. Najprej denimo, da eksistira A-drevo z rezultatom alf. Trditev $A \overset{*}{\Longrightarrow} alf$ dokažemo z indukcijo po številu n notranjih vozlišč. V primeru n = 0 (drevo ima eno samo vozlišče, koren) je trditev očitna. V primeru n = 1 je edino notranje vozlišče koren in iz definicije A-drevesa sledi, da je koren označen z A, nasledniki korena pa z $alf = X_1 X_2 \ldots X_k$. Obstaja tudi produkcija $A \to X_1 X_2 \ldots X_k$ in torej $A \overset{*}{\Longrightarrow} alf$. Sedaj pa denimo, da smo trditev dokazali za

vsa števila manjša od n. Dokažimo, da velja za n. Imamo A-drevo z rezultatom alf. Koren je označen z A, njegovi nasledniki pa z $X_1 X_2 \ldots X_k$. Torej obstaja produkcija $A \to X_1 X_2 \ldots X_k$. Lahko tudi trdimo, da velja $alf = alf_1 alf_2 \ldots alf_k$, kjer je alf_i rezultat nekega X_i-drevesa. V slednjem primeru imamo na podlagi induktivne predpostavke izpeljavo $X_i \overset{*}{\Longrightarrow} alf_i$. Torej imamo naslednjo izpeljavo besede alf:

$$A \Longrightarrow X_1 X_2 \ldots X_k \overset{*}{\Longrightarrow} alf_1 X_2 \ldots X_k \overset{*}{\Longrightarrow} alf_1 alf_2 \ldots X_k \overset{*}{\Longrightarrow} \ldots \overset{*}{\Longrightarrow} alf_1 alf_2 \ldots alf_k \overset{*}{\Longrightarrow}$$

Sedaj pa, denimo, da velja nasprotno, $A \overset{*}{\Longrightarrow} alf$. Dokaz izpeljemo z indukcijo po dolžini n izpeljave. Primer n = 0 je očiten. Naj bo sedaj n = 1. Torej velja $A \Longrightarrow alf = X_1 X_2 \ldots X_k$. Zato eksistira produkcija $A \to X_1 X_2 \ldots X_k$ in obenem A-drevo z rezultatom alf (in enim notranjim vozliščem). Sedaj pa denimo, da trditev velja za vsa števila manjša od n in dokažimo, da velja za n. Naj bo prvi korak izpeljave $A \Longrightarrow X_1 X_2 \ldots X_k$. Potem velja $X_i \overset{*}{\Longrightarrow} alf_i$, $i = 1,2,\ldots,k$ pri določenih besedah $X_i \in \{V \mid\_\mid T\}^*$. Na podlagi induktivne predpostavke vemo da

eksistirajo X_i-drevesa z rezultatom alf$_i$ in torej lahko zgradimo A-drevo z rezultatom alf tako, da koren označimo z A, naslednike pa z $X_1, X_2, \ldots, X_k$, pri čemer so nasledniki koreni X_i dreves, ki eksistirajo na podlagi induktivne predpostavke. S tem je dokaz izreka končan. |_|

Sedaj se pa vprašajmo, kakšna je relacija med drevesom izpeljav, oziroma v splošnem A-drevesom, in izpeljavo? Očitno dobimo izpeljavo iz A-drevesa tako, da najprej "obiščemo" koren A-drevesa in nato na vsakem koraku obiščemo naslednike nekega že obiskanega vozlišča. Ob vsakem obisku izpišemo rezultat A-drevesa, ki ga sestavljajo obiskana vozlišča. To počenjamo dokler niso vsa vozlišča obiskana. Če v tem postopku vedno obiščemo skrajno levo vozlišče med vsemi možnimi, pravimo ustrezni izpeljavi skrajno leva izpeljava. Podobno definiramo skrajno desno izpeljavo. Očitno obstaja bijekcija med A-drevesi in skrajno levimi (oz. desnimi) izpeljavami.

S pojmom KNG je povezan še važen pojem dvoumnosti. Za neko KNG G pravimo da je dvoumna, če eksistira $x \in L(G)$ z lastnostjo, da obstajata dve različni drevesi izpeljav z rezultatom x. Za neki KN jezik L pa pravimo, da je dvoumen, če je vsaka KNG G z lastnostjo $L = L(G)$ dvoumna.

4.4 Poenostavljanje kontekstno neodvisnih gramatik

Iz različnih razlogov je zaželeno, da neko KNG G spravimo v nekakšno normalno (standardno) obliko, ki dovoljuje manj svobode pri izbiri produkcij kot splošna KNG, vendar obenem dovoljuje tudi generiranje vseh KN jezikov. Doslej se uporabljata predvsem dve normalni obliki za KNG. Prva je normalna oblika po Chomskemu. V tej normalni obliki so vse produkcije oblike A --> a ali pa A --> BC. A,B,C so seveda nekončni simboli, a pa je končen simbol. Druga je normalna oblika po Greibachovi. V drugi normalni obliki so vse produkcije oblike A --> aoalf, kjer je A nekončni simbol, a je končni simbol, alf pa je zaporedje, ki je sestavljeno iz nekončnih in končnih simbolov. Ostanek tega poglavja je posvečen pretvarjanju neke poljubne KNG G v obe normalni obliki. Obe normalni obliki imata eno samo zahtevo do jezika gramatike L(G), in sicer $\epsilon \notin L(G)$. V kolikor L(G) vsebuje prazno besedo ϵ, moramo obravnavati L(G) kot $L(G) = L' \,|\_|\, \{\epsilon\}$ pri čemer velja $\epsilon \notin L'$ in iskati normalne oblike le za gramatike za L'.

4.5 Normalna oblika po Chomskemu

Če je podana neka poljubna KNG G, jo spravimo v normalno obliko po Chomskemu z zaporedjem transformacij, ki ohranjajo L(G). Za uspešno pretvorbo v normalno obliko po Chomskemu moramo predvsem zagotoviti, da gramatika ne vsebuje produkcij oblike A --> ϵ in A --> B. Vendar je dobro pred tem tudi odpraviti vse nekončne simbole, ki ne igrajo nobene koristne

vloge. Kako pa definiramo koristne nekončne simbole? Za neki nekončni simbol A pravimo, da je koristen, če eksistirajo $w \in T^*$ in $alf_1, alf_2 \in (V \cup T)^*$ z lastnostjo $S \Rightarrow^* alf_1 A alf_2 \Rightarrow^* w$, torej če A dejansko nastopa v izpeljavi neke besede, ki je sestavljena iz končnih simbolov. Očitno je, da ,če odpravimo iz gramatike vse simbole, ki niso koristni (ki so nekoristni) in vse produkcije, kjer taki simboli nastopajo bodisi na levi ali na desni strani, bo tako dobljena gramatika generirala isti jezik kot prvotna. Odpravljanje nekoristnih simbolov bomo opisali z dvema lemama.

4.7. LEMA. Naj bo podana KNG G=<V,T,P,S> z lastnostjo $L(G) \neq \emptyset$. Potem obstaja ekvivalentna KNG G' =<V ,T,P',S> (torej L(G') = L(G)), pri kateri za vsak neskončen simbol $A \in V'$ eksistira $x \in T^*$ z lastnostjo $A \Rightarrow^* x$.

Dokaz. G' dobimo tako, da odpravimo določene elemente V in obenem odpravimo vse produkcije, ki imajo odpravljene simbole bodisi na levi ali desni strani. Algoritem za odpravljanje nekoristnih simbolov je prikazan na sl. 4.2. Kako pa se prepričamo, da res pravilno deluje? Naj velja $A \Rightarrow^* x$ pri $A \in V$. Na podlagi izreka 4.6 eksistira A-drevo T z rezultatom x. Sedaj definirajmo nivo nekega vozlišča na naslednji način:

1. Končna vozlišča imajo nivo 0.
2. Za notranje vozlišče A velja nivo (A) = max(nivo(X)) + 1, kjer gre X preko vseh naslednikov A.

Iz delovanja algoritma s sl. 4.2 se vidi, da se vozlišča na nivoju 1 postavijo v V' (oz. NEWV) pred vstopom v zanko. V splošnem se pa volzišča na nivoju k postavijo v V' najkasneje v k-1 izvajanju jedra zanke. Nasprotno, če velja $A \in V'$, potem denimo, da je algoritem postavil A v V' v k izvajanju jedra zanke k = 0,1,... Sedaj je možno dokazati z indukcijo po k, da velja $\exists x[A \Rightarrow^* x]$ (vaja). Morali bi še dokazati, da se algoritem ustavi, toda tudi to prepuščamo bralcu za vajo. S tem je dokaz končan. □

```
BEGIN
1.  oldv := ∅ ;
2.  newv := {A | ∃w ∈ T*,A --> w ∈ P[A ⇒* w}
3.  WHILE oldv ≠ newv DO
    BEGIN
4.    oldv := newv ;
5.    newv := oldv ∪
          {A | ∃alf ∈ (T ∪ oldv)* A --> alf ∈ P pri
alf ∈ (T ∪ oldv) }
    END ;
6.  V' := newv
END
```

sl. 4.2 Algoritem za iskanje spremenljivk, ki se lahko razvijejo v končne simbole

Na naslednjem koraku odpravimo nekončne simbole, ki se ne pojavljajo v nobeni besedi, ki jo lahko izpeljemo iz S.

4.8. LEMA. Naj bo dana KNG G = <V,T,P,S>. Potem obstaja ekvivalentna G' = <V',T,P',S> z lastnostjo

$$\forall A \in V' \exists |alf_1, alf_2 \in (V' |_| T)\ [S \Longrightarrow^* alf_1 A alf_2]$$

Dokaz. Postopek je podoben onemu, ki je opisan v dokazu leme 4.7. Torej iz množice V izločimo simbole, ki nimajo lastnosti iz trditve v lemi, obenem pa odpravimo produkcije, v katerih se ti simboli pojavljajo. Sedanjo nalogo lahko opravimo z algoritmom na sl. 4.2, ki vsebuje naslednje spremembe: (1) vrstica 2. postane

newv := {A | $\exists |alf_1, alf_2 \in (V |_| T)^*$ [$\exists |$produkcija $S \rightarrow alf_1 A alf_2$]}

in (2) vrstica 5. postane

newv := oldv |_| {A | $\exists |alf_1, alf_2 \in (V |_| T)^*$ $\exists |B \in$ oldv[B --> $alf_1 A alf_2$]}.

Dokaz pravilnosti algoritma je zelo podoben dokazu pravilnosti algoritma iz leme 4.7 in ga zato prepuščamo za vajo bralcu. |¯|

Torej imamo:

4.9. IZREK. Naj bo podana KNG G = <V,T,P,S> z lastnostjo L(G) ≠ Ø. Potem obstaja algoritem, ki poišče ekvivalentno G' = <V', T, P',S> z lastnostjo

$$\forall A \in V' \exists |w \in T^*, alf_1, alf_2 \in (V |_| T)^*$$

$$[(S \Longrightarrow^* alf_1 A alf_2) \text{ and } (alf_1 A alf_2 \Longrightarrow^* w)]$$

Dokaz. Najprej uporabimo algoritem iz dokaza leme 4.7, nato algoritem iz dokaza leme 4.8.

4.10. PRIMER. Vzemimo gramatiko

S --> AB | a
A --> a

Algoritem iz dokaza leme 4.7 nam odpravi nekončni simbol B, ker iz njega ne moremo izpeljati nobenega končnega zaporedja. Obenem izločimo produkcijo S --> AB, ki vsebuje B. Torej preostaneta produkciji

S --> a
A --> a

Sedaj pa nam algoritem iz dokaza leme 4.8 odpravi tudi A, saj se ne pojavlja v nobenem zaporedju, ki ga lahko izpeljemo iz S.

Bodimo pa pozorni na to, da omenjena dva algoritma ne moremo uporabljati v poljubnem vrstnem redu. Če v primeru 4.10 obrnemo vrstni red, nam drugi algoritem ne odpravi nobenega simbola, saj se tako A kot B pojavljata na desni strani produkcije S --> AB. Prvi algoritem odpravi B, vendar ker obenem odpravimo produkcijo S --> AB, postane simbol A nekoristen, saj kljub temu, da iz

njega lahko izpeljemo neko končno zaporedje, ne nastopa v zaporedju, ki ga je možno izpeljati iz S.

Odslej torej lahko predpostavljamo, da gramatika, ki nas zanima vse- buje le koristne simbole, kot to določa izrek 4.9.

Sedaj pa sledita dva koraka, ki ju opišemo z dvema lemama, ki nam pripravita gramatiko za pretvorbo v normalno obliko po Chomskemu.

4.11. LEMA. Podana je KNG G = <V,T,P,S> z lastnostjo $\epsilon \notin L(G)$. Potem eksistira ekvivalentna gramatika G' = <V',T,P',S>, ki ne vsebuje produkcij oblike A --> ϵ.

Dokaz. G' vsebuje vse produkcije G, razen tistih ki imajo obliko A --> ϵ. Poleg tega pa vsebuje dodatne produkcije, ki jih poiščemo takole: najprej poiščemo vse nekončne simbole $A \in V$ z lastnostjo $A \Longrightarrow^{*} \epsilon$. Za te simbole pravimo, da se dajo izbrisati. Simbole, ki se dajo izbrisati poiščemo s podobnim algoritmom, kot je opisan v lemah 4.7 in 4.8 (vaja). Sedaj pa denimo, da v P obstaja produkcija A --> alf in da alf vsebuje simbole $X_1, X_2, \ldots X_k$, ki se dajo izbrisati. Potem k P' dodamo produkcijo A --> alf kakor tudi vse dodatne produkcije oblike A --> alf', kjer alf' dobimo iz alf tako, da izbrišemo enega ali več simbolov X_i, i = 1,2,...,k, z omejitvijo, da alf' $\neq \epsilon$ (torej v primeru, ko alf vsebuje samo simbole, ki se dajo izbrisati, jih ne smemo izbrisati vseh). Sedaj moramo dokazati, da velja L(G) = L(G'). Naj $x \in L(G)$. Potem eksistira drevo T izpeljav z rezultatom x. V T poiščemo vsa maksimalna A-drevesa z

rezultatom ϵ oziroma $\epsilon\epsilon\ldots\epsilon = \epsilon^{k}$, pri nekem k. Če vsa taka drevesa skupaj s koreni izločimo, dobimo drevo izpeljav T' za katerega se brez težav prepričamo, da uporablja le produkcije iz P' . Nasprotno, naj velja $x \in L(G')$. Zopet imamo drevo izpeljav T', ki uporablja produkcije iz P'. Ni se težko prepričati, da lahko dobimo drevo T, ki uporablja produkcije iz P tako, da dodamo določena A-drevesa z rezultati ϵ^{k}, pri nekem k. S tem je dokaz končan. |_|

4.12. PRIMER. Naj bo dana gramatika s produkcijami

```
S --> A | AB | a
A --> CD | aB
B --> b
C --> ϵ | Ba
D --> ϵ | AB
```

Takoj ugotovimo, da se C in D dasta izbrisati neposredno. Posredno pa se da izbrisati še A zaradi produkcije A --> CD. Torej odpraviti moramo produkciji C --> ϵ in D --> ϵ ter še dodati produkcije

```
S --> A | B
A --> C | D
D --> B
```

Vendar je med njimi S --> A že prisotna v gramatiki.

Zadnji pripravljalni korak za spreminjanje KNG G v normalno obliko po Chomskemu je odpravljanje produkcij oblike A --> B.

4.13. LEMA. Podana je KNG G = <V,T,P,S>. Potem obstaja ekvivalentna gramatika G' = <V',T,P',S>, ki ne vsebuje produkcij oblike A --> B.

Dokaz. V prvi fazi poiščemo vse pare A,B z lastnostjo $A \overset{*}{\Longrightarrow} B$. Algoritem za to je kaj preprost: na prvem koraku poiščemo vse pare A,B za katere obstaja produkcija A --> B, oziroma, za katere velja $A \overset{*}{\Longrightarrow} B$. V i-tem koraku, i > 1, pa poiščemo i-to potenco $\overset{i}{\Longrightarrow}$ relacije $\Longrightarrow$, pri čemer seveda $\overset{i}{\Longrightarrow}$ definiramo kot

1. $A \overset{0}{\Longrightarrow} B$ natanko ko A = B,
2. $A \overset{i}{\Longrightarrow} B$ natanko ko $\exists C \in V[(A \Longrightarrow C)$ and $(C \overset{i-1}{\Longrightarrow} B)]$

Korakov je seveda potrebno narediti največ toliko kot je nekončnih simbolov.

V drugi fazi pa spremenimo produkcije. Najprej odpravimo vse pro- dukcije oblike A --> B in nato za vsako produkcijo B --> alf dodamo vse produkcije A --> alf, če velja $A \overset{*}{\Longrightarrow} B$.

Dokaz, da opisani postopek resnično poišče ekvivalentno gramatiko, je preprost. Zopet razpade na dva dela: iz hipoteze $x \in L(G)$ moramo izpeljati $x \in L(G')$ in nasprotno, iz $x \in L(G')$ mora slediti $x \in L(G)$. Noben od teh dveh dokazov ne bi več smel delati bralcu težav in zato mu jih prepuščamo za vajo. □

4.14. PRIMER. Na koncu primera 4.12 smo dobili gramatiko s produkcijami

S --> A | B | AB | a
A --> C | D | aB | CD
B --> b
C --> Ba
D --> B | AB

Z upoštevanjem relacije $\overset{1}{\Longrightarrow}$ ugotovimo da velja $S \overset{*}{\Longrightarrow} A$, $S \overset{*}{\Longrightarrow} B$, $A \overset{*}{\Longrightarrow} C$, $A \overset{*}{\Longrightarrow} D$, $D \overset{*}{\Longrightarrow} B$ in nato še na podlagi $\overset{2}{\Longrightarrow}$ dobimo $S \overset{*}{\Longrightarrow} C$, $S \overset{*}{\Longrightarrow} D$, $A \overset{*}{\Longrightarrow} B$.

Kot rezultat dobimo naslednji seznam produkcij

S --> a | b | aB | Ba | AB | CD
A --> b | aB | Ba | AB | CD
B --> b
C --> Ba
D --> b | AB

Sedaj se pa lahko lotimo spreminjanja kontekstno neodvisne gramatike v normalno obliko po Chomskemu.

4.15. IZREK. Naj bo G = <V,T,P,S> KNG z lastnostmi $L(G) \neq \emptyset$, $\epsilon \notin L(G)$. potem obstaja ekvivalentna KNG G' = <V',T, P', S>, kjer so vsi nekončni simboli koristni, in kjer imajo vse produkcije obliko A --> BC ali A --> a (normalna oblika po Chomskemu).

VRSTVI RED!

Dokaz. Nekoristne nekončne simbole odpravimo z uporabo izreka 4.9, nato pa odpravimo produkcije oblik A --> Є in A --> B z uporabo lem 4.11 in 4.13. Torej ostanejo dve vrsti produkcij: take ki imajo obliko A --> a in druge, ki imajo obliko A --> alf pri alf ≠ a ter alf Є $(V \cup T)^{+1}$. Prve so že v eni od dovoljenih oblik in jih lahko postavimo v P'. Sedaj pa vsako produkcijo oblike A --> alf, alf ≠ a, nadomestimo takole: Najprej za vse končne simbole a, ki se pojavljajo v alf, vpeljemo nekončne simbole C_a in produkcije C_a --> a (seveda, v kolikor smo tak simbol in produkcijo vpeljali že pri obravnavi neke druge produkcije B --> bta, ga sedaj ni treba) nato v alf vse končne simbole a zamenjamo z nekončnimi C_a in tako dobimo besedo alf' sestavljeno iz samih nekončnih simbolov. Očitno je, da produkcije A --> alf' skupaj s produkcijami C_a --> a generirajo isti jezik L(G). Sedaj smo dobili gramatiko, ki vsebuje produkcije oblik A --> a ter A --> alf, alf Є V^{+}. V tej novi gramatiki vsako produkcijo A --> alf, alf ≠ a, nadomestimo takole: Naj bo alf = $A_1 A_2 \ldots A_k$. Potem vpeljemo nove nekončne simbole $[A_2 A_3 \ldots A_k]$, $[A_3 \ldots A_k]$, ..., $[A_{k-1} A_k]$ (seveda v kolikor jih nismo že prej vpeljali) in A --> alf nadomestimo s produkcijami

---

1 $A^{+} = A^{*} - \{Є\}$.

$A \to A [A_2 A_3 \ldots A_k]$

$[A_2 A_3 \ldots A_k] \to A [A_2 A_3 \ldots A_k]$

.
.
.

$[A_{k-1} A_k] \to A A_{k-1} A_k$.

Bralca posebno opozarjamo na to, da predstavlja vsako zaporedje znotraj oklepajev en sam nekončen simbol. Sedaj so vse produkukcije v obliki, ki jo zahteva normalna oblika po Chomskemu, in očitno je, da gramatika, ki smo jo tako dobili generira isti jezik, saj so vse transformacije produkcij, ki smo jih izvajali, imele za edino posledico to, da smo en korak v izpeljavah s prvotno gramatiko nadomestili z več koraki v izpeljavah z novo gramatiko. S tem je dokaz končan. |¯|

4.16. PRIMER. Podana je gramatika s produkcijami

S --> bA | aB
A --> bAA | aS | a
B --> aBB | bS | b

gramatika je že brez nekoristnih simbolov, kakor tudi brez produkcij oblike A --> Є oziroma A --> B. Ko vpeljemo nova nekončna simbola C_a in C_b dobimo naslednji nabor produkcij:

S --> C_b A | C_a B

A --> C AA | C S | a
b a
B --> C BB | C S | b
a b
C --> a
C --> b

Sedaj pa vpeljemo nekončna simbola [AA] in [BB] in končno dobimo produkcije

S --> C A | C B
b a
A --> C [AA] | C S | a
b a
B --> C [BB] | C S | b
a b
C --> a
a
C --> b
b
[AA] --> AA
[BB] --> BB

v zahtevani obliki, ki generirajo isti jezik.

4.6 Normalna oblika po Greibachovi

Normalno obliko po Greibachovi prav tako dobimo z določenimi transformacijami produkcij, ki ohranjajo L(G). Kot osnovo uporabljamo dve transforciji, ki bosta opisani z lemama 4.17 in 4.18.

Naj bo G = <V,T,P,S> neka KNG in naj bo A neki nekončen simbol. Potem pravimo produkcijam, ki imajo A na levi strani, A-produkcije. Sedaj imamo:

4.17. LEMA. G = <V,T,P,S> je neka KNG. Naj bo $A \to \mathrm{alf}_1 B \mathrm{alf}_2$ neka A-produkcija in naj bo $B \to \mathrm{bta}_1 \mid \mathrm{bta}_2 \mid \ldots \mid \mathrm{bta}_r$ množica B-produkcij. Če $A \to \mathrm{alf}_1 B \mathrm{alf}_2$ zamenjamo z množico produkcij $A \to \mathrm{alf}_1 \mathrm{bta}_i \mathrm{alf}_2$, $i = 1,2,\ldots,r$, dobimo ekvivalentno gramatiko.

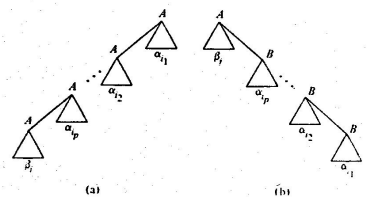
Dokaz. V nekem drevesu izpelav gramatike G ima vsako A-vozlišče, ki ustreza $A \to \mathrm{alf}_1 B \mathrm{alf}_2$, za enega od naslednikov neko B-vozlišče, ki ustreza $B \to \mathrm{bta}_i$, pri nekem i. Učinek opisane transformacije je združitev obeh vozlišč v eno samo.

|¯|

4.18. LEMA. G = <V,T,P,S> je neka KNG. Naj bo $A \to \mathrm{Aalf}_1 \mid \mathrm{Aalf}_2 \mid \ldots \mid \mathrm{Aalf}_r$ množica A-produkcij, katerih desna stran se začenja z A. Naj bo $A \to \mathrm{bta}_1 \mid \mathrm{bta}_2 \mid \ldots \mid \mathrm{bta}_s$ množica preostalih A-produkcij. Če vpeljemo dodaten nekončen simbol B in množico $A \to \mathrm{Aalf}_i$, $1 \leq i \leq r$, zamenjamo s produkcijami $A \to \mathrm{bta}_j B$, $1 \leq j \leq s$, ter $B \to \mathrm{alf}_i$,

B --> alf B, $1 \leq i \leq r$, dobimo ekvivalentno gramatiko.
 i

Dokaz. Trditev zopet dokažemo s premislekom, ki sloni na drevesih izpeljav. Drevo (oziroma poddrevo) izpeljav, ki uporablja A-produkcije prvotne gramatike, je prikazano na sl. 4.3a. Pravzaprav imamo opravka z A-drevesom z rezultatom bta alf alf ...alf alf .
 j i(k) i(k-1) i(2) i(1)



sl. 4.3 Dve poddrevesi izpeljav.

Na sl. 4.3b pa vidimo A-drevo z istim rezultatom, vendar je zgrajeno na podlagi spremenjene gramatike. Ni se težko prepričati tudi v nasprotno dejstvo: poljubno A-drevo, ki sloni na novi gramatiki ustreza nekemu A-drevesu prvotne gramatike z istim rezultatom. Dokaz je končan. |_|

Sedaj se pa lahko lotimo spreminjanja gramatike v normalno obliko po Greibachovi.

4.19. IZREK. Vsak KNJ L z lastnostjo $\epsilon \notin L$ in $L \neq \emptyset$ lahko generiramo z gramatiko, kjer so vse produkcije oblike

A --> aalf, pri čemer je A nekončen simbol, a je končen simbol, alf pa je beseda (morebiti prazna), ki je sestavljena iz ~~končnih~~ nekončnih simbolov.

Dokaz: Naj bo L = L(G) pri neki KNG G = <V,T,P,S>. Na podlagi izreka 4.15 lahko predpostavimo, da je G v normalni obliki po Chomskemu. Vse nekončne simbole oštevilčemo tako, da velja $V = \{A_1, A_2, \ldots, A_m\}$, vpeljemo pa tudi nove spremenljivke $B_1, B_2, \ldots, B_m$. Sedaj v prvem koraku pretvorbe spremenimo produkcije z algoritmom na sl. 4.4, ki ga z besedami opišemo takole: pri vseh A_i-produkcijah, $1 \leq i \leq m$, zagotovimo obliko

A_i --> a (že v predpisani obliki) ali A_i --> A_j alf, pri $i < j$.

Denimo, da smo nalogo že opravili za vse nekončne simbole A_k, pri $k < i$. Pokažimo, kako nalogo opravimo še za A_i. Vse A_i-produkcije, ki imajo obliko A_i --> A_j alf, lahko razvrstimo v tri skupine, in sicer $i < j$, $i = j$ in $i > j$. Prva skupina je že v želeni obliki, tretjo pa lahko postopno spravimo v drugo obliko. Namreč naj bo A_i --> A_1 alf neka produkcija, katere desna stran začenja z A_1. Če uporabimo lemo 4.17 in induktivno predpostavko, jo lahko nadomestimo z vrsto produkcij, pri katerih velja v primeru A_i --> A_j bta $j \geq 2$. Če ta postopek ponavljamo, dobimo končno pri produkcijah oblike A_i --> A_j alf i

= j (ali pa i < j). Nato produkcije oblike A_i --> A_i alf nadomestimo tako kot predpisuje lema 4.18.

V drugem koraku pretvorbe spravimo A_i-produkcije v zahtevano obliko. Zaradi lastnosti i < j pri produkcijah oblike A_i --> A_j alf, imamo pri i = m dve možnosti. Prva je, da je A_m-produkcija v obliki A_m --> a, druga pa, da je v obliki A_m --> B_k bta. Vendar druga možnost pravzaprav ne more nastopiti, ker so vse A_m produkcije, kjer nastopa B_k na desni, pravzaprav oblike A_m --> btaB$_k$, pri čemer je bta neprazna beseda (gl. sl. 4.4), vrstici 15 in 17), kajti bta izvira iz desne strani neke prvotne produkcije, ki pa ima normalno obliko po Chomskemu. Torej imamo le produkcije oblike A_m --> a, ki pa so že v predpisani obliki. Glede A_{m-1}-produkcij ugotovimo, da desna stran prav tako ne more začeti z B spremenljivko. Torej se začne bodisi s končnim simbolom ali pa s simbolom A_m. V slednjem primeru dobimo končni simbol na začetku desne strani z uporabo leme 4.17. Ta postopek nadaljujemo dokler se pri vseh A_i-produkcijah desne strani ne začenjajo s končnimi simboli. V tretjem koraku pa spremenimo B_k-produkcije. Pri B_k-produkcijah, podobno kot prej, ugotovimo, da se ne morejo začeti z B simbolom, kajti vse te produkcije so v obliki B_k --> alf ali

B_k --> alfB_k (sl. 4.19, vrstica 12) in je alf neprazno zato, ker je prvotna gramatika v normalni obliki po Chomskemu. Torej če še enkrat uporabimo lemo 4.17, dobimo pri vseh produkcijah desne strani, ki se začenjajo s končnim simbolom. Dokaz je končan. |_|

```
    BEGIN
 1.   FOR k:=1 TO m DO
 2.   BEGIN
 3.     FOR j:=1 TO k-1 DO
 4.       FOR vse produkcije v obliki A  --> A alf DO
                                       k      j
 5.       BEGIN
 6.         FOR vse produkcije A  --> bta DO
                               j
 7.           dodaj produkcijo A  --> btaalf ;
                              K
 8.         odpravi produkcijo A  --> A alf
                               k      j
 9.       END ;
10.       FOR vse produkcije v obliki A  --> A  DO
                                       k      k
11.       BEGIN
12.         dodaj produkcije B  --> alf ter B  --> alfB ;
                             k                k          k
13.         odpravi produkcijo A  --> A alf
                               k      k
14.       END ;
15.       FOR vse produkcije v obliki A  --> bta, kjer
                                       k
16.          se bta ne pričenja z A
                                  k
17.       DO dodaj produkcijo A  --> btaB
                               k         k
18.   END
    END
```

sl. 4.4 Predelava produkcij pri pretvarjanju v normalno obliko po Greibachovi

4.20. PRIMER. V normalno obliko po Greibachovi bomo spremenili gramatiko

$G = \langle \{A_1, A_2, A_3\}, \{a,b\}, P, A_1 \rangle$

pri čemer P vsebuje naslednje produkcije:

$A_1 \rightarrow A_2 A_3$
$A_2 \rightarrow A_3 A_1 \mid b$
$A_3 \rightarrow A_1 A_2 \mid a$

Prvi korak. Desna stran A_1- in A_2-produkcij se že začenja bodisi s končnim simbolom ali pa z višje oštevilčeno spremenljivko. Torej je potrebno zamenjati le A_3-produkcijo $A_3 \rightarrow A_1 A_2$. V tem primeru zamenjamo A_1 na desni strani z desno stranjo produkcije $A_1 \rightarrow A_2 A_3$ in tako dobimo novo množico produkcij:

$A_1 \rightarrow A_2 A_3$
$A_2 \rightarrow A_3 A_1 \mid b$
$A_3 \rightarrow A_2 A_3 A_2 \mid a$

Desna stran produkcije $A_3 \rightarrow A_2 A_3 A_2$ se zopet začenja z nekončnim simbolom z indeksom, ki je manjši od 3, zato nadaljujemo z istim postopkom. Sedaj moramo A_2 nadomesili z vsemi možnimi desnimi stranmi A_2-produkcij. Tako dobimo:

$A_1 \rightarrow A_2 A_3$
$A_2 \rightarrow A_3 A_1 \mid b$

$A_3 \rightarrow A_2 A_3 A_2 A_3 \mid b A_3 A_2 \mid a A_2$

Sedaj pa uporabimo lemo 4.18 na produkcijah

$A_3 \rightarrow A_2 A_3 A_2 A_3 \mid b A_3 A_2 \mid a A_2$

Potrebno je vpeljati simbol B_3 ter produkcijo $A_3 \rightarrow A_2 A_3 A_2 A_3$ zamenjati z $A_3 \rightarrow b A_3 A_2 B \mid a B$, $B \rightarrow A_2 A_3 A_2 A_3 \mid A_2 A_3 A_2 B$.

Celotna množica produkcij postane tako:

$A_1 \rightarrow A_2 A_3$
$A_2 \rightarrow A_3 A_1 \mid b$
$A_3 \rightarrow b A_3 A_2 B \mid a B \mid b A_3 A_2 \mid a$
$B \rightarrow A_2 A_3 A_2 A_3 \mid A_2 A_3 A_2 B$

Drugi korak. Sedaj se vse desne strani A_3-produkcij pričenjajo s končnimi simboli. Na podlagi leme 4.17 lahko spremenimo A_2-produkcije v to obliko in v naslednjem koraku še A_1-produkcijo. Kot posledica dobimo naslednjo množico produkcij:

$A_1 \rightarrow a \mid a B \mid b A_3 A_2 \mid b A_3 A_2 B$
$A_2 \rightarrow a A_3 \mid a B A_3 \mid b \mid b A_3 A_2 A_3 \mid b A_3 A_2 B A_3$
$A_3 \rightarrow a A_3 A_2 A_3 \mid a B A_3 A_2 A_3 \mid b A_3 \mid b A_3 A_2 A_3 A_2 A_3 \mid b A_3 A_2 B A_2 A_3$
$B \rightarrow A_2 A_3 A_2 A_3 \mid A_2 A_3 A_2 B$

Tretji korak. Sedaj spremenimo še dve B-produkciji v zahtevano obliko, kar da 10 dodatnih produkcij. Vsako od dveh B_3-produkcij zamenjamo tako, da A_1 na začetku desnih strani zamenjamo z vsemi možnimi desnimi stranmi A_1-produkcij. Tako na

primer B --> A A A postane:
3 1 3 2

B --> aA A A A | aB A A A A | bA A A A A A |
3 1 3 3 2 3 1 3 3 2 3 2 1 3 3 2
bA A B A A A A. | bA A A
3 2 3 1 3 3 2 3 3 2

Na podoben način zamenjamo drugo B -produkcijo. Končno
3

dobimo naslednji nabor produkcij:

A --> aA A | aB A A | bA | bA A A A | bA A B A A
1 1 3 3 1 3 3 2 1 3 3 2 3 1 3
b
A --> aA | aB A | b A A A | bA A A A
2 1 3 1 3 2 1 3 2 3 1
A --> a | aB | bA A | bA A A
3 3 3 2 3 2 3
B --> aA A A A | aA A A A A | aB A A A A |
3 1 3 3 2 1 3 3 2 3 3 1 3 3 2
aB A A A A B | bA A A A A A |
3 1 3 3 2 3 3 2 1 3 3 2
bA A A A A A A | bA A A | bA A A B
3 2 1 3 3 2 3 3 3 2 3 3 2 3

4.7 Naloge

4.1 Poiščite kontekstno neodvisne gramatike, ki generirajo naslednje množice.

R
1. L = {x | x = x }
2. Množica nizov iz pravilno vgnezdenih oklepajev (tak niz dobimo, če vzmemo poljuben aritmetični izraz in odstranimo vse razen oklepajev);
3. Množica nizov nad {a,b}, ki vsebujejo natanko dvakrat več a-jev kot b-jev;
4. Množica nizov nad {a,b, o ,+,*,),(,∈,∅}, ki predstavljajo pravilne regularne izraze nad {a,b}. Upoštevaj dejstvo, da so vsi simboli v opisani abecede znaki in ne imena za operacije oziroma posebne množice;
5. Množica nizov nad {a,b}, ki nimajo obliko ww, za neki niz w;
i j k
6. {a b c | i ≠ j or j ≠ k}.

4.2 Naj bo G gramatika

S --> aS | aSbS | ∈.

Dokaži da velja

L(G) = {x | vsak začetek x ima več ali enako b-jev kot a-jev}.

4.3 Pri vsakem $i \geq 1$ naj bo b_i niz iz $1(0+1)^*$, ki je dvojiška predstavitev i. Poišči KNG, ki generira

$$\_{\{0,1,\#\}}^{+} - \{b_1 \# b_2 \# \ldots \# b_n \mid n \geq 1\}.$$

4.4 Poišči KNG, ki generira množico

$$\{w\#w^R \# \mid w \in (0+1)^*\}.$$

4.5 Gramatika

E --> E+E | E*E | (E) | ime

generira množico aritmetičnih izrazov, ki vsebujejo +, * ter ime. Opisana gramatika je dvoumna, ker lahko ime+ime*ime generiramo z dvema različnima skrajno levima izpeljavama.

1. Poišči enakovredno nedvoumno gramatiko;
2. Poišči nedvoumno gramatiko za vse aritmetične izraze brez odvečnih oklepajev. Par oklepajev je odvečen v primeru, ko ima brez njega izraz isti pomen. N.pr. v izrazu ime+(ime*ime) sta oklepaja odvečna, v izrazu (ime+ime)*ime pa ne.

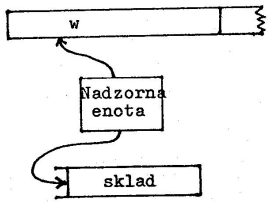
5. POGLAVJE

SKLADOVNI AVTOMATI

5.1 Neformalen opis

V tem poglavju bomo opisali vrsto avtomata, ki sprejema natanko kontekstno neodvisne jezike.

Skladovni avtomat (SA) si predstavljamo tako (gl. sl. 5.1), kot da je vhodna beseda zapisana na nekem traku, ki je neskončen v desno. Vhodni trak ima to lastnost, da z njega lahko le beremo, ne moremo pa ga uporabljati za zapisovanje simbolov. Bralna glava se giblje od leve proti desni, lahko pa tudi zastane in v tem primeru nadzorna enota predeluje podatke ne, da bi brala vhodnih simbolov. Poleg vhodnega traku obstaja še delovni trak, ki je neskončen v levo, dovoljuje pa zapisovanje in branje podatkov na poseben način: Simbol zapišemo tako, da bralno glavo premaknemo v levo in nato zapišemo želeni simbol, beremo pa tako, da simbol preberemo s celice, na kateri bralna glave trenutno stoji, nato pa simbol izbrišemo s traku in glavo premaknemo za eno polje v desno.



sl. 5.1 Skladovni avtomat

V splošnem deluje skladovni avtomat nedeterministično. Skladovni avtomat vhodno besedo sprejme na en od dveh načinov (kako pa deluje v nekem konkretnem primeru, mora biti v naprej dogovorjeno). Prvi način je, da je vhodna beseda sprejeta, ko se, v trenutku, ko bralna glava na vhodnem traku prestopi desno mejo vhodne besede, nadzorna enota nahaja v nekem končnem stanju. Ta način delovanja torej spominja na delovanje končnih avtomatov. Pri drugem načinu delovanja pa je vhodna beseda sprejeta, ko je v opisanem trenutku sklad prazen. Torej ne glede na stanje nadzorne enote.

5.2 Definicije

Torej formalno lahko definiramo skladovni avtomat takole:

5.1. DEFINICIJA. Skladovni avtomat (SA) je sedmerka $M = \langle Q, SIGMA, GAMA, q_0, Z_0, F, dlt \rangle$, pri čemer je Q množica stanj, SIGMA

je vhodna abeceda, GAMA je skladovna abeceda, $q_0 \in Q$ je začetno stanje, $Z \in GAMA$ je začetni skladovni simbol, $F \subseteq Q$ je množica končnih stanj in dlt je korespondenca

$$Q \times (SIGMA \cup \{\epsilon\}) \times GAMA \rightarrow Q \times GAMA^*$$

Stanje računskega postopka skladovnega avtomata je popolnoma določeno s trenutnim stanjem, trenutnim položajem vhodne bralne glave oziroma preostalim delom vhodne besede in trenutno vsebino sklada. Z drugimi besedami stanje računskega postopka je podano z elementom množice $Q \times SIGMA^* \times GAMA^*$, ki jo zaznamujemo s TO (množica trenutnih opisov). Sedaj bomo pa na tej množici definirali relacijo "neposredno daje", ki jo označujemo z $\vdash_M$:

5.2. DEFINICIJA. Naj bo M skladovni avtomat, <q,u,alf> in <r,v,bta> pa dva trenutna opisa. Potem velja $\langle q,u,alf\rangle \vdash_M \langle r,v,bta\rangle$ natanko v naslednjem primeru:

1. $u = av$, $a \in SIGMA$, $alf = Zgma_2$, $Z \in GAMA$, $bta = gma_1 ogma_2$, $gma_1, gma_2 \in GAMA^*$ in $\langle r, gma_1 \rangle \in dlt(q,a,Z)$ ali
2. $u = v$, $alf = Zgma_2$, $Z \in GAMA$, $bta = gma_1 ogma_2$, $gma_1, gma_2 \in GAMA^*$ in $\langle r, gma_1 \rangle \in dlt(q,\epsilon,Z)$

Tako kot v drugih podobnih primerih ima relacija $a \vdash_M b$ pomen, da je možno iz trenutnega opisa a dobiti opis b z enim korakom stroja M.

Na podlagi relacije $\vdash_M$ lahko definiramo njeno tranzitivno ovojnico $\vdash_M^*$, kar beremo "daje".

Sedaj pa lahko definiramo jezik, ki ga sprejema skladovni avtomat, in sicer na oba načina, ki smo ju prej omenili.

5.3. DEFINICIJA. (Jezik skladovnega avtomata po kriteriju končnega stanja). Jezik skladovnega avtomata $M = \langle Q,SIGMA,GAMA,q_0,Z_0,F,dlt\rangle$ definiramo kot

$$L(M) = \{x \mid \langle q_0,x,Z_0\rangle \vdash_M^* \langle q,\epsilon,alf\rangle,\ q \in F, alf \in GAMA^*\}$$

5.4. DEFINICIJA. (Jezik skladovnega avtomata po kriteriju praznega sklada). Jezik skladovnega avtomata $M = \langle Q,SIGMA,GAMA,q_0,Z_0,F,dlt\rangle$ definiramo kot

$$L(M)) = \{x \mid \langle q_0,x,Z_0\rangle \vdash_M^* \langle q,\epsilon,\epsilon\rangle,\ q \in Q\}$$

5.5. PRIMER. Na sl. 5.2 je prikazan formalen skladovni avtomat, ki sprejema jezik $\{wcw^R \mid w \in (0+1)^*\}$ na podlagi

praznega sklada. Bodi pozoren na to, da pri korakih, ko avtomat vpisuje v sklad ima dlt vrednost <q,gma> pri |gma| = 2. Na primer $dlt(q_1,0,R) = \{<q_1,BR>\}$. Pri |gma| = 1 bi prišlo preprosto do zamenjave vrhnjega simbola sklada in se dolžina sklada ne bi povečala. Tako lahko skrajšamo sklad, če postavimo gma = ϵ.

Pravilo $dlt(q_2,\epsilon,R) = \{<q_2,\epsilon>\}$ pomeni, da lahko skladovni avtomat v stanju q_2 in vrhnjim skladovnim simbolom enakim R, izbriše R neodvisno od vhodnega simbola. V tem primeru se bralna glava ne premakne in pravzaprav je lahko ostanek vhodnega zaporedja prazen.

$M = (\{q_1, q_2\}, \{0, 1, c\}, \{R, B, G\}, \delta, q_1, R, \varnothing)$

$\delta(q_1, 0, R) = \{(q_1, BR)\}$ $\quad$ $\delta(q_1, 1, R) = \{(q_1, GR)\}$

$\delta(q_1, 0, B) = \{(q_1, BB)\}$ $\quad$ $\delta(q_1, 1, B) = \{(q_1, GB)\}$

$\delta(q_1, 0, G) = \{(q_1, BG)\}$ $\quad$ $\delta(q_1, 1, G) = \{(q_1, GG)\}$

$\delta(q_1, c, R) = \{(q_2, R)\}$

$\delta(q_1, c, B) = \{(q_2, B)\}$

$\delta(q_1, c, G) = \{(q_2, G)\}$

$\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$ $\quad$ $\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$

$\delta(q_2, c, R) = \{(q_2, \epsilon)\}$

sl. 5.2 Formalen skladovni avtomat, ki sprejema $\{wcw^R \mid w \in (0+1)^*\}$ na podlagi praznega sklada.

5.6. PRIMER. Sl. 5.3 prikazuje skladovni avtomat za

jezik $\{ww^R \mid w \in (0+1)^*\}$. Pravila (1) do (6) omogočajo M, da vpisuje v sklad. Pri pravilih (3) ter (6) ima M izbirmo med dvema prehodoma. Ena odločitev (druga izbira) ustreza sredini vhoda: v tem primeru je naslednje stanje q_2 in v nadaljevanju izračuna se opravlja primerjava med vsebino sklada ter preostankom vhoda. V kolikor je bila odločitev pravilna in v kolikor ima vhod obliko ww^R, se bo sklad izpraznil in M sprejme vhodno zaporedje.

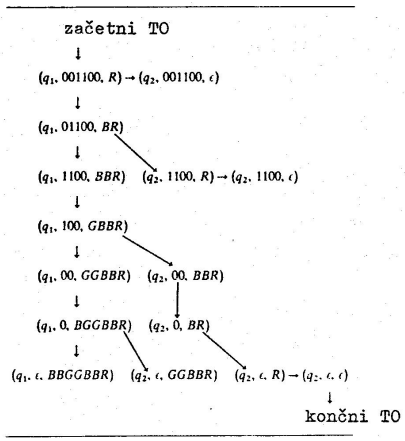
Podobno kot nedeterminističen končni avtomat, nedeterminističen skladovni avtomat sprejme svoj vhod v primeru, ko obstaja neko zaporedje odločitev, ki izprazni sklad. Na ta način se M vedno "pravilno odloča", kajti nepravilne odločitve, same po sebi, ne povzročijo, da se vhod odkloni. Neki vhod se odkloni (se ne sprejme) le v primeru, ko "pravilnih odločitev" ni. Sl. 5.4 prikazuje dosegljive trenutne opise stroja M, ko slednji računa na vhodu 001100.

$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G\}, \delta, q_1, R, \emptyset)$

1) $\delta(q_1, 0, R) = \{(q_1, BR)\}$
2) $\delta(q_1, 1, R) = \{(q_1, GR)\}$
3) $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \epsilon)\}$
4) $\delta(q_1, 0, G) = \{(q_1, BG)\}$
5) $\delta(q_1, 1, B) = \{(q_1, GB)\}$
6) $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \epsilon)\}$
7) $\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$
8) $\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$
9) $\delta(q_1, \epsilon, R) = \{(q_2, \epsilon)\}$
10) $\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$

sl. 5.3 Nedeterminističen skladovni avtomat, ki sprejema

$$\{ww^R \mid w \in (0+1)^*\}$$ na podlagi praznega sklada.



sl. 5.4 Dosegljivi TO skladovnega avtomata na sl. 5.3 pri vhodu 001100.

Deterministični skladovni avtomati

Skladovni avtomat iz primera 5.5 je deterministične v tem pomenu, da je iz poljubnega TO neposredno dosegljiv kvečjemu en TO. Formalno povedano, za skladovni avtomat M = $\langle Q, SIGMA, GAMA, dlt, q_0, Z_0, F\rangle$ pravimo, da je deterministične (DSA) v primeru, ko velja

(1) za vse $q \in Q$ in $Z \in GAMA$ velja, da iz $dlt(q, \epsilon, Z)$ neprazno sledi $dlt(q, a, Z)$ je prazno pri vseh $a \in SIGMA$;

(2) $dlt(q, a, Z)$ vsebuje največ en element pri vseh $q \in Q$, $a \in SIGMA \cup \{\epsilon\}$ in $Z \in GAMA$.

Pogoj 1 onemogoča izbiro med prehodom brez vhoda (ϵ prehodom) in prehodom z vhodom. Pogoj 2 pa onemogoča izbiro pri poljubni trojki $\langle q, a, Z\rangle$ ali $\langle q, \epsilon, Z\rangle$. Bodimo pozorni na to, da pri skladovnem avtomatu predpostavljamo, da je nedeterminističen, razen, če ni izrecno določeno drugače.

Pri končem avtomatu sta bila deterministična in nedeterministična modela ekvivalentna glede na razred sprejemanih jezikov. Isto pa ne velja za skladovne avtomate. Na primer, ww^R je jezik nedeterminističnega skladovnega avtomata, ni pa jezik nobenega determinističnega skladovnega avtomata.

5.3 SA in kontekstno neodvisni jeziki

Oba kriterija sprejemanja besed (definiciji 5.3 in 5.4) sta pravzaprav enakovredna v naslednjem pomenu:

5.7. IZREK. Podan je skladovni avtomat M = <Q,SIGMA,GAMA,q_0,Z_0,F,dlt>. Naj bosta L' in L" jezika avtomata M na podlagi definicij 5.3 in 5.4, po vrsti. Potem obstaja skladovni avtomat M', ki sprejema L' po kriteriju praznega sklada, in M", ki sprejema L" po kriteriju končnega stanja.

Dokaz. M' definiramo na podlagi definicije M tako, da dodamo ustrezne elemente, ki izpraznijo sklad po tem, ko M pride v končno stanje.

$$M' = \langle Q \mid\_\mid \{q'\}, SIGMA, GAMA, q_0, Z_0, \emptyset, dlt' \rangle$$

pri čemer velja $q' \notin Q$ in $dlt' = dlt \mid\_\mid dlt_1$ ter

$$dlt_1(q,a,V) = \begin{cases} \langle q', \epsilon \rangle \text{ pri vseh } q \in F, a = \epsilon, V \in GAMA \\ \langle q, \epsilon \rangle \text{ pri } q = q' \\ \text{nedefiniran sicer} \end{cases}$$

Posledica novega stanja q' in novih prehodov je, da v trenutku, ko bralna glava prestopi desni rob vhodne besede in v primeru ko je stanje končno, imamo na voljo prehod v novo stanje q', v katerem lahko izpraznimo sklad. Poudariti velja, da nas ta izbira pripelje v "slepo ulico", če jo naredimo prezgodaj,

torej pred prestopom desnega roba vhodne besede.

M" pa definiramo tako, da vpeljemo možnost ugotavljanja trenutka, ko M izprazni sklad, in prehoda v končno stanje.

$$M'' = \langle Q \mid\_\mid \{q'',q''\}, SIGMA, GAMA \mid\_\mid \{Z\}, q'', Z, \{q''\}, dlt'' \rangle$$

dlt" dobimo is dlt tako, da dodamo naslednje prehode: $\langle q_0, Z_0 Z \rangle \in dlt(q'', \epsilon, Z)$, $\langle q'', \epsilon \rangle \in dlt(q, \epsilon, Z)$ pri vseh $q \in Q$.

Ugotovimo lahko, da M" uporablja M kot podprogram in sicer tako, da ga pokliče takoj na začetku, pri čemer je edina sled o obstoju M" simbol Z na samem dnu sklada. Ko M izprazni svoj sklad ostane v skladu avtomata M" le simbol Z. V tem primeru pa lahko izbiramo prehod v stanje q''_2. Zopet ugotovimo, da, če izberemo ta prehod prezgodaj, torej pred prestopom desnega roba vhodne besede, pride stroj v "slepo ulico". |‾|

Naš osnovni namen v tem poglavju je pokazati enakovrednost kontekstno neodvisnih gramatik in skladovnih avtomatov. Z drugimi besedami, za vsako KNG G = <V,T,P,S> obstaja skladovni avtomat M =<Q,SIGMA,GAMA,q_0,Z_0,F,dlt> z lastnostjo L(G) = L(M). Po drugi strani za vsak skladovni avtomat M, kot smo ga pravkar definirali, obstaja KNG G z lastnostjo L(M) = L(G). To dokažemo v obliki dveh izrekov:

5.8. IZREK. Podana je KNG G = <V,T,P,S>. Potem eksistira skladovni avtomat M = <Q,SIGMA,GAMA,q_0,Z_0,F,dlt> z lastnostjo L(G) = L(M).

Dokaz. Ne da bi izgubili na splošnosti, se domenimi, da je G v normalni obliki po Greibachovi. Edina težava je v morebitni prisotnosti besede ϵ v L(G). V tem primeru obravnavamo gramtiko v normalni obliki po Greibachovi za jezik L(G) - {ϵ}, za jezik {ϵ} pa zgradimo poseben (končni) avtomat, ki ga kasneje povežemo z M iz izreka. Torej vse produkcije so v obliki V --> a o alf, a ϵ T, alf ϵ V^*. Sedaj definirajmo elemente M takole: Q = $\{q_0\}$, SIGMA = T, GAMA = V, $q_0 = q_0$, $Z_0 = S$, F = $\emptyset$, dlt pa vsebuje naslednje vrednosti: $\langle q_0, bta\rangle \in dlt(q_0, a, A)$ za vse a ϵ T, A ϵ V, bta ϵ V^* pri katerih eksistira produkcija A --> a o bta. Dokaz, da M ustreza pogoju izreka, je preprost: denimo, da velja $w = a_1 a_2 \ldots a_n \in L(G)$. Naj bo

$$S \Longrightarrow alf_1 \Longrightarrow alf_2 \Longrightarrow \ldots \Longrightarrow alf_n \qquad (5.1)$$

skrajno leva izpeljava besede w. Potemtakem ima alf_i obliko $a_1 a_2 \ldots a_i bta_i$, kjer velja $bta_i \in V^*$. Zaporedje trenutnih opisov M, ki se konča s praznim skladom, dobimo iz opisane skrajno leve izpeljave (5.1) takole:

$$\langle q_0, a_1 a_2 \ldots a_n, S\rangle \vdash \langle q_0, a_2 \ldots a_n, bta_1\rangle \vdash$$

$$\langle q_0, a_3 \ldots a_n, bta_2\rangle \vdash \ldots \vdash \langle q_0, a_n, bta_n\rangle\langle q_0, \epsilon, \epsilon\rangle$$

Nasprotno, če je podana w ϵ L(M), lahko iz "uspešnega" ("sprejemajo- čega"?!) zaporedja trenutnih opisov dobimo skrajno levo izpeljavo.

Sedaj pa le nekaj besed, kako združimo končni avtomat za jezik {ϵ} v primeru $\epsilon \in L(G)$ z opisanim skladovnim avtomatom. Najprej pridobljeni skladovni avtomat spremenimo v avtomat, ki sprejema po kriteriju končnega stanja (na podlagi izreka 5.7 je to možno) Naj bo ta avtomat M = $\langle Q,T,V,q_0,Z_0,F,dlt\rangle$. Končni avtomat za {$\epsilon$} ima samo eno stanje q. Denimo $q \notin Q$. Avtomat za L(M) |_| {ϵ} zgradimo takole: M' = $\langle Q \cup \{q\}, T, V, q, Z_0, F \cup \{q\}, dlt'\rangle$, dlt' = dlt |_| dlt_1, pri čemer $\langle q_0, Z_0\rangle \in dlt(q, \epsilon, Z_0)$, v ostalih primerih je $dlt(r,a,Z) = dlt_1(r,a,Z)$, r ϵ Q', a ϵ T |_| {ϵ}, Z ϵ GAMA prazno. Bralec se bo brez težav prepričal, da M' resnično sprejema L(M) |_| {ϵ}. |¯|

Sedaj moramo dokazati naslednji izrek:

5.9. IZREK. Naj bo $M = \langle Q, SIGMA, GAMA, q_0, \emptyset, dlt\rangle$ skladovni avtomat, ki sprejema po kriteriju praznega sklada. Potem eksistira KNG $G = \langle V,T,P,S\rangle$ z lastnostjo L(G) = L(M).

Dokaz. Gramatiko G definiramo takole:

G = <V,T,P,S>
V = {S} |_| {<q,Z,r> | q,r ϵ Q; Z ϵ GAMA},
T = SIGMA,
P vsebuje produkcije

1. S --> <q_0 ,Z ,r> pri vseh r $\in$ Q in
2. <q,Z,q_{m+1} > --> x<q_1 ,Z ,q_1 ><q_2 ,Z ,q_2 >...<q_m ,Z ,q_{m+1} > za vse možne $q_1, q_2, \ldots, q_{m+1} \in Q$, če velja <$q_1$,Z Z ...Z > $\in$ dlt(q,x,Z). V primeru m = 0 imamo produkcijo <q,Z,q_1 > --> x pri x $\in$ SIGMA ali x = ϵ.

Pomen nekončnega simbola <q,Z,r> je, da je iz njega možno izpeljati vse besede $w \in T^*$ z lastnostjo

<q,wy,Zalf> $\vdash^*$ <r,y,alf>, pri vseh možnih besedah $y \in T^*$ in $alf \in V^*$. Z drugimi besedami, w je tisti del vhodne besede, ki jo preberemo med trenutkom, ko je nadzorna enota v stanju q in je na vrhu sklada simbol Z, do trenutka, ko se omenjeni simbol ravno izbriše iz sklada in je nadzorna enota v stanju r. Simbolično to zapišemo kot

$$\forall q,r \in Q, Z \in GAMA, w,y \in SIGMA^*, alf \in GAMA^* \; [\langle q,Z,r\rangle \Longrightarrow w \iff \langle q,wy,Zalf\rangle \vdash^* \langle r,y,alf\rangle] \qquad (5.2)$$

Če postavimo $q = q_0$, $Z = Z_0$, $y = \epsilon$, alf = ϵ, dobimo na levi strani ekvivalence izpeljavo besede w (brez prve produkcije), na desni strani pa uspešno zaporedje trenutnih opisov. Sedaj bomo trditev (5.2) dokazali.

(==>) Indukcija po globini drevesa izpeljav besede w, ki ima minimalno globino. V primeru d = 1 je trditev očitna. Namreč v tem primeru imamo <q,z,r> ==> a, oziroma obstaja produkcija <q,z,r> --> a, iz česar sklepamo, da obstaja prehod <r,ϵ> $\in$ dlt(q,a,Z), kar potegne za seboj resničnost desne strani (5.2). Primer d > 1. Produkcija, ki ustreza prvemu koraku je <q,Z,r> --> x<q_1 ,Z ,q_1 >...<q_m ,Z ,q_{m+1} >. Torej mora veljati w = $xw_1 w_2 \ldots w_m$, pri čemer <q_i ,Z ,q_{i+1} > $\vdash^*$ w_i. Na podlagi induktivne hipoteze velja

$$S_i = \langle q_i, w_i w_{i+1} \ldots w_m y, Z_i Z_{i+1} \ldots Z_m alf\rangle \vdash^* \langle q_{i+1}, w_{i+1} \ldots w_m y, Z_{i+1} \ldots Z_m alf\rangle$$

Ko zaporedja korakov S_i zložimo skupaj, dobimo zaporedje

<q,wy,Zalf> $\vdash^*$ <r,y,alf>.

(<==) Indukcija po številu korakov k stroja M. V primeru k = 1 trditev drži po definiciji. Obravnavajmo sedaj primer k > 1. Naj bo w kot prej. Potem je prvi korak stroja M

$$\langle q,xw'y,Zalf\rangle \vdash \langle q_1,w'y,B_1 B_2 \ldots B_m alf\rangle, \qquad (5.3)$$

pri čemer imamo <q_1 ,B_1 B_2 ...B_m > $\in$ dlt(q,x,Z) ter

$$\langle q_1, w'y, B_1 B_2 \ldots B_m \text{ alf}\rangle \overset{*}{\vdash} \langle r, y, \text{alf}\rangle \qquad (5.4)$$

Sedaj pa v zaporedju (5.4) poiščemo skrajno levi trenutni opis, kjer B_1 izgine. Očitno imamo

$$\langle q_1, w_1 w'y, B_1 B_2 \ldots B_m \text{ alf}\rangle \overset{*}{\vdash} \langle q_2, w'y, B_2 \ldots B_m \text{ alf}\rangle$$

za neka $w_1, w' \in \text{SIGMA}^*$, pri $w_1 ow' = w'$, ter $q_2 \in Q$. Torej na podlagi induktivne hipoteze lahko sklepamo, da velja $\langle q_1, B_1, q_2\rangle \overset{*}{\Longrightarrow} w_1$. Sedaj nadaljujemo ter poiščemo trenutni opis, ko B_2 izgine in dobimo $\langle q_2, B_2, q_3\rangle \overset{*}{\Longrightarrow} w_2$, itn. Skratka, dobili smo drevesa izpeljav $\langle q_i, B_i, q_{i+1}\rangle \overset{*}{\Longrightarrow} w_i$, ki jih lahko priključimo k produkciji

$$\langle q,Z,r\rangle \longrightarrow x\langle q_1, B_1, q_2\rangle\langle q_2, B_2, q_3\rangle \ldots \langle q_m, B_m, r\rangle,$$

ki ustreza koraku (5.3). Tako smo popolnoma sestavili drevo izpeljav, ki ustreza izpeljavi na levi strani ekvivalence (5.2).

|_|

5.10. PRIMER. Vzemimo naslednji skladovni avtomat

$$M = \langle\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, q_0, Z_0, \emptyset, \text{dlt}\}$$

pri čemer imamo

$$\text{dlt}(q_0, 0, Z_0) = \{\langle q_0, XZ_0\rangle\},\ \text{dlt}(q_0, 1, X) = \{\langle q_1, \epsilon\rangle\}$$
$$\text{dlt}(q_0, 0, X) = \{\langle q_0, XX\rangle\},\ \text{dlt}(q_1, \epsilon, X) = \{\langle q_1, \epsilon\rangle\}$$
$$\text{dlt}(q_0, 1, X) = \{\langle q_1, \epsilon\rangle\},\ \text{dlt}(q_1, \epsilon, Z_0) = \{\langle q_1, \epsilon\rangle\}$$

Poiščimo ekvivalentno gramatiko s postopkom, ki je opisan v dokazu izreka 5.10. Množica nekončnih simbolov je

$$V = \{S, \langle q_0,X,q_0\rangle, \langle q_0,X,q_1\rangle, \langle q_1,X,q_0\rangle, \langle q_1,X,q_1\rangle, \langle q_0,Z_0,q_0\rangle, \langle q_0,Z_0,q_1\rangle, \langle q_1,Z_0,q_0\rangle, \langle q_1,Z_0,q_1\rangle\},$$

produkcije pa so

$$S \longrightarrow \langle q_0,Z_0,q_0\rangle \mid \langle q_0,Z_0,q_1\rangle$$
$$\langle q_0,Z_0,q_0\rangle \longrightarrow 0\langle q_0,X,q_0\rangle\langle q_0,Z_0,q_0\rangle \mid 0\langle q_0,X,q_1\rangle\langle q_1,Z_0,q_0\rangle$$
$$\langle q_0,Z_0,q_1\rangle \longrightarrow 0\langle q_0,X,q_0\rangle\langle q_0,Z_0,q_1\rangle \mid 0\langle q_0,X,q_1\rangle\langle q_1,Z_0,q_1\rangle$$
$$\langle q_0,X,q_0\rangle \longrightarrow 0\langle q_0,X,q_0\rangle\langle q_0,X,q_0\rangle \mid 0\langle q_0,X,q_1\rangle\langle q_1,X,q_0\rangle$$
$$\langle q_0,X,q_1\rangle \longrightarrow 0\langle q_0,X,q_0\rangle\langle q_0,X,q_1\rangle \mid 0\langle q_0,X,q_1\rangle\langle q_1,X,q_1\rangle$$
$$\langle q_0,X,q_1\rangle \longrightarrow 1$$
$$\langle q_1,Z_0,q_1\rangle \longrightarrow \epsilon$$
$$\langle q_1,X,q_1\rangle \longrightarrow \epsilon \mid 1$$

Sedaj pa ugotovimo, da ni produkcij, ki bi imele spremenljivko $\langle q_1,X,q_0\rangle$ ali $\langle q_1,Z_0,q_0\rangle$ na levi, medtem ko imata spremenljivki

<q ,Z ,q > in <q ,X,q > lastnost, da iz njih ne moremo izpeljati
 0 0 0   0   0

besed, ki so sestavljene iz končnih simbolov. Tako ostanejo naslednje produkcije:

S --> <q ,Z ,q >,
       0 0 1
<q ,Z ,q > --> 0<q ,X,q ><q ,Z ,q >
 0 0 1   0   1 1 0 1
<q ,X,q > --> 0< ,X,q ><q ,X,q >,
 0   1   0   1 1   1
<q ,X,q > --> 1,
 0   1
<q ,Z ,q > --> ϵ,
 1 0 1
<q ,X,q > --> ϵ | 1.
 1   1

5.4 Naloge

5.1 Sestavi SA za vsakega izmed jezikov iz naloge 3.1.

5.2 Sestavi SA, ki je ekvivalenten naslednji gramatiki:

S --> aAA, A --> aS | bS | a.

5.3 Dokaži, da v primeru, ko je L KNJ, potem obstaja SA M, ki sprejema L s končnim stanjem, pri čemer M nima več kot 2 stanj in nima ϵ-prehodov.

5.4

1. Pokaži, da, če je L KNJ, potem je L = L(M) za neki SA M, pri katerem velja, da iz <p,gma> ϵ dlt(q,a,X) sledi |gma| ≤ 2;
2. Pokaži, da lahko M iz prvega odstavka še bolj omejimo, in sicer: <p,gma> ϵ dlt(q,a,X) ima lastnost, da je gma bodisi enak ϵ (zmanjševanje sklada), ali X (brez spremembe sklada), ali pa YX pri nekem skladovnem simbolu (povečanje sklada);

3. Ali lahko pri SA M iz prvega odstavka omejimo število stanj in še vedno dobimo SA za poljuben KNJ?
4. Ali lahko omejimo število stanj pri avtomatu iz drugega odstavka?

5.5 Deterministični skladovni avtomat ni ekvivalenten nedeterminističnemu skladovnemu avtomatu. Na primer jezik

$$L = \{0^n 1^n \mid n \geq 1\} \;|\_|\; \{0^n 1^{2n} \mid n \geq 1\}$$

je primer KNJ, ki ni jezik nobenega DSA.

1. Dokaži, da je L KNJ;
2. Dokaži, da ne obstaja DSA, ki bi sprejemal L.

6. POGLAVJE

LASTNOSTI KONTEKSTNO NEODVISNIH JEZIKOV

To poglavje je po snovi podobno poglavju 3 le, da se nanaša na kontekstno neodvisne jezike.

6.1 Lema o napihovanju za KNJ

Lemo o napihovanju tudi v primeru kontekstno neodvisnih jezikov uporabljamo za dokazovanje dejstva, da nekateri jeziki niso kontekstno neodvisni. Pri regularnih jezikih veljavnost leme o napihovanju sloni na tem, da avtomat, ki sprejema neki regularen jezik začne ponavljati stanja, kakor hitro vhodna beseda preseže neko dolžino. V primeru KNJ pa bomo izrabili dejstvo, da pri zadosti dolgi generirani besedi ustrezno drevo izpeljav vsebuje neko vejo, na kateri se nekončni simboli ponavljajo.

6.1. LEMA (o napihovanju za kontekstno neodvisne jezike). Naj bo L neki KNJ. Potem obstaja konstanta n, ki je odvisna le od L, z lastnostjo, da v primeru $z \in L$ in $|z| \geq n$ lahko zapišemo

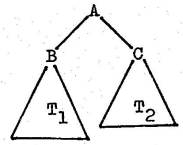
$z = uvwxy$, pri čemer

1. $|vx| \geq 1$.
2. $|vwx| \leq n$.
3. Pri vseh $i \geq 0$ velja $uv^i wx^i y \in L$.

Dokaz. V dokazu uporabljamo gramatiko $G = \langle V,T,P,S \rangle$ v normalni obliki po Chomskemu za jezik $L - \{\epsilon\}$. Prvi korak v dokazu je izpeljava dejstva, da iz $A \in V$, $A \overset{*}{\Longrightarrow} z \in T^*$ ter $|z|$ "veliko" sledi, da A-drevo z rezultatom z vsebuje "dolgo" vejo. To je najlaže dokazati v obliki trditve, da je dolžina rezultata $z \in T^*$ A-drevesa, kjer imajo vse veje manj ali enako k povezav, manjša ali enaka 2^{k-1}. Na tem mestu ugotovimo, da je število povezav na neki veji enako številu nekončnih vozlišč. Dokaz je zgrajen z indukcijo po k. Pri k = 1 ima A-drevo obliko, ki jo prikazuje sl. 6.1. To sledi iz posebne oblike produkcij v normalni obliki po Chomskemu. Torej smo trditev o dolžini rezultata pri k = 1 preverili. Sedaj pa naj trditev drži za vsa števila manjša od k in dokažimo veljavnost za k. Podano je A-drevo z rezultatom $z \in T^*$ in lastnostjo, da vsaka veja vsebuje manj ali enako $k > 1$ nekončnih vozlišč oziroma povezav. Zaradi oblike produkcij v gramatiki, ki je v normalni obliki po Chomskemu ima drevo obliko, kot jo prikazuje sl. 6.2. Koren ustreza produkciji $A \to BC$. Lahko zapišemo $z = z_1 z_2$, pri čemer sta z_1 in z_2 rezultata nekega B- oziroma C-drevesa. Prikazano B- in C-drevo ima na vsaki veji manj ali enako k-1 nekončnih vozlišč in na podlagi induktivne predpostavke ugotovimo, da

velja $|z_1|, |z_2| \leq 2^{k-2}$, iz česar izpeljemo $|z| = |z_1 z_2| \leq 2*2^{k-2} = 2^{k-1}$.

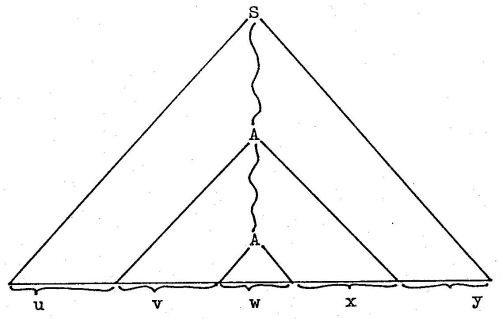
sl. 6.1 A-drevo višine 1.



sl. 6.2 A-drevo, ki ustreza k > 1.

Sedaj pa naj bo k število nekončnih simbolov v G in naj bo konstanta n iz trditve v lemi enaka 2^k. Naj velja $z \in L(G)$ in $|z| \geq n$. V tem primeru sklepamo, na podlagi pravkar dokazanega, da eksistira v drevesu izpeljav besede z neka veja z več kot k nekončnimi vozlišči. Opazujmo tako vejo in ji sledimo od končnega vozlišča proti korenu. Na tej poti se neki nekončni simbol mora ponoviti (to sledi iz števila nekončnih simbolov v V in števila nekončnih simbolov na veji). Ustavimo se pri prvem ponovljenem simbolu, denimo A (gl. sl. 6.3). Besede u,v,w,x,y ustrezajo delom rezultata z, kot je to prikazano na sl. 6.3.

Sedaj pa dokažimo lastnosti 1,2 in 3 iz leme. Lastnost 1, $|vx| \geq 1$ sledi iz dejstva, da je dolžina vx najmanjša v primeru, ko zgornjemu simbolu A ustreza produkcije A --> AB ali A --> BA, oziroma, ko je spodnji simbol A pravzaprav naslednik zgornjega A in pa ko je naslednik B končen simbol. V tem primeru je $|vx| = 1$, sicer je vx daljše. Lastnost 2, $|vwx| \leq n$ sledi iz naslednjega premisleka: dolžina besede w je $\leq 2^{k-1}$, ker pod spodnjim simbolom A ni ponavljanja nekončnih simbolov; iz istega razloga je skupna dolžina besed v in x tudi $\leq 2^{k-1}$. Torej velja lastnost 2. Lastnost 3, $\forall i \geq 0[uv^i wx^i y \in L(G)]$, pa sledi iz dejstva, da lahko spodnje A-drevo zamenjamo z zgornjim poljubno krat (to ustreza primerom $i \geq 1$), oziroma zgornje s spodnjim (kar ustreza primeru i = 0). |‾|



sl. 6.3 Drevo izpeljav besede z pri pogoju $|z| \geq n$.

6.2. PRIMER. Dokazali bomo, da jezik $L = \{a^i b^i c^i \mid i \geq 1\}$ ni kontekstno neodvisen. Podobno kot pri uporabah napihovalne leme za regularne jezike, bomo zopet uporabili reductio ad absurdum. Naj bo L KNJ. Izberimo i = n, kjer je n konstanta iz leme. Velja torej $a^n b^n c^n = uvwxy$. Sedaj obravnavamo dva primera: (1) beseda vwx ne vsebuje znakov c in (2) beseda vwx vsebuje znake c, vendar ne vsebuje znakov a. Ta dva primera pokrivata vse možnosti zaradi lastnosti 2 v lemi 6.1. Ko besedo $a^n b^n c^n$ "napihnemo", dobimo na podlagi lastnosti 1 poljubno dolge besede, pri katerih pa ostane bodisi število simbolov c nespremenjeno (primer 1) ali pa število simbolov a nespremenjeno (primer 2). V obeh primerih dobimo torej besede, ki po

definiciji jezika L ne spadajov v L, kar je v protislovju z lastnostjo 3 leme 6.1.

Ogdenova lema

Obstajajo določeni jeziki, ki niso kontekstno neodvisni, pri katerih nam pa lema o napihovanju ne pomaga. Na primer

$$L_3 = \{a^i b^j c^k d^l \mid \text{bodisi } i = 0 \text{ ali pa } j = k = l\}$$

ni kontekstno neodvisen. Na primer, če izberemo $z = b^j c^k d^l$, potem, ko zapišemo z = uvwxy, je vedno možno izbrati u, v, w, x in y tako, da $uv^m wx^m y$ pripada L_3 pri vseh m. Na primer, ko vwx vsebuje le b-je. Če pa izberemo $z = a^i b^j c^j d^j$, sta lahko v in x sestavljena izključno iz a-jev, zaradi česar je $uv^m wx^m y$ zopet v L_3 pri vseh m.

V takem primeru potrebujemo neko močnejšo različico leme o napihovanju, ki nam omogoča, da se osredotočimo na neko omejeno število mest v besedi in jih napihujemo. Podobno razširitev v primeru regularnih množic ni težko dobiti, saj poljubno zaporedje n+1 stanj nekega končnega avtomata z n stanji vsebuje ponavaljanje in lahko podbesedo med ponavaljanji stanj napihnemo. V primeru KNJ pa je podoben rezultat veliko težje (vendar možno) dobiti. Na tem mestu opišemo in dokažemo neko šibko različico t.im. Ogdenove leme.

6.3. LEMA (Ogdenova lema). Naj bo L KNJ. Potem obstaja konstanta n (ki je pravzaprav lahko enaka oni iz leme o napihovanju) tako, da, če je z ∈ L in zaznamujemo n ali več mest v z kot "posebna mesta", potem lahko zapišemo z=uvwxy in velja:

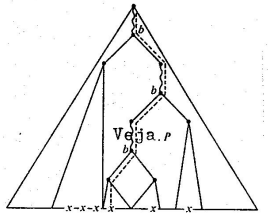
1. v in x skupaj imata vsaj eno posebno mesto;
2. vwx ima največ n posebnih mest in
3. pri vseh $i \geq 0$ velja $uv^i wx^i y \in L$.

Dokaz. Naj bo G gramatika v normalni obliki po Chomskemu, ki generira L-{ϵ}. Naj ima G k spremenljivk in izberimo $n = 2^k + 1$. Tudi sedaj moramo poiskati neko vejo v drevesu izpeljav, podobno kot v dokazu leme o napihovanju, vendar, ker nas zanimajo le posebna mesta, nas ne bo zanimalo vsako vozlišče na veji, temveč le razvejišča, ki so vozlišča z dvema naslednikoma, od katerih peljejo veje do posebnih mest.

Opisano vejo P bomo poiskali takole: najprej postavimo v P koren. Naj bo r zadnje vozlišče, ki smo ga vključili v P. V primeru, ko je r list smo končali. V primeru, ko ima r le enega naslednika, preko katerega vodijo veje do posebnih mest, vključimo tega nasledniko v P in postopek ponovimo. V primeru pa, ko preko dveh naslednikov vodijo veje do posebnih mest, je r razvejišče in v P vključimo tistega naslednika, ki vsebuje (pomen besede "vsebuje" je očiten) več posebnih mest (v primeru enakega števila posebnih mest pri dveh naslednikih, vključimo poljubnega). Postopek je prikazan na sl. 6.4.

Razvejišča imajo lastnost, da vsebujejo vsaj polovico posebnih mest kot predhodno razvejišče. Ker pa imamo vsaj n posebnih mest v z in vsa ta mesta vsebuje koren, sledi, da je v P vsaj k+1 razvejišč. Torej med zadnjimi k+1 razvejišči sta vsaj dva z isto oznako. Ti dve razvejišči potem igrata isto vlogo kot dve vozlišči s ponovljenima oznakama v dokazu leme o napihovanju. Zato tudi podrobnosti dokaza ne bomo ponavljali.

□



sl. 6.4 Veja P. Posebna mesta so zaznamovana z x; razvejišča z b.

6.4. PRIMER. Naj bo $L_4 = \{a^i b^j c^k \mid i \neq j,\ j \neq k$ in $i \neq k\}$. Denimo, da je L_4 kontekstno neodvisen. Naj bo n konstanta iz Ogdenove leme in vzemimo niz $z = a^n b^{n+n!} c^{n+2n!}$.

Naj bodo položaji a-jev posebna mesta in naj z = uvwxy zadošča pogojem Ogdenove leme. Če bodisi v ali x vsebuje dva ~~posebni~~ naključna mesti, potem uv wx y ne pripada L_4. (Na primer, če je v oblike a^+b^+, potem v uv wx y neki b stoji pred a.) Ugotovimo lahko, da vsaj eden izmed v in x vsebuje a-je, ker le a-ji ustrezajo posebnim mestom. Torej, če x pripada b^* ali c^*, mora v pripadati a^+. Če x pripada a^+, mora v pripadati a^*, kajti sicer bi neko b ali c stalo pred a. Sedaj bomo podrobno obravnavali primer ko x pripada b^*. Ostale primere lahko obravnavamo podobno. Naj x pripada b^* in v pripada a^+. Naj bo $p = |v|$. Potem velja $1 \leq p \leq n$ in torej p deli n!. Naj bo q celo število tako, da velja pq = n!. Potem

$$z' = uv^{2q+1}wx^{2q+1}y$$

pripada L_4. Vendar velja $v^{2q+1} = a^{2pq+p} = a^{2n!+p}$. Ker pa uwy vsebuje natanko (n-p) a-jev, ima z' (2n!+n) a-jev. Vendar, ker v in x ne vsebujejo c-jev, ima z' prav tako (2n!+n) c-jev in torej ne pripada L_4, kar je protislovje. Do podobnega protislovja pridemo, če x pripada a^+ ali c^*. Torej L_4 ni kontekstno neodvisen jezik.

Lahko ugotovimo, da je lema o napihovanju poseben primer Ogdenove leme, kjer so vsa mesta posebna.

6.2 Nekatere lastnosti kontekstno neodvisnih jezikov

6.5. IZREK. Vsak regularen jezik je kontekstno neodvisen.

Dokaz. Naj bo L regularen jezik. Pokazali bomo da obstaja kontekstno neodvisna gramatika, ki generira L. Naj bo L =L(M), M = $\langle Q,SIGMA,dlt,q_0,F\rangle \in$ NKA. Ustrezna gramatika je G = $\langle Q,SIGMA,P,q_0\rangle$, pri čemer P vsebuje produkcijo q --> ar natanko, ko velja r $\in$ dlt(q,a) in produkcijo q --> $\in$ v primeru q $\in$ F. Lastnost L = L(G) ni težko dokazati, saj velja $q_0 \overset{*}{\Longrightarrow} xr$ pri x $\in$ L natanko v primeru r $\in$ F, nato pa se simbola r znebimo z enkratno uporabo produkcije r --> $\in$. |_|

6.6. IZREK. Operacije unije, stik ter iteracija (*) ohranjajo KNJ.

Dokaz. Za dokaz je potrebno iz gramatik za L_1 in L_2 sestaviti gramatike za $L_1 \cup L_2$, $L_1 \circ L_2$ in L_1^*. Naj bo $L_1 = L(G_1)$ in $L_2 = L(G_2)$, pri čemer velja $G_1 = \langle V_1,T_1,P_1,S_1\rangle$, $G_2 = \langle V_2,T_2,P_2,S_2\rangle$. Trdimo, da velja $L_1 \cup L_2 = L(G_u)$,

$G_u = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S\rangle$,

$L_1 \circ L_2 = L(G_o)$,

$G_o = \langle Vk_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 S_2, S\rangle$,

$L_1^* = L(G_*)$,

$G_* = \langle V_1, T_1, P_1 \cup \{S \to S_1 S \mid \epsilon\}, S\rangle$.

V vseh primerih predpostavljamo $S \notin V_1, V_2$. Dokaz dejstev da opisane gramatike generirajo želene jezike prepuščamo bralcu. $\overline{\square}$

Operacije iz izreka 6.6 se pri KNJ vedejo podobno kot pri regularnih jezikih. Razlika med KNJ in regularnimi jeziki pa se pokaže pri operaciji preseka.

6.7. IZREK. Operacija preseka ne ohranja KNJ.

Dokaz. Zadostuje, da prikažemo dva kontekstno neodvisna jezika, katerih presek ni KNJ. Naj bo $L_1 = \{a^i b^j c^j \mid i \geq 1, j \geq 1\}$ in $L_2 = \{a^i b^i c^j \mid i \geq 1, j \geq 1\}$. Z lahkoto se prepričamo, da velja $L_1 \overline{\square} L_2 = L_3 = \{a^i b^i c^i \mid i \geq 1\}$, za katerega pa smo že pokazali, da ni KNJ (gl. primer 6.2). Torej je potrebno le sestaviti kontekstno neodvisni gramatiki za L_1 in L_2, kar pa prepuščamo bralcu za vajo, saj obe gramatiki lahko

definiramo na podlagi gramatike za jezik $\{a^i b^i \mid i \geq 1\}$. $\overline{\square}$

Seveda, glede na $L_1 \overline{\square} L_2 = \overline{L}_1 \cup \overline{L}_2$ velja tudi

6.8. POSLEDICA. Komplement ne ohranja KNJ.

Čeprav presek v splošnem ne ohranja KNJ, kot smo to pravkar videli, pa le velja, da je presek nekega KNJ z regularnim jezikom vendar kontekstno neodvisen.

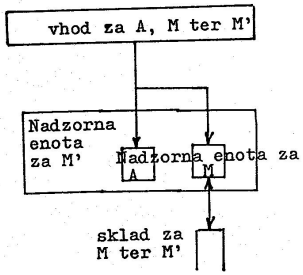
6.9. IZREK. $L \in$ KNJ, R pa je regularen jezik. Potem je $L \overline{\square} R \in$ KNJ.

Dokaz. Zadostuje, da opišemo skladovni avtomat, ki sprejema $L \overline{\square} R$. Denimo da je $L = L(M)$, pri čemer je $M = \langle Q_M, SIGMA, GAMA, dlt_M, q_{M0}, Z_0, F_M\rangle$ skladovni avtomat, ki sprejema po kriteriju končnega stanja, in $R = L(A)$, pri čemer je $A = \langle Q_A, SIGMA, dlt_A, p_{A0}, F_A\rangle \in$ NKA. Jezik $L \overline{\square} R$ sprejema skladovni avtomat M' po kriteriju končnega stanja, pri čemer M' vsebuje kot "podprogram" tako A kot M (gl. sl. 6.5). Osnovni elementi M' so določeni takole:

$M' = \langle Q_A \times Q_M, SIGMA, GAMA, dlt, \langle p_0, q_0 \rangle, Z, F_A \times F_M \rangle$

dlt je definirano tako, da se prehodi leve komponente stanja opravljajo na podlagi dlt_A, razen v primeru ϵ-prehodov, ko leva komponenta ostane nespremenjena, desna komponenta in vsebina sklada pa se spreminjata na podlagi dlt_M. Vhod je sprejet natanko, ko v trenutku prestopa desnega roba vhodne besede obe komponenti stanje pripadata ustreznima množicama končnih stanj.

|_|



sl. 6.5 Skladovni avtomat za presek KNJ in regularnega jezika

6.3 Odločitveni algoritmi za vprašanja v zvezi s KNJ

Tipična vprašanja, ki jih postavljamo v zvezi s kontekstno neodvisnimi jeziki, so npr. ali je neki KNJ prazen (enak Ø), končen, neskončen in ali neka beseda pripada podanemu KNJ. Odgovore na ta vprašanja lahko poiščemo z algoritmom. Vendar opozarjamo, pa na dejstvo, da odgovore na mnoga druga vprašanja ne moremo poiskati z algoritmom, in sicer ne zato, ker ustreznih algoritmov še ne poznamo, temveč zato, ker v principu ne obstajajo. Med vprašanja o KNJ slednje vrste spadajo npr., ali sta dve KNG ekvivalentni (ali generirata isti jezik), ali je komplement nekega KNJ končen, ali je komplement nekega KNJ prav tako KNJ in ali je neka KNG dvoumna. O tehnikah za dokazovanje neeksistence algoritmov za ta vprašanja bomo spregovorili kasneje.

6.10. IZREK. Na vprašanje, ali je neki KNJ (a) prazen, (b) končen, (c) neskončen lahko odgovorimo z algoritmom.

Dokaz. Odgovor na vprašanje, ali je neki KNJ prazen, poiščemo z algoritmom iz leme 4.7. Namreč potrebno je le preveriti, ali začetni simbol S pripada množici simbolov A z lastnostjo $A \stackrel{*}{\Longrightarrow} x, x \in T^*$.

Odgovor na vprašanje o končnosti ali neskončnosti nekega KNJ pa lahko poiščemo z algoritmom, ki izrablja naslednjo lastnost KNJ. Naj bo podan $L \in$ KNJ. Poiščimo gramatiko G za $L - \{\epsilon\}$ v normalni obliki po Chomskemu (očitno je L končen natanko, ko je $L - \{\epsilon\}$, končen). Sedaj pa sestavimo graf,

katerega vozlišča so nekončni simboli G, usmerjena povezava A --> B pa eksistira natanko, ko eksistira produkcija A --> BC ali A --> CB, pri neki spremenljivki C. Trdimo, da je L(G) končen natanko, ko opisani graf nima ciklov.

Naj obstaja cikel $A_0, A_1, \ldots, A_n, A_0$. V gramatiki, ki je v normalni obliki po Chomskemu, so vsi simboli koristni. Torej

$$S \overset{*}{\Longrightarrow} alf\ o\ A_0\ bta \overset{*}{\Longrightarrow} alf\ o\ gma\ o\ A_0\ dlt\ o\ bta \overset{*}{\Longrightarrow} alf\ o\ gma^i$$

$A_0^i\ dlt\ bta$ pri vseh $i \geq 0$. Obenem pa imajo vse spremenljivke lastnost, da iz njih lahko izpeljemo besedo nad T neničelne dolžine (G ne vsebuje produkcij v obliki A --> Є). Torej sklapamo, da iz S lahko izpeljemo poljubno dolgo besedo, oziroma, da je L(G) neskončen.

Sedaj pa nasprotno, denimo, da graf nima cikla. Pokazali bomo, da je L(G) končen. Naj bo <u>rang neke spremenljivke</u> A dolžina najdaljše poti v grafu, ki začenja pri A. Glede na acikličnost grafa je rang vsake spremenljivke končen. Naj bo rang A enak r. Pokazali bomo, da je dolžina besede, sestavljene iz končnih simbolov, ki jo lahko generiramo iz A, manjša ali enaka 2^r. Ker to velja med ostalim za S, je potemtakem L(G) končen. Dokaz izpeljemo z indukcijo po r.

r = 0. Iz A ne pelje nobena povezava, kar pomeni, da imajo vse A-produkcije obliko A --> a, a Є T. S tem smo trditev pri r = 0 dokazali.

r > 0. Naj trditev velja za vse števila, ki so manjša od r. A ima rang $r \geq 1$. Torej obstajata povezavi A --> B in A --> C oziroma produkcija A --> BC. Sklepamo lahko, da sta ranga B in C strogo manjša od A. Naj velja $B \overset{*}{\Longrightarrow} x_1$, $C \overset{*}{\Longrightarrow} x_2$ in torej $A \overset{*}{\Longrightarrow} x_1 x_2$. Na podlagi induktivne predpostavke velja

$$|x_1|, |x_2| \leq 2^{r-1} \quad \text{iz česar sledi} \quad |x_1 x_2| \leq 2*2^{r-1} = 2^r. \quad |\_|$$

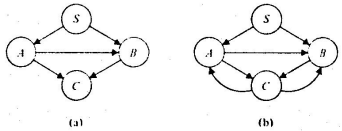
Na tem mestu se ne bomo spuščali v algoritme za ugotavljanje acikličnosti grafov, ker predpostavljamo, da so bralcu znani od drugod.

6.11. PRIMER. Podana je gramatika

S --> AB
A --> BC | a
B --> CC | b
C --> a

Graf te gramatike je prikazan na sl. 6.6a. Brez težav ugotovimo, da graf nima ciklov. Rang spremenljivk S,A,B in C je 3,2,1 in 0, po vrsti. Iz tega sledi, da jezik gramatike ne vsebuje besed, ki bi bile daljše od $2^3 = 8$, in je torej končan. Pavzaprav je najdaljša beseda, ki jo lahko izpeljemo is S

S ==> AB ==> BCB ==> CCCB ==> CCCCC ==> aaaaa



sl. 6.6 Grafi KN gramatik

Če pa podani gramatiki dodamo produkcijo C --> AB, dobimo graf s sl. 6.6b. Ta graf vsebuje nekaj ciklov kot npr. ABCA. Na podlagi izpeljave v dokazu izreka 6.10 sledi, da velja

$$S \overset{*}{\Longrightarrow} alf\ o\ A\ o\ bta \overset{*}{\Longrightarrow} alf\ o\ gma\ o\ A\ o\ dlt\ o\ bta \overset{*}{\Longrightarrow} alf\ o\ gmaiA\ o\ dlt^{i}\ bta$$

pri vseh $i \leq 0$. Konkretno imamo

$$S \Longrightarrow AB \Longrightarrow BCB \Longrightarrow CCCB \Longrightarrow CABCB$$

oziroma $S \overset{*}{\Longrightarrow} C\ AB^{i}\ C\ B^{i}$ ali $S \overset{*}{\Longrightarrow} a\ ab^{i}\ a\ b^{i}$ pri vseh $i \geq 0$. Torej je L(G) neskončen.

Izredno važen problem v zvezi s KNJ je problem pripadnosti. Natančneje povedano, če sta podana KNG $G = \langle V,T,P,S \rangle$ in $x \in T^{*}$, je potrebno ugotoviti, ali velja $x \in L(G)$. Ta problem je praktično pomemben zato, ker praktično uporabljeni računalniški programski jeziki temeljijo na kontekstno neodvisnih gramatikah, pravilno prevajanje nekega programa ("besede" jezika ustrezne gramatike) pa je tesno povezano s postopkom ugotavljanja ali je

program pravilno zapisana "beseda" jezika.

Problem pripadnosti ni težko rešiti s preprostim algoritmom, ki ga bomo opisali, vendar je izredno počasen. Torej podana sta G in x. V primeru $x = \epsilon$ lahko preizkusimo trditev $S \overset{*}{\Longrightarrow} \epsilon$ z algoritmom iz dokaza leme 4.11. Denimo torej, da velja $x \neq \epsilon$. Sedaj G spremenimo v normalno obliko po Greibachovi za jezik $L(G) - \{\epsilon\}$. Glede na to, da se desna stran vsake produkcije začenja s končnim simbolom, je izpelava besede x (v kolikor $x \in L(G)$) dolga največ $|x|$. Torej je potrebno sistematično generirati vse izpeljave dolžine največ $|x|$ in če se med njimi pojavi izpeljava besede x, smo $x \in L(G)$ dokazali, sicer pa velja $x \notin L(G)$. Ker pa je število takih izpeljav eksponentna funkcija od $|x|$, tudi sam algoritem potrebuje v najslabšem primeru eksponentno (od $|x|$) število korakov, kar je nezaželeno.

Obstajajo pa algoritmi, ki za isti problem potrebujejo velikostnega reda $|x|^{3}$ ali celo nekaj manj korakov. Algoritem, ki ga bomo tukaj opisali, je znan kot Algoritem Cocke-Younger-Kasami a CYK algoritem in temelji na dinamičnem programiranju.

Za gramatiko G predpostavimo, da je v normalni obliki po Chomskemu in, da velja $x \notin L(G) \setminus \{\epsilon\}$ (v primeru $x = \epsilon$ postopamo tako, kot smo opisali prej). Algoritem deluje tako, da pri $j = 1,2,\ldots,n$ ugotavlja, ali velja $A \overset{*}{\Longrightarrow} x_{ij}$, kjer je A neka spremenljivka gramatike, x_{ij} pa je tisti del besede x, ki

začenja pri i-tem znaku in je dolg j znakov. Pri j = 1 je $A \overset{*}{\Longrightarrow} x_{ij}$ možno le, ko obstaja produkcija $A \longrightarrow a_i$, kjer je a_i i-ti znak besede x. Pri j > 1 pa je $A \overset{*}{\Longrightarrow} x_{ij}$ možno le, ko obstaja produkcija A --> BC in obenem velja $B \overset{*}{\Longrightarrow} x_{ik}$ ter $C \overset{*}{\Longrightarrow} x_{i+k,j-k}$, pri nekem $i \leq k \leq j - 1$. Vendar smo slednji dve relacijo že izračunali, ker velja k < j in zato lahko izračunamo tudi prvo. Celoten algoritem je prikazan na sl. 6.7.

```
   BEGIN
1.   FOR i := 1 TO n DO
       V   := {A | A --> a je produkcija, a pa je
        i1
                  i-ti simobl besede x};
2.   FOR j := 2 TO n DO
3.     FOR i := 1 TO n-j+1 DO
       BEGIN
4.       V   := Ø ;
          ij
5.       FOR k := 1 TO j-1 DO
6.         V   := V   |_| {A | A --> BC je
             ij     ij
             produkcija, B ∈ V   , C ∈ V       }
                              ik       i+k,j-k
       END
   END
```

sl. 6.7 CYK algoritem za problem x ∈ L(G)

Za razumevanje algoritma je potrebno le pojasnilo, da V_{ij} predstavlja množico nekončnih simbolov A z lastnostjo $A \overset{*}{\Longrightarrow} x_{ij}$. Upoštevati pa moramo še dejstvo, da velja $1 \leq i \leq n - j + 1$,

kajti desna vrednost i je največja pri kateri še obstaja podzaporedje x, ki začenja pri simbolu i in ki je dolgo j znakov. Sedaj pa k oceni števila korakov, ki ga porabi algoritem pri |x| = n. Stavek 1 zahteva po velikostnem redu n korakov, saj je število produkcij konstanta. Vgnezdeni zanki 2 in 3 povzročita po velikostnem redu n^2 izvajanj jedra 4 - 6. Jedro 6 zanke 5 zahteva po velikostnem redu konstantno število korakov, saj V_{ik} vsebuje največ vse nekončne simbole, njih število pa je konstanta. Torej zanka 5 porabi po velikostnem redu j korakov. Iz tega sledi, da jedro 4-6 zahteva prav tako po velikostnem redu j korakov. Na podlagi vsega zapisanega sledi, da celoten algoritem porabi po velikostnem redu n^3 korakov.

6.12. PRIMER. Podana je naslednja KNG

```
S --> AB | BC
A --> BA | a
B --> CC | b
C --> AB | a
```

ter beseda baaba. Tabela vrednosti V_{ij} je prikazana na sl. 6.8. Prvo vrstico tabele izračuna stavek 1 iz algoritma na sl. 6.7, kasnejše pa stavek 2. Kot primer vzemimo izračun V_{24}.

Stavek 4 najprej priredi V_{24} začetno vrednost Ø. Za tem stavek 5 pri k = 1 doda spremenljivko S zaradi produkcije S --> AB in ker velja $A \in V_{21}$ ter $B \in V_{31}$. Pri k = 1 stavek 5 prav tako

doda C zaradi produkcije C --> AB. Pri k = 2 stavek 5 doda k

V_{24} spremenljivko A, zaradi produkcije A --> BA ter $B \in V_{22}$ in

$A \in V_{42}$. Pri k = 3 bi lahko dodali S zaradi produkcije S --> BC

in $B \in V_{24}$ ter $C \in V_{51}$, prav tako A zaradi produkcije A --> A in

$B \in V_{23}$ ter $A \in V_{51}$, vendar sta obe spremenljivki že v V_{24}.

Končno ugotovimo, zaradi $S \in V_{15}$, da velja $x \in L(G)$.

	b	a	a	b	a	
	1	2	3	4	5	i
1	B	A,C	A,C	B	A,C	
2	S,A	B	S,C	S,A		
3	Ø	B	B			
4	Ø	S,A,C				
5	S,A,C					

sl. 6.8 Tabela vrednosti V_{ij}

6.4 Naloge

6.1 Pokaži, da naslednji jeziki niso kontekstno neodvisni:

1. $\{a^i b^j c^k \mid i<j<k\}$
2. $\{a^{i^2} b^j \mid j = i\}$
3. $\{a^i \mid i$ je praštevilo$\}$
4. množica nizov nad {a,b,c}, ki vsebujejo ista števila vseh;
5. $\{a^n b^n c^m \mid n \leq m \leq 2n\}$

6.2 Kateri izmed naslednjih jezikov so KNJ?

1. $\{a^i b^j \mid i \neq j$ and $i \neq 2j\}$
2. $(a+b)^* - \{(a^n b^n)^R \mid n \geq 1\}$
3. $\{ww^R \mid w \in (a+b)^*\}$
4. $\{b^{\#b} \mid b$ je dvojiški zapis i, $i > 1\}$
5. $\{wxw^R \mid w$ ter x pripadata $(a+b)^*\}$
6. $(a+b)^* - \{(a^n b^n)^* \mid n \geq 1\}$

6.3 Pokaži, da naslednji jeziki niso KNJ.

1. $\{a^i b^j c^k \mid j = \max(i,k)\}$
2. $\{a^n b^n c^i \mid i \neq n\}$

(Napotek: uporabljaj Ogdenovo lemo na nizu oblike $a^n b^n c^n$.)

6.4 Dokaži, da naslednje operacije ohranjajo KNJ:

1. INIT
2. CYCLE
3. zrcaljenje

INIT ter CYCLE sta definirani v nalogi 3.4

6.5 Dokaži, da naslednje operacije ne ohranjajo KNJ:

1. MIN
2. MAX
3. inverzna substitucija
4. INV, če je $INV(L) = \{x \mid x=wyz, wy^R z \in L\}$

MIN ter MAX sta definirana v nalogi 3.4

7. POGLAVJE

TURINGOVI STROJI

V hierarhiji avtomatov stoji Turingov stroj na najvišjem mestu, ker je sposoben izračunati vse, kar se sploh izračunati da (v to vsaj verjamemo). Potemtakem je po svoji moči enakovreden najmočnejšemu računalniku, ne glede na svojo izredno preprosto zgradbo. Seveda to velja le, če moč merimo po tem, kaj vse je sposoben narediti, ne spuščamo pa se v ceno, ki jo moramo plačati za rezultat (n.pr. izraženo s številom korakov, ki jih stroj porabi za določeno opravilo). Doslej so nas različni avtomati zanimali predvsem kot jezikovni sprejemniki, torej zanimivi so bili jeziki, ki so jih ti avtomati sprejemali, pri Turingovih strojih pa postanejo enako zanimive funkcije, ki so jih sposobni izračunati.

7.1 Uvod

Z intuitivnim pojmom učinkovite (efektivne) procedure ali algoritma smo se že nekajkrat srečali. Na primer v poglavju 3 smo opisali algoritem, ki ugotovi, ali je množica besed, ki jo sprejema neki podan končen avtomat prazna, končna ali neskončna. Na podlagi tega bi lahko na hitro sklepali, da podobni algoritmi obstajajo za vse jezike, ki imajo nekakšne končne opise.

Vendar temu ni tako. Na primer ni algoritma, ki bi nam razodel, ali je komplement nekega KNJ prazen (čeprav je vprašanje, ali je sam KNJ prazen, rešljivo z algoritmom). Bodimo pozorni, da nas ne zanima algoritem, ki bi odgovoril na tako vprašanje o nekem posebnem, določenem KNJ, temveč algoritem, ki bi bil sposoben podati odgovor na vprašanje za vse KNJ. Očitno je, da, če potrebujemo odgovor za en sam KNJ, potem algoritem zanj gotovo obstaja: namreč tak algoritem, ki odgovori bodisi "da" ali "ne", ne glede na vhodne podatke (seveda pa vnaprej ne vemo, kateri od teh dveh algoritmov je pravi).

Na začetku tega stoletja se je David Hilbert lotil problema poiskati algoritem, ki bi ugotovil resničnost ali neresničnost poljubne matematične trditve. Bolj natančno, zanimalo ga je vprašanje, ali je neka poljubna formula predikatnega računa prvega reda resnična ali neresnična. Glede na dejstvo, da je predikatni račun prvega reda dovolj močan, da v njem zapišemo trditev, da je jezik neke KNG enak $SIGMA^*$, pri neki abecedi SIGMA, bi uspeh Hilbertovih prizadevanj tudi zagotovil

eksistenco algoritma za praznost komplementa KNJ. Vendar je l. 1931 Kurt Gödel iznašel svoj znani izrek o nepopolnosti, ki zagotavlja, da takega algoritma ni. Rezultat je dosegel tako, da je sestavil neko formulo predikatnega računa prvega reda, ki, če jo uporabimo na naravnih številih, zagotavlja že po svoji obliki, da je ne moremo dokazati niti moremo dokazati njenega nasprotja. Gödelov dosežek in kasnejši dosežki v zvezi z izračunljivostjo spadajo med velike matematične rezultate tega stoletja.

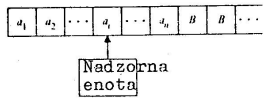
Ko je bil pojem algoritma formalno določen, se je kmalu pokazalo, da za mnoge konkretne funkcije ne eksistirajo algoritmi. Pravzaprav se o eksistenci takih funkcij lahko prepričamo brez težav -- s preprostim "preštevanjem". Vzemimo funkcije, ki preslikajo naravna števila v {0, 1}. Med takimi funkcijami in realnimi števili lahko zgradimo bijekcijo. Sedaj pa, če privzamemo, da imajo algoritmi končne opise, sledi, da obstaja bijekcija med množico algoritmov in množico naravnih števil. Ker pa med naravnimi števili in realnimi števili ni bijekcije (množica realnih števil ima večjo moč), sklepamo, da obstajajo funkcije, za katere ne eksistirajo algoritmi. Funkcij je preprosto preveč -- neštevno mnogo -- da bi jih lahko zajeli z vsemi razpoložljivimi algoritmi. Torej eksistenca funkcij, ki niso algoritmično izračunljive, ni presenetljiva. Presenetljivo pa je, da imajo tako lastnost mnoge funkcije, ki niso umetno skonstruirane temveč izvirajo iz realnih matematičnih problemov.

Dandanes je Turingov stroj sprejet kot standardna formalizacija pojma algoritma. Ker pojem "vse možne formalizacije pojma algoritma" ne moremo formalizirati, tudi ne moremo dokazati, da so vse takšne formalizacije ekvivalentne Turingovemu stroju. Vendar v to verjamemo na podlagi empiričnih izkušenj. Tej trditvi (da so vse možne formalizacije pojma algoritma ekvivalentne Turingovemu stroju), ki jo sicer ne moremo dokazati, pravimo Churchova hipoteza.

7.2 Model Turingovega stroja

Formalen model algoritma mora zadoščati pogojema, da je zgrajen iz končne množice elementov in da deluje po diskretnih korakih. Tak model je opisal Turing l. 1936.

Osnovni model je prikazan na sl. 7.1. Predstavljamo si ga kot nekakšen trak, ki je neskončen v desno in je razdeljen na celice. V vsaki celici je zapisan en simbol iz neke končne abecede, pri čemer je v vsakem trenutku vsebina le končnega števila celic različna od nekega posebnega simbola, ki mu pravimo presledek, in ki ga zaznamujemo z B (za celice, katerih vsebina je B, pravimo, da so prazne). Poleg traku imamo še nekakšno končno nadzorno enoto, ki bere in spreminja simbole na traku. V vsakem trenutku je nadzorni enoti dostopna le ena celica traku skozi nekakšno čitalno okno. V začetku okno pokriva skrajno levo celico traku.



Sl. 7.1 Osnovni model Turingovega stroja

V enem koraku svojega delovanja Turingov stroj opravi naslednje:

1. spremeni stanje nadzorne enote,
2. zapiše nov simbol v dostopno celico ter
3. premakne čitalno glavo (spremeni dostopno celico) za eno mesto bodisi v levo ali desno.

Razlika med dvosmernim končnim avtomatom, kot smo ga opisali v poglavju 2, in Turingovim strojem, je v sposobnosti Turingovega stroja, da spreminja vsebino traku. Imamo naslednjo formalno definicijo:

7.1. DEFINICIJA. Turingov stroj je sedmerka

$M = \langle Q, SIGMA, GAMA, dlt, q_0, B, F \rangle$

pri čemer velja:

Q je končna množica stanj;

SIGMA je množica vhodnih simbolov;

GAMA vsebuje SIGMA in je množica tračnih simbolov;

B je poseben element GAMA, ki mu pravimo presledek;

dlt: $Q \times GAMA \rightarrow Q \times GAMA \times \{L, D\}$ je (parcialna) funkcija prehodov;

q_0 je začetno stanje; in

F je podmnožica Q sestavljena iz končnih stanj $|\underline{\ }|$

Delovanje Turingovega stroja spremljamo s trenutnimi opisi. Množica trenutnih opisov $TO = GAMA^* \times Q \times GAMA^*$ je definirana podobno kot v poglavju 2 (v zvezi z dvosmernimi končnimi avtomati) oziroma v poglavju 5 (v zvezi s skladovnimi avtomati). Pomen trenutnega opisa $\langle alf_1, q, alf_2 \rangle$ pa je, da alf_1 predstavlja tisti del traku, ki je levo od okna, q je trenutno stanje nadzorne enote, prvi simbol alf_2 predstavlja položaj okna, alf_2 pa je vsebina dela traku, ki se začne pri oknu, končuje pa se z zadnjim simbolom, ki je različen od B. Če je $alf_1 = \epsilon$, je okno na skrajno levi celici traku. Če je $alf_2 = \epsilon$, je vsebina celice pod oknom kakor tudi vseh celic desno od njega enaka B. (Simbol B pa vseeno lahko nastopa v $alf_1 o alf_2$.) Da si omogočimo pisanje trenutnih opisov brez oklepajev in vejic, se domenimo, da sta si Q in GAMA medsebojno tuji.

Nadaljujemo enako kot v citiranih primerih: najprej na množici TO definiramo relacijo $|-_M$, pri čemer $x |-_M y$ beremo kot "y sledi iz x z enim korakom delovanja stroja M" ali krajše "x neposredno daje y". Recimo, da je $Z_1 = X_1 X_2 \ldots X_{i-1} q X_i \ldots X_n$

trenutni opis. Naj velja $dlt(q,X_i) = \langle p,Y,L\rangle$, pri čemer v primeru $i = n + 1$ velja $X_i = B$. V primeru $i = 1$ ne eksistira $Z_2 \in TO$, za katerega bi veljalo $Z_1 \vdash_M Z_2$ zato, ker oknu ne dovolimo, da zaide s traku v levo. Torej domenili smo se, da se v takem primeru izračun neha. V primeru $i > 1$ velja

$$X_1 X_2 \ldots X_{i-1} q X_i \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-2} p X_{i-1} Y X_{i+1} \ldots X_n \qquad (7.1)$$

Domenimo se tudi, da v primeru, ko je neki rep zaporedja $X_1 X_2 \ldots X_{i-2} p X_{i-1} Y X_{i+1} \ldots X_n$ sestavljen iz samih presledkov B, tako zaporedje v (7.1) preprosto izpustimo. Če pa velja $dlt(q,X_i) = \langle p,Y,D\rangle$, potem imamo

$$X_1 X_2 \ldots X_{i-1} q X_i X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-1} Y p X_{i+1} \ldots X_n \qquad (7.2)$$

Opazimo lahko, da je v primeru $i - 1 = n$ zaporedje $X_i \ldots X_n$ prazno in je desna stran (7.2) daljša od leve.

Tranzitivno ovojnico relacije $\vdash_M$ zaznamujemo z $\vdash_M^*$. Kot že vemo, $x \vdash_M^* y$ pomeni, da eksistira k in zaporedje $x = x_0, x_1, \ldots, x_k = y$ z lastnostjo

$$(x_0 \vdash_M x_1) \text{ and } (x_1 \vdash_M x_2) \text{ and } \ldots \text{ and } (x_{k-1} \vdash_M x_k)$$

oziroma, da y sledi iz x po k korakih delovanja stroja M.

Jezik nekega Turingovega stroja M definiramo kot

$$L(M) = \{w \mid w \in SIGMA^* \text{ and } (q_0 w \vdash_M^* w_1 q_1 w_2 \text{ pri nekih } w_1, w_2 \in GAMA^*, q_1 \in F)\}$$

Če je podan TS M, ki sprejema neki jezik, potem velja, da se po sprejetju neke besede x M ustavi, vendar v primeru $x \notin L(M)$ (torej ko M ne sprejme x) se lahko zgodi, da se M ne ustavi.

7.2. PRIMER. Tu podajamo opis Turingovega stroja M, ki sprejema jezik $L = \{0^n 1^n \mid n \geq 1\}$. Na začetku vsebuje trak stroja M zaporedje $0^n 1^n$, ki mu sledi neskončno presledkov. M deluje tako, da ponavlja naslednji osnovni cikel: najprej zamenja skrajno levi znak 0 z znakom X. Za tem se premakne na desno do prvega znaka 1, in le-tega zamenja z znakom Y, po čemer gre zopet v levo in poišče skrajno desni znak X ter se nato premakne v desno do skrajno levega znaka 0. Tu je konec cikla. V primeru, ko pri iskanju znaka 1 stroj naleti na presledek, se M ustavi ne, da bi vhod sprejel. V primeru pa, ko po najdenem znaku 1 stroj M več ne more najti znaka 0, stroj preveri, ali je tudi znakov 1 zmanjkalo in v takem primeru vhod sprejme, sicer

ga ne sprejme.

Naj imamo Q = {q_0, q_1, q_2, q_3, q_4}, SIGMA = {0, 1}, GAMA = {0,1,X,Y,B} in F = {q_4}. Neformalni pomen stanj je, da predstavljajo stavke oziroma skupine stavkov nekega programa. Stanje q_0 je začetno stanje in stanje pred zamenjavo skrajno levega znaka 0 z znakom X. V stanju q_1 iščemo proti desni prvi znak 1. V primeru, ko M najde znak 1 in ga spremeni v Y pride v stanje q_2. V stanju q_2 išče M proti levi prvi znak X. Ko M najde X, se postavi v stanje q_0 ter premakne v desno v poskusu poiskati skrajno levi znak 0. Ko M v stanju q_1 išče proti desni in naleti na znak B ali X pred znakom 1, se vhod ne sprejme. V tem primeru je bodisi preveč znakov 0 ali pa vhod ni oblike $0^* 1^*$.

Stanje q_0 ima tudi dodatno vlogo. V primeru, ko stanje q_2 najde skrajno desni simbol X in je desno od njega simbol Y, je to znak, da simbolov 0 ni več. V tem primeru, ko je M v stanju q_0 in bere simbol Y, se opravi prehod v stanje q_3, ki preišče zaporedje znakov Y in preveri, da ni nobenega znaka 1. V tem primeru, ko za zaporedjem znakov Y stoji B, se opravi prehod v končno stanje q_4. V nasprotnem primeru se vhodna beseda ne sprejme. Funkcija prehodov dlt je prikazana na sl. 7.2, sl. 7.3 pa prikazuje izračun stroja M pri vhodu 0011. Če si

ogledamo obe sliki, vidimo n.pr., da prvi korak nastane zaradi dlt(q_0,0) = <q_1,X,D>, zadnji korak pa zaradi dlt(q_3,B) = <q_4,B,D>. Bralec naj po možnosti simulira stroj M na nekaterih vhodih, ki jih M odkloni, n.pr. 001101, 001 ali 011.

Stanje	Simbol 0	1	X	Y	B
q_0	<q_1,X,D>	--	--	<q_3,Y,D>	--
q_1	<q_1,0,D>	<q_2,Y,L>	--	<q_1,Y,D>	--
q_2	<q_2,0,L>	--	<q_0,X,D>	<q_2,Y,L>	--
q_3	--	--	--	<q_3,Y,D>	<q_4,B,D>
q_4	--	--	--	--	--

sl. 7.2 Funkcija prehodov dlt

$q_0$0011 |- Xq_1)11 |- X0$q_1$1 |- X$q_2$0Y1 |-

q_2X0Y1 |- X$q_0$0Y1 |- XXq_1Y1 |- XXY$q_1$1 |-

XXq_2YY |- Xq_2XYY |- XXq_0YY |- XXYq_3Y |- XXYYq_3 |- XXYYBq_4

sl. 7.3 Primer izračuna stroja M

7.3 Izračunljivi jeziki in funkcije

Jezikom, ki jih sprejemajo Turingovi stroji pravimo kar Turingovi jeziki. Ta jezikovni razred je izredno obsežen in (na primer) strogo vsebuje kontekstno neodvisne jezike.

Med Turingovimi jeziki so tudi takšni, pri katerih se ne moremo algoritmično odločiti o pripadnosti (torej ne moremo odgovoriti z "da" ali "ne" na vprašanje, ali velja $x \in L$ pri poljubnem x). V primeru, ko je L tak jezik, se noben Turingov stroj M, ki sprejema L (torej $L = L(M)$), ne more ustaviti prav na vseh elementih komplementa $\overline{L}(M)$. Če velja $w \in L$, potem se M zagotovo ustavi na w. Vendar dokler pa M teče, ne moremo (v splošnem primeru) nikakor presoditi, ali bi se M ustavil, če bi ga pustili delovati dovolj dolgo.

Iz razreda Turingovih jezikov bomo izločili poseben podrazred, ki mu pravimo rekurzivni jeziki, in ki se odlikuje po tem, da za vsak tak jezik eksistira neki Turingov stroj, ki se ustavi na vseh vhodnih besedah (torej tudi na takih, ki ne pripadajo jeziku stroja). Kasneje bomo videli, da so rekurzivni jeziki strogi podrazred Turingovih jezikov. Lahko omenimo n.pr., da iz algoritma na sl. 6.7 sledi, da so kontekstno neodvisni jeziki podrazred rekurzivnih jezikov.

Turingov stroj kot računalnik funkcij

Turingov stroj lahko poleg sprejemanja jezikov računa tudi vrednosti funkcij $NN^k \to NN$ pri končnem k. Običajno je, da naravna števila predstavljamo v unarnem zapisu. To pomeni, da naravno število i zapišemo kot 0^i. V primeru več argumentov posamezne argumente ločimo z znakom 1: $0^{i(1)}10^{i(2)}1 \ldots 10^{i(k)}$.

Če se TS ustavi (ni važno ali je stanje končno), in je na traku zapisano 0^m pri nekem m, potem pravimo, da je $f(i_1, i_2, \ldots, i_k) = m$, kjer je f k-argumentna funkcija, ki jo računa ta stroj. Zanimiva je ugotovitev, da en in isti TS M računa eno funkcijo enega argumenta, drugo funkcijo dveh argumentov in tako naprej. Prav tako lahko ugotovimo, da ni nujno, da ima funkcija, ki jo računa M, pri vseh k-tericah argumentov definirane vrednosti.

Funkcijam, ki jih računajo Turingovi stroji, pravimo parcialno rekurzivne funkcije (čeprav je lahko neka parcialno rekurzivna funkcija pravzaprav totalna). V primeru, ko je $f(i_1, i_2, \ldots, i_k)$ definirana pri vseh k-terkah $\langle i_1, i_2, \ldots, i_k \rangle$ pa pravimo, da je f totalna rekurzivna funkcija. V nekem smislu ustrezajo parcialno rekurzivne funkcije Turingovim jezikom, ker ni nujno, da se ustrezni stroj ustavi medtem, ko totalno rekurzivne funkcije ustrezajo rekurzivnim jezikom, ker se v tem primeru stroj vedno ustavi. Ugotovimo lahko, da so vse običajne aritmetične funkcije, ki so definirane na naravnih številih, kot

n.pr. množenje, n!, $\lfloor \log_2 n \rfloor$ in exp(2,exp(2,n)) totalne rekurzivne funkcije.

7.3. PRIMER. Čisto odštevanje $m \dot{-} n$, ki je definirano kot

$$m \dot{-} n = \begin{cases} m - n \text{ pri } m \geq n \\ 0 \text{ sicer} \end{cases}$$

Če Turingov stroj

$$M = \langle \{q_0, q_1, \ldots, q_6\}, \{0,1\}, \{0,1,B\}, dlt, q_0, B, \emptyset \rangle,$$

ki je definiran spodaj, pošljemo na vhodu $0^m 10^n$, se ustavi s tračno vsebino $0^{m \dot{-} n}$. M deluje tako, da ponavlja osnovni cikel, ki ga sestavlja (1) zamenjava skrajno levega znaka 0 s presledkom, (2) iskanje zaporedja 10, (3) zamenjava znaka 0 z 1 in (4) premik proti levi do presledka. Ponavljanje se konča v dveh primerih:

(i) Takrat ko pri desnem premikanju in iskanju znaka 0 M sreča presledek. To pomeni, da je M vseh n ničel v zaporedju $0^m 10^n$ že spremenil v 1, medtem ko je n + 1 izmed m ničel spremenil v presledek B. V tem primeru M spremeni n + 1 znakov 1 v eno samo ničlo ter n znakov B, kar pomeni, da je na traku ostalo m - n znakov 0.

(ii) Takrat ko na začetku cikla M ne najde znaka 0, ki bi ga spremenil v presledek, ker je že prej vseh m prvih ničel bilo spremenjenih. Torej $n \geq m$ in $m \dot{-} n = 0$. V tem primeru M spremeni vse preostale znake 1 in 0 v presledek B.

Sledi opis funkcije dlt:

(1) $dlt(q_0, 0) = \langle q_1, B, D \rangle$

(Začetek cikla in spreminjanje začetnega znaka 0 v B.)

(2) $dlt(q_1, 0) = \langle q_1, 0, D \rangle$
$dlt(q_1, 1) = \langle q_2, 1, D \rangle$

(Premikanje proti desni in iskanje prvega znaka 1.)

(3) $dlt(q_2, 1) = \langle q_2, 1, D \rangle$
$dlt(q_2, 0) = \langle q_3, 1, L \rangle$

(Premikanje v desno po znakih 1 do prvega znaka 0. Slednji znak M spremeni v 1.)

(4) $dlt(q_3, 0) = \langle q_3, 0, L \rangle$
$dlt(q_3, 1) = \langle q_3, 1, L \rangle$
$dlt(q_3, B) = \langle q_0, B, D \rangle$

(Premikanje v levo do presledka. Prehod v stanje q_0 in začetek novega cikla.)

(5) $dlt(q_2, B) = \langle q_4, B, L \rangle$
$dlt(q_4, 1) = \langle q_4, B, L \rangle$
$dlt(q_4, 0) = \langle q_4, 0, L \rangle$
$dlt(q_4, B) = \langle q_6, 0, D \rangle$

(Če v stanju q_2 M sreča B predno sreča 0, ustreza to primeru (i) zgoraj. Tedaj se M postavi v stanje q_4 ter se premika v levo in spreminja vse znake 1 v B ter preskoči ničle do znaka B. Slednji presledek M spremeni nazaj v 0, se postavi v q_6 in se ustavi.)

(6) $dlt(q_0,1) = \langle q_5,B,D\rangle$
$dlt(q_5,0) = \langle q_5,B,D\rangle$
$dlt(q_5,1) = \langle q_5,B,D\rangle$
$dlt(q_5,B) = \langle q_6,B,D\rangle$

(Če v stanju q_0 M sreča znak 1 predno sreča 0, to pomeni, da je M v prvem argumentu že spremenil vse ničle, kar pa ustreza primeru (ii) zgoraj. Tedaj se M postavi v q_5, izbriše ostanek traku, po čemer se postavi v q_6 ter se ustavi.)

Sledi primer izračuna stroja M na vhodu 0010:

$q_0 0010 \vdash B q_1 010 \vdash B0 q_1 10 \vdash B01 q_2 0 \vdash$

$B0 q_3 11 \vdash B q_3 011 \vdash q_3 B011 \vdash B q_0 011 \vdash$

$BB q_1 11 \vdash BB1 q_2 1 \vdash BB11 q_2 \vdash BB1 q_4 1 \vdash$

$BB q_4 1 \vdash B q_4 \vdash B0 q_6$

Če pa je vhod 0100, deluje M takole:

$q_0 0100 \vdash B q_1 100 \vdash B1 q_2 00 \vdash B q_3 110 \vdash$

$q_3 B110 \vdash B q_0 110 \vdash BB q_5 10 \vdash BBB q_5 0 \vdash$

$BBBB q_5 \vdash BBBBB q_6$

7.4 Konstrukcija Turingovih strojev

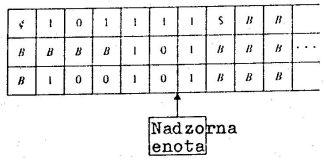
Na začetku tega poglavja smo zapisali trditev, da je Turingov stroj, kljub svoji preprostosti, najmočnejši možen avtomat. Z drugimi besedami, kar se sploh da izračunati, se da izračunati s Turingovim strojem (Churchova hipoteza). V resničnost te trditve se prepričamo tako, da, prvič, "sprogramiramo" čim več nalog na Turingovem stroju (s čimer se prepričamo v njegovo moč) in, drugič, da pokažemo, da so navidez drugačni stroji pravzaprav kvečjemu enakovredni Turingovemu stroju.

Najprej se bomo lotili problema programiranja oz. sestavljanja Turingovih strojev. Bralec se je že lahko prepričal v to, da je Turingov stroj precej okorna naprava (sicer ne bi bil preprost za preučevanje) in zato potrebujemo "bližnice", ki nam nekoliko olajšajo delo. Tukaj bomo na kratko opisali nekaj trikov, ki nam poenostavljajo sestavljanje Turingovih strojev.

Uporaba nadzorne enote kot pomnilnika. Eden od osnovnih prijemov pri sestavljanju Turingovih strojev je simulacija prirejanja x := A, kjer je A neki simbol tračne abecede. V tem primeru ima pri navadnem Turingovem stroju, ki simulira prirejanje, množica stanj obliko Q = GAMA x K, kjer je GAMA tračna abeceda, K pa neka množica "čistih" stanj. Stavek x := A, kombiniran s prehodom q --> r in zamenjavo A --> B ter smerjo gibanja d pa simuliramo s prehodom dlt(<A,q>,B) = <<B,r>,C,d>.

Bralec lahko za vajo poskuša sestaviti Turingov stroj, ki sprejema jezik, ki je sestavljen iz besed z lastnostjo, da prvi znak besede ne nastopa drugje v besedi.

Večsledni trak. Pogosto je ugodno, če si predstavljamo trak kot, da je razdeljen na več sledi (sl. 7.4). To dosežemo tako, da tračno abecedo GAMA definiramo kot kartezični produkt $GAMA_1 \times GAMA_2 \times \ldots \times GAMA_k$ abeced posameznih sledi.



sl. 7.4 Večsledni Turingov stroj

Za vajo naj bralec sestavi Turingov stroj, ki preveri, ali je število, ki je zapisano na traku v dvojiškem zapisu, praštevilo. (Napotek: uporabljamo trisledni trak, pri čemer je v prvi sledi zapisano vhodno število, v drugi sledi je zapisano neko število, ki je manjše od vhodnega, in v tretji sledi računamo večkratnike števila iz druge sledi ter preverjamo, ali obstaja večkratnik, ki je enak vhodnemu številu.)

Prestavljanje vsebine traku. Zelo pogosto želimo izprazniti določen del traku tako, da vse znake, ki niso presledki, prestavimo v desno za neko določeno število celic.

Kot primer naj bralec poskuša sestaviti dvosledni Turingov stroj, ki sprejema jezik $L = \{x^*x^n \mid x \in \{0, 1\}, n \geq 0\}$.

Podprogrami. O pomembnosti podprogramov pri programiranju ni potrebno izgubljati besed. Omenimo pa naj, da podprogrami obstajajo v nekaj različicah, od katerih nekatere zahtevajo razmeroma zapletene mehanizme za svojo realizacijo.

Najpreprostejši so nerekurzivni podprogrami brez parametrov. Realiziramo jih tako, da "povratne naslove" oziroma stanja hranimo v nadzorni enoti, podobno kot smo tam hranili tračne simbole. V tem primeru se klic podprograma sestoji iz prirejanja povratnega naslova ustrezni spremenljivki in prehoda v začetno stanje podprograma.

Za primer naj bralec poskuša sestaviti podprogram, ki prestavi vsebino traku do prvega presledka, desno od trenutne vsebine.

7.5 Različice Turingovih strojev

V prejšnem razdelku smo se seznanili s prijemi, ki nam omogočajo, da z več ali manj napora sprogramiramo kakršenkoli problem v obliki Turingovega stroja. Če opravimo večje število takih nalog pridobimo zaupanje v zmogljivosti Turingovega stroja in smo pripravljeni verjeti v Churchovo hipotezo. Vendar pravo zaupanje vanjo dobimo šele, ko se nam posreči dokazati, da so mnogi navidez drugačni računalniki pravzaprav enakovredni Turingovemu stroju. Postopek pri takih dokazih je vedno enak: Naj bo definiran neki razred strojev MM, za katerega želimo pokazati, da je enakovreden Turingovim strojem. Potem je potrebno iznajti neko sistematično proceduro, ki nam za neki poljuben konkreten stroj M iz razreda MM poišče ustrezni Turingov stroj M', ki je sposoben modelirati ali simulirati M, oziroma z drugimi besedami, izračunati isti rezultat. Postopek bomo prikazali na nekaj primerih.

Stroji z dvosmernim trakom. Edina razlika med takimi stroji in Turingovimi stroji iz definicije 7.1 je v tem, da je trak stroja neskončen v levo in desno. Na začetku je vhodna beseda zapisana na nekem mestu na traku, ostale celice vsebujejo simbol B, okno pa je postavljeno na skrajno levi znak vhodne besede. V tem primeru je gibanje okna možno tudi v levo od začetnega položaja. Formalizacija takega modela je pravzaprav celo preprostejša od definicije navadnega Turingovega stroja, saj je le potrebno opustiti določilo, da se pri prehodu leve meje traku izračun neha. Kako pokažemo, da tak stroj ni močnejši od stroja z enosmernim trakom? (Očitno pa tudi ni šibkejši.)

Naj bo M neki dvotračni stroj. Potem lahko opišemo eno različico "enosmernega" simulatorja M' stroja M takole: enosmerni trak stroja M' razdelimo na dve sledi, zgornjo in spodnjo, od katerih ima vsaka abecedo, ki je enaka tračni abecedi stroja M. Poleg tega imamo še prvotno vhodno abecedo SIGMA, kajti vhod mora biti zapisan v prvotni obliki, in končno imamo poseben simbol #, ki ni vsebovan v tračni abecedi stroja M, in ki naznanja levo mejo na traku. Zgornja sled vsebuje trak stroja M, od začetnega položaja čitalne glave na desno, spodnja pa vsebuje trak stroja M od začetnega položaja čitalne glave na levo. Skrajno leva celica pa vsebuje posebni simbol #, ki oznanja levi rob traku in rabi kot signal, da je potrebno pri čitanju vsebine traku v primeru gibanja v levo zamenjati sled. Torej stroj M' deluje tako, da bere vedno iz ene tračne sledi (razen na začetku, ko prepisuje vhodno besedo v dvosledno

obliko); ko pa pri premiku v levo zadene ob posebni znak, ki oznanja levi rob traku, se zamenja sled na traku in smisel smeri gibanja (L dobi pomen desnega premika, D pa pomen levega premika). Formalizacija tega modela je preprosta, vendar jo tu podajamo kot primer -- kasneje se v take podrobnosti ne bomo več spuščali. Torej je podan dvosmerni TS M = $\langle Q,SIGMA,GAMA,dlt,q_0,B,F\rangle$. Ustrezni enosmerni simulator M' definiramo takole:

$$M' = \langle Q',SIGMA,GAMA',\langle B,B\rangle,dlt',q_0',F'\rangle,$$

pri čemer velja:

$$Q' = \{r_0,r_1\} \sqcup \{r_X \mid X \in SIGMA\} \sqcup \{a, b\} \times Q$$

$$GAMA' = \{\#\} \sqcup SIGMA \sqcup GAMA \times GAMA$$

$$q_0' = q_0$$

$$F' = \{a\} \times F \sqcup \{b\} \times F,$$

dlt' pa definiramo takole: najprej imamo določeno število prehodov, katerih naloga je prepisati vhod v dvosledno obliko.

$dlt'(r_0,X) = \langle r_X,\#,D\rangle$ pri vseh $X \in SIGMA$

$dlt'(r_X,Y) = \langle r_Y,\langle X,B\rangle,D\rangle$ pri vseh $X, Y \in SIGMA$

$dlt'(r_X,\langle B,B\rangle) = \langle r_1,\langle X,B\rangle,L\rangle$ pri vseh $X \in SIGMA$

$dlt'(r_1,X) = \langle r_1,X,L\rangle$ pri vseh $X \in SIGMA \times \{B\}$

$dlt'(r_1,\#) = \langle\langle a,q_0\rangle,\#,D\rangle$

nato vsakemu prehodu $dlt(q,X) = \langle r,Y,d\rangle$ stroja M priredimo prehoda

$dlt'(\langle a,q\rangle,\langle X,Z\rangle) = \langle\langle a,r\rangle,\langle Y,Z\rangle,d\rangle$ pri vseh $Z \in GAMA$ in
$dlt'(\langle b,q\rangle,\langle Z,X\rangle) = \langle\langle b,r\rangle,\langle Z,Y\rangle,f\rangle$ pri vseh $Z \in GAMA$,

pri čemer je f nasprotna smer od smeri d. Poleg tega vpeljemo še prehoda

$dlt'(\langle a,q\rangle,\#) = \langle\langle b,q\rangle,\#,D\rangle$ in
$dlt'(\langle b,q\rangle,\#) = \langle\langle a,q\rangle,\#,D\rangle$

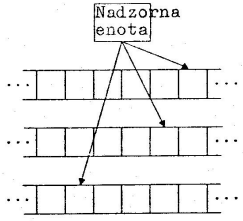
Sedaj je definicija M' popolnoma določena. Seveda ostane še naloga dokazati, da je M' enakovreden (ekvivalenten) stroju M. Kako pa tak dokaz izgleda? Pravzaprav v tem primeru preprosto "vidimo", da to velja, in smo s tem zadovoljni. Vendar se je prav tudi pri taki stvari vsaj enkrat zamisliti in se vprašati, kako bi tako reč tudi formalno dokazali. Bistvo dokaza je, da zgradimo bijekcijo med trenutnimi opisi stroja M in trenutnimi opisi stroja M' (vsaj tistimi trenutnimi opisi M', ki ustrezajo dvosledni fazi delovanja M') in sicer tako bijekcijo, ki ohranja relacijo $\vdash$. Tu naletimo na majhno težavo; namreč med trenutnimi opisi M nobeden ne ustreza opisom M' oblike alfq#bta. Seveda lahko težavo odpravimo tako, da v opisani bijekciji omenjeni nevšečni opis priključimo svojemu predhodniku. Druga rešitev pa je, da posebni znak # vključimo v sosedno celico traku tako, da uporabimo posebno kopijo abecede GAMA x GAMA. Na koncu še pokažemo, da končni trenutni opisi stroja M' natanko ustrezajo končnim trenutnim opisom stroja M, kar zagotavlja, da, ko stroj M' pride v končno stanje, bi se to zgodilo tudi s strojem M. Torej upravičeni smo zapisati

7.4. IZREK. Za neki jezik L obstaja Turingov stroj z dvosmernim trakom natanko takrat, ko zanj obstaja tudi navaden Turingov stroj.

Večtračni Turingovi stroji. Večtračni Turingov stroj je prikazan na sl. 7.5. Vsebuje nadzorno enoto s k okni ter k trakovi. Vsak trak je neskončen v obe smeri. Na vsakem koraku stroj opravi naslednje operacije v odvisnosti od stanja nadzorne enote ter od znakov, ki jih pokrivajo okna:

1. Sprememba stanja.
2. Zamenjava znakov na vseh trakovih.
3. Levi ali desni premik ali pa mirovanje vseh čitalnih glav, neodvisno ene od drugih.

Na začetku je vhod zapisan na prvem traku, ostali so prazni. V podrobno formalizacijo se ne bomo spuščali, saj je bil postopek prikazan v prejšnem razdelku na primeru dvosmernih Turingovih strojev.



sl. 7.5 Večtračni Turingov stroj

Sedaj pa lahko zapišemo

7.5. IZREK. Za neki jezik L obstaja večtračni Turingov stroj natanko takrat, ko zanj obstaja tudi navaden Turingov stroj.

Dokaz. Očitno lahko vsak večtračni stroj simulira enotračnega. Torej je potrebno dokazati le nasprotno. Podan je večtračni Turingov stroj M. Postopamo natanko kot v prejšnem razdelku in sestavimo navadni stroj M', ki oponaša stroj M. M' uporablja 2k-sledni trak (seveda mora M' najprej prepisati vhod v 2k-sledno obliko podobno, kot je to moral storiti simulator dvosmernega stroja v prejšnem razdelku), pri čemer vsakemu traku stroja M ustrezata dve sledi traku stroja M'. Abeceda ene od dveh sledi je enaka tračni abecedi stroja M in ta sled rabi za predstavljanje vsebine ustreznega traku stroja M. Druga sled

ima le dva simbola, B in X, pri čemer X rabi kot znamka za položaj okna (gl. sl. 7.5). Pri simulaciji enega koraka delovanja stroja M opravi M' naslednje: najprej poišče položaje vseh oken (kako to stori prepuščamo bralcu za vajo), ter si zapomni vsebino ustreznih celic, kot je to opisano v razdelku 7.4, stran 181, za tem izračuna potreben prehod stroja M ter končno zamenja vsebino celic in po potrebi premakne znamke, ki ustrezajo čitalnim glavam. |_|

7.6. PRIMER. Primer naloge, ki jo večtračni stroj opravi mnogo učinkoviteje od enotračnega je razpoznavanje jezika $L = \{ww^R \mid w \in SIGMA^*\}$. Če uporabljamo enotračni stroj, moramo s čitalno glavo potovati od enega konca do drugega in na vsakem prehodu primerjati en simbol. Vhodno besedo tak stroj sprejme, če so znaki na ustreznih mestih enaki. Zaradi potovanja sem in tja porabi enotračni stroj, število korakov, ki je sorazmerno kvadratu dolžine vhodne besede. Na dvotračnem stroju pa za isto opravilo porabimo linearen čas, ker je potrebno le prekopirati vhod še na drugi trak, nato pa s sočasnim gibanjem čitalnih glav v nasprotnih smereh primerjati simbole na isti razdalji od obeh koncev.

Nedeterministični Turingovi stroji. Morda je najpomembnejša različica Turingovih strojev nedeterministični stroj. V tem primeru nimamo funkcije prehodov temveč korespondenco prehodov. Z drugimi besedami pri enem stanju in pri določeni vsebini celice, ki jo pokriva okno, ima stroj morebiti na izbiro več naslednjih stanj (in ustrezno temu tudi več zamenjav vsebine traku ter več smeri gibanja). Zopet se izogibamo natančni formalizaciji tega modela, ker sedaj ni več potrebno zahajati v take podrobnosti. Tudi v tem primeru imamo

7.7. IZREK. Za neki jezik L obstaja nedeterminističen Turingov stroj natanko takrat, ko zanj obstaja navaden Turingov stroj.

Dokaz. Podan je nedeterminističen Turingov stroj M. Navadni Turingov stroj M', ki simulira stroj M, uporablja trisledni trak. V prvi sledi je zapisan vhod, v drugi je zapisano navodilo za delovanje M in v tretji se dejansko opravlja simulacijo tako, da se računajo zaporedni trenutni opisi stroja M. Navodilo za delovanje je dolgo toliko kot je korakov v izračunu stroja M, ki ga trenutno M' simulira. Vsaki kombinaciji stanja in tračnega simbola pri stroju M ustreza neka končna množica trojic oblike $\langle q,X,d \rangle$, kjer je q neko stanje, X je tračni simbol in d je smer gibanja. Denimo, da je v neki taki množici največ k elementov in da so vse urejene. Tedaj abeceda druge sledi vsebuje k znakov (poleg B), znak a_i, $1 \leq i \leq k$, na n-tem mestu v drugem kanalu pa pomeni, da je v simulaciji n-tega koraka potrebno vzeti i-ti element v množici

prehodov. Če takega elementa ni (če je v tem primeru moč množice manjša od i), se domenimo, da se izračun neha. Sedaj pa je potrebno po vrsti generirati (pri nekem podanem vhodu) vsa možna navodila v drugi sledi dolžin 1, 2, ... , n, ... in opraviti ustrezne simulacije v tretjem kanalu. Očitno je, da, če M sprejme podani vhod, ga bo sprejel v določenem številu korakov recimo N, pač pri neki izbiri dovoljenih prehodov. Tedaj bo to dejstvo M' ugotovil pri nekem navodilu dolžine N in ker generira prav vsa navodila dolžine N, ga bo zagotovo našel. Ugotovimo lahko, da se M' pri vseh simulacijah ustavi, kajti simulacija se nadaljuje le za število korakov, ki je enako dolžini navodila. Če po tolikšnem številu korakov M ne sprejme vhoda (pri dani izbiri prehodov), se simulacija preneha in se generira naslednje navodilo. |¯|

Nekaj dodatnih različic Turingovih strojev. Z opisanimi različicami še zdaleč nismo izčrpali vseh modelov računalnikov, ki so jih raziskovali, in ki se v različnih okoliščinah pokažejo koristni. Tukaj naj na kratko opišemo še nekaj tipov.

Večdimenzionalni Turingovi stroji. V tem primeru nadzorna enota nima na voljo traku temveč k-dimenzionalno strukturo, katere celice lahko identificiramo s k-tericami naravnih števil (v kolikor želimo definirati "večsmerno" strukturo take vrste, uporabljamo k-terice celih števil). V celicah prav tako kot prej hranimo znake iz neke končne abecede, razlika do navadnega Turingovega stroja pa je v tem, da je gibanje možno po vsaki od k dimenzij (elementaren premik ustreza prištevanju ali odštevanju 1 ustrezni komponenti celičnega "naslova"). Tudi tak

stroj se seveda izkaže za enakovrednega Turingovemu stroju, bralec pa naj za vajo definira navaden Turingov stroj, ki opravlja simulacijo.

Turingov stroj z več okni. Definiran je kot navaden Turingov stroj le, da ima na enem traku več namesto enega okna, ki se lahko premikajo neodvisno eden od drugega. Tudi za tak stroj simulatorja ni težko narediti.

7.6 Churchova teza

Domneva, da lahko intuitiven pojem "izračunljive funkcije" poistovetimo z razredom parcialno rekurzivnih funkcij, je znana kot Churchova hipoteza ali Church-Turingova teza. Čeprav je ne moremo "dokazati", ker se sklicuje na neformalen pojem "izračunljivosti", pa jo vseeno lahko podkrepimo z vrsto podatkov, ki ji govorijo v prid. Če naš inutuitiven pojem "izračunljivega" ne omejuje števila računskih korakov ali obsega uporabljenega pomnilnika, potem se nam zdi, da so parcialno rekurzivne funkcije intuitivno izračunljive. Priznati pa je treba, da nekateri menijo, da neka funkcija ni "izračunljiva", če ne moremo vnaprej postaviti mejo na število korakov, ki je potrebno za njen izračun, ali pa vsaj ugotoviti, ali se njen izračun ustavi.

Manj pa je očitno, ali razred parcialno rekurzivnih funkcij obsega vse "izračunljive" funkcije. Logiki so preučevali tudi druge formalizacije "izračunljivih" funkcij kot so n.pr. lmd-račun, Postovi sistemi ter splošno rekurzivne funkcije. Za

vse se je izkazalo, da so istovetni s parcialno rekurzivnimi funkcijami. Poleg tega pa tudi abstraktni modeli računalnikov kot so n.pr. stroji z naslovljivim pomnilnikom (SNP) (angleško: Random Access Machines) porajajo parcialno rekurzivne funkcije.

SNP vsebuje neskončno pomnilnih celic, ki so oštevilčene z 0, 1, ..., in ki lahko vsebujejo poljubno celo število. Poleg tega ima končno mnogo aritmetičnih registrov, ki prav tako lahko vsebujejo poljubno število. Med ostalim lahko vsako celo število dekodiramo v enega od običajnih računalniških ukazov. Model SNP ne bomo definirali bolj podrobno, vendar je očitno, da, v primeru, ko je nabor ukazov primerno izbran, lahko s SNP simuliramo poljuben računalnik. Sedaj bomo podali dokaz, da je model Turingovega stroja enako močan kot model SNP.

Simulacija SNP s Turingovim strojem.

7.8. IZREK. **Turingov stroj lahko simulira SNP pod pogojem, da so tudi elementarni ukazi SNP izračunljivi po Turingu.**

Dokaz. Za simulacijo uporabljamo večtračni TS M. Eden od trakov stroja M vsebuje celice SNP, ki smo jim pripisali vrednost. Ta trak izgleda takole:

$$\#0*v_0\ \#1*v_1\ \#10*v_2\ \#\ldots\#i*v_i\ \#\ldots,$$

pri čemer je v_i vsebina i-te celice v dvojiškem zapisu. Do poljubnega trenutka je le končno mnogo celic SNP bilo uporabljenih in torej mora stroj M imeti v evidenci le vsebino celic, katerih indeks (naslov) ne presega največji indeks, ki je

bil med izračunom že uporabljen.

SNP vsebuje neko končno število aritmetičnih registrov. M uporablja za vsebino vsakega registra en trak, en trak za lokacijski števec, ki vsebuje indeks celice, iz katere naj bi vzeli naslednji ukaz, in en trak za naslovni register, v katerega lahko zapišemo indeks (naslov) neke pomnilne celice.

Denimo, da prvih 10 bitov ukaza predstavlja kodo za enega od običajnih računalniških ukazov, n.pr. LOAD, STORE, ADD i.t.n., in da preostali biti predstavljajo naslov operanda. Na tem mestu ne bomo obravnavali podrobnosti realizacije vseh standardnih računalniških ukazov, pač pa bomo uporabljeno tehniko pojasnili na primeru. Naj trak lokacijskega števca stroja M vsebuje število i v dvojiškem zapisu. Na začetku ukaznega cikla M preišče svoj pomnilni trak od leve proti desni, ali se pojavlja zaporedje #i*. Če med iskanjem naleti na presledek, je to znak, da celica i ne vsebuje ukaza, in se SNP (kakor tudi M) ustavi. Če je zaporedje #i* najdeno, se najprej opravi analiza zaporedja med znakom * in naslednjim znakom #. Kot primer vzemimo, da prvih deset bitov predstavlja ukaz "prištej k registru 2", preostali biti pa predstavljajo število j. V tem primeru poveča M vsebino traku lokacijskega števca za 1 ter prekopira j v trak naslovnega registra. Nato M preišče svoj pomnilni trak (od leve), ali vsebuje podzaporedje #j*o. V primeru, ko takega podzaporedja ni, predpostavljamo, da vsebuje celica j 0 in nadaljujemo z naslednjim ukazom SNP. V nasprotnem primeru pa se v_j prišteje k vsebini traku registra 2. S tem je ukazni cikel končan in se začne izvajati naslednji ukazni cikel.

Morda je umestna pripomba, da, čeprav smo uporabljali večtračni stroj, bi lahko, na podlagi izreka 7.5, uporabili tudi enotračnega.

Splošno rekurzivne funkcije. Poleg računalnikov lahko za pridobivanje "izračunljivih" funkcij uporabljamo tudi drugačne metode, ki so bolj neposredne in bolj v duhu matematične tradicije. Kot primer rabijo splošno rekurzivne funkcije, ki izhajajo iz določenega omejenega nabora izhodiščnih funkcij $NN^k \to NN$, $k \geq 0$, iz katerih je nato možno graditi nove funkcije s tremi pravili (kompozicija, primitivna rekurzija in minimalizacija).

7.9. DEFINICIJA. Splošno rekurzivna je vsaka funkcija $NN^k \to NN$, ki bodisi pripada naboru začetnih funkcij:

1. Ničelna funkcija $Z(x) = 0$ pri vseh x,
2. Naslednik $N(x) = x + 1$,
3. Projekcija $U^n_i(x_1,\ldots,x_n) = x_i$ pri vseh $n \geq 1$ in $1 \leq i \leq n$,

ali pa je pridobljena iz drugih splošno rekurzivnih funkcij s kompozicijo, primitivno rekurzijo ali minimalizacijo.

Za splošno rekurzivno funkcijo f pravimo, da je pridobljena s kompozicijo iz splošno rekurzivnih funkcij $g: NN^m \to NN$ in $h_i: NN^n \to NN$, pri $1 \leq i \leq m$, če velja

$$f(x_1,\ldots,x_n) = g(h_1(x_1,\ldots,x_n),\ldots,h_m(x_1,\ldots,x_n)).$$

Za splošno rekurzivno funkcijo f pravimo, da je pridobljena iz splošno rekurzivnih funkcij $g: NN^n \to NN$ in $h: NN^{n+2} \to NN$ s primitivno rekurzijo, če velja

$$f(x_1,\ldots,x_n,0) = g(x_1,\ldots,x_n)$$

ter

$$f(x_1,\ldots,x_n,y+1) = h(x_1,\ldots,x_n,y,f(x_1,\ldots,x_n,y))$$

pri vseh $y \in NN$.

Končno, naj bo $g: NN^{n+1} \to NN$ splošno rekurzivna funkcija. Potem za funkcijo f:

$$f(x_1,\ldots,x_n) = \mu y[g(x_1,\ldots,x_n,y)] = \begin{cases} \text{najmanjše } z \text{ pri čemer je} \\ \quad y, \text{ pri katerem velja } g(x_1,\ldots,x_n,y) = 0, \\ \text{če tako } y \text{ obstaja} \\ \text{nedefinirano sicer} \end{cases}$$

pravimo, da je pridobljena z minimalizacijo iz g.

Pripomba. Splošno rekurzivnim funkcijam, ki za svojo definicijo ne potrebujejo minimalizacije, pravimo primitivno rekurzivne funkcije.

Zopet lahko pokažemo, da velja

7.10. IZREK. Splošno rekurzivne funkcije natanko sovpadajo s parcialno rekurzivnimi funkcijami, oziroma funkcijami, ki jih računajo Turingovi stroji.

Dokaz tega izreka je dokaj zamuden in zato se vanj ne bomo spuščali. V eno smer bi ga sicer izpeljali brez težav. Namreč dejstvo, da splošno rekurzivne funkcije lahko računamo s Turingovimi stroji, se dokaže na ta način, da, prvič, pokažemo, da lahko vse začetne funkcije izračunamo s Turingovim strojem, in nato, da lahko realiziramo kompozicijo, primitivno rekurzijo ter minimalizacijo s Turingovim strojem. V drugi smeri je dokaz nekoliko bolj zapleten, saj moramo izraziti rezultat izračuna poljubnega Turingovega stroja z nekakšno splošno rekurzivno funkcijo. Bralec lahko najde tak dokaz n.pr. v delu Davisa [DM].

7.7 Turingovi stroji kot generatorji

Doslej smo opisali Turingove stroje kot sprejemnike jezikov ali kot računalnike funkcij. Včasih pa je koristno, če jih obravnavamo kot generatorje množic ali jezikov.

Turingov stroj kot generator je definiran kot dvotračni stroj, pri čemer enemu traku pravimo delovni, drugemu pa izhodni. Na izhodnem traku se okno lahko premika samo v desno. Na začetku sta oba traka prazna, okni pa sta postavljeni na skrajno levi celici. Ko stroj poženemo, le-ta uporablja delovni

trak kot delovni pomnilnik, na izhodni trak pa izpisuje besede (nad neko abecedo SIGMA), ki so med seboj ločene z nekakšnim posebnim znakom #. Seznam besed, ki ga stroj izpiše na izhodni trak potem, ko ga poženemo, je jezik, ki ga stroj generira. Seveda, v primeru, ko je jezik neskončen, se stroj ne bo nikoli ustavil, vendar je kljub temu s takšnim Turingovim generatorjem jezik enolično določen. Seveda lahko tudi v tem primeru definiramo različice Turingovih generatorjev kot n.pr. večtračne generatorje i.pd., vendar se v to ne bomo spuščali. V tem razdelku bomo le dokazali dva izreka, ki postavljata zvezo med Turingovimi stroji kot generatorji in Turingovimi stroji kot sprejemniki jezikov.

7.11. IZREK. Razred jezikov, ki jih je možno pridobiti s Turingovimi generatorji, je razred Turingovih jezikov.

Dokaz. Postopamo podobno kot pri drugih dokazih ekvivalence: najprej pokažemo, da je jezik nekega Turingovega generatorja, možno tudi sprejemati s Turingovim strojem, in nasprotno, vsak Turingov jezik je možno tudi generirati s Turingovim generatorjem.

Prvi del dokaza je trivialen: podan je Turingov generator M in jezik L = L(M). Potrebno je zgraditi Turingov stroj M', ki sprejema L in ki uporablja M kot "podprogram". Na vhodu stroja M' je beseda w. M' postopa tako, da sproži M in vsakič, ko M zgenerira neki element L, M' primerja w z generirano besedo. V primeru enakosti stroj M' besedo w sprejme, v nasprotnem primeru pa M nadaljuje z generiranjem jezika L. Če velja $w \in L$, se bo

stroj M' očitno ustavil in besedo w sprejel, v nasprotnem primeru pa bo besedo w bodisi odklonil (če je L končna množica), ali pa se sploh ne bo ustavil (če je L neskončna množica).

Drugi del dokaza pa je rahlo netrivialen. V tem primeru je podan Turingov stroj M, ki sprejema jezik L = L(M), potrebno pa je sestaviti generator M', ki generira isti jezik. Očitna metoda, da po vrsti generiramo vse besede w nad SIGMA in preverjamo z M, ali velja w ∈ L, ne deluje, kajti ni nujno, da se M ustavi na vseh elementih $\overline{L}$. V takem primeru bi se zgodilo, da ne bi mogli generirati tistih elementov L, ki nastopajo v naši razvrstitvi $SIGMA^*$ za prvim elementom $\overline{L}$, na katerem se M ne ustavi. Zato uporabljamo metodo, ki je v takšnih primerih standardna. Najprej bomo nehali govoriti o elementih $SIGMA^*$ in namesto tega govorili le o naravnih številih, kajti znano je, da lahko postavimo bijekcijo med naravnimi števili in $SIGMA^*$ (vsako besedo nad SIGMA pojmujemo kot naravno število v nekakšnem pozicijskem zapisu). Potem pa sestavimo generator parov naravnih števil <i, j> pri i, j ∈ NN. Tudi za to obstaja standarden prijem: števila generiramo po naraščajoči vrednosti i + j, pri določeni vrednosti te količine, pa jih generiramo po naraščajoči vrednosti i. Torej prvih nekaj elementov v tej razrstitvi je <0,0>, <0,1>, <1,0>, <0,2>, ... Sedaj pa generator M' deluje takole: najprej se sproži generator parov, kot smo ga pravkar opisali; ko pa je par <i, j> zgeneriran, se sproži stroj M z vhodom j za i korakov. Če M sprejme j natanko

v i-tem koraku, potem generator M' zapiše j na svoj izhodni trak. Sicer na izhodni trak ne izpiše ničesar in nadaljuje z generiranjem naslednjega para <i,j>. Ni se težko prepričati, da bo M' na svoj izhodni trak izpisal natanko elemente L. |_|

V razdelku 7.3 smo že omenili, da jezikom, ki jih $sprejemajo$ stroji, ki se ustavijo na vseh elementih $SIGMA^*$, pravimo rekurzivni jeziki. Postavlja se vprašanje, ali imajo Turingovi generatorji takšnih jezikov kakšno posebnost? Predno nanj odgovorimo, se domenimo, da izberemo nekakšno razvrstitev elementov $SIGMA^*$ in jo poimenujemo kanonska razvrstitev. Kanonska razvrstitev mora zadoščati le pogoju, da obstaja Turingov stroj, ki generira $SIGMA^*$ v takšnem zaporedju. V primeru SIGMA = {0, 1} je to n. pr. lahko razvrstitev po naraščajoči dolžini, pri podani dolžini pa razvrščamo besede v leksikografskem redu (denimo 0 < 1). Torej prvih nekaj elementov je ∈, 0, 1, 00, 01, 10, 11, 000, ... Potem imamo

7.12. IZREK. L je rekurziven jezik natanko v primeru, ko ga lahko generiramo v kanonskem vrstnem redu.

Dokaz. Najprej naj bo podan Turingov generator M nekega jezika L v kanonskem vrstnem redu. Za L želimo definirati Turingov sprejemnik M', ki se ustavi na vseh vhodih. Naj bo vhod stroja M' beseda w. Stroj M' sproži generator M in v primeru, ko M zgenerira w, M' besedo w sprejme in se ustavi. V nasprotnem primeru pa M' ugotovi dejstvo w ∉ L po tem, da se na

izhodu M pojavi beseda, ki je v kanonski razvrstitvi postavljena za w. Torej se tudi v tem primeru M' lahko ustavi s pravilno diagnozo.

Sedaj pa naj bo podan Turingov sprejemnik M jezika L. Kako sestaviti generator M' za L, ki bo besede generiral v kanonskem vrstnem redu? M' vsebuje generator $SIGMA^*$ v kanonski razvrstitvi in vsakič ko slednji generator zgenerira neki element $w \in SIGMA^*$, ga M' podtakne na vhod stroju M. M' potem izpiše w na svoj izhodni trak odvisno od tega, ali M sprejme w ali ne. V tem primeru ne moremo priti v težave, ker smo predpostavili, da se M vedno ustavi. |_|

7.8 Naloge

7.1 Sestavi Turingove stroje, ki sprejemajo naslednje jezike:

(a) $\{0^n 1^n 0^n \mid n \geq 1\}$.

(b) $\{ww^R \mid w \in (0 + 1)^*\}$.

(c) Množica besed z istim številom znakov 0 in 1.

7.2 Sestavi Turingove stroje, ki računajo naslednje funkcije:

(a) $\lfloor \log_2 n \rfloor$ (b) n! (c) n^2

7.3 Podan je Turingov stroj

$M = \langle Q, \{0,1\}, GAMA, dlt, q_0, B, F \rangle$.

Simulirajte ga s strojem, katerega tračna abeceda vsebuje le znake 0, 1 in B.

7.4 Poiščite rekurzivne definicije za naslednje funkcije:

(a) n + m (b) n ' m (c) n*m (d) n!

7.5 Sestavite Turingov generator za jezik $\{0^n 1^n \mid n \geq 1\}$.

8. POGLAVJE

NEODLOČLJIVOST

V tem poglavju bomo obravnavali rekurzivne in Turingove jezike. Najbolj nas bodo zanimali jeziki, ki so pravzaprav nekakšni zakodirani konkretni problemi. Vzemimo problem, ali neki poljuben Turingov stroj sprejema prazno besedo. Problem lahko preformuliramo v problem, ki se nanaša na jezike, če Turingove stroje kodiramo z besedami nad abecedo {0,1}. Množica vseh besed, ki predstavljajo kode Turingovih strojev, ki sprejemajo prazno besedo, je (kot bomo kasneje videli) Turingov jezik, vendar ni rekurziven. Na podlagi tega bomo sklepali, da ne eksistira algoritem, ki bi nam razodel, ali neki Turingov stroj sprejme prazno besedo ali ne.

Pokazali bomo, da za mnoga vprašanja o Turingovih strojih, kakor tudi nekatera vprašanja o kontekstno neodvisnih jezikih in drugih strukturah, ni algoritmov, ki bi odgovorili bodisi "da" ali "ne".

8.1 Problemi

V vsakdanjem pomenu se beseda problem nanaša na vprašanje kot je n.pr. "Ali je podana KNG dvoumna?" Konkretna oblika tega problema (ki mu pravimo problem dvoumnosti kontekstno neodvisnih gramatik) je neka določena gramatika, za katero nas zanima odgovor o dvoumnosti. Torej razlikujemo dva pojma: (splošen) problem, ki je neko vprašanje s toliko "da" oziroma "ne" odgovori, kolikor je elementov v množici, na katero se splošni problem nanaša, in konkreten problem, ki je vprašanje, ki se nanaša samo na en element množice. Če se omejimo na vprašanja z odgovori "da" in "ne" in če konkretne probleme kodiramo z besedami nad neko končno abecedo, potem vprašanje o eksistenci algoritmov za takšna vprašanja prevedemo na vprašanje o rekurzivnosti ustreznih jezikov.[1] Prvi hip se nam zdi, da smo se s tem dogovorom preveč utesnili. Vendar temu ni tako. Mnogi splošnejši problemi imajo da-ne različice, ki so prav tako težke kot splošni problem.

Kot primer vzemimo omenjeni problem dvoumnosti. Ta problem bomo označili z AMB (angleški izraz za dvoumnost je ambiguity). AMB ima splošnejšo različico, ki jo zaznamujemo s FIND, in pri kateri se zahteva ne le, da ugotovimo, ali je neka gramatika dvoumna, temveč, da v primeru dvoumnosti celo poiščemo stavek z dvemi ali več drevesi izpeljav. V primeru nedvoumnosti

---

[1] Poudarjamo, da je pojem algoritma (v grobem) sinonimen s pojmoma programa ali n.pr. Turingovega stroja; algoritem, ki na konkreten problem odgovori bodisi z "da" ali "ne" ustreza Turingovemu stroju, ki se vedno ustavi bodisi v končnem ali nekončnem stanju.

zahtevamo od FIND le, da odgovori z "ne". Za FIND pravimo, da je bolj splošen problem med ostalim tudi zato, ker lahko iz algoritma zanj sestavimo algoritem za AMB: v primeru, ko FIND odgovori z "ne", da isti odgovor tudi AMB; če pa FIND poišče besedo z več drevesi izpeljav, potem AMB preprosto odgovori "da". Presenetljivo pa je, da lahko iz algoritma za AMB prav tako sestavimo algoritem za FIND: ko AMB odgovori "ne", to stori tudi FIND; ko pa AMB odgovori "da" pa algoritem za FIND začne sistematično generirati vse možne besede nad abecedo gramatike, graditi vsa mogoča drevesa izpeljav, ter preverjati, ali je njihov rezultat obravnavana beseda. Bodimo pozorni, da z generiranjem besed počakamo dokler ne vemo, da je gramatika zagotovo dvoumna. To nam zagotavlja uspešnost postopka iskanja, ker se program v vsakem primeru ustavi. V podrobnosti postopka generiranja se na tem mestu ne bomo spuščali. Bralec lahko opisano nalogo izdela za vajo, ali pa naj počaka do konca poglavja, ko se bo seznanil z več primeri podobnih postopkov.

Sestavljanju algoritma za neki problem (kot n.pr. FIND) z uporabo domnevnega algoritma za neki drugi problem (n.pr. AMB) pravimo prevedba (problema FIND na AMB). V splošnem, ko problem A prevedemo na B, smo pokazali, da je B vsaj toliko težak kot A. Torej v primeru problemov FIND in AMB to pomeni, da različica da-ne ni preprostejša od bolj splošne oblike problema. (Več o pojmu prevedbe problemov kasneje.) Kasneje bomo pokazali, da za problem AMB ni algoritma. Zaradi zgoraj opisane prevedbe problema FIND na AMB pa sklepamo, da prav tako ni algoritma za FIND, kajti iz eksistence algoritma za FIND sledi eksistenca

algoritma za AMB, kar bi bilo protislovno s pravkar napovedano nemožnostjo algoritma za AMB.

V zvezi z opisanim problemom se je morda koristno ustaviti pri načinu kodiranja gramatik $G = \langle V,T,P,S \rangle$ z nizi nad neko končno abecedo, n.pr. {0, 1}. Predvsem obravnavamo vse možne gramatike G pri določeni množici T. Torej so elementi T vnaprej znani, isto pa velja tudi za spremenljivko S. Ostale spremenljivke so odvisne od konkretne gramatike. Če velja $T = \{a_1, a_2, \ldots, a_k\}$ in $V = \{A_1, A_2, \ldots, A_m\}$, potem zakodiramo S z zaporedjem 0^{i+1}, a_i z zaporedjem 0^{i+k+2}, A_i pa z 0^{i+2}.

Produkcijo $A \rightarrow X_{i(1)} X_{i(2)} \ldots X_{i(n)}$ zakodiramo z besedo

$z(A)1z(X_{i(1)})1z(X_{i(2)})1\ldots1z(X_{i(n)})$, kjer je z(X) koda znaka X.

Množico produkcij $P = \{P_1, P_2, \ldots, P_N\}$ in s tem tudi gramatiko G lahko zakodiramo v obliki

$111Z(P_1)11Z(P_2)11\ldots11Z(P_N)111$, kjer je $Z(P_i)$ koda produkcije P_i. Bodimo pozorni, da mnoge besede nad abecedo {0, 1} niso veljavne kode gramatik. Lahko se zgodi, na primer, da koda ustreza gramatiki s produkcijo, ki ima na svoji levi strani znak iz množice končnih simbolov T, ali pa koda sploh ni v pravilni obliki. Vendar, ker je zaželeno, da prav vsi nizi nad {0, 1} predstavljajo nekakšno gramatiko, se domenimo, da vse neveljavne kode predstavljajo neko vnaprej domenjeno gramatiko (n.pr. $\langle \{S\},T,\{S \rightarrow \epsilon\},S \rangle$.

Odločljivi in neodločljivi problemi. Za neki problem, čigar jezik je rekurziven, pravimo, da je odločljiv. Sicer je neodločljiv. Z drugimi besedami neki problem je neodločljiv, če ni algoritma, ki bi na vhodu sprejel neki konkreten problem in bi odgovoril bodisi z "da" ali "ne".

Presenetljiva posledica naših definicij je, da so problemi z enim samim konkretnim problemom trivialno odločljivi. Vzemimo naslednji problem, ki sloni na Fermat-jevi domnevi. Ali obstaja rešitev v pozitivnih celih številih za enačbo $x^i + y^i = z^i$ pri $i \geq 3$? Poudarjamo, da x, y, z in i niso parametri problema (po katerih bi se konkretni problemi medsebojno razlikovali) temveč vezane spremenljivke v problemu (torej takšne, na katere delujejo kvantifikatorji). Vemo, da obstajata dva Turingova stroja, eden od katerih odgovarja z "da" na vse vhode in drugi, ki odgovarja z "ne" na vse vhode. Prav gotovo bo eden od njih pravilno odgovoril na zastavljeno vprašanje, čeprav trenutno še ne vemo, kateri od teh dveh je to.

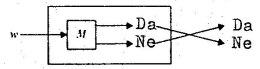
Bralca pa naj dejstvo, da je eden od najtršihmatematičnih orehov "odločljiv", ne moti. To le priča o tem, da smo si pri naših raziskavah izbrali za pomen besede "odločljiv" nekaj posebnega -- nekaj kar se nanaša na eksistenco ali neeksistenco algoritmov za probleme, ki imajo neskončno konkretnih primerov.

8.2 Osnovne lastnosti Turingovih jezikov

V tem razdelku bomo prikazali nekaj osnovnih lastnosti jezikovnih razredov, ki nas zanimajo. Pri tem se bomo srečali s preprosto obliko metod dokazovanja, ki jih bomo uporabljali tudi kasneje za izpeljavo mnogo zahtevnejših izrekov. Osnovni prijem je konstrukcija različnih Turingovih strojev, ko je znan neki drugi Turingov stroj. Za primer naj rabi

8.1. IZREK. Komplement $\overline{L}$ nekega rekurzivnega jezika je prav tako rekurziven.

Dokaz. Iz izhodiščne predpostavke sklepamo, da obstaja TS M_1, ki se vedno ustavi, z lastnostjo $L = L(M_1)$. Potem je stroj M_2, ki sprejema $\overline{L}$, prikazan na sl. 8.1. |_|

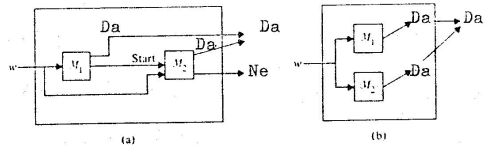


sl. 8.1 Stroj za komplement nekega rekurzivnega jezika

8.2. IZREK. Naj bosta L_1 in L_2 rekurzivna (Turingova) jezika. Potem je njihova unija $L_1 \cup L_2$ tudi rekurziven (Turingov) jezik.

Dokaz. Iz predpostavke sklepamo, da obstajata Turingova stroja M_1 in M_2 z lastnostima $L_1 = L(M_1)$ in $L_2 = L(M_2)$. Sedaj uporabimo M_1 in M_2 za sestavljanje stroja M z lastnostjo $L(M) = L_1 \sqcup L_2$. V primeru rekurzivnih L_1 in L_2, M dela tako, da svoj vhod w podtakne stroju M_1 in v primeru, ko le-ta besedo w sprejme, jo sprejme tudi M, v nasprotnem primeru, pa jo M poda na vhod stroja M_2. M v tej, drugi, fazi delovanja besedo w sprejme natanko, ko jo sprejme tudi M_2 (gl. sl. 8.2(a)). V primeru Turingovih L_1 in L_2 pa tako ne moremo postopati, ker ni nujno, da se M_1 ustavi na vhodu w. Problem rešimo tako, da uporabljamo generator celih pozitivnih števil 1, 2, ... in ko le-ta generira število j, sprožimo stroj M_1 za j korakov. Če M_1 sprejme w na j-tem koraku, potem tudi M sprejme vhod w (in se ustavi). Sicer sprožimo stroj M_2 za j korakov. Zopet M sprejme w v primeru, ko ga sprejme M_2 na j-tem koraku. Če niti M_1 niti M_2 ne sprejme vhoda na j-tem koraku, generiramo naslednje celo število (n.pr. j + 1) in postopek ponovimo. Očitno se M ustavi in vhodno besedo w sprejme natanko, ko jo sprejme bodisi M_1 ali M_2. Torej M sprejema jezik $L_1 \sqcup L_2$. Opisani izračun pojmujemo kot "vzporeden izračun M_1 in M_2" (gl. sl. 8.2(b)).

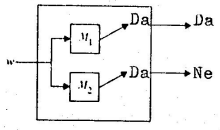
$\square$



sl. 8.2 Stroj za unijo dveh Turingovih jezikov

8.3. IZREK. Naj bosta jezik L in njegov komplement $\overline{L}$ Turingova jezika. Potem je L rekurziven.

Dokaz. Izrek bomo dokazali, če za L sestavimo Turingov stroj, ki se ustavi na vseh vhodih (ne glede na dejstvo, ali pripadajo jeziku L). Na podlagi izhodiščne predpostavke sklepamo, da eksistirata stroja M_1 in M_2 z lastnostima $L = L(M_1)$ in $\overline{L} = L(M_2)$. Diagram stroja M, ki sprejema L in se vedno ustavi je prikazan na sl. 8.3. Na vhodu dobi besedo w, ki jo nato "vzporedno" (v smislu, ki je opisan v dokazu izreka 8.2) obdelujeta M_1 in M_2. Zagotovo se eden od teh dveh strojev ustavi na besedi w (velja bodisi $w \in L$ ali $w \in \overline{L}$), na podlagi česar lahko M napravi pravilno odločitev. $\square$



sl. 8.3 Stroj, ki se vedno ustavi, za L, ki ima Turingov komplement

V zvezi s tem izrekom je morda umestno, da se zamislimo nad pomenom pojma "paralelna obdelava", ki smo ga pravkar uporabili. Kako bi lahko omenjeno paralelno obdelavo uresničili na Turingovem stroju? Takoj prideta na misel najmanj dva načina: ena možnost je, da uporabimo dvotračni Turingov stroj in vsak trak uporabljamo za ponazoritev traku enega od strojev M_1 ali M_2, po čemer lahko tak dvotračni Turingov stroj simuliramo z navadnim Turingovim strojem, kot je to opisano v razdelku 7.5, stran 187. Drugi način je, da generiramo po vrsti števila 1, 2, ... , n, ... in po vsakem generiranem številu n najprej simuliramo M_1 za n korakov in nato še M_2. Skratka, pri dokazovanju bomo pogosto uporabljali kratice, ki so podobne pojmu "paralelna obdelava", in za katere bomo predpostavljali, da je njihov pomen očiten, vendar je očiten le za bralca, ki je seznanjen z določenimi osnovnimi pojmi, ki so bili vpeljani

prej.

Izreka 8.1 in 8.3 imata pomembno posledico: pri nekem jeziku in njegovem komplementu imamo le tri možnosti. Prva je, da sta oba Turingova (in torej tudi rekurzivna), druga je, da nobeden ni in tretja, da je eden Turingov, drugi pa ne. V nadaljevanju bomo veliko pozornost usmerili k dokazom, da določeni jeziki niso rekurzivni. Pogosto bomo postopali tako, da bomo pokazali, da komplement jezika ni Turingov.

8.3 Univerzalni Turingov stroj

V tem razdelku bomo pokazali, da je problem "Ali Turingov stroj M sprejme besedo w?" neodločljiv. Ker pa smo v zvezi s tem že poudarili, da gre za nerekurzivnost nekega jezika, ki je sestavljen iz besed nad neko abecedo, se bomo najprej ukvarjali s tem, kako neko konkretno obliko opisanega problema zakodirati v obliki niza simbolov. Z namenom, da si poenostavimo delo, bomo obravnavali le Turingove stroje, katerih vhodi so sestavljeni iz simbolov 0 in 1. To je vsekakor preprostejši problem kot, če bi obravanvali stroje z vsemi možnimi abecedami, vendar če je že preprostejša različica neodločljiva, bo to tem bolj držalo za splošnejšo obliko. V zvezi s tem glej nalogo 7.3.

Kodiranje Turingovih strojev

Pokazali bomo, da lahko zakodiramo poljuben Turingov stroj nad vhodno abecedo {0,1} in tračno abecedo {0,1,B} z neko besedo nad abecedo {0, 1}. Naj bo podan stroj

$$M = \langle Q,\{0,1\},\{0,1,B\},dlt,q_1,B,\{q_2\}\rangle,$$

pri čemer velja $Q = \{q_1, q_2, \ldots, q_n\}$. Pri vseh strojih tega razreda je začetno stanje q_1, edino končno stanje pa q_2. Bralec naj se za vajo prepriča, da opisane omejitve niso bistvene. (Torej vsak Turingov jezik nad abecedo {0,1} sprejema tudi neki stroj iz opisanega razreda. Velikost množice stanj Q ni omejena, omejena je le množica končnih stanj.)

Najprej določimo način kodiranja stanj, simbolov in smeri gibanja (L ali D). Stanje q_i kodiramo v obliki 0^i, 0, 1 in B, po vrsti, pa kot 0, 00 in 000. L kodiramo kot 0 in D kot 00. Nato vsak prehod dlt(q,X) = <r,Y,d> zakodiramo v obliki koda(q)1koda(X)1koda(r)1koda(Y)1koda(d) kjer je koda(A) koda posameznega elementa, kot je to bilo pravkar opisano. Končno zakodiramo opis stroja M v obliki

$$111koda(d_1)11koda(d_2)11\ldots11koda(d_N)111,$$

pri čemer je $koda(d_i)$ koda i-tega prehoda v nekakšni razvrstitvi vseh prehodov dlt(q,X) = <r,Y,d>.

Če je podan opis stroja M, lahko zakodiramo par <M, w>, pri čemer je w beseda nad abecedo {0,1}, preprosto v obliki koda(M) o w, pri čemer je koda(M) opis stroja M.

8.4. PRIMER. Naj bo podan stroj

$$M = \langle\{q_1,q_2,q_3\},\{0,1\},\{0,1,B\},dlt,q_1,B,\{q_2\}\rangle,$$

pri čemer je funkcija prehodov definirana takole:

$$dlt(q_1,1) = \langle q_3,0,D\rangle,$$
$$dlt(q_3,0) = \langle q_1,1,D\rangle,$$
$$dlt(q_3,1) = \langle q_2,0,D\rangle,$$
$$dlt(q_3,B) = \langle q_3,1,L\rangle.$$

Potem lahko par <M,1011> zakodiramo v obliki

111010010001010011000101010010011000101001010011
000100010001001011111011

Lahko omenimo, da je jezik vseh mogočih veljavnih parov <M,w>, ki so sestavljeni iz opisa stroja ter neke vhodne besede, možno podati z naslednjim regularnim izrazom:

$$[111111 + 111E(11E)^* 111](0 + 1)^*, \qquad (8.1)$$

pri čemer E predstavlja regularni izraz

$$(0^+1(0 + 00 + 000)10^+1(0 + 00 + 000)1(0 + 00)),$$

F^+ pa predstavlja $F^* - \{\epsilon\}$.

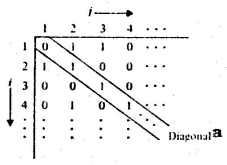
Pri vsem tem je važno upoštevati dejstvo, da lahko en in isti stroj zakodiramo na več načinov (z več besedami) s preprostim permutiranjem seznama prehodov. Bralec je morda že opazil dejstvo, da niso vse besede nad abecedo {0, 1} veljavne kode. Vendar je včasih pomembno, da nam prav vse besede predstavljajo stroje oziroma pare <M, w>. S tem namenom se domenimo, da v kolikor neka koda iz kakršnegakoli razloga ni

veljavna, potem predstavlja neki določen stroj, n.pr. stroj, ki ne sprejme nobenega vhoda.

Primer jezika, ki ni Turingov. Vzemimo množico besed nad abecedo {0,1}. Sedaj definirajmo naslednji jezik:

$$L_d = \{w \mid w \notin M_w\},$$

pri čemer M_w označuje Turingov stroj, ki ima opis (kot smo ga pravkar definirali) enak besedi w. Torej z besedami: L_d vsebuje vse besede w, ki jih ne sprejema Turingov stroj z istim opisom. L_d si lahko nazorno predstavljamo takole. V razdelku 7.7 smo vpeljali kanonsko razvrstitev besed nad neko poljubno abecedo. Potemtakem vsaki besedi w ustreza neko naravno število i (mesto besede w v kanonski razvrstitvi). Sedaj si predstavljajmo neskončno tabelo (sl. 8.4), katere i-ta vrstica je karakteristična funkcija jezika stroja z opisom, ki je enak besedi w z indeksom i (pravzaprav bi lahko rekli kar w-ta vrstica). Z drugimi besedami $T(i,j) = 1$ pri $w_j \in L(M_{w(i)})$ in $T(i,j) = 0$ pri $w_j \notin L(M_{w(i)})$. L_d ima potem karakteristično funkcijo, ki ima nasprotne vrednosti od diagonale v tabeli.



sl. 8.4 Tabela karakterističnih funkcij strojev M_w

Sedaj bomo dokazali

8.5. LEMA. L_d ni Turingov jezik.

Dokaz. Denimo, da lema ni resnična. Potem obstaja Turingov stroj M z $L_d = L(M)$. Vemo, da ima ta Turingov stroj lahko tračno abecedo {0,1,B}. Torej nastopa v naši razvrstitvi Turingovih strojev, ki smo jo opisali. In torej njemu tudi ustreza neka vrstica v tabeli iz sl. 8.4. Naj bo indeks te vrstice k. Na podlagi opisa tabele in predpostavke, da vrstica k predstavlja karakteristično funkcijo L_d, vemo, da je vrstica k enaka nasprotnim vrednostim diagonale. To pa ni možno, kajti diagonala in vrstica k imata en skupen element, namreč element T(k, k), in ta element ne more biti enak svoji nasprotni vrednosti. |_|

Univerzalni jezik. Sedaj je naš namen definirati nekakšen kanonski (standardni) primer nerekurzivnega jezika tako, kot je L_d kanonski primer neTuringovega jezika. Jezik, ki ga imamo v mislih, je

$$L_u = \{\langle M,w\rangle \mid w \in L(M)\},$$

torej jezik vseh parov $\langle M,w\rangle$ z lastnostjo, da M sprejema w.

Najprej dokažimo, da velja

8.6. IZREK. L_u je Turingov jezik.

Dokaz. Potrebno je le sestaviti Turingov stroj M_u, ki sprejema L_u. Ta stroj bomo na tem mestu opisali le z besedami, na bralca pa se zanašamo, da zna spremeniti ta besedni opis v podroben opis Turingovega stroja.

M_u najprej analizira besedo $\langle M,w\rangle$ in preveri, ali predstavlja veljaven opis nekega Turingove stroja skupaj z vhodno besedo. Preverjanje pravilnosti lahko opravi končen avtomat na podlagi (8.1). Torej tudi Turingov stroj ne bo imel težav. V primeru neveljavnosti vhoda, ga M_u ne sprejme.

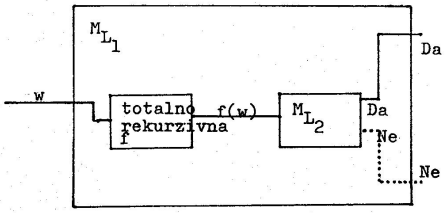
Če pa je vhodna beseda veljaven par $\langle M,w\rangle$, potem stroj M_u opravi simulacijo tako, da na posebni sledi traku zapiše začetni trenutni opis $q_1 w$, nato pa generira zaporedne trenutne opise uporabljajoč pri tem opis stroja $\langle M\rangle$. V primeru, ko M_u generira

trenutni opis oblike $xq_2 y$, kjer sta x in y besedi nad tračno abecedo {0,1,B}, stroj M_u svoj vhod $\langle M,w\rangle$ sprejme, sicer pa nadaljuje s simulacijo stroja M. |_|

Predno nadaljujemo je morda umestno ponovno spregovoriti o pojmu prevedbe ali redukcije problemov oziroma jezikov. Naj bosta podana jezika L_1 in L_2. Za L_2 imamo domnevni Turingov stroj oziroma sprejemnik M_2 medtem, ko za L_1 iščemo sprejemnik. V tem primeru pogosto postopamo tako, da poiščemo totalno rekurzivno funkcijo f: NN --> NN z lastnostjo, da velja

$$w \in L_1 \iff f(w) \in L_2, \tag{8.2}$$

potem pa zgradimo sprejemnik M_1 za L_1 tako, kot prikazuje sl. 8.5. Vhodno besedo w podamo na vhod računalniku za totalno rekurzivno funkcijo f, nakar vrednost f(w) podamo na vhod sprejemniku jezika L_2. Pri tem seveda velja, da v primeru, ko se sprejemnik jezika L_2 vedno ustavi, velja to tudi za pridobljeni sprejemnik za L_1.



sl. 8.5 Prevedba ali redukcija L_1 na L_2

Pravzaprav pa bomo redukcijo problemov največ uporabljali za to, da pokažemo, da algoritem za L_2 ni možen. Namreč opisana konstrukcija nam zagotavlja implikacijo

"iz eksistence sprejemnika za L_2 sledi eksistenca sprejemnika za L_1"

(8.3)

Kot vemo je ta trditev logično enakovredna trditvi

"iz neeksistence sprejemnika za L_1 sledi neeksistenca sprejemnika za L_2".

(8.4)

Po navadi bomo izhajali iz predpostavke, da algoritma ali sprejemnika za L_1 ni in bomo s prevedbo (na podlagi (8.4)) dokazali, da tudi za L_2 ni algoritma oziroma sprejemnika.

Naslednji rezultat je

8.7. IZREK. L_u ni rekurziven jezik.

Dokaz. Na podlagi izrekov 8.3 in 8.6 zadostuje, da dokažemo, da $\overline{L}_u$ ni Turingov jezik. To pa opravimo s prevedbo L_d na $\overline{L}_u$. Funkcija f na sl. 8.5 mora biti definirana takole:

$$f(w) = \langle M_w, w \rangle$$

Lastnost (8.2) naj bralec dokaže za vajo. Sedaj pa na podlagi (8.4) sledi, da algoritma oziroma sprejemnika za $\overline{L}_u$ ni. |_|

8.4 Riceov izrek

V prejšnem razdelku smo opisali primer Turingovega jezika, ki ni rekurziven. Ustrezni problem "Ali M sprejema w?" ni odločljiv, to dejstvo pa z metodo prevedbe uporabljamo, da pokažemo, da tudi drugi problemi niso odločljivi. Najprej bomo pokazali, da nekateri problemi, ki so v zvezi Turingovimi jeziki niso odločljivi, nato pa bomo izpeljali neodločljivost drugih problemov, ki niso neposredno povezani s Turingovimi stroji.

8.8. PRIMER. Sedaj bomo obravnavali problem "Ali velja $L(M) \neq \emptyset$?" Ustrezni jezik je

$$L_{ne} = \{<M> \mid L(M) \neq \emptyset\},$$

njegov komplement pa,

$$L_{e} = \{<M> \mid L(M) = \emptyset\}.$$

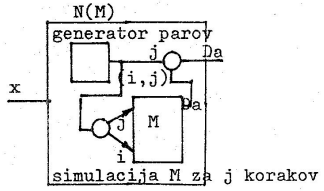
Pokazali bomo, da je L_{ne} Turingov jezik, ni pa rekurziven. To pa tudi pomeni, da L_{e} ni Turingov jezik. Dokaz izpeljemo tako, da pokažemo, da je L_{ne} možno prevesti na L_{u} in nasprotno, da je L_{u} možno prevesti na L_{ne}.

Obe trditvi dokažemo tako, da definiramo ustrezni totalni rekurzivni funkciji f s sl. 8.5. Za prevedbo L_{ne} na L_{u} postopamo takole. Vhod za sprejemnik za L_{ne} in torej argument funkcije f je neki opis Turingovega stroja <M>. L_{u} pa pričakuje na vhodu par <N,w>, ki predstavlja opis Turingovega stroja in neki vhod zanj. Torej mora f imeti vrednost, ki je v obliki $f(<M>) = <N(M),w(M)>$. Stroj N(M) je shematično prikazan na sl. 8.6. N(M) ima na vhodu besedo x, vsebuje pa generator parov <i,j> ter stroj M. Deluje tako, da vsakič, ko zgenerira neki par <i, j> sproži M za j korakov s vhodom i. V kolikor M sprejme i natanko v j-tem koraku se delovanje N(M) neha in se x sprejme ne glede na njegovo vrednost. V nasprotnem primeru pa

N(M) generira naslednji par. Za w(M) lahko izberemo poljubno vrednost. Očitno velja

$$L(N(M)) = \begin{cases} \Sigma^* & \text{pri } L(M) \neq \emptyset \\ \emptyset & \text{sicer} \end{cases}$$

Bralec naj se prepriča v veljavnost trditve $<M> \in L_{ne} \iff <N(M),w(M)> \in L_{u}$, prav tako pa naj se prepriča v dejstvo, da je f totalna rekurzivna funkcija (torej, da obstaja Turingov stroj, ali kakršenkoli računalnik, ki računa f). S to prevedbo smo dokazali, da je L_{ne} Turingov jezik na podlagi implikacije (8.3).

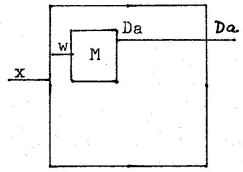


sl. 8.6 Stroj N(M) pri prevedbi L_{ne} na L_{u}

Sedaj ostane problem prevedbe L_u na L_{ne}. Vhod za stroj za L_u je neki par oblike <M,w> medtem, ko je vhod za stroj za L_{ne} neki opis stroja <M>. Imamo torej f(<M,w>) = <N(M,w)>, pri čemer je stroj N(M,w) shematično prikazan na sl. 8.7. N(M,w) deluje tako, da vsebuje M kot podprogram in ga ob začetku delovanja sproži z vhodom w. V kolikor M sprejme w, tudi N(M,w) sprejme svoj vhod, ne glede na njegovo vrednost. Očitno velja

$$L(N(M,w)) = \begin{cases} \text{SIGMA}^* & \text{pri } w \in L(M) \\ \emptyset & \text{sicer} \end{cases}$$

Tudi v tem primeru se lahko bralec prepriča v veljavnost trditve $\langle M,w\rangle \in L_u \iff \langle N(M,w)\rangle \in L_{ne}$, očitno pa je f totalna rekurzivna funkcija. S to prevedbo smo dokazali, na podlagi implikacije (8.4), da L_{ne} ni rekurziven jezik. Dejstvo, da L_{ne} ni Turingov jezik sledi iz izreka 8.3.



sl. 8.7 Stroj N(M,w) pri prevedbi L_u na L_{ne}

8.9. PRIMER. Sedaj bomo obravnavali komplementarna jezika

L_r = {<M> | L(M) je rekurziven}

ter

L_{nr} = {<M> | L(M) ni rekurziven}.

Bodimo pozorni, da jezik L_r ne sovpada z jezikom

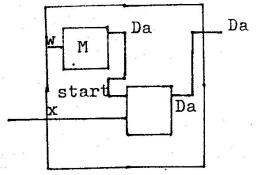
{<M> | M se ustavi na vseh vhodih},

kajti lahko se zgodi, da je neki stroj "slabo konstruiran" in se ne ustavi na vseh vhodih, čeprav je jezik, ki ga sprejema rekurziven (torej obstaja neki drugi stroj, ki sprejema isti jezik in se ustavi na vseh vhodih).

V tem primeru imamo opravka s parom komplementarnih jezikov z lastnostjo, da nobeden ni Turingov. Dokažimo najprej, da L_r

ni Turingov jezik. To opravimo s prevedbo jezika $\overline{L}_u$ na L_r. V tem primeru je argument za totalno rekurzivno funkcijo f s sl. 8.5 neki par <M,w> medtem, ko je vhod za sprejemnik za L_r neki opis stroja <N(M,w)>. Stroj N(M,w) je prikazan na sl. 8.8, deluje pa takole. Vhod stroja je beseda x medtem, ko stroj vsebuje M kot podprogram. Ko se sproži N(M,w), se sproži tudi M z vhodom w, in v primeru, ko M besedo w sprejme, se sproži sprejemnik M_u za L_u z vhodom x, sicer stroj N(M,w) besede x ne sprejme Očitno velja

$$L(N(M,w)) = \begin{cases} L_u & \text{pri } w \in L(M) \\ \emptyset & \text{sicer} \end{cases} \qquad (8.5)$$

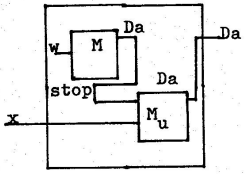


sl. 8.8 Stroj N(M,w) pri prevedbi $\overline{L}_u$ na L_r

Od dveh jezikov, ki nastopata na desni strani enačbe (8.5), je prvi nerekurziven drugi pa rekurziven in torej v primeru, ko sprejemnik za L_r sprejme <N(M,w))>, imamo zagotovilo, da velja $w \notin L(M)$.

Sedaj pa dokažimo, da L_{nr} ni Turingov jezik. V tem primeru uporabimo redukcijo jezika $\overline{L}_u$ na L_{nr}. Stroj N(M,w) za ta primer je prikazan na sl. 8.9, deluje pa takole. Zopet je vhod zanj beseda x. Ko sprožimo N, začne delovati M z vhodom w, istočasno pa začne delovati sprejemnik za L_u z vhodom x. M_u preneha delovati bodisi, ko sprejme besedo x ali pa ko M sprejme w. V prvem primeru N sprejme x, v drugem pa ne. Očitno velja

$$L(N(M,w)) = \begin{cases} \text{neka končna podmnožica } L_u & \text{pri } w \in L_u \\ L_u \text{ sicer} & \end{cases} \qquad (8.6)$$



Sl. 8.9 Stroj N(M,w) pri redukciji $\overline{L}_u$ na L_{nr}

Prvi jezik na desni strani enačbe (8.6) je rekurziven, drugi pa ne. Torej vemo, da velja $w \notin L(M)$ natanko v primeru, ko je $L(N(M,w))$ nerekurziven in smo s tem v celoti dokazali, da nobeden od L_r ter L_{nr} ni Turingov.

Sedaj pa naj bo podana neka podmnožica Turingovih jezikov. Po navadi je podobna podmnožica podana z nekakšno lastnostjo, ki jo imajo elementi množice. Zato bomo taki podmnožici rekli kar jezikovna lastnost. Torej naj bo dana jezikovna lastnost SS in

definirajmo jezik

$$L_{SS} = \{\langle M\rangle \mid L(M) \in SS\}.$$

Primer. V primeru 8.8 zgoraj smo imeli opravka z jezikovno lastnostjo $L = \emptyset$ (torej ustrezna podmnožica Turingovih jezikov vsebuje le en element in sicer prazen jezik), v primeru 8.9 pa z lastnostjo "L je rekurziven jezik".

Za neko jezikovno lastnost SS pravimo, da je rekurzivna v primeru, ko je jezik L_{SS} rekurziven, in da je Turingova v primeru, ko je L_{SS} Turingov.

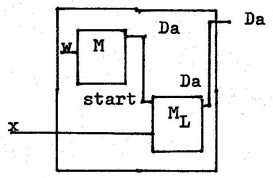
Sedaj lahko zapišemo

8.10. IZREK (Riceov izrek za rekurzivne jezikovne lastnosti). Neka jezikovna lastnost je rekurzivna natanko v primeru, ko je trivialna, kar pomeni, da bodisi ne vsebuje nobenega Turingovega jezika ali pa vsebuje vse.

Dokaz. Denimo, da izrek ne velja. Potem obstaja neka netrivialna jezikovna lastnost SS, ki je rekurzivna, ali z drugimi besedami obstaja sprejemnik M_{SS} za L_{SS}, ki se vedno ustavi. Izrek dokažemo s prevedbo jezika L_u na L_{SS}. Denimo sedaj, da jezik $\emptyset \notin SS$. (V nasprotnem primeru zamenjamo vlogi SS in njenega komplementa.) Glede na netrivialnost SS lahko sklepamo, da sta tako SS kot njen komplement neprazna. Torej obstaja $L \in SS$ kakor tudi TS M_L tako, da velja $L = L(M_L)$. Pri redukciji L_u na L_{SS} moramo z neko totalno rekurzivno funkcijo f

spremeniti vhod <M,w> v <N(M,w)>. N(M,w) je prikazan na sl. 8.10, deluje pa takole. Vhod za N je beseda x. N vsebuje M in M_L kot podprograma. Na začetku se sproži M z vhodom w. V primeru, ko M sprejme w, se sproži M_L z vhodom x. Izhod stroja N je potem enak izhodu M_L. Očitno velja

$$L(N(M,w)) = \begin{cases} L \text{ pri } w \in L(M) \\ \emptyset \text{ sicer} \end{cases}$$



sl. 8.10 Stroj N(M,w) za redukcijo L_u na L_{SS}

Očitno lahko na podlagi izhoda M_{SS} razlikujemo med $w \in L(M)$ in $w \notin L(M)$, kar pa ni možno. |_|

Obstaja tudi rezultat, ki se nanaša na Turingove jezikovne lastnosti, vendar je sam izrek, posebno pa njegov dokaz, nekoliko bolj zapleten, zaradi česar presega zastavljeni okvir tega dela.

8.5 Postov korespondenčni problem

Primeri, ki smo jih doslej obravnavali, imajo to hibo, da se nam zdijo nekoliko odmaknjeni, oziroma, da so sestavljeni nalašč za dokazovanje neodločljivosti. Za nas pa bi bilo zanimivejše izpeljati neodločljivost kakšnega naravnega problema oziroma problema, ki je povezan z drugimi področji. V tem razdelku se bomo s takšnim problemom prvič srečali.

Postov korespondenčni problem (PKP). Podana sta dva seznama besed nad neko izbrano abecedo:

$$X = \langle x_1, x_2, \ldots, x_n \rangle$$

ter

$$Y = \langle y_1, y_2, \ldots, y_n \rangle.$$

Vprašujemo se, ali obstaja algoritem, ki bi ugotovil, ali eksistira neko k in zaporedje indeksov i(1), i(2), ... , i(k) z lastnostjo

$$x_{i(1)} \circ x_{i(2)} \circ \ldots \circ x_{i(k)} = y_{i(1)} \circ y_{i(2)} \circ \ldots \circ y_{i(k)}.$$

Ponovno poudarjamo, da nas zanima algoritem, ki problem reši ne glede na konkretne vrednosti seznamov X in Y. Tak algoritem (Turingov stroj, ki se vedno ustavi) dobi na svojem vhodu neko besedo, ki predstavlja seznama X in Y v nekakšni

zakodirani obliki. Če pa nas zanima rešljivost vprašanja, ki smo si ga postavili, le za neka konkretna seznama X in Y je problem pogosto rešljiv.

8.11. PRIMER. Naj bo SIGMA = {0, 1} in naj bosta seznama X in Y taka, kot ju prikazuje sl. 8.11. V tem primeru ima PKP rešitev pri k = 4 in <i(1),i(2),i(3),i(4)> = <2,1,1,3>. Tedaj imamo

$$x_2 x_1 x_1 x_3 = y_2 y_1 y_1 y_3 = 101111110.$$

	Seznam X	Seznam Y
i	x_i	y_i
1	1	111
2	10111	10
3	10	0

sl. 8.11 Konkreten primer PKP

8.12. PRIMER. Naj bo zopet SIGMA = {0,1} in naj bosta seznama X in Y podana s sl. 8.12.

	Seznam X	Seznam Y
i	x_i	y_i
1	10	101
2	011	11
3	101	011

sl. 8.12 Drug konkreten primer PKP

Denimo, da ima tudi ta konkreten primer PKP neko rešitev, n.pr. i(1), i(2), .. , i(k). Očitno velja i(1) = 1, kajti nobena beseda, ki se začenja z x_2 = 011, ne more biti enaka besedi, ki se začenja z y_2 = 11 in nobena beseda, ki se začenja z x_3 = 101, ne more biti enaka besedi, ki se začenja z y_3 = 011. Besedo s seznama X bomo zapisovali nad ustrezno besedo s seznama Y. Doslej imamo

10
101

Naslednja beseda, ki jo izberemo s seznama X, se mora začeti z 1. Torej mora veljati i(2) = 1 ali i(2) = 3. Vendar i(2) = 1 ne ustreza, ker nobena beseda, ki se pričenja z $x_1 x_1$ = 1010, ne more biti enaka besedi, ki se pričenja z $y_1 y_1$ = 101101. Pri i(2) = 3 pa imamo

10101

101011

Ker pa je beseda, sestavljena iz seznama Y, zopet daljša za en simbol od besede, ki jo gradimo iz X, nam podoben premislek kot prej razodene, da velja i(3) = i(4) = ... = 3. Torej je zaporedje indeksov, ki definira združljive nize, enolično določeno, pri teh nizih pa je niz, ki ga tvorimo iz Y vedno za en simbol daljši. Zato ta konkreten PKP nima rešitve.

Omejeni Postov korespondenčni problem (OPKP). Neodločljivost PKP bomo pokazali tako, da nanj prevedemo L_u (v tem primeru sledi neodločljivost na podlagi (8.4). Vendar to moramo storiti v dveh korakih, in sicer tako, da najprej prevedemo L_u na neki soroden problem, ki mu pravimo omejeni Postov korespondenčni problem (OPKP), in nato slednjega prevedemo na PKP.

OPKP je definiran tako kot PKP z eno samo omejitvijo, da mora pri rešitvi veljati i(1) = 1.

8.13. LEMA. OPKP lahko prevedemo na PKP. Z drugimi besedami, če obstaja algoritem za PKP, obstaja tudi za OPKP.

Dokaz. Na podlagi sl. 8.5 je potrebno poiskati totalno rekurzivno funkcijo f, ki preslika neki opis konkretnega OPKP w v neki opis konkretnega PKP f(w) tako, da velja

"w ima rešitev" <==> "f(w) ima rešitev"

Smer ==> ne dela težav, saj je rešitev OPKP obenem rešitev ustreznega PKP. Kaj pa nasprotna smer? Implikacijo v nasprotni smeri pa dobimo tako, da iz w pridobimo takšen konkreten PKP, ki ne more imeti druge rešitve kot je rešitev OPKP. Za to vpeljemo dva nova simbola v abecedo SIGMA, n.pr. [in]. Nato spremenimo vse besede iz seznama X tako, da po vsakem simbolu vstavimo dodaten znak [. Tako dobimo seznam U, ki vsebuje besede $u_1, u_2, \ldots, u_k$. Seznam Y pa spremenimo tako, da v vseh besedah vstavimo pred vsakim znakom poseben znak [. Tako dobimo seznam V, ki vsebuje besede $v_1, v_2, \ldots, v_k$.

Nato dodamo še naslednje besede

$$u_0 = [u_1 \qquad v_0 = v_1$$

$$u_{k+1} =] \qquad v_{k+1} = []$$

Na sl. 8.13 vidimo seznama U in V, ki ustrezata primeru 8.11.

	Seznam X	Seznam Y		Seznam Z	Seznam W
i	x_i	y_i	i	z_i	w_i
1	1	111	0	[1[	[1[1[1
2	10111	10	1	1[	[1[1[1
3	10	0	2	1[0[1[1[1	[1[0
			3	1[0[	[0
			4	]	[]

sl. 8.13 Konkreten primer PKP in ustrezni primer OPKP

Sedaj pa trdimo, da ima OPKP rešitev natanko v primeru, ko ima njemu ustrezni PKP rešitev. Saj res, če ima OPKP neko rešitev i(1) = 1, i(2), ... , i(r), lahko to rešitev zlahka prevedemo v rešitev 0, i(2), i(3), ... , i(r), k + 1 ustreznega PKP. Nasprotno pa, če ima ustrezni PKP rešitev j(1), j(2), ... , j(r), potem mora nujno veljati j(1) = 0, saj je u_0 edina beseda na seznamu U, ki se začenja z [medtem, ko imajo prav vse besede na seznamu V to lastnost. Poleg tega velja tudi j(r) = k + 1, saj je u_{k+1} edina beseda na seznamu X, ki se končuje z] medtem, ko se vse besede na seznamu Y tako končujejo. Če pa je to res, pa lahko takšno rešitev takoj prevedemo v reštiev OPKP 1, j(2), ... , j(r - 1)

S tem smo opisali totalno rekurzivno funkcijo f in smo dokazali prevedbo OPKP na PKP. |¯|

Neodločljivost PKP. Izpeljali bomo

8.14. IZREK. PKP je neodločljiv.

Dokaz. Sedaj bomo dokazali, da je možno L_u prevesti na OPKP. Potem iz implikacije (8.4) sledi, da ni algoritma za OPKP in nato tudi, da ga ni za PKP. Pri jeziku L_u so elementi pari <M,w>. Tak par bo totalno rekurzivna funkcija f s sl. 8.5 preslikala v neki opis OPKP, ki bo imel to značilnost, da bodo besede, ki jih bomo gradili iz elementov seznamov A in B predstavljale zaporedja trenutnih opisov stroja M. Rešitev OPKP bo obstajala natanko v primeru, ko velja w ∈ L(M), torej ko M sprejme besedo w. Torej pri rešitvi OPKP bo beseda, ki jo bomo gradili iz elementov A in B (oziroma njen začetek), imela videz $\#q_0 w\#alf\ q_1 bta\ \#...\#alf\ q_m bta\ \#$, pri čemer so nizi med znaki # trenutni opisi stroja M, stanje q_m pa je končno.

Pri definiciji seznamov A in B se je potrebno le domeniti, kateri elementi so na prvem mestu, pri ostalih pa vrstni red ni važen (seveda, če permutiramo elemente enega od seznamov, moramo na isti način tudi permutirati elemente drugega seznama). Zaradi tega elemente seznamov ne bomo opremili z vrstnimi številkami, nakazali bomo le, kateri elementi v obeh seznamih si ustrezajo.

Prvi par je:

Seznam A	Seznam B
#	$\#q_0 w\#$

Preostali pari pa so razvrščeni takole:

SKUPINA 1 za vse X ∈ GAMA

Seznam A	Seznam B
X	X
#	#

SKUPINA 2. Za vse q ∈ Q - F, p ∈ Q ter X, Y, Z ∈ GAMA:

Seznam A	Seznam B	
qX	Yp	pri dlt(q,X) = <p,Y,D>
ZqX	pZY	pri dlt(q,X) = <p,Y,L>
q#	Yp#	pri dlt(q,B) = <p,Y,D>
Zq#	pZY#	pri dlt(q,B) = <p,Y,L>

SKUPINA 3. Za vse q ∈ F ter X, Y ∈ GAMA:

Seznam A	Seznam B
XqY	q
Xq	q
qY	q

SKUPINA 4

Seznam A	Seznam B	
q##	#	za vse q ∈ F

Rekli bomo, da je <x, y> delna rešitev konkretnega OPKP s seznamoma A in B, če je x neka predpona (začetni del) y in če sta x in y stika ustreznih besed s seznamov A in B (po vrsti). Če velja xz = y, potem bomo z poimenovali ostanek para <x, y>.

Naj obstaja zaporedje dolžine k + 1 trenutnih opisov s prvim elementom $q_0 w$. Trdimo, da eksistira delna rešitev

$$\langle x,y\rangle = \langle \#q_0 w\#alf_1 q_1 bta_1 \#\ldots$$

$$\#alf_{k-1} q_{k-1} bta_{k-1} \#,\#q_0 w\#alf_1 q_1 bta_1 \#\ldots\#alf_k q_k bta_k \#\rangle.$$

Poleg tega je to edina delna rešitev, katere daljša beseda je dolga |y|.

To trditev ni težko dokazati z indukcijo po k. Pri k = 0 je očitna, saj moramo najprej izbrati par $\langle \#,\#q_0 w\#\rangle$.

Sedaj pa naj bo trditev resnična pri nekem k in naj velja $q_k \notin F$. Pokazali bomo, da trditev velja pri k + 1. Ostanek para <x, y> je $z = alf_k q_k bta_k \#$. Pari, ki sledijo, morajo biti tako izbrani, da stik besed s seznama A tvori z. Ne glede na znake, ki so desno in levo od q_k, obstaja največ en par iz skupine 2, ki omogoča, da delno rešitev podaljšamo mimo q_k. Ta par na naraven način predstavlja korak, ki ga naredi M pri trenutnem opisu $alf_k q_k bta_k$. Ostali znaki besede z nas prisilijo na določene izbire parov v skupini 1. Nobene druge izbire nam ne omogočajo, da sestavimo z iz elementov seznama A.

Tako dobimo novo delno rešitev

$$\langle y, y alf_{k+1} q_{k+1} bta_{k+1} \#\rangle.$$

Ni se težko prepričati v dejstvo, da je $alf_{k+1} q_{k+1} bta_{k+1}$ edini trenutni opis, v katerega

stroj M lahko pride z enim korakom iz opis alf q bta . Prav
k k k

tako ni nobene druge delne rešitve, pri kateri je dolžina druge besede enaka |yalf q bta #|.
k + 1 k + 1 k + 1

Če pa je q ∈ F ni težko najti pare iz skupin 1 in 3, ki,
k

če jih dodamo delni rešitvi <x,y> in nato še vse zaključimo s parom iz skupine 4, tvorijo rešitev konkretnega OPKP s seznamoma A in B.

Torej, v primeru, ko M iz začetnega trenutnega opisa q w
0

pride v neko končno stanje, ima ustrezni konkretni OPKP rešitev. V nasprotnem primeru pa ne moremo uporabljati parov iz skupin 3 ali 4. V tem primeru, čeprav obstajajo delne rešitve, je beseda, sestavljena iz seznama B, vedno daljša od besede, sestavljene iz A, zaradi česar rešitve ni.

Na koncu ugotovimo, da ima konkreten OPKP rešitev natanko v primeru, ko M sprejme vhod w. Ker pa opisano konstrukcijo lahko izpeljemo za poljubna M in w, sklepamo, da bi nam algoritem za OPKP zagotovil algoritem za L , kar je v protislovju z izrekom
u

8.7.

8.15. PRIMER. Naj bo podan

M = <{q ,q ,q },{0,1,B},{),1},dlt,q ,B,{q }>,
1 2 3 1 3

pri čemer je dlt definirano z naslednjo tabelo:

q_i	dlt(q_i,0)	dlt(q_i,1)	dlt(q_i,B)
q₁	(q₂,1,D)	(q₂,0,L)	(q₂,1,L)
q₂	(q₃,0,L)	(q₁,0,D)	(q₂,0,D)
q₃	--	--	--

Naj bo w = 01. Sedaj bomo sestavili ustrezni konkreten OPKP s seznamoma A in B. Prvi par je sestavljen s # s seznama A ter #q 01# a seznama B. Preostali pari pa so:
1

SKUPINA 1

Seznam A	Seznam B
0	0
1	1
#	#

SKUPINA 2

Seznam A	Seznam B	
q 0 1	1q 2	zaradi dlt(q ,0) = <q ,1,R> 1 2
0q 1 1	q 00 2	in
1q 1 1	q 10 2	zaradi dlt(q ,1) = <q ,0,L> 1 2
0q # 1	q 01# 2	in
1q # 1	q 11# 2	zaradi dlt(q ,B) = <q ,1,L> 1 2
0q 0 2	q 00 3	in

---

2 Ker se B ne izpisuje, lahko izpustimo pare, pri katerih je B desno od stanja. Pari iz skupine III prav tako ne vsebujejo takih s presledkom B na eni ali obeh straneh stanja

$1q_2\ 0$	$q_3\ 10$	zaradi $dlt(q_2,0) = \langle q_3,0,L\rangle$
$q_2\ 1$	$0q_1$	zaradi $dlt(q_2,1) = \langle q_1,0,R\rangle$
$q_2\ \#$	$0q_2\ \#$	zaradi $dlt(q_2,B) = \langle q_2,0,R\rangle$

SKUPINA 3

Seznam A	Seznam B
$0q_3\ 0$	q_3
$0q_3\ 1$	q_3
$1q_3\ 0$	q_3
$1q_3\ 1$	q_3
$0q_3$	q_3
$1q_3$	q_3
$q_3\ 0$	q_3
$q_3\ 1$	q_3

SKUPINA 4

Seznam A	Seznam B
$q_3\ \#\#$	#

Ugotovimo lahko, da stroj sprejme vhod w = 01 na podlagi zaporedja trenutnih opisov:

$q_1 01,\ 1q_2 1,\ 10q_1,\ 1q_2 01,\ q_3 101.$

Sedaj poskušajmo sestaviti rešitev za opisani konkreten OPKP. Prvi par predstavlja delno rešitev $\langle \#, \#q_1 01\#\rangle$. Ogled parov nam odkrije, da lahko dobimo neko daljšo delno rešitev edino z uporabo para $\langle q_1 0, 1q_2\rangle$. Iz tega dobimo delno rešitev

$\langle \#q_1 0, \#q_1 01\#1q_2\rangle$. Sedaj je ostanek $1\#1q_2$. Naslednji trije pari so nujno $\langle 1,1\rangle, \langle \#,\#\rangle$ in $\langle 1,1\rangle$, kar nam da delno rešitev $\langle \#q_1 01\#1, \#q_1 01\#1q_2 1\#1\rangle$. Sedaj je ostanek $q_2 1\#1$. Ko nadaljujemo s podobnimi premisleki, ugotovimo, da je edina delna rešitev, pri kateri je dolžina druge komponente 14, enaka $\langle x, x0q_1 \#1\rangle$, pri x = $q_1 01\#1q_2 1\#1$.

Sedaj imamo navidez izbiro, kajti nadaljevali bi lahko s parom $\langle 0, 0\rangle$ ali pa $\langle 0q_1 \#, q_2 01\#\rangle$. V prvem primeru dobimo delno rešitev $\langle x0, x0q_1 \#10\rangle$, vendar je ta rešitev "slepa ulica". K tej delni rešitvi ne moremo dodati nobenega para, da bi dobili neko novo delno rešitev, in torej tudi ne vodi do rešitve.

Na podoben način smo vedno prisiljeni izbirati en določen par, če hočemo na koncu priti do rešitve. Končno pridemo do delne rešitve $\langle y, y1\#q_3 10\rangle$ pri

$y = q_1 01\#1q_2 1\#10q_1 \#1q_2 0.$

Ker pa je q_3 končno stanje, lahko sedaj začnemo uporabljati pare iz skupin 1, 3 in 4 zato, da dokončno pridemo do rešitve konkretnega OPKP. Potrebna izbira parov je

$\langle 1, 1\rangle, \langle \#, \#\rangle, \langle q_3 1, q_3\rangle, \langle 0, 0\rangle, \langle 1, 1\rangle, \langle \#, \#\rangle, \langle q_3 0, q_3\rangle,$

$\langle 1, 1\rangle, \langle \#, \#\rangle, \langle q_3 1, q_3\rangle, \langle \#, \#\rangle, \langle q_3 \#\#, \#\rangle.$

Torej ima najkrajša beseda, ki jo lahko sestavimo iz ustreznih elementov seznamov A in B, če začnemo s parom številka 1, naslednji videz:

#q 01#1q 1#10q #1q 01#q 101#q 01#q 1#q ##.
  1   2   1 2   3   3 3 3

Primer uporabe PKP. Postov korespondenčni problem nam omogoča, da dokažemo naslednji

8.16. IZREK. Problem, ali je neka poljubna kontekstno neodvisna gramatika dvoumna, je neodločljiv.

Opisani problem bomo označevali z AMB.

Dokaz. Izrek bomo dokazali tako, da bomo prevedli PKP na AMB. Vhod za PKP sta dva seznama $X = x_1, x_2, \ldots, x_n$ in $Y = y_1, y_2, \ldots, y_n$, ki sta sestavljena iz besed nad neko končno abecedo SIGMA. Vhod za AMB pa je opis neke kontekstno neodvisne gramatike. Kot po navadi, moramo definirati neko totalno rekurzivno f, ki nam preslika seznama X in Y v opis gramatike f(X, Y) z lastnostjo, da ima PKP, ki je definiran z X in Y, rešitev natanko, ko je ustrezna gramatika dvoumna. Postopamo tako, da vzamemo nove simbole $a_1, a_2, \ldots, a_n$, ki niso v SIGMA, in definiramo jezika

$$L_X = \{x_{i(1)} x_{i(2)} \ldots x_{i(m)} a_{i(m)} a_{i(m-1)} \ldots a_{i(1)} \mid (m \geq 1) \text{ and } i(1)i(2)\ldots i(m) \in \{1,2,\ldots,n\}^* \}$$

ter

$$L_Y = \{y_{i(1)} y_{i(2)} \ldots y_{i(m)} a_{i(m)} a_{i(m-1)} \ldots a_{i(1)} \mid (m \geq 1) \text{ and } i(1)i(2)\ldots i(m) \in \{1,2,\ldots,n\}^* \}$$

pri čemer $i(1)i(2)\ldots i(m) \in \{1,2,\ldots,n\}^*$ pomeni, da gre za poljubno zaporedje elementov množice {1,2,...,n}.

Naj bo G kontekstno neodvisna gramatika

$$\langle \{S, S_1, S_2\}, SIGMA \cup \{a_1, \ldots, a_n\}, P, S \rangle,$$

pri čemer P vsebuje produkcije $S \to S_X$, $S \to S_Y$ ter pri $1 \leq i \leq n$ $S_X \to x_i S_X a_i$, $S_X \to x_i a_i$, $S_Y \to y_i S_Y a_i$ ter $S_Y \to y_i a_i$. Gramatika G generira jezik $L_X \cap L_Y$.

Če ima konkreten PKP <X, Y> rešitev, denimo i(1), i(2), ..., i(m), potem v jeziku L_X obstaja beseda $x_{i(1)} x_{i(2)} \ldots x_{i(m)} a_{i(m)} a_{i(m-1)} \ldots a_{i(1)}$, ki je enaka besedi $y_{i(1)} y_{i(2)} \ldots y_{i(m)} a_{i(m)} a_{i(m-1)} \ldots a_{i(1)}$ iz jezika L_Y. Ta beseda ima drevo izpeljav, ki ima na vrhu produkcijo $S \to S_X$ in neko drugo drevo, ki ima na vrhu produkcijo $S \to S_Y$. Torej je v tem primeru G dvoumna.

Nasprotno pa denimo, da je G dvoumna. Glede na to, da zaporedje simbolov a določa uporabljene produkcije, ni težko dokazati, da imajo vse besede, ki so izpeljane iz S_X le eno drevo izpeljav s korenom S_X. Na podoben način nima nobena beseda, ki je izpeljana iz S_Y več kot eno drevo izpeljav s korenom S_Y. Torej mora veljati, da ima neka beseda drevesi izpeljav, eno od katerih uporablja pri vrhu produkcijo $S \to S_X$, drugo pa produkcijo $S \to S_Y$. Če je ta beseda $a_{i(m)} a_{i(m-1)} \ldots a_{i(1)}$ pri $y \in SIGMA^*$, potem je i(1), i(2), ... i(m) rešitev podanega PKP.

S tem smo dokončali prevedbo PKP na AMB, iz česar sklepamo, da AMB ni odločljiv. |_|

8.6 Veljavni in neveljavni izračuni TS

Čeprav lahko PKP brez težav prevedemo na večino problemov o jezikih, za katere vemo, da so neodločljivi, pa obstaja tudi ka druga, bolj neposredna metoda, ki je poučna. V tem zdelku bomo izpeljali prevedbo problema pripadnosti za TS na zlične probleme o kontekstno neodvisnih jezikih. S tem menom vpeljemo pojma veljavnih in neveljavnih izračunov ringovih strojev.

Za potrebe tega razdelka je veljaven izračun Turingovega stroja $M = \langle Q, SIGMA, GAMA, dlt, q_0, B, F\rangle$ niz $w_1 \#w_2 \#w_3 \#w_4 \#\ldots$ tako, da:

1. vsaka beseda w_i je trenutni opis stroja M, torej neki element $GAMA^* Q GAMA^*$, ki se ne končuje z B;
2. w_1 je začetni trenutni opis, torej je element $q_0 SIGMA^*$;
3. w_n je končni trenutni opis, torej je element $GAMA^* F GAMA^*$; in končno
4. $w_i \vdash_M w_{i+1}$, pri $1 \leq i < n$.

Ne da bi se s tem omejili, se domenimo, da sta Q in GAMA medsebojno tuji in da # ne pripada nobeni od teh dveh množic.

Množica neveljavnih izračunov nekega Turingovega stroja je komplement množice veljavnih izračunov glede na abecedo GAMA |_| Q |_| {#}.

Pojma veljavnih in neveljavnih izračunov sta koristna pri dokazovanju, da so mnoge lastnosti KNJ neodločljive. To izvira iz dejstva, da je množica neveljavnih izračunov KNJ, množica veljavnih izračunov pa presek dveh KNJ.

8.17. LEMA. Množica veljavnih izračunov Turingovega stroja M je presek dveh KNJ L_1 in L_2, ki ju lahko efektivno konstruiramo iz M.

Dokaz. Naj bo M = <Q,SIGMA,GAMA,dlt,q_0,B,F> Turingov stroj. Oba jezika, L_1 in L_2 bosta vsebovala besede oblike

$x_1 \# x_2 \# \ldots \# x_m \#$. L_1 uveljavlja pogoj $x_i \vdash (x_{i+1})^R$ pri lihih i, L_2 pa pogoj $x_i^R \vdash x_{i+1}$ pri sodih i. L_2 prav tako uveljavlja pogoj, da je x_1 začetni trenutni opis. Pogoj, da je x_m končni trenutni opis ali njegova zrcalna slika, uveljavlja bodisi L_1 ali L_2, odvisno od tega, ali je m liho ali sodo (po vrsti). V tem primeru je potem $L_1 \barwedge L_2$ množica veljavnih izračunov stroja M.

Naj bo L_3 enak $\{y\#z^R \mid y \vdash_m z\}$. Potem ni težko konstruirati skladovni avtomat P, ki sprejema L_3: P bere y (vhodna beseda do znaka #) ter preverja v svoji nadzorni enoti, ali ima y obliko $GAMA^* QGAMA^*$. Obenem postavi P na svoj sklad trenutni opis z tako, da velja $y \vdash_M z$, pri čemer je y del vhodne besede do znaka #. P to opravi tako, da v primeru, ko je na vhodu znak iz GAMA, vstavi ta znak v sklad, v primeru pa, ko je na vhodu simbol za stanje q, spravi q v svojo nadzorno enoto in prebere naslednji vhodni simbol, na primer X (v primeru, ko je naslednji simbol #, ga obravnavamo kot B). V primeru dlt(q,X) =

<p,Y,D>, postavi P v sklad Yp, v primeru pa dlt(q,X) = <p,Y,L>, pa zamenja zadnji skladovni simbol Z z pZY (če pa je zadnji vhodni znak bil # in Y = B, se Z zamenja z pZ ali preprosto s p v primeru, ko je tudi Z enak B). Potem, ko P prebere #, preverja vhodne simbole z zadnjim simbolom na skladu. Če sta simbola različna, P nima prehodov in se ustavi ne, da bi besede sprejel. Če sta simbola ista, P izbriše zadnji simbol v skladu in nadaljuje. P sprejme vhod, če se sklad izprazni.

Naj bo jezik $L_1 = (L_3 \#)^* (\{\epsilon\} \cup GAMA^* FGAMA^* \#)$. Na podlagi izrekov 5.9 in 6.6 obstaja algoritem, ki sestavi KNG G_1 za L_1.

Na podoben način lahko sestavimo skladovni avtomat za $L_4 = \{y\#z^R \mid y \vdash_M z\}$. Podobno kot za L_1 lahko tudi sedaj sestavimo algoritem, ki nam izgradi KNG G_2 za jezik

$$L_2 = q_0 SIGMA^* \#(L_4 \#)^* (\{\epsilon\} \cup GAMA^* FGAMA^* \#).$$

Presek $L_1 \barwedge L_2$ je množica veljavnih izračunov stroja M.

Resnično, če $x_1 x_2 \ldots x_m$ pripada $L_1 \barwedge L_2$, potem L_1 uveljavlja $x_i \vdash_M (x_{i+1})^R$ pri lihih i, L_2 pa terja, da je x_1 začetni trenutni opis in da velja $x_i^R \vdash_M x_{i+1}$ pri sodih i. Pogoj, da je

zadnji trenutni opis končni uveljavlja L_1 pri lihih m in L_2 pri sodih m.

8.18. IZREK. Problem, ali je presek jezikov poljubnih dveh kontekstno neodvisnih gramatik prazen, je neodločljiv.

Dokaz. Na podlagi leme 8.17 lahko iz stroja M konstruiramo gramatiki G_1 in G_2 tako, da je $L(G_1)|^-|L(G_2)$ množica veljavnih izračunov M. Če obstaja algoritem A za preizkus praznosti preseka jezikov dveh KNG, lahko tudi konstruiramo algoritem B za problem L(M) = Ø pri poljubnem TS M: potrebno je preprosto sestaviti B tako, da konstruira G_1 in G_2 iz M tako, kot je opisano v lemi 8.17, in nato uporabiti algoritem A za preizkus, ali je $L(G_1)|^-|L(G_2)$ prazen. V primeru ko presek ni prazen, velja $L(M) \neq Ø$. Z drugimi besedami pokazali smo, da lahko problem praznosti Turingovih jezikov prevedemo na problem preseka KNG.

Vendar algoritma B ni, ker je L(M) = Ø neodljočljivo na podlagi izreka 8.10. Torej tudi algoritma A ni in je neodločljivo, ali je presek jezikov dveh poljubnih KNG prazen.

|¯|

Čeprav potrebujemo dva KNJ za predstavitev veljavnih izračunov Turingovega stroja, je množica neveljavnih izračunov enaka nekemu KNJ. Razlog za to je, da ni več potrebno

zagotoviti $w_i \vdash w_{i+1}$ pri vsakem i. Potrebno je le uganiti, kje je prišlo do napake, oziroma, z drugimi besedami potrebno je le pri enem i preveriti, da ne velja $w_i \vdash w_{i+1}$.

8.19. LEMA. Množica neveljavnih izračunov Turingovega stroja $M = \langle Q, SIGMA, GAMA, dlt, q_0, B, F\rangle$ je KNJ.

Dokaz. Če je niz w neveljaven izračun, potem je izpolnjen eden od naslednjih pogojev:

1. w ni oblike $x_1 \# x_2 \# \ldots \# x_m \#$, kjer je vsak x_i TO stroja M;
2. x_1 ni začetni TO oziroma ne pripada množici $q_0 SIGMA^*$;
3. x_m ni končen TO oziroma ne pripada množici $GAMA^* FGAMA^*$;
4. pri nekem lihem i ne velja $x_i \vdash_M^R (x_{i+1})$;
5. pri nekem sodem i ne velja $x_i \vdash_M^R x_{i+1}$.

Množica nizov, ki izpolnjujejo pogoje 1, 2 in 3, je regularna in brez težav konstruiramo KA, ki jo sprejema. Množici, ki izpolnjujeta pogoja 4 in 5 sta obe KNJ. To trditev bomo dokazali za 4 (podoben premislek pa velja za 5): skladovni avtomat P za 4 izbere nedeterministično neki x_i, ki sledi sodemu številu znakov #, in medtem, ko bere x_i shrani v svoj sklad TO z

tako, da velja x_i |- z, (desni konec z je na vrhu sklada). Po naslednjem znaku # P primerja z z naslednjim x_{i+1}. V primeru $z \neq x_{i+1}$ P prebere preostanek svojega vhoda ter vhod sprejme.

Množica neveljavnih izračunov je unija dveh KNJ in neke regularne množice. Na podlagi izreka 6.9 je ta množica KNJ in lahko konstruiramo njeno gramatiko efektivno. |_|

8.20. IZREK. Ni odločljivo, ali za neko poljubno KNG G velja $L(G) = SIGMA^*$.

Dokaz. Pri poljubnem podanem TS M lahko efektivno konstruiramo KNG G s končno abecedo SIGMA tako, da velja L(G) = SIGMA natanko, ko velja L(M) = Ø. Na podlagi leme 8.19 lahko konstruiramo KNG G, ki generira neveljavne izračune stroja M. Torej če bi bil za poljubno G problem L(G) = SIGMA odločljiv, potem bi bil odločljiv za poljuben TS M tudi problem L(M) = Ø, kar pa je v protislovju z izrekom 8.10. |_|

Dodatne posledice opisa izračunov s KNJ. Iz izreka 8.20 lahko izpeljemo mnoge rezultate.

8.21. IZREK. Naj bodo G_1 in G_2 poljubne KNG, R pa naj bo poljubna regularna množica. Lahko pokažemo, da so naslednji proglemi so neodločljivi:

1. $L(G_1) = L(G_2)$. 2. $L(G_2)$ |_ $L(G_1)$.

3. $L(G_1) = R$. 4. R |_ $L(G_1)$.

Dokaz. Sestavi G_2 tako, da generira $SIGMA^*$, pri čemer je SIGMA končna abeceda gramatike G_1. V tem primeru sta 1 in 2 ekvivalentna problemu $L(G_1) = SIGMA^*$. Nato sestavi R = $SIGMA^*$ in sta 3 in 4 ekvivalentna problemu $L(G_1) = SIGMA^*$. Torej lahko neodločljivi problem, ali je neki KNJ enak $SIGMA^*$, prevedemo na 1 preko 4 in so tako tudi ti problemi neodločljivi. |_|

Naj pripomnimo, da lahko na podlagi izrekov 5.8 in 5.9 spreminjamo opise gramatik v opise strojev (in nasprotno) efektivno, in torej ostanejo izreki 8.18, 8.20 in 8.21 v veljavi, tudi ko KNJ predstavimo s skladovnim avtomatom namesto z gramatiko. Podobno lahko predstavimo regularno množico R iz izreka 8.21 bodisi z determinističnim ali nedeterminističnim avtomatom, ali pa z regularnim izrazom.

Zanimivo je dejstvo, da je problem L(G) |_ R odločljiv. Razlog temu je, da velja L(G) |_ R natanko ko velja $L(G) \cap \bar{R}$ = Ø. Jezik $L(G) \cap \bar{R}$ pa je KNJ in je torej njegova praznost odločljiva.

Sedaj lahko ugotovimo, da v primeru, ko ima neki TS veljavne izračune na neskončni množici vhodov, slednja množica, v splošnem, ni KNJ. Iz tega sledi, da je še nekaj lastnosti KNJ neodločljivih. Da bi to dosegli pa moramo vsakemu TS M dodati dve novi stanji, katerih edini namen je v tem, da zagotovijo, da M naredi vsaj dva koraka. To je možno narediti, ne da bi kakorkoli spremenili rezultat izračuna M. Namen te spremembe je, da zagotovimo, da vsak veljaven izračun vsebuje najmanj tri trenutna opisa, s čimer dosežemo, da je množica veljavnih izračunov KNJ, natanko ko M sprejema končno množico.

8.22. LEMA. Naj bo M TS, ki napravi najmanj tri korake pri vsakem vhodu. Množica veljavnih izračunov stroja M je KNJ natanko v primeru, ko je L(M) končna množica.

Dokaz. V primeru, ko je L(M) končna, je tudi množica veljavnih izračunov končna in torej tudi KNJ. Sedaj pa predpostavimo, da je L(M) neskončna in da je kljub temu množica veljavnih izračunov L KNJ. Ker je L(M) neskončna obstaja veljaven izračun

$$w_1 \#_R w_2 \#_R w_3 \#_R \ldots$$

kjer so w_i TO in kjer je $|w_2|$ večje kot konstanta n iz Ogdenove leme. Zaznamuj znake w_2. Tedaj lahko w_2 "napihnemo", ne da bi napihovali w_1 in w_3, zaradi česar dobimo neveljaven izračun, ki mora biti v L. Iz tega sklepamo, da veljavni izračuni ne

tvorijo KNJ. |‾|

8.23. IZREK. Problem, ali je za poljubni gramatiki G_1 in G_2

1. $\overline{L}(G_1)$ kontekstno neodvisen ali
2. $L(G_1)|‾|L(G_2)$ kontekstno neodvisen

je neodločljiv.

Dokaz.

1. Pri poljubnem podanem TS M, spremeni M, ne da bi pri tem spremenil L(M) tako, da M naredi najmanj dva koraku pri vsakem vhodu. Sestavi KNG G, ki generira neveljavne izračune. $\overline{L}(G)$ je KNJ, natanko ko je L(M) končna.

2. Postopaj natanko kot v primeru 1, vendar sestavi G_1 in G_2 tako, da je $L(G_1)|‾|L(G_2)$ množica veljavnih izračunov stroja M. |‾|

8.7 Naloge

8.1 Denimo, da so tračne abecede vseh Turingovih strojev končne podmnožice neke števno neskončne množice simbolov $a_1, a_2, \ldots$ Pokažite, kako bi v tem primeru zakodirali Turingove stroje z binarnimi nizi.

8.2 Pokažite, da je neodločljivo, ali se neki TS ustavi na vseh vhodih.

8.3 Pokažite, da so naslednji problemi, ki se nanašajo na neki resničen programski jezik, neodločljivi.

1. Ali se podani program zacikla na nekem vhodu.
2. Ali podani program kdajkoli izpiše neki rezultat.
3. Ali dva podana programa izpišeta isti rezultat na vseh vhodih.

8.4 Pokažite, da je PKP za besede nad enosimbolno abecedo odločljiv.

9. POGLAVJE

HIERARHIJA CHOMSKEGA

Od treh jezikovnih razredov, ki smo jih preučevali (regularni jeziki, kontekstno neodvisni jeziki in Turingovi jeziki), smo z gramatikami opisali le kontekstno neodvisne jezike. V tem poglavju podajamo gramatične opise tudi regularnih jezikov in Turingovih jezikov. Vpeljali bomo tudi nov razred jezikov, ki leži med KNJ in Turingovimi jeziki, in zanj podali tako opis z avtomati kot opis z gramatikami. Omenjeni štirje jezikovni razredi tvorijo t.im. hierarhijo Chomskega, po jezikoslovcu Noamu Chomskemu, ki jih je definiral kot potencialne modele naravnih jezikov.

9.1 Regularni jeziki in gramatike.

Najpreprostejši jezikovni razred, ki ga bomo preučevali, je vsekakor razred regularnih jezikov (RJ), oziroma jezikov, ki jih sprejemajo končni avtomati. Predvsem nas zanima to, s kakšnimi gramatikami lahko opisujemo jezike v tem razredu. Izkaže se, da so to kontekstno neodvisne gramatike z neko dodatno omejitvijo.

Tem gramatikam bomo rekli regularne gramatike.

9.1. DEFINICIJA. Neka kontekstno neodvisna gramatika <V,T,P,S> je desno (levo) linearna, če imajo vse produkcije obliko A --> wB (A --> Bw) ali pa obliko A --> w, pri A,B ∈ V in $w \in T^*$. Če je neka gramatika bodisi desno ali pa levo linearna, ji pravimo regularna.

9.2. PRIMER. Jezik $0(10)^*$ generira desno linearna gramatika

S --> 0A

A --> 10A | ∈ (9.1)

kot tudi levo linearna gramatika

S --> S10 | 0 (9.2)

Ekvivalenca regularnih gramatik in končnih avtomatov

Regularne gramatike opisujejo regularne množice v tem smislu, da je neki jezik regularen natanko, ko ima levo linearno gramatiko in natanko ko ima desno linearno gramatiko. Te rezultate dokažemo v naslednjih dveh izrekih.

9.3. IZREK. Če ima L regularno gramatiko, je L regularna množica.

Dokaz. Najprej predpostavimo, da velja L = L(G) pri neki desno linearni gramatiki G = <V,T,P,S>. Sestavili bomo NKA z ∈-prehodi M = <Q,T,dlt,[S],{[∈]}>, ki simulira izpeljave gramatike G.

Q je sestavljena iz simbolov [alf] tako, da je alf bodisi A ali neka predpona (ni nujno, da je stroga predpona) desne strani neke produkcije iz P.

dlt definiramo takole:

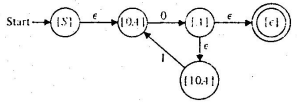
1. Če je A spremenljivka, potem velja dlt([A],∈) = {[alf] | A --> alf je produkcija }.
2. Če je a ∈ T in $alf \in T^* \cup T^* V$, potem je dlt([aalf],a) = {[alf]}.

S preprosto indukcijo po dolžini izpeljave ali izračuna ugotovimo, da dlt([S],w) vsebuje [alf] natanko, ko velja $S \Rightarrow^* xA \Rightarrow xyalf$, pri čemer je A --> yalf produkcija in velja xy = w ali pa ko velja alf = S in w = ∈. Ker je [∈] edino končno stanje, sprejme M besedo w natanko ko velja $S \Rightarrow^* xA \Rightarrow w$. Vendar, ker ima vsaka izpeljava nekega končnega zaporedja vsaj en korak, je razvidno, da M sprejme w natanko ko G generira w. Torej vsaka desno linearna gramatika generira regularno množico.

Sedaj pa naj bo G = <V,T,P,S> levo linearna gramatika. Naj bo G' = <V,T,P',S>, pri čemer sestavljajo P' produkcije G z zrcaljenimi desnimi stranmi, z drugimi besedami:

$P' = \{A \to alf \mid A \to alf^R \text{ pripada } P\}$.

Če zrcalimo produkcije levo linearne gramatike dobimo desno linearno gramatiko in nasprotno. Torej je G' desno linearna gramatika in ni težko pokazati, da velja $L(G') = L(G)^R$. Na podlagi prejšnjega odstavka je L(G') regularna množica (gl. nalogo 3.4g), torej je tudi $L(G')^R = L(G)$ regularna. Tako smo pokazali, da vsaka desno ali levo linearna gramatika definira regularno množico. |_|

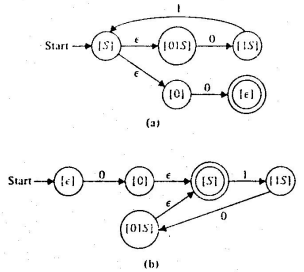


Sl. 9.1. NKA za jezik $0(10)^*$

9.4. PRIMER. NKA, ki nastane z uporabo izreka 9.3 iz gramatike (9.1) je prikazan na sl. 9.1.

Sedaj pa vzemimo gramatiko (9.2). Če njene produkcije zrcalimo, dobimo

S --> 01S | 0



Sl. 9.2. Konstrukcija NKA za $0(10)^*$ iz levo linearne gramatike.

Postopek iz izreka 9.3 v tem primeru daje NKA na sl. 9.2(a). Če pa spremenimo smer povezav tega NKA ter zamenjamo končna in začetna stanja, dobimo drug NKA za $0(10)^*$.

9.5. IZREK. Če je L regularna množica, potem jo je možno generirati z neko levo linearno kakor tudi desno linearno gramatiko.

Dokaz. Naj bo L = L(M) za neki DKA $M = \langle Q, SIGMA, dlt, q_0, F \rangle$. Najprej denimo, da q_0 ni končno stanje. Tedaj velja L = L(G) za neko desno linearno gramatiko $G = \langle Q, SIGMA, P, q_0 \rangle$, pri čemer P vsebuje produkcijo $p \to aq$, kadarkoli velja dlt(p,a) = q, in

tudi produkcijo p --> a v primeru, ko je dlt(p,a) končno stanje. Očitno je dlt(p,w) = q natanko ko velja $p \overset{*}{\Longrightarrow} wq$. Naj M sprejme wa in naj bo $dlt(q_0,w) = p$, iz česar sklepamo $q_0 \overset{*}{\Longrightarrow} wp$. Prav tako je dlt(p,a) končno stanje, torej je p --> a produkcija. Torej končno velja $q_0 \overset{*}{\Longrightarrow} wa$. Nasprotno pa naj velja $q_0 \overset{*}{\Longrightarrow} x$. Tedaj velja x = wa in $q_0 \overset{*}{\Longrightarrow} wp \Longrightarrow wa$ pri nekem stanju (spremenljivki) p. Tedaj velja $dlt(q_0,w) = p$ in je dlt(p,a) končno stanje. Torej je x v L(M). Torej velja L(M) = L(G) = L.

Sedaj pa naj bo q_0 v množici F, tako da velja $\epsilon \in L$. Ugotovimo lahko, da gramatika G, ki smo jo pravkar definirali, generira L - {ϵ}. G sedaj spremenimo tako, da dodamo nov začetni simbol S s produkcijama S --> q_0 | ϵ. Tako pridobljena gramatika je še vedno desno linearna in generira L.

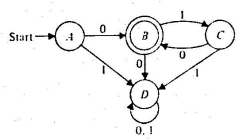
Levo linearno gramatiko za L dobimo tako, da izhajamo iz NKA za L^R in nato zrcalimo desne strani vseh produkcij ustrezne desno linearne gramatike. |_|

9.6. PRIMER. Na sliki 9.3 vidimo DKA za $0(10)^*$. Desno linearna gramatika, ki jo dobimo iz tega avtomata je

A --> 0B | 1D | 0
B --> 0D | 1C |
C --> 0B | 1D | 0
D --> 0D | 1D

Ker pa je simbol D nekoristen, jo lahko prevedemo na gramatiko

A --> 0B | 0
B --> 1C
C --> 0B | 0



Sl. 9.3. DKA za $0(10)^*$.

9.2 Gramatike brez omejitev (gramatike tipa 0)

Po svoji izrazni moči so najmočnejše v hierarhiji Chomskega gramatike, ki dovoljujejo vse možne produkcije oblike alf --> bta, kjer sta alf in bta poljubni besedi nad abecedo V |_| T z edinim pogojem, da velja alf ≠ ϵ. Poleg naziva gramatike brez omejitev se pogosto uporabljajo še poimenovanja gramatike tipa 0 ali gramatike frazne strukture. Tudi v tem primeru uporabljamo oznako G = <V,T,P,S>. Prav tako kot doslej vpeljemo relacijo neposredne izpeljave ==> , pri čemer velja gma o alf o dlt ==> gma o bta o dlt natanko v primeru, ko je alf --> bta neka produkcija. Tako kot doslej predstavlja $\overset{*}{\Longrightarrow}$ refleksivno in tranzitivno ovojnico relacije ==> . Potem pa

zapišemo podobno kot v primeru kontekstno neodvisnih gramatik

$$L(G) = \{w \mid w \in T^* \text{ in } S \Longrightarrow^* w\}$$

9.7. PRIMER. Vzemimo jezik $\{a^{i} \mid i$ je pozitivna potenca 2$\}$. Možno ga je generirati z naslednjo gramatiko brez omejitev:

(1) S --> ACaB, (2) Ca --> aaC, (3) CB --> DB

(4) CB --> E, (5) aD --> Da, (6) AD --> AC

(7) aE --> Ea, (8) AE --> ϵ

A in B sta leva in desna meja generirane besede. C je simbol, ki se pomika po nizu simbolov a od A do B in med vsakim prehodom podvoji število simbolov a (zaradi produkcije (2)). Ko C pride do desne meje B, se spremeni v D ali E in pomika proti levi (zaradi produkcije (5)) dokler ne pride do leve meje A. Takrat se D spremeni v C s produkcijo (6) in se celoten postopek ponovi. V primeru pa, ko se C spremeni v E, je desna meja uničena, in se E pomika proti levi s produkcijo (7), nato pa še uniči sebe in levo mejo A, po čemer ostane le niz a^{i}, pri neki vrednosti i > 0. Z indukcijo po številu korakov v izpeljavi lahko dokažemo, da je v primeru, ko nismo uporabili produkcijo (4), generirana beseda v eni od naslednjih oblik

1. S,
2. $Aa^{i}Ca^{j}B$, pri čemer je i + 2j neka pozitivna potenca 2 ali pa
3. $Aa^{i}Da^{j}B$, pri čemer je i + j neka pozitivna potenca 2.

Po uporabi produkcije (4) ostane beseda $Aa^{i}E$, pri čemer je i pozitivna potenca 2. Nato je edina možnost, da pridemo do konca ta, da i krat uporabimo produkcijo (7), kar da AEa^{i} ter enkrat produkcijo (8), po čemer imamo a^{i}, kjer je i pozitivna potenca 2.

Ekvivalenca gramatik tipa 0 in Turingovih strojev

Namen tega razdelka je pokazati, da gramatike tipa 0 generirajo natanko Turingove jezike. Kot po navadi razpade dokaz na dva dela. V prvem pokažemo, da vsaka gramatika tipa 0 generira neki Turingov jezik, v drugem pa, da se da jezik vsakega Turingovega stroja generirati z neko gramatiko tipa 0. Torej

9.8. IZREK. Naj bo L jezik, ki ga generira neka gramatika tipa 0 G = <V,T,P,S>. Potem je L Turingov jezik.

Dokaz. Sestavili bomo nedeterminističen Turingov stroj M, ki sprejema L. M ima en trak, na katerem je zapisan vhod, na drugem traku pa hrani trenutno generirano besedo alf. Uvodoma M priredi besedi alf vrednost S, nato pa začne izvajati naslednjo zanko:

(1) Na nedeterminističen način izbere neki položaj znotraj besede alf. Z drugimi besedami, postavi se na levi konec besede alf in se začne premikati proti desni, pri vsakem znaku pa naredi odločitev med alternativama, ali nadaljevati, ali pa izbrati trenutni znak.

(2) Na nedeterminističen način izbere neko produkcijo bta --> gma gramatike G.

(3) V primeru, ko se pri izbranem znaku začenja podzaporedje bta, se slednje podzaporedje spremeni v gma. Pri tem M uporablja tehniko prestavljanja vsebine traku, ki je bila opisana v razdelku 7.4, stran 182. Prestavljanje se opravlja v desno v primeru |bta| < |gma| in v levo v nasprotnem primeru.

(4) Tako pridobljeno besedo M primerja z vhodno besedo w na prvem traku. V primeru enakosti, M besedo w sprejme. V nasprotnem primeru se zanka ponovi.

Ni se težko prepričati v dejstvo, da M sprejema le besede, ki jih generira gramatika G. Torej je jezik L(G) Turingov. |_|

Sedaj pa imamo še

9.9. IZREK. V primeru, ko je L Turingov jezik, obstaja neka gramatika tipa 0 G tako, da velja L = L(G).

Dokaz. Naj bo $M = \langle Q,SIGMA,GAMA,dlt,q_0,B,F\rangle$ Turingov stroj, ki sprejema L. Sedaj bomo definirali gramatiko G, ki "nedeterministično" generira dve kopiji neke besede nad SIGMA in nato oponaša delovanje stroja M na eni izmed kopij. V primeru, ko M besedo w sprejme, spremeni G prvo kopijo w v besedo nad končno abecedo, v nasprotnem primeru pa ni možno generirati besede nad končno abecedo.

Naj bo $G = \langle V,SIGMA,P,A_1\rangle$ pri

$$V = ((SIGMA \,|\_|\, \{\epsilon\}) \times GAMA) \,|\_|\, \{A_1, A_2, A_3\}.$$

Produkcije iz P pa tvorimo po naslednjih pravilih:

1. $A_1 \to q_0 A_2$
2. $A_2 \to [a,a]A_2$ za vse $a \in SIGMA$
3. $A_2 \to A_3$
4. $A_3 \to [\epsilon,B]A_3$
5. $A_3 \to \epsilon$
6. $q[a,X] \to [a,Y]p$ za vse $a \in SIGMA \,|\_|\, \{\epsilon\}$ ter vse $q \in Q$, $X, Y \in GAMA$ z lastnostjo $dlt(q,X) = \langle p,Y,D\rangle$.
7. $[b,Z]q[a,X] \to p[b,Z][a,Y]$ za vse X, Y ter $Z \in GAMA$, $a, b \in SIGMA \,|\_|\, \{\epsilon\}$ in $q \in Q$ z lastnostjo $dlt(q,X) = \langle p,Y,L\rangle$.
8. $[a,X]q \to qaq$, $q[a,X] \to qaq$ ter $q \to \epsilon$ za vse $a \in SIGMA \,|\_|\, \{\epsilon\}$ ter $X \in GAMA$ in $q \in F$.

Na podlagi produkcij 1 in 2 ugotovimo

$$A_1 \overset{*}{\Longrightarrow} q_0[a_1,a_1][a_2,a_2]\ldots[a_n,a_n]A_2,$$

pri čemer velja $a_i \in SIGMA$ za vse vrednosti i. Naj stroj M sprejema besedo $a_1 a_2 \ldots a_n$. Torej v tem primeru porabi M neko določeno število m dodatnih celic, desno od vhoda w. Če uporabimo produkcijo 3, nato pa m krat produkcijo 4 in končno produkcijo 5 dobimo

$$A_1 \overset{*}{\Longrightarrow} q_0[a_1,a_1][a_2,a_2]\ldots[a_n,a_n][\epsilon,B]^m.$$

Odslej uporabljamo le produkcije na podlagi pravil 6 in 7 dokler ne generiramo neko končno stanje. Bodimo pozorni na dejstvo, da prve komponente spremenljivk iz množice (SIGMA |_| {Є}) x GAMA ne spreminjamo. Z indukcijo po številu korakov stroja M lahko dokažemo, da iz

$$q_0 a_1 a_2 \ldots a_n \;|\!\!-^{*}_M\; X_1 X_2 \ldots X_{R-1} q X_r \ldots X_s , \tag{9.3}$$

sledi

$$q_0 [a_1,a_1][a_2,a_2]\ldots[a_n,a_n][Є,B] \;\overset{m\ *}{\Longrightarrow}\; [a_1,X_1][a_2,X_2]\ldots[a_{r-1},X_{r-1}]q[a_r,X_r]\ldots [a_{n+m},X_{n+m}], \tag{9.4}$$

pri $a_1, a_2, \ldots, a_n \in$ SIGMA, $a_{n+1} = a_{n+2} = \ldots = a_{n+m} = Є, X_1, X_2, \ldots, X_{n+m} \in$ GAMA ter $X_{s+1} = X_{s+2} = \ldots = X_{n+m} = B$.

Induktivna hipoteza je trivialno resnična za število korakov enako nič, kajti r = 1 in s = n. Sedaj pa denimo, da je resnična pri številu korakov enako k - 1, ter, da velja

$$q_0 a_1 a_2 \ldots a_n \;|\!\!-^{k-1}_M\; X_1 X_2 \ldots X_{r-1} q X_r \ldots X_s$$

$$|\!\!-\; Y_1 Y_2 \ldots Y_{t-1} p Y_t \ldots Y_u .$$

Na podlagi induktivne hipoteze velja

$$q_0 [a_1,a_1]\ldots[a_n,a_n][Є,B] \;\overset{m\ *}{\Longrightarrow}\; [a_1,X_1]\ldots[a_{r-1},X_{r-1}]\, q [a_r,X_r]\ldots[a_{n+m},X_{n+m}],$$

pri čemer znaki a ter X zadoščajo istim pogojem kot v (9.4).

V primeru t = r + 1 je k-ti korak stroja M zvezan z desnim premikom, torej velja $dlt(q,X_r) = \langle p,Y_r,R\rangle$. Na podlagi pravila 6 je $q[a_r,X_r] \dashrightarrow [a_r,Y_r]p$ produkcija gramatike G in torej imamo

$$q_0 [a_1,a_1]\ldots[a_n,a_n][Є,B] \;\overset{m\ *}{\Longrightarrow}\; [a_1,Y_1]\ldots[a_{t-1},Y_{t-1}]p[a_t,Y_t]\ldots[a_{n+m},Y_{n+m}], \tag{9.5}$$

pri $Y_i = B$ pri $i > u$.

V primeru t = r - 1 pa je k-ti korak stroja M zvezan s premikom proti levi in lahko dokažemo (9.5) na podlagi pravila 7 ter ugotovitve, da velja r > 1 ter $dlt(q,X_r) = \langle p,Y_r,L\rangle$.

Na podlagi pravila (8), če velja p Є F, potem

$$[a_1,Y_1]\ldots[a_{t-1},Y_{t-1}]p[a_t,Y_t]\ldots[a_{n+m},Y_{n+m}] \overset{*}{\Longrightarrow} a_1 a_2 \ldots a_n .$$

Torej smo pokazali, da iz $w \in L(M)$ sledi $A_1 \overset{*}{\Longrightarrow} w$, oziroma

w ∈ L(G).

Kaj pa glede nasprotne trditve, da iz w ∈ L(G) sledi w ∈ L(M)? S podobno indukcijo kot smo jo uporabili pravkar dokažemo najprej, da iz (9.4) sledi (9.3). Ta del prepuščamo bralcu. Nato ugotovimo, da brez uporabe pravila (8) ne moremo odpraviti znaka za stanje stroja M. Torej gramatika G ne more generirati besedo nad končno abecedo, če se v generirani besedi ne pojavi končno stanje. Zaradi oblike pravila (8) je beseda nad končno abecedo, ki jo generiramo, enaka prvim komponentam spremenljivk, ki pripadajo množici (SIGMA |_| {∈}) x GAMA, slednje komponente pa ostanejo nespremenjene med simulacijo delovanja stroja M. |¯|

9.3 Kontekstno odvisni jeziki in gramatike

Denimo, da se pri gramatiki tipa 0 domenimo za dodatno omejitev, da mora biti desna stran produkcije alf --> bta vsaj toliko dolga kot leva stran. V takem primeru gre za kontekstno odvisno gramatiko (KOG) oziroma jezik (KOJ). Beseda kontekstno odvisen se nanaša na neko normalno obliko takih gramatik, kjer so vse produkcije oblike $alf_1 A alf_2 \to alf_1 bta alf_2$ pri bta ≠ ∈. Produkcije take oblike so zelo podobne kontekstno neodvisnim produkcijam le, da je spremenljivko A možno nadomestiti z besedo bta samo znotraj "konteksta" (okolice) $alf_1 \_ alf_2$. Izpeljavo te normalne oblike prepuščamo bralcu za vajo.

Skoraj vsak jezik, ki nam pride na misel, je kontekstno odvisen. Jezike, ki niso kontekstno odvisni, za enkrat znamo opisati le z diagonalizacijo. Med njimi so n.pr. jezik L_u iz 8. poglavja, kakor tudi jeziki, na katere lahko L_u prevedemo. Kasneje v razdelku 9.5 bomo dokazali, da eksistirajo jeziki, ki so rekurzivni, niso pa kontekstno odvisni. Pri tem bomo prav tako uporabljali diagonalizacijo.

9.10. PRIMER. Oglejmo si zopet gramatiko iz primera 9.7. Ugotovimo lahko, da obstajata dve produkciji, ki ne zadoščata pogojem kontekstne odvisnosti. To sta produkciji CB --> E ter AE --> ∈. Izkaže pa se, da lahko tudi za jezik $\{a^{2i} \mid i \geq 1\}$ poiščemo kontekstno odvisno gramatiko. Potrebno se je le zavedati dejstva, da rabijo spremenljivke A, B, C, D in E izključno kot znamenja, in jih lahko vključimo v simbole a na ta način, da definiramo sestavljene spremenljivke kot n.pr. [CaB], ki jih obravnavamo kot en sam simbol, ki nadomešča niz CaB.

Celoten nabor sestavljenih simbolov, ki jih potrebujemo za to, da oponašamo gramatiko iz primera 9.7 je [ACaB], [Aa], [ACa], [ADa], [AEa], [Ca], [Da], [Ea], [aCB], [CaB], [aDB], [aE], [DaB] in [aB]. Kontekstno odvisne produkcije za to gramatiko smo tu oštevilčili na podlagi produkcij, ki jih oponašajo:

1. S --> [ACaB]

2. [Ca]a --> aa[Ca], [Ca][aB] --> aa[CaB],
   [ACa]a --> [Aa]a[Ca], [ACa][aB] --> [Aa]a[CaB],
   [ACaB] --> [Aa][aCB], [CaB] --> [Aa][aCB],
   [CaB] --> a[aCB]
3. [aCB] --> [aDB]
4. [aCB] --> [aE]
5. a[Da] --> [Da]a, [aDB] --> [DaB], [Aa][Da] --> [ADa]a,
   a[DaB] --> [Da][aB], [Aa][DaB] --> [ADa][aB]
6. [ADa] --> [ACa]
7. a[Ea] --> [Ea]a, [aE] --> [Ea], [Aa][Ea] --> [AEa]a
8. [AEa] --> a

Ni se težko prepričati v to, da za gramatiko iz primera 9.7 velja $S \overset{*}{\Longrightarrow} alf$ natanko, ko velja $S \overset{*}{\Longrightarrow} alf'$ pri sedanji gramatiki, pri čemer alf' pridobimo iz alf tako, da k vsakemu znaku a pridružimo še znamenja A do E, ki se pojavljajo levo od tega znaka a do naslednjega znaka a. Prav tako moramo k skrajno levemu znaku a pridružiti vsa znamenja levo od njega, k skrajno desnemu znaku a pa vsa znamenja, ki stojijo desno. N.pr. pri alf = AaaCaB imamo alf' = [Aa]a[CaB].

Linearno omejeni avtomati

Sedaj bomo opisali razred avtomatov, ki sprejemajo natanko kontekstno odvisne jezike. Linearno omejeni avtomat (LOA) je nedeterministični Turingov stroj, ki izpolnjuje naslednje pogoje:

1. Vhodna abeceda vsebuje dva posebna znaka, < in >, ki jima pravimo leva in desna meja (po vrsti).
2. LOA se ne more premakniti levo od < in desno od >, niti more zamenjati ta dva simbola z drugimi.

Linearno omejeni avtomati je preprosto Turingov stroj, ki je omejen na tisti del traku, ki vsebuje vhod (pa še dve dodatni celici, ki vsebujeta < ter >). Kasneje bomo videli, da se sposobnosti takega stroja ne spremenijo, če bi namesto toliko traku stroju namenili dolžino traku, ki je linearna funkcija od dolžine vhoda -- odtod ime "linearno omejeni avtomat".

LOA bomo označevali z $M = \langle Q, SIGMA, GAMA, dlt, q_0, <, >, F \rangle$, pri čemer pomenijo simboli Q, SIGMA, GAMA, dlt, q_0 in F isto kot pri navadnem nedeterminističnem Turingovem stroju medtem, ko sta znaka < in > elementa SIGMA (leva in desna meja). Jezik, ki ga sprejema M definiramo kot

$$\{w \mid w \in (SIGMA - \{<,>\})^* \text{ and } q_0 \langle w \rangle \vdash_M^* alfqbta \text{ pri določenih } alf, bta \in GAMA^* \text{ in } q \in F\}.$$

Ekvivalenca med LOA in KOG. Sedaj bomo dokazali, da LOA sprejemajo natanko kontekstno odvisne jezike z edino razliko, da lahko LOA sprejme besedo $\in$ medtem, ko jo nobena KOG ne more generirati.

9.11. IZREK. Naj bo L neki kontekstno odvisen jezik. Potem ga sprejema neki LOA.

Dokaz. Izrek dokažemo skoraj na isti način kot izrek 9.8. Edina razlika je v tem, da medtem, ko je TS iz izreka 9.8 generiral stavčne oblike neke gramatike tipa 0 na drugem traku, jih LOA generira na drugi sledi vhodnega traku. Pri vhodu <w> LOA najprej zapiše znak S v drugo sled pod prvim znakom besede

w. V primeru w = Є se LOA ustavi ne, da bi svojega vhoda sprejel. V nadaljevanju LOA ponavlja neterministično izbiro produkcije ter položaja v drugi sledi. Če je produkcijo možno uporabiti, jo uporabi, pri čemer stavčno obliko, ki jo generira, morebiti širi proti desni. V primeru pa, ko se stavčna oblika, ki jo generira, razširi preko meja besede w, se LOA ustavi, ne da bi svojega vhoda sprejel. Tako bo LOA sprejel besedo w natanko v primeru, ko obstaja izpeljava $S \overset{*}{\Longrightarrow} w$ z lastnostjo, da nobena vmesna stavčna oblika ni daljša od w. Vendar, ker pri kontekstno odvisnih gramatikah desna stran nobene produkcije ni daljša od leve strani, tudi ni možno, da bi neka vmesna stavčna oblika bila daljša od generirane besede. Torej LOA sprejema natanko besede, ki jih generira KOG.

9.12. IZREK. Naj bo L = L(M), pri čemer je M = $\langle Q,SIGMA,GAMA,dlt,q_0,<,>,F\rangle$ LOA. Tedaj je L - {Є} kontekstno odvisen jezik.

Dokaz. Dokaz je podoben konstrukciji gramatike tipa 0, ki oponaša poljuben TS, ki je bila opisana v dokazu izreka 9.9. Vendar sedaj moramo vsa pomožna znamenja, kakor tudi znak za stanje stroja, priključiti k nekemu tračnemu simbolu, kajti v nasprotnem primeru, bi bili prisiljeni krajšati generirano stavčno obliko, ko bi se želeli znebiti teh pomožnih simbolov. Krajšanje stavčne oblike pa zahteva produkcije, ki imajo daljšo levo stran od desne strani; to pa pri KOG ni dovoljeno. Najprej generiramo pare, katerih prve komponente predstavljajo

znake besede, ki jo želimo generirati, druge komponente pa predstavljajo trak LOA. To opravimo s produkcijami

$$A_1 \longrightarrow [a,q_0\langle a]A_2,\quad A_1 \longrightarrow [a,q_0\langle\rangle],$$
$$A_2 \longrightarrow [a,a]A_2,\quad A_2 \longrightarrow [a,a\rangle],$$

za vse a Є SIGMA - {<,>}.

Pravila, ki oponašajo delovanje LOA, so podobna pravilom 6 in 7 v izreku 9.9 in jih prepuščamo bralcu za vajo.

V primeru, ko je q končno stanje, obstaja produkcija

[a,alfqbta] --> a

za vse znake a Є SIGMA - {<,>} ter alf in bta z lastnostjo, da vsebujejo poleg < in > še en tračni simbol. Očitno je število produkcij, ki jih tako dobimo, končno. Poskrbimo tudi za to, da se druga komponenta neke spremenljivke izbriše, če je zraven nekega končnega simbola:

[a,alf]b --> ab, b[a,alf] --> ba,

za vse a in b Є SIGMA - {<,>} ter vsa možna zaporedja alf.

Produkcije, ki smo jih prikazali, so očitno kontekstno odvisne. Tiste produkcije pa, ki oponašajo delovanje LOA, lahko brez težav tako definiramo, da so kontekstno odvisne, iz česar sledi, da je tudi gramatika, ki jo dobimo kontekstno odvisna. Dokaz, da opisana gramatika generira neko besedo w nad končno abecedo natanko, ko isto besedo sprejme podani linearno omejeni avtomat, je podoben dokazu izreka 9.9 in zato ga ne bomo ponavljali. Lahko opazimo, da opisana gramatika ne more zapisati vhodne besede za LOA, ki je enaka <>, niti more oponašati delovanje M na takem vhodu. Torej G ne more generirati besede Є, ne glede na to, ali je ta beseda v L(M) ali

no. |_|

9.4 Relacije med jezikovnimi razredi

Štiri jezikovne razrede, o katerih smo razpravljali (Turingovi jeziki, kontekstno odvisni jeziki, kontekstno neodvisni jeziki in regularni jeziki), pogosto imenujemo jezike tipov 0, 1, 2 in 3 (po vrsti). Lahko dokažemo, da jeziki tipa i strogo vsebujejo jezike tipa i + 1 (če izvzamemo prazno besedo ϵ, ki nam dela preglavice). Najprej pa bomo dokazali, da je vsak KOJ rekurziven in celo, da obstajajo rekurzivni jeziki, ki niso kontekstno odvisni.

Razred KOJ in rekurzivne množice

9.13. IZREK. Vsak KOJ je rekurziven.

Dokaz. Naj bo podana KOG G = <V,T,P,S> ter neka beseda w $\in$ SIGMA dolžine n. Potem lahko preizkusimo, ali velja w $\in$ L(G) takole: Najprej sestavimo graf, katerega vozlišča so besede nad V |_| T dolžine n ali manj. Povezava <alf,bta> pa obstaja natanko, ko velja alf ==> bta. Torej poti v grafu ustrezajo izpeljavam gramatike G in velja w $\in$ L(G) natanko, ko eksistira pot od vozlišča S do vozlišča w. Za preverjanje tega dejstva lahko uporabimo kateregakoli od mnogih algoritmov za iskanje poti po grafih.

9.14. PRIMER. Vzemimo KOG iz primera 9.10 te. vhodno

besedo w = aa. Eden izmed načinov iskanja poti v grafu je, da začnemo z besedo S in na i-tem koraku poiščemo besede dolžine n ali manj, do katerih pelje pot dolžine i ali manj iz S. Če je SS množica takih poti pri i - 1, potem je ustrezna množica pri i enaka SS |_| {bta | alf ==> bta pri nekem alf $\in$ SS ter |bta| $\leq$ n}. V našem primeru dobimo naslednje množice:

```
i = 0:  {S}
i = 1:  {S, [ACaB]}
i = 2:  {S, [ACaB], [Aa][aCB]}
i = 3:  {S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE]}
.
.
.
i = 6:  {S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE],
         [Aa][DaB], [Aa][Ea], [ADa][aB], [AEa]a, [ACa][aB],
aa}
```

Ker pri i = 6 ugotovimo, da do aa pelje pot iz S, se lahko ustavimo. V splošnem primeru pa sklepamo iz dejstva, da je število stavčnih oblik dolžine n ali manj končno pri poljubni podani gramatiki in podanem n, da od nekega trenutka naprej ne bomo generirali novih stavčnih oblik. Ker je množica za i odvisna le od množice za i - 1, ne bomo dodali novih stavčnih oblik in torej, če dotlej nismo generirali besede w, jo tudi kasneje ne bomo, iz česar sklepamo, da w ni v jeziku.

Za dokaz dejstva, da so KOG strogi podrazred rekurzivnih jezikov, bomo dokazali nekaj bolj splošnega. Recimo, da imamo neki razred jezikov, ki jih lahko efektivno naštejemo tako, da z algoritmom generiramo neki seznam Turingovih strojev, ki se ustavijo na vseh vhodih, in da za vsak jezik razreda obstaja vsaj en stroj na seznamu. Potem je ta razred strogi podrazred rekurzivnih jezikov.

9.15. LEMA. Naj bo $M_1, M_2, \ldots$ neki efektiven seznam Turingovih strojev, ki se ustavijo na vseh vhodih. Potem obstaja neki rekurziven jezik, ki je različen od $L(M_i)$ pri vseh i.

Dokaz. Naj bo L podmnožica $(0 + 1)^*$ z lastnostjo, da velja $w \in L$ natanko v primeru, ko velja $w \notin L(M_i)$, pri čemer je i število z dvojiškim zapisom w. Jezik L je rekurziven, kajti pri podanem w lahko generiramo M_i ter ugotovimo, ali velja $w \in L(M_i)$. Vendar noben stroj M_i na seznamu ne sprejema jezika L. Kajti denimo, da velja $L = L(M_j)$ in naj je x dvojiški zapis števila j. V primeru $x \in L$ velja $x \notin L(M_j)$ in v primeru $x \notin L$ velja $x \in L(M_j)$. Torej v vsakem primeru velja $L \neq L(M_j)$, kot smo zatrjevali. Iz tega sklepamo, da pri nobenem j L ni enako $L(M_j)$. |‾|

9.16. IZREK. Eksistira rekurziven jezik, ki ni kontekstno odvisen.

Dokaz. Na podlagi leme 9.15 je potrebno le pokazati, da lahko naštejemo vse Turingove stroje za kontekstno odvisne jezike nad abecedo $\{0,1\}$. Potrebno je najprej določiti neko dvojiško kodiranje za predstavitev kontekstno odvisnih gramatik

v obliki četverk. Naj bo M_j Turingov stroj, ki realizira algoritem iz izreka 9.11, za jezik, ki ga generira KOG z binarno kodo j. Očitno se M_j vedno ustavi ne glede na to, ali sprejme svoj vhod. Potem izrek takoj sledi iz leme 9.15. |‾|

Izrek o hierarhiji

9.17. IZREK. (a) Regularni jeziki so strogi podrazred kontekstno neodvisnih jezikov. (b) Kontekstno neodvisni jeziki, ki ne vsebujejo prazne besede, so strogi podrazred kontekstno odvisnih jezikov. (c) Kontekstno odvisni jeziki so strogi podrazred rekurzivnih jezikov in (d) rekurzivni jeziki so strogi podrazred Turingovih jezikov.

Dokaz. (a) sledi iz dejstva, da je vsaka linearna gramatika kontekstno neodvisna, $\{0^n 1^n \mid n \geq 1\}$ pa je primer jezika, ki je kontekstno neodvisen, ni pa regularen. (b) dokažemo tako, da ugotovimo, da je vsaka kontekstno neodvisna gramatika v normalni obliki po Chomskyju tudi kontekstno odvisna. Za jezik $\{a^{2i} \mid i \geq 1\}$ pa brez težav lahko dokažemo, da je kontekstno odvisen, ni pa kontekstno neodvisen. (c) sledi iz izreka 9.16, (d) pa iz izrekov 8.6 in 8.7. |‾|

9.5 Naloge

9.1. Sestavite levo linearne in desno linearne gramatike za jezike

(a) $(0+1)^*\,00(0+1)^*$

(b) $0\,(1(0+1)^*)^*$

(c) $(((01+10)^*\,11)^*\,00)^*$

9.2. Sestavite gramatike brez omejitev za naslednje jezike

(a) $\{ww^R \mid w \in (0+1)^*\}$

(b) $\{0^{i*i} \mid i \geq 1\}$

(c) $\{0^i \mid i$ ni praštevilo$\}$

(d) $\{0^{i^i} \mid i \geq 1\}$

9.3. Sestavite kontekstno odvisne gramatike za jezike iz naloge 9.3 (brez ϵ v jeziku (a)).

9.4. Pokažite, da

(a) vsak kontekstno odvisen jezik sprejema neki deterministiČen LOA.

(b) je Booleovo zaprtje KNJ vsebovano v razredu jezikov, ki jih sprejemajo deterministični LOA.

(c) je vsebovanost iz (b) stroga. (Navodilo: premišljujte o jezikih nad enosimbolno abecedo.)

9.5. Pokažite, da je problem, ali je neki KOJ prazen, neodločljiv.

10. POGLAVJE

KOMPLEKSNOST IZRAČUNOV

V poglavju 9 smo zgradili hierarhijo jezikov na podlagi "strukturalnih" lastnosti avtomatov (oziroma gramatik), ki so te jezike sprejemali (oziroma generirale). Sedaj se bomo seznanili z neko drugačno hierarhijo, ki sloni na porabi določenih računskih "zmogljivosti" kot je n.pr. število korakov Turingovega stroja (čas računanja) ali pa število celic traku, ki ga stroj potrebuje za to, da izračuna rezultat (prostor). Cela zadeva sloni na občutku, da je Turingov stroj (ali kakršenkoli drug računalnik) zmožen izračunati več stvari oziroma bolj kompliciranе stvari, če mu damo na voljo več časa (več korakov) ali mu dovolimo uporabljati več traku. V tem poglavju želimo ta megleni občutek precizirati. Kot primera računskih zmogljivosti smo navedli število računalniških korakov ali pa število celic traku. Vendar sta to le dva konkretna primera med nešteto možnimi. Kot primer drugačne "zmogljivosti" lahko rabi n.pr. število sprememb smeri gibanja okna Turingovega stroja. Poleg teorije, ki preučuje takšne "konkrentne" zmogljivosti, pa obstaja t.im. aksiomatična

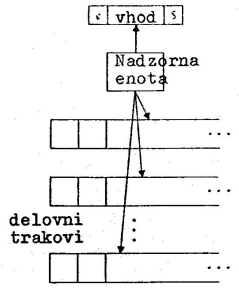
teorija kompleksnosti računanja, ki ne preučuje konkretnih primerov računskih zmogljivosti, temveč postulira določene lastnosti, ki jih morajojo imeti kakršnekoli računalniške zmogljivosti, in iz takšnih lastnosti izpeljuje določene posledice.

10.1 Definicije

Prostorska kompleksnost

Naj imamo Turingov stroj M na sl. 10.1, ki ima poseben vhodni trak z znamenji za levo in desno mejo vhodne besede. Svoj vhodni trak lahko M le bere. Poleg vhodnega traku ima M še k delovnih trakov, ki se prostirajo neskončno v desno. Če za vsako besedo dolžine n M porabi največ S(n) celic na vsakem delovnem traku, potem za M pravimo, da je Turingov stroj s prostorsko omejitvijo S(n). Za jezik stroja M, L(M), prav tako pravimo, da ima prostorsko omejitev ali kompleksnost S(n).

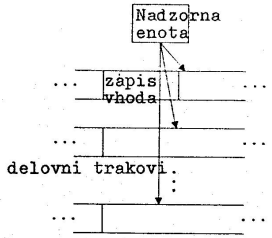
Bodimo pozorni na to, da stroj ne more pisati po svojem vhodu, kakor tudi na to, da upoštevamo le porabo delovnih trakov. Namen takšne omejitve je, da omogočimo preučevanje funkcij S(n), ki rastejo počasneje kot linearna funkcija. Namreč v primeru, ko bi dovolili pisanje po vhodni besedi, bi morali upoštevati celotno vhodno besedo in torej ne bi obstajale prostorske omejitve, ki rastejo počasneje kot n.



sl. 10.1 Večtračni Turingov stroj z vhodnim trakom.

Časovna kompleksnost

Sedaj pa si oglejmo večtračni Turingov stroj M na sl. 10.2. M ima k trakov, ki se razprostirajo na obe strani v neskončnost. M lahko piše po vseh trakovih, vključno s trakom, na katerem je zapisan vhod. Če na vseh možnih vhodih dolžine n stroj M naredi največ T(n) korakov preden se ustavi, potem za M pravimo, da je Turingov stroj s časovno omejitvijo T(n). Za L(M) prav tako pravimo, da ima časovno omejitev ali kompleksnost T(n).



sl. 10.2 Večtračni Turingov stroj

Malenkostne razlike med stroji, ki jih uporabljamo pri preučevanju prostorske in časovne kompleksnosti, izvirajo iz želje, da določene dokaze čim bolj poenostavimo. Če pa n.pr. velja $S(n) \geq n$, potem lahko nadomestimo opisani model z navadnim enotračnim Turingovim strojem ne, brez vpliva na razred jezikov, ki jih takšen stroj sprejema. Pri časovni kompleksnosti pa ne moremo uporabljati enotračnih strojev ali stroje s kakršnimkoli določenim številom trakov ne, da bi pri tem spremenili razreda jezikov, ki jih stroj sprejema.

10.1. PRIMER. Vzemimo jezik

$$L = \{wcw^R \mid w \in (0 + 1)^*\}$$

L ima časovno kompleksnost n + 1, ker za L obstaja dvotračni Turingov stroj M_1, ki deluje tako, da tisti del vhoda, ki leži levo od znaka c, prepiše na drugi trak, potem pa, ko pride do znaka c, spremeni smer gibanja čitalnega okna na drugem traku pri nespremenjeni smeri čitanja na vhodnem traku in primerja vsebini prvega (vhodnega) in drugega traku. V primeru enakosti vsebin je vhodna beseda sprejeta, sicer ne. Očitno potrebuje M_1 toliko korakov, kolikor jih potrebuje za branje vhoda, torej n + 1.

Obstaja pa drugi stroj M_2 za L, ki ima prostorsko omejitev $\log_2 n$. M_2 uporablja dva trakova kot dvojiška števca. Deluje tako, da najprej preveri vhod, ali vsebuje le en znak c in ali je levi del enako dolg kot desni del. Po tem levi del primerja z desnim, pri čemer za iskanje ustreznih simbolov uporablja dva delovna trakova kot števca položaja čitalne glave.

Posebnosti prostorskih in časovnih omejitev

Očitno je, da vsak TS porabi vsaj eno celico traku tako, da velja $S(n) \geq 1$, če je S(n) neka prostorska omejitev. Če za neki stroj ali jezik pravimo, da ima prostorsko omejitev S(n), potem v resnici mislimo na $\max(1, \lceil S(n) \rceil)$. Na primer v primeru 10.1 smo ugotovili, da ima M_2 prostorsko omejitev $\log_2 n$. V primeru n = 0 ali n = 1 to nima smisla razen, če se ne domenimo, da je

$\log_2 n$ pravzaprav okrajšava za $\max(1, \lceil S(n) \rceil)$.

Na podoben način je smiselno, da se domenimo, da je časovna omejitev T(n) večja ali enaka n + 1, kar je število korakov, ki je potrebno, da preberemo vhod in prvi presledek za vhodom[1]. Torej se domenimo, da "časovna omejitev T(n)" pravzaprav pomeni $\max(n + 1, \lceil T(n) \rceil)$. Na primer vrednost časovne omejitve $n \log_2 n$ pri n = 1 je 2 in ne 0, pri n = 2 pa je vrednost 3.

Nedeterministična prostorska in časovna kompleksnost

Opisana pojma lahko prav tako uporabimo pri nedeterminističnih strojih. Na primer nedeterministični TS s časovno omejitvijo T(n) je stroj, pri katerem za vse besede dolžine n obstaja izračun, ki vsebuje največ T(n) korakov. Nedeterministični TS ima prostorsko omejitev S(n) v primeru, ko na nobeni besedi dolžine n ne porabi več kot S(n) celic traku.

Kompleksnostni razredi

Družino jezikov s prostorsko omejitvijo S(n) označujemo z DSPACE(S(n)), družino jezikov z nedeterministično prostorsko omejitvijo S(n) pa označujemo z NSPACE(S(n)). Družino jezikov s časovno omejitvijo T(n) označujemo z DTIME(T(n)), tisto z nedeterministično časovno omejitvijo T(n) pa z NTIME(T(n))[2]. Takim družinam pravimo kompleksnostni razredi. Na primer jezik

---

[1] Seveda obstajajo stroji, ki sprejmejo svoj vhod v času, ki je manjši od dolžine vhodne besede, vendar se domenimo, da takih strojev ne bomo upoštevali.

iz primera 10.1 pripada razredoma DTIME(n)[3] ter DSPACE($\log_2 n$). Potemtakem je L tudi v razredih NTIME(n) in NSPACE($\log_2 n$) kakor tudi v večjih razredih kot n.pr. DTIME(n^2) ali NSPACE($V\overline{n}$).

10.2 Linearna pospešitev in druge transformacije

Turingov stroj ima lahko poljubno število stanj kakor tudi poljubno veliko tračno abecedo. Zaradi tega lahko vedno skrajšamo trak oziroma zmanjšamo število porabljenih celic traku za konstanten faktor tako, da prekodiramo več tračnih celic v eno. Na podoben način lahko pospešimo izračun (zmanjšamo število porabljenih korakov) za konstanten faktor. Na podlagi tega nas bo odslej pravzaprav zanimal le velikostni razred prostorskih oziroma časovnih omejitev (n.pr. linearen, kvadratičen, eksponencialen), konstantne faktorje pa ne bomo upoštevali. Na primer govorimo lahko o kompleksnosti log n, ne da bi izbrali logaritemsko osnovo, kajti $\log_b n$ in $\log_c n$ se razlikujeta le za konstanten faktor $\log_b c$. V tem razdelku bomo izpeljali osnovne rezultate, ki se nanašajo na linearno pospešitev in krajšanje traku kakor tudi na vpliv števila trakov

---

[2] Oznake seveda izvirajo iz angleških besed za prostor in čas.

[3] Spomnimo se, da n pravzaprav pomeni $\max(n + 1, n) = n + 1$ v primeru časovne kompleksnosti

na kompleksnost.

Krajšanje trakov

10.2. IZREK. Če za jezik L obstaja k-tračni Turingov stroj s prostorsko omejitvijo S(n), potem obstaja zanj tudi k-tračni Turingov stroj s prostorsko omejitvijo cS(n), pri poljubnem $c > 0$.[4]

Dokaz Naj bo M_1 Turingov stroj s prostorsko omejitvijo S(n), ki sprejema L. Izrek dokažemo tako, da sestavimo nov Turingov stroj M_2, ki oponaša M_1, in ki hrani v eni svoji tračni celici vsebino r tračnih celic stroja M_1 pri določeni vrednosti konstante r. Nadzorna enota M_2 pa upošteva, katero celico (med r celicami) stroja M_1 čitalna glava dejansko bere.

Podrobnosti definicije stroja M_2 prepuščamo bralcu. r izberemo tako, da velja $rc \geq 2$. M_2 potem lahko oponaša delovanje M_1, pri čemer ne porabi več kot $|\overline{S(n)/r}|$ celic na kateremkoli traku. V primeru $S(n) \geq r$ to število ne presega cS(n). Če pa velja $S(n) < r$, potem lahko M_2 shrani vsebino

---

[4] Po našem dogovoru cS(n) pravzaprav predstavlja omejitev $\max(1, |\overline{cS(n)}|)$.

eno celico. |_|

10.3. POSLEDICA. Če L pripada razredu NSPACE(S(n)), potem pripada tudi razredu NSPACE(cS(n)), kjer je c poljubno število > 0.

Dokaz. V kolikor je M_1 iz predhodnega dokaza nedeterminističen stroj, je edina potrebna sprememba v tem, da mora tudi M_2 biti nedeterminističen. |_|

Odpravljanje trakov pri prostorskih kompleksnostnih razredih

10.4. IZREK. Če je L jezik nekega k-tračnega TS s prostorsko omejitvijo S(n), potem je L tudi jezik nekega enotračnega TS s prostorsko omejitvijo S(n).

Dokaz. Naj bo M_1 k-tračni TS s prostorsko omejitvijo S(n), ki sprejema L. Sedaj lahko sestavimo nov TS M_2 z enim samim trakom, ki oponaša vseh k trakov stroja M_1 uporabljajoč pri tem k sledi. Podrobnosti so opisane v dokazu izreka 7.5. M_2 ne uporablja več kot S(n) celic. |_|

Odslej bomo predpostavljali, da ima vsak TS s prostorsko omejitvijo S(n) le en trak in v primeru S(n) $\geq$ n bo to navaden enotračni TS in ne stroj s posebnim vhodnim trakom, ki je opisan zgoraj.

Linearna pospešitev delovanja Turingovega stroja

Predno se začnemo ukvarjati s časovnimi mejami bomo vpeljali naslednjo notacijo. Naj bo f(n) neka funkcija od n. Izraz $\sup_{n \to \infty} f(n)$ je okrajšava za limito najmanjše zgornje meje zaporedja

f(n),f(n+1), f(n+2), ...

pri n --> oo. Na podoben način predstavlja $\inf_{n \to \infty} f(n)$ limito največje spodnje meje istega zaporedja pri n --> oo. V kolikor f(n) konvergira proti neki limiti pri n --> oo, potem je ta limita enaka tako $\inf_{n \to \infty} f(n)$ kot $\sup_{n \to \infty} f(n)$.

10.5. PRIMER. Naj bo f(n) = 1/n pri sodih n in f(n) = n pri lihih n. Najmanjša zgornja meja zaporedja f(n), f(n+1), ... je očitno oo pri poljubnem n zaradi vrednosti f(n) pri lihih n. Torej $\sup_{n \to \infty} f(n) = \infty$. Vendar zaradi vrednosti f pri sodih n velja tudi $\inf_{n \to \infty} f(n) = 0$.

Sedaj pa denimo, da velja f(n) = n/(n + 1). V tem primeru je najmanjša zgornja meja zaporedja n/(n + 1), (n + 1)/(n + 2), ... enaka 1 pri poljubnem n. Torej velja

$$\sup_{n \to \infty} n/(n+1) = 1.$$

Največja spodnja meja zaporedja n/(n + 1),(n + 1)/(n + 2), ... je n/(n + 1), in ker velja $\lim_{n \to \infty} n/(n+1) = 1$ prav tako velja $\inf_{n \to \infty} n/(n+1) = 1$.

10.6. IZREK. Če je L jezik nekega k-tračnega TS M_1 s časovno omejitvijo T(n), potem je L tudi jezik nekega k-tračnega TS M_2 s časovno omejitvijo cT(n) pri poljubnem c > 0, pod pogojem da velja k > 1 ter $\inf_{n \to \infty} T(n)/n = \infty$.

Dokaz. TS M_2, ki oponaša M_1, je možno sestaviti na naslednji način. Najprej M_2 prepiše svoj vhod na neki delovni trak tako, da stisne m znakov v enega (vrednost m bomo določili naknadno). Odslej uporablja M_2 ta trak kot vhodni trak in vhodni trak namesto delovnega. M_2 prekodira vsebino delovnih trakov stroja M_1 tako, da stisne m znakov v enega. Pri svojem delovanju M_2 oponaša več korakov stroja M_1 z enim sestavljenim korakom, ki ga tvori osem elementarnih korakov (stroja M_2). Naj bodo celice, ki jih trenutno pokrivajo okna stroja M_2 trenutne

celice. Nadzorna enota M_2 ima za vsak trak v evidenci, katerega od m simbolov M_1, ki jih hrani trenutna celica M_2, pravzaprav stroj M_2 bere.

Na začetku sestavljenega koraka M_2 premakne vsa čitalna okna enkrat v levo, dvakrat v desno in zopet enkrat v levo. Tako lahko nadzorna enota ugotovi vsebino celic levo in desno od trenutne celice. Za to operacijo so potrebni štirje koraki stroja M_2. Po končani operaciji pa M_2 zopet bere trenutne celice.

Po tem M_2 izračuna vsebino vseh celic stroja M_1, ki so shranjene v trenutnih celicah stroja M_2 in njihovih levih ter desnih sosedih v trenutku, ko neka čitalna glava M_1 zapusti področje traku, ki je shranjeno v trenutni celici in njenih sosedih. (Ta izračun stroja M_2 ne zahteva nobenih dodatnih korakov, ker ga lahko vgradimo v funkcijo prehodov nadzorne enote M_2.) V primeru, ko M_1 sprejme svoj vhod pred tem trenutkom, ga sprejme tudi M_2. V primeru, ko se M_1 ustavi pred tem trenutkom, se ustavi tudi M_2. Sicer pa M_2 obišče na vseh trakovih obe sosedi trenutne celice ter spremeni njihovo vsebino, kakor tudi vsebino trenutne celice, če je to potrebno. Končno M_2 premakne vsa čitalna okna do celic, ki vsebujejo tiste

znake M_1, ki jih čitalna okna M_1 sedaj berejo. Za vse to so potrebni največ štirje koraki stroja M_2.

Stroj M_1 porabi najmanj m korakov za to, da premakne neko svoje čitalno okno izven področja, ki ga predstavlja trenutna celica in njene sosede. Torej z osmimi koraki je stroj M_2 oponašal najmanj m korakov stroja M_1. Sedaj pa izberemo m tako, da velja $cm \geq 16$.

Če M_1 napravi T(n) korakov, potem jih M_2 oponaša z največ svojimi $8\lceil T(n)/m \rceil$ koraki. Poleg tega mora M_2 prepisati ter prekodirati svoj vhod (m celic stisne v eno) ter vrniti čitalno okno novega vhodnega traku do levega roba. Vse to zahteva $n + \lceil n/m \rceil$ korakov in je torej celotna poraba časa

$$n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil \qquad (10.1)$$

korakov. Ker velja $\lceil x \rceil < x + 1$ pri vseh x, je zgornja meja za (10.1)

$$n + n/m + 8T(n)/m + 9. \qquad (10.2)$$

Ker smo predpostavljali $\inf_{n \to \infty} T(n)/n = \infty$, pa za poljubno konstanto d obstaja neki n_d z lastnostjo, da za vse

$n \geq n_d$ velja $T(n)/n \geq d$, ali z drugimi besedami $n \leq T(n)/d$. Torej kadarkoli velja $n \geq 9$ (v tem primeru velja tudi $n + 9 \leq 2n$) in $n \geq n_d$ je (10.2) omejeno zgoraj s količino

$$T(n)\left(\frac{8}{m} + \frac{2}{d} + \frac{1}{md}\right). \qquad (10.3)$$

Doslej še nismo izbrali vrednosti d. m naj bo tako, da velja $cm \geq 16$. Če sedaj izberemo $d = m/4 + \frac{1}{8}$ ter v (10.3) zamenjamo m z 16/c, ugotovimo, da pri vseh $n \geq \max(2, n_d)$ število korakov M_2 ne bo presegalo cT(n).

Kako pa obravnavati besede, ki so krajše od $\max(9, n_d)$? Takih besed je končno mnogo in jih lahko M_2 razpozna po n + 1 korakih, uporabljajoč pri tem le svojo nadzorno enoto. To pa je čas, ki je potreben za branje vhodne besede. Torej je časovna kompleksnost M_2 cT(n). Bralec naj upošteva dejstvo, da pri časovni kompleksnosti cT(n) pravzaprav predstavlja $\max(n + 1, \lceil cT(n) \rceil)$. |_|

10.7. POSLEDICA. Če velja $\inf_{n \to \infty} T(n)/n = \infty$ ter $c > 0$, potem velja tudi

DTIME(T(n)) = DTIME(cT(n)).

Dokaz. Že sam izrek 10.6 predstavlja dokaz posledice v primeru, ko imamo za jezik L stroj z 2 ali več trakovi. Če pa je jezik L tak, da zanj obstaja enotračni TS, potem zanj obstaja očitno tudi dvotračni TS z isto časovno mejo in torej lahko zopet uporabimo isti izrek. |_|

Izrek 10.6 ni uporaben, če je T(n) večkratnik n, kajti v tem primeru je količina $\inf_{n \to \infty} T(n)/n$ konstanta in ne neskončno. Vendar nam konstrukcija iz izreka 10.6, če si jo natančneje ogledamo, razodene naslednje dejstvo.

10.8. IZREK. Če je L jezik nekega k-tračnega TS s časovno mejo cn pri $k > 1$ ter poljubnem c, potem je za vsako $\epsilon > 0$ L tudi jezik nekega k-tračnega TS s časovno omejitvijo $(1 + \epsilon)n$.

Dokaz. V dokazu izreka 10.6 izberemo $m = 1/16\epsilon$. |_|

10.9. POSLEDICA. Če velja T(n) = cn pri nekem $c > 1$, potem DTIME(T(n)) = DTIME($(1 + \epsilon)n$) pri poljubnem $\epsilon > 0$.

10.10. POSLEDICA (izrekov 10.6 in 10.8).

(a) Če velja $\inf_{n \to \infty} T(n)/n = \infty$, velja tudi NTIME(T(N)) = NTIME(cT(n)) pri poljubnem $c > 0$.

(b) Če velja $T(n) = cn$ pri nekem konstantnem c, potem velja $NTIME(T(n)) = NTIME((1 + \epsilon)n)$ pri poljubnem $\epsilon > 0$.

Dokaz. Dokaza sta podobna dokazoma izrekv 10.6 in 10.8. |‾|

Odpravljanje trakov pri časovnih kompleksnostnih razredih

Oglejmo si sedaj, kaj se zgodi s časovno kompleksnostjo, če se omejimo samo na en trak. N.pr. jezik $L = \{wcw^R | w \in (a + b)^*\}$ zahteva le linearen čas na dvotračnem stroju, kot smo to videli v primeru 10.1. Če pa uporabljamo enotračni stroj je potrebno število korakov cn^2 pri nekem $c > 0$ (to dejstvo ne bomo dokazali). Torej, ko smo zmanjšali število trakov na ena, se je število potrebnih korakov povečalo s kvadratom. To pa je tudi najhuje, kar se lahko zgodi, kar nam je zagotovljeno z naslednjim rezultatom:

10.11. IZREK. Če velja $L \in DTIME(T(n))$, potem je L jezik nekega enotračnega TS s časovno mejo $T^2(n)$.

Dokaz. V konstrukciji iz dokaza izreka 7.5 uporablja enotračni TS za simulacijo T(n) korakov večtračnega TS največ $cT^2(n)$ pri neki konstanti c. Izrek 10.6 nam zagotavlja, da prvotni večtračni TS lahko pospešimo tako, da bo porabil le $T(n)/V\bar{a}$ korakov. Potemtakem bo pridoblejni enotračni stroj porabil kvečjemu $T^2(n)$ korakov. |‾|

10.12. POSLEDICA. Če velja $L \in NTIME(T(n))$, potem je L jezik nekega enotračnega nedeterminističnega TS s časovno omejitvijo $T^2(n)$.

Dokaz. Podoben dokazu izreka. |‾|

Če se omejimo na dva trakova, nam naslednji rezultat razodene, da je izguba na hitrosti znatno manjša kot, če dovolimo le en trak.

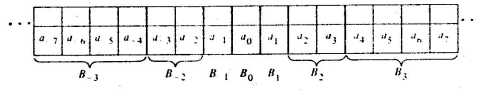
10.13. IZREK. Če je L jezik nekega k-tračnega TS M_1 s časovno mejo T(n), potem je tudi jezik nekega dvotračnega TS M_2 s časovno mejo $T(n)\log T(n)$.

Dokaz. Prvi delovni trak stroja M_2 ima za vsak delovni trak stroja M_1 dve sledi. Razumevanje si olajšamo tako, da se osredotočimo le na tisti dve sledi, ki ustrezata nekemu določenemu traku. Preostale trakove oponašamo na isti način. Drugi trak stroja M_2 ima le pomožno vlogo pri prestavljanju podatkov na prvem traku.

Ena celica prvega traku, ki jo zaznamujemo z B_0, bo vsebovala tračne znake, ki jih pokrivajo vsa okna stroja M_1. M_2 bo deloval tako, da bo premikal podatke skozi B_0 v nasprotni smeri od smeri gibanja oken namesto, da bi premikal okna. Na ta način lahko M_2 oponaša en korak stroja M_1 tako, da prebere B_0.

Desno od celice B_0 so bloki celic B_1, B_2, ... , katerih dolžina raste eksponentno. Dolžina B_i je 2^{i-1}. Podobno so na levo od B_0 bloki B_{-1}, B_{-2}, ..., pri čemer ima B_{-i} dolžino 2^{i-1}. Med bloki predpostavljamo, da obstajajo znamenja, ki pa so lahko pridružena vsebini prve celice bloka.

Naj je a_0 vsebina celice, ki je trenutno pod čitalnim oknom stroja M_1. Vsebine celic, ki so desno od te celice, so a_1, a_2, ..., vsebine celic na levi pa so a_{-1}, a_{-2}, ... Vrednosti a_i se lahko spreminjajo pri vstopu v celico B_0, vendar vrednosti niti niso važne, pomembni so le njihovi položaji na prvem traku stroja M_2. Uvodoma je zgornja sled prvega traku prazna, druga pa vsebuje ..., a_{-2}, a_{-1}, a_0, a_1, a_2, ... Te vrednosti so shranjene v blokih ..., B_{-2}, B_{-1}, B_0, B_1, B_2, ... , kot je to prikazano na sl. 10.3.



sl. 10.3 Bloki prvega traku.

Kot smo že omenili, se bodo podatki pretakali skozi celico B_0, kjer bodo morebiti podvrženi tudi spremembam. Po oponašanju vsakega koraka stroja M_1 veljajo naslednji pogoji.

(1) Pri vseh $i > 0$ sta bodisi obe sledi bloka B_i polni in je B_{-i} popolnoma prazen, ali nasprotno (obe sledi bloka B_{-i} polni in B_i prazen), ali pa sta spodnji sledi obeh blokov B_i in B_{-i} polni medtem, ko sta zgornji sledi prazni.

(2) Vsebini B_i ali B_{-i} predstavljata zaporedne celice na traku stroja M_1. Pri $i > 0$ predstavlja zgornja sled celice, ki stojijo levo od celic, ki ustrezajo spodnji sledi medtem, ko pri $i < 0$ velja nasprotno: zgornja sled vsebuje celice, ki stojijo desno od celic, ki so predstavljene v spodnji sledi.

(3) Pri $i < j$ predstavlja B_i celice, ki stojijo levo od celic iz B_j.

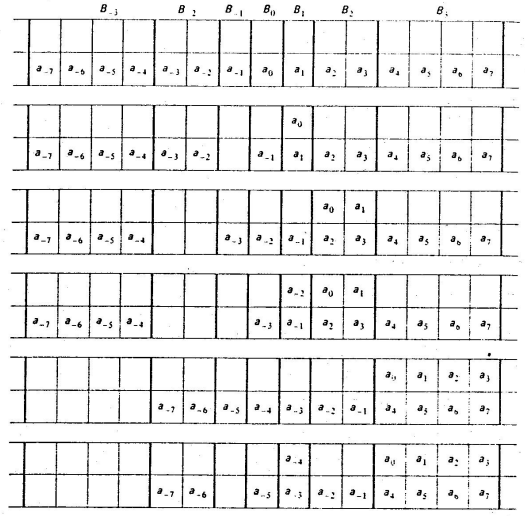
(4) B_0 ima zasedeno le spodnjo sled medtem, ko je zgornja sled posebno zaznamovana.

Da si bomo predstavljali, kako prenašamo podatke, denimo, da se ustrezno čitalno okno stroja M_1 premakne v levo. Torej mora M_2 prestaviti ustrezne podatke proti desni. M_2 to opravi tako, da najprej premakne svoje okno na prvem traku proti desni do prvega bloka, denimo B_i, ki nima obe sledi zasedeni. Nato M_2 prepiše vse podatke iz $B_0, B_1, \ldots, B_{i-1}$ na drugi trak in jih nazaj zapiše v spodnjo sled $B_1, B_2, \ldots, B_{i-1}$ ter spodnjo sled B_i, pod pogojem, da slednja sled še ni zasedena. V primeru, da je spodnja sled B_i zasedena, se podatki zapišejo v zgornjo sled B_i. V obeh primerih je prostora ravno dovolj za opisane podatke. Važno si je pri tem zapomniti, da je omenjeno operacijo možno opraviti v številu korakov, ki je sorazmerno dolžini B_i.

Po tem lahko M_2 poišče blok B_{-i} v številu korakov, ki je prav tako sorazmerno dolžini B_i (pri tem uporablja drugi trak za odmero razdalje med B_0 in B_i). V primeru, ko je B_{-i} popolnoma poln, M_2 prepiše zgornjo sled B_{-i} na drugi trak. Če pa je B_{-i} le napol poln, se prepiše spodnja sled. V obeh primerih se

vsebina drugega traku prepiše nazaj na prvi trak v spodnjo sled blokov $B_{-(i-1)}, B_{-(i-2)}, \ldots, B_0$ (na podlagi prvega pravila zgoraj je spodnja sled omenjenih blokov prazna, ker so simetrično ležeči bloki popolnoma polni). Zopet ugotavljamo, da je prostora ravno dovolj za omenjene podatke in da opisano operacijo lahko opravimo v času, ki je sorazmeren dolžini B_i. Prav tako ugotavljamo, da po opravljeni operaciji zopet veljajo pravila oziroma pogoji 1-4 zgoraj.

Temu, kar smo opisali, pravimo B_i operacija. V primeru, ko se čitalno okno stroja M_1 premakne proti levi, postopamo podobno. Za ilustracijo je na sl. 10.4 prikazana vsebina prvega traku M_2 pri oponašanju petih zaporednih premikov v levo stroja M_1.



sl. 10.4 Bloki prvega traku stroja M2.

Ugotovimo lahko, da mora M_2 za vsak trak M_1 opraviti B_i-operacijo največ enkrat na 2^{i-1} korakov stroja M_1, kajti toliko korakov je potrebnih, da se bloki $B_1, B_2, \ldots, B_{i-1}$, ki so po predhodni B_i operaciji polprazni, zopet napolnijo. Prav tako velja, da do prve B_i operacije ne more priti pred 2^{i-1} korakom stroja M_1. Iz tega sledi, da v primeru, ko ima M_1 časovno mejo T(n), bo M_2 opravil B_i operacije kvečjemu pri $i \leq \log_2 T(n) + 1$.

Izvedeli smo, da B_i operacija zahteva $m2^i$ korakov pri neki konstanti m. Če ima M_1 časovno mejo T(n), potem M_2 naredi kvečjemu

$$T_1(n) = \text{VSOTA}_{i=1}^{\log_2 T(n)+1} m2^i \left\lceil \frac{T(n)}{2^{i-1}} \right\rceil \qquad (10.4)$$

korakov pri simulaciji enega traku stroja M_1.

Iz (10.4) dobimo

$$T_1(n) = 2mT(n)\left\lceil \log_2 T(n) + 1 \right\rceil, \qquad (10.5)$$

nato pa iz (10.5)

$$T_1(n) < 4mT(n)\log_2 T(n).$$

Bralec bi moral videti, da M_2 potrebuje čas, ki je sorazmeren $T_1(n)$ tudi v primeru, ko M_1 uporablja različne trakove in ne le tistega, na katerega smo se osredotočili. Nato pa lahko na podlagi izreka 11.6 M_2 priredimo, da bo porabil

kvečjemu $T(n)\log_2 T(n)$ korakov. |¯|

10.14. POSLEDICA. Če za L obstaja k-tračni nedeterministični TS s časovno mejo T(n), potem zanj obstaja tudi dvotračni nedeterministični TS s časovno mejo $T(n)\log T(n)$.

Dokaz. Podoben dokazu prejšnega izreka. |¯|

10.3 Hierarhije

Občutek nam narekuje, da, če si vzamemo več časa, potem bi morali razpoznati večji razred jezikov. Vendar nam izreka o linearni pospešitvi ter stisljivosti razodevata, da moramo čas ali prostor povečati za več kot konstanten faktor. Kaj pa če časovno omejitev pomnožimo z neko izredno počasi naraščajočo funkcijo, kot je n.pr. log(log(n))? Ali se lahko zgodi, da v takem primeru ne moremo razpoznati nič več kot smo z manjšo časovno omejitvijo? Ali morda obstaja neka časovna oziroma prostorska omejitev f(n) z lastnostjo, da je vsak rekurziven jezik v razredu DTIME(f(n)) ali morda DSPACE(f(n))?

Odgovor na zadnje vprašanje je "ne", kot bo to razvidno iz naslednjega izreka. Odgovor na prejšnje vprašanje pa je odvisen od tega, ali začnemo z nekakšno "lepo" funkcijo. V tem razdelku bomo podali primerno definicijo "lepih" funkcij in pokazali, da nam majhne spremembe v časovni ali prostorski omejitvi povečajo razred jezikov, ki jih lahko sprejemamo.

Končno bomo prikazali na primeru izreka o "vrzelih", da moramo biti v primeru, ko omejitev ni "lepa" funkcija, pripravljeni na vse in lahko dokažemo marsikatero "patološko" lastnost časovnih oziroma prostorskih omejitev.

10.15. IZREK. Naj bo T(n) poljubna rekurzivna časovna ali prostorska omejitev. Potem obstaja rekurziven jezik L, ki ni v DTIME(T(n)) ali DSPACE(T(n)) (po vrsti).

Dokaz. Trditev bomo izpeljali za primer časovne omejitve. Prostorsko omejitev obravnavamo podobno. Dokaz se v osnovi opira na diagonalizacijo. Ker je T(n) totalna rekurzivna funkcija, obstaja zanjo TS M, ki se vedno ustavi. Sestavili bomo stroj $\bar{M}$ za jezik $L \subseteq (0+1)^*$, ki je rekurziven, ni pa v razredu DTIME(T(n)).

Naj bo x_i i-ta beseda v kanonski urejenosti $(0+1)^*$. V poglavju 9 smo uredili enotračne TS s tračno abecedo {0, 1, B}. Na podoben način lahko uredimo večtračne TS s poljubno tračno abecedo. Pri kodiranju Turingovih strojev z vhodno abecedo {0, 1} in poljubno tračno abecedo z dvojiškimi besedami sprejmemo dogovor, da 0, 1 in B zakodiramo z 0, 00 in 000, po vrsti, ostale znake, ki jih lahko istovetimo z $X_4, X_5, \ldots, X_k$ za neko končno k, pa z 0^i, $4 \leq i \leq k$. V tem primeru se še domenimo, da pred vsako kodo nekega TS dovolimo poljubno število znakov 1, s čimer dosežemo, da ima vsak TS poljubno dolge kode.

Na ta način lahko govorimo o i-tem večtračnem stroju M_i. Sedaj definiramo jezik

$$L = \{x_i \mid M_i \text{ ne sprejme } x_i \text{ v manj kot } T(|x_i|) \text{ korakih}\}.$$

Trdimo, da je L rekurziven jezik. Za sprejemanje jezika L je potrebno izvajati naslednji algoritem, za katerega je očitno, da ga je možno uresničiti na TS. Pri podanem vhodu w dolžine n najprej simuliramo M in izračunamo T(n). Nato poiščemo i tako, da velja $w = x_i$. Dvojiški zapis števila i je funkcija prehodov nekega večtračnega TS M_i (v primeru, ko dvojiški zapis i ni v pravilni obliki, je stroj M_i brez prehodov). Sedaj simuliramo M_i na w za T(n) korakov in besedo w sprejmemo v primeru, ko se bodisi M_i ustavi, ne da bi sprejel w, ali pa ko porabi na w več kot T(n) korakov.

Pokažimo sedaj, da L ne pripada DTIME(T(n)). Denimo, da velja $L = L(M_i)$, in da ima M_i časovno omejitev T(n). Ali velja $x_i \in L$? Če je odgovor da, M_i porabi za x_i kvečjemu T(n) korakov pri $n = |x_i|$. Na podlagi definicije L potem x_i ne pripada L, kar je protislovje. Če je odgovor ne, potem M_i ne sprejme x_i, iz česar pa sledi ponovno protislovje, na podlagi definicije L. Ker obe predpostavki pripeljeta v protislovje, je hipoteza, da ima M_i časovno omejitev T(n), neresnična. |_|

V primeru $T'(n) \geq T(n)$ pri vseh n takoj sledi iz definicije časovnega kompleksnostnega razreda, da velja

$\text{DTIME}(T(n)) \subseteq \text{DTIME}(T'(n))$. V primeru, ko je T(n) totalna rekurzivna funkcija, nam izrek 10.15 zagotavlja eksistenco rekurzivne množice L, ki ni v DTIME(T(n)): naj bo $\overline{T}(n)$ časovna omejitev nekega TS za jezik L in naj bo $T'(n) = \max(T(n), \overline{T}(n))$. Tedaj velja $\text{DTIME}(T(n)) \subseteq \text{DTIME}(T'(n))$, ne pa $\text{DTIME}(T(n)) = \text{DTIME}(T'(n))$, ker je L v drugi množici, ne pa v prvi. Na podlagi tega sklepamo, da obstaja neskončna hierarhija deterministični časovnih kompleksnostnih razredov. Podoben rezultat velja za deterministične prostorske razrede kakor tudi za nedeterministične časovne in prostorske razrede.

Izrek 10.15 nam zagotavlja, da za poljubno totalno rekurzivno časovno ali prostorsko omejitev f(n) obstaja f'(n), katere razred vsebuje neki jezik, ki ni vsebovan v razredu f(n). Sedaj bomo pokazali, da mora v primeru prostorskih razredov f'(n) rasti le neznatno hitreje od f(n), če je f(n) "lepa" funkcija. Naslednja dva izreka nam povesta nekaj o tem kako se mora obnašati f'(n) za to, da pridobimo nov razred. Tukaj pa opozarjamo, da je podobne rezultate za nedeterministične razrede izredno težko dobiti.

Prostorska hierarhija

Sedaj vpeljemo našo definicijo "lepe" kompleksnostne omejitve. Za funkcijo S(n) pravimo, da je prostorsko verna, če obstaja TS M, ki ima prostorsko omejitev S(n), in za vsak n obstaja neki vhod dolžine n, pri katerem M dejansko porabi S(n) celic. Množica prostorsko vernih funkcij vsebuje $\log n$, n^k, 2^n in n! Če sta $S_1(n)$ ter $S_2(n)$ prostorsko verni, so to tudi $S_1(n)S_2(n)$, $2^{S1(n)}$ ter $S_1(n)^{S2(n)}$. Torej je množica prostorsko vernih funkcij zelo bogata.

Poudarjamo: ni nujno, da opisani stroj M porabi S(n) prostora pri vseh vhodih dolžine n, temveč le na nekem vhodu takšne dolžine. V primeru pa, ko M porabi natanko S(n) celic na vseh vhodih dolžine n, pravimo, da je S(n) popolnoma prostorsko verna. Za vajo naj bralec dokaže, da je vsaka prostorsko verna funkcija $S(n) \geq n$ popolnoma prostorsko verna.

Da bi poenostavili rezultat, ki sledi, pa bomo zapisali še naslednjega:

10.16. LEMA. Naj bo L jezik nekega TS s prostorsko omejitvijo $S(n) \geq \log_2 n$. Potem obstaja za L tudi TS s prostorsko omejitvijo S(n), ki se ustavi na vseh vhodih.

Dokaz. Naj bo M TS s posebnim vhodnim trakom in prostorsko omejitvijo S(n), ki ima s stanj in t tračnih znakov. Naj bo L jezik M. Če M sprejema neko besedo, je za to potrebno kvečjemu $(n + 2)sS(n)t^{S(n)}$ korakov, kajti v nasprotnem primeru pride do

ponovitve nekega trenutnega opisa. To sledi iz dejstva, da obstaja n + 2 položajev okna na vhodnem traku, s stanj, S(n) položajev oken na delovnih trakovih in $t^{S(n)}$ različnih tračnih vsebin. Če dodamo še eno sled kot števec korakov, se lahko M ustavi po $(4st)^{S(n)} \geq (n + 2)sS(n)t^{S(n)}$ korakih. Pravzaprav si M pripravi števec dolžine $\log n$, šteje pa s številčno osnovo 4st. Kadarkoli M porabi novo celico, ki je izven področja, ki ga trenutno zaseda števec, si M podaljša tudi števec. Na ta način, če M zaide v cikel pri porabljenem prostoru i, se bo to pokazalo na števcu, ko bo vseboval vrednost $(4st)^{\max(i,\log_2 n)}$, kar ni manjše od $(n + 2)sS(n)t^{S(n)}$. |_|

10.17. IZREK. Če je $S_2(n)$ popolnoma prostorsko verna, če velja

$$\inf_{n \to \infty} S_1(n)/S_2(n) = 0$$

in če sta $S_1(n)$ ter $S_2(n)$ najmanj enaki $\log_2 n$, potem obstaja jezik, ki je v $DSPACE(S_2(n))$, ni pa v $DSPACE(S_1(n))$.

Dokaz. Izrek dokažemo z diagonalizacijo. Opirali se bomo na neko oštevilčenje TS s posebnim vhodnim trakom in z vhodno abecedo {0, 1} ter z enim delovnim trakom z dodatno lastnostjo, da ima vsak TS poljubno dolge kode (glej dokaz izreka 11.15).

Sestavili bomo TS M, ki porabi prostor $S_2(n)$, in čigar izračun se vsaj pri enem vhodu ne sklada s poljubnim TS s prostorsko omejitvijo $S_1(n)$.

Pri vhodu w M najprej zaznamuje $S_2(n)$ celic na traku, kjer je n dolžina w. Ker je $S_2(n)$ popolnoma prostorsko verna, lahko to naredimo tako, da simuliramo neki TS, ki porabi natanko $S_2(n)$ celic na vsakem vhodu dolžine n. V nadaljevanju predpostavljamo, da v kolikor M poskuša zapustiti zaznamovane celice, se M ustavi in odkloni w. Tako imamo zagotovilo, da ima M časovno omejitev $S_2(n)$.

Za tem prične M simulirati TS M_w pri vhodu w, kjer M_w predstavlja stroj z binarno kodo w. V primeru, ko ima M_w prostorsko omejitev $S_1(n)$ ter t tračnih simbolov, zahteva simulacija prostor $\lceil \log_2 t \rceil S_1(n)$. M sprejme w le, ko lahko opravi simulacijo v prostoru $S_2(n)$ in se M_w ustavi ter odkloni w.

Ker ima M prostorsko omejitev $S_2(n)$, pripada L(M) razredu $DSPACE(S_2(n))$. Trdimo, da L(M) ni v razredu $DSPACE(S_1(n))$. Denimo, da to ni res. Potem obstaja za L(M) TS $\bar{M}$ s prostorsko

omejitvijo $S_1(n)$ in t tračnimi simboli. Na podlagi leme 10.16 lahko predpostavljamo, da se $\bar{M}$ ustavi na vseh vhodih. Ker naše oštevilčenje vsebuje neskončno kod stroja $\bar{M}$ in ker velja

$$\inf_{n \to \infty} S_1(n)/S_2(n) = 0,$$

obstaja zadosti dolga beseda w pri $|w| = n$ z lastnostjo $\lceil \log_2 t \rceil S_1(n) < S_2(n)$ ter $M_w = \bar{M}$. Pri vhodu w ima M dovolj prostora, da simulira M_w ter sprejme w natanko, ko jo M_w odkloni. Torej velja $L(M_w) \neq L(M)$, kar je protislovje. Iz tega sklepamo, da L(M) pripada razredu $DSPACE(S_2(n))$ ne pa razredu $DSPACE(S_1(n))$. $\lceil\_\rceil$

Časovna hierarhija

Deterministična časovna hierarhija ni tako tesna kot je prostorska hierarhija. To pa je zaradi tega, ker TS, ki diagonalizira preko vseh večtračnih TS, ima sam neko določeno število trakov. Za simulacijo TS z večjim številom trakov uporabljamo dvotračni simulator, kar nam prinese logaritemsko upočasnitev. Predno pa opišemo konstrukcijo, pa bomo vpeljali pojem časovne vernosti.

Neka funkcija T(n) je časovno verna v primeru, ko obstaja neki večtračni TS M s časovno omejitvijo T(n) z lastnostjo, da za vsak n obstaja neki vhod dolžine n, pri katerem M dejansko uporabi T(n) korakov. Podobno kot v primeru prostorsko vernih funkcij obstaja bogata hierarhija časovno vernih funkcij. Za T(n) pravimo, da je popolnoma časovno verna v primeru, obstaja neki TS, ki pri vseh vhodih dolžine n porabi T(n) korakov. Tudi v tem primeru ugotovimo lahko, da je večina običajnih funkcij popolnoma časovno verna.

10.18. IZREK. Če je $T_2(n)$ popolnoma časovno verna funkcija in če velja

$$\inf_{n \to \infty} \frac{T_1(n)\log T_1(n)}{T_2(n)} = 0,$$

potem obstaja neki jezik, ki je v razredu $DTIME(T_2(n))$, ni pa v $DTIME(T_1(n))$.

Dokaz. Dokaz je podoben dokazu izreka 10.17, zato bomo podali le kratek oris konstrukcije. TS M s časovno omejitvijo $T_2(n)$ konstruiramo takole. M obravnava svoj vhod w kot neki opis stroja $\hat{M}$ ter simulira $\hat{M}$ na vhodu w. Tu naletimo na majhno težavo, ker ima $\hat{M}$ neko določeno, konstantno število trakov, in

bo torej pri nekaterih w $\hat{M}$ vseboval več trakov kot M. To sreče pa sta na podlagi izreka 10.13 potrebna le dva traku za simulacijo poljubnega $\hat{M}$ -- seveda za ceno faktorja $\log T_1(n)$.

Ker ima $\hat{M}$ lahko poljubno mnogo tračnih znakov, ki jih moramo zakodirati z omejenim številom znakov, bo simulacija $T_1(n)$ korakov $\hat{M}$ na stroju M zahtevala čas $cT_1(n)\log T_1(n)$, kjer je c neka konstanta, ki je odvisna od $\hat{M}$.

Časovno omejitev $T_2(n)$ pri $\hat{M}$ zagotovimo tako, da M istočasno izvaja izračun nekega TS, ki porabi natanko $T_2(n)$ korakov na vseh vhodih dolžine n (to je možno opraviti z uporabo dodatnih trakov). To je tudi razlog, zakaj mora biti $T_2(n)$ popolnoma časovno verna. Po $T_2(n)$ korakih se M ustavi. M sprejme w edino v primeru, ko se simulacija $\hat{M}$ konča in $\hat{M}$ odkloni w. Kodiranje $\hat{M}$ je definirano tako kot v prejšnem izreku, naradi česar ima vsak stroj poljubno dolge opise. Torej če ima $\hat{M}$ časovno omejitev $T_1(n)$, bo zanj obstajal zadosti dolg opis w z lastnostjo

$$c_1 T_1(|w|)\log T_1(|\$|) \leq T_2(|w|),$$

zaradi česar se bo simulacija lahko končala. V tem primeru je $w \in L(M)$ natanko ko $w \notin L(\hat{M})$. Torej velja $L(M) \neq L(\hat{M})$ za poljuben $\hat{M}$, ki ima časovno omejitev $T_1(n)$. Torej pripada $L(M)$ razredu $DTIME(T_2(n))$-$DTIME(T_1(n))$. $\square$

10.19. PRIMER. Naj bo $T_1(n) = 2^n$ in $T_2(n) = n\,2^n$. Tedaj velja

$$\inf_{n \to \infty} \frac{T_1(n)\log T_1(n)}{T_2(n)} = \inf_{n \to \infty} \frac{1}{n} = 0.$$

Torej je izrek 10.18 uporaben in velja $DTIME(2^n) \neq DTIME(n\,2^n)$. Ker pa tudi velja $T_1(n) \leq T_2(n)$ pri vseh n, lahko sklepamo, da velja $DTIME(2^n) \subseteq DTIME(n\,2^n)$.

10.4 Relacije med različnimi merami kompleksnosti

Med štirimi merami kompleksnosti, ki smo jih preučevali, obstaja za neki podan jezik L nekaj preprostih relacij in ena, ki ni čisto očitna. Omenjene preproste relacije opišemo z naslednjim izrekom.

10.20. IZREK. (a) Če je L v razredu $DTIME(f(n))$, potem je tudi v razredu $DSPACE(f(n))$. (b) Če je L v razredu $DSPACE(f(n))$ in če velja $f(n) \geq \log_2 n$, potem eksistira konstanta c, ki je odvisna od L, tako, da L pripada razredu $DTIME(c^{f(n)})$. (c) Če L pripada razredu $NTIME(f(n))$, potem eksistira konstanta c, ki je odvisna od L, tako, da L pripada razredu $DTIME(c^{f(n)})$.

Dokaz. (a) Če TS M ne opravi več kot f(n) korakov, potem tudi na nobenem traku ne more obiskati več kot f(n) + 1 celic. Če M spremenimo tako, da vsaka celica vsebuje dva simbola, smo zmanjšali potrebno število celic na $\lceil (f(n)+1)/2 \rceil$, kar znaša največ f(n).

(b) Najprej ugotovimo, da, če ima TS M_1 s stanj in t tračnih simbolov in če uporablja največ f(n) prostora, potem je število različnih TO stroja M_1 pri dolžini vhoda n največ $s(n+2)f(n)t^{f(n)}$. Ker je $f(n) \geq \log_2 n$, eksistira neka konstanta c tako da velja $n \geq 1$, $c^{f(n)} \geq s(n+2)f(n)t^{f(n)}$.

Sedaj konstruiramo na podlagi M_1 večtračni TS M_2, ki uporablja en trak za štetje do $c^{f(n)}$ in preostala dva trakova za simulacijo M_1. V primeru, ko M_1 ne sprejme svojega vhoda in je števec enak $c^{f(n)}$, se M_2 ustavi ne, da bi sprejel svojega vhoda. Namreč po tolikšnem številu korakov se je pri M_1 neki TO ponovil in se torej M_1 ne bo nikoli ustavil. Očitno ima torej stroj M_2 časovno omejitev $c^{f(n)}$.

(c) Naj ima M_1 časovno omejitev f(n) in naj ima s stanj, t tračnih simbolov in k trakov. Število različnih TO stroja M_1 pri dolžini vhoda n ne presega $s(f(n)+1)^k t^{kf(n)}$ (zmnožek števila stanj, števila položajev oken in števila tračnih vsebin). Torej količina $d = s(t+1)^{3k}$ zadošča pogoju

$$d^{f(n)} \geq s(f(n)+1)^k t^{kf(n)} \quad \text{pri vseh } n \geq 1.$$

Neki deterministični večtračni TS lahko ugotovi, ali M_1 sprejme vhod w dolžine n tako, da sestavi seznam vseh TO stroja M_1, ki so dosegljivi iz začetnega TO. Ta postopek se da opraviti v času, ki je omejeno s kvadratom dolžine seznama. Ker pa ima seznam dosegljivih TO dolžino, ki ne presega $d^{f(n)}$ krat dolžina nekega TO, katerega pa je možno zakodirati s 1 + k(f(n)

+ 1) simboli, je čas delovanja stroja omejen z $c^{f(n)}$, pri neki konstanti c. □

10.21. IZREK. (Savitchev izrek) Če L pripada razredu NSPACE(S(n)), potem pripada tudi razredu $DSPACE(S^2(n))$ pod pogojem, da je S(n) popolnoma prostorsko verna in da velja $S(n) \geq \log_2 n$.

Dokaz. Naj bo $L = L(M_1)$, ko je M_1 neki nedeterministični TS s prostorsko omejitvijo S(n). Pri dolžini vhoda n obstaja neka konstanta c tako, da je največ $c^{S(n)}$ različnih TO. Torej, če M_1 sprejme svoj vhod, je dolžina ustreznega izračuna kvečjemu $c^{S(n)}$, kajti noben TO se ne more ponoviti v najkrajšem izračunu stroja M_1, ki se konča s sprejetjem.

```
BEGIN
  naj bosta |w| = n in m = |⁻log c⁻| ;
                           2
  naj bo I  začetni TO stroja M  pri vhodu w ;
          0                    1
  FOR vsi končni TO I  dolžine največ S(n) DO
                    f
      IF TEST(I ,I ,mS(n)) THEN ACCEPT ;
               0  f
END ;
PROCEDURE TEST(I ,I ,i) ;
               1  2

  IF (i = 0) in (I  = I  ali I  |- I ) THEN
                  1    2      1     2
      RETURN TRUE;
  IF i ≥ 1 THEN
     FOR vsi TO I' dolžine največ S(n) DO
       IF TEST(I ,I',i-1)  and  TEST(I',I ,i-1) THEN
                1                            2
          RETURN TRUE ;
  RETURN FALSE
END TEST
```

Sl. 10.5 Algoritem, ki simulira M_1.

Naj $I_1 \vdash^{(i)} I_2$ pomeni, da TO I_2 dosegljiv od I_1 z zaporedjem največ 2^i korakov. Pri vseh $i \geq 1$ je možno ugotoviti, ali velja $I_1 \vdash^{(i)} I_2$ tako, da preverimo za vsak I', ali velja $I_1 \vdash^{(i-1)} I'$ in $I' \vdash^{(i-1)} I_2$. Torej je prostor za preverjanje, ali lahko pridemo iz enega TO do drugega v 2^i korakih, enak prostoru, ki je potreben za zapis TO I', ki ga

trenutno preizkušamo, plus prostor, ki je potreben za preverjanje, ali lahko pridemo od enega TO do drugega v 2^{i-1} korakih. Bodimo pozorni na to, da lahko za dva preverjanja, ali lahko pridemo od enega do drugega TO v 2^{i-1} korakih, uporabimo isti prostor.

Podrobnosti preizkusa, ali w pripada $L(M_1)$ so prikazane na sl. 10.5. Algoritem na sl. 10.5 je možno uresničiti na TS M_2, ki uporablja trak kot sklad za lokalne podatke procedure TEST. Vsakemu klicu ustreza en nabor lokalnih podatkov, ki vsebujejo I_1, I_2, i ter vmesni parameter I'. I_1, I_2 ter I' so TO, ki ne zahtevajo več kot S(n) celic. Dvojiški zapis položaja vhodnega okna zahteva $\log n \leq S(n)$ celic. Bodimo pozorni na to, da je vhodni trak pri vseh TO isti in istoveten z vhodnim trakom stroja M_2 tako, da ni potrebno prepisovati vhoda iz enega v drugi TO. Parameter i lahko zakodiramo v dvojiški obliki v mS(n) celicah. Torej zahteva en nabor lokalnih podatkov prostor O(S(n)).

Ker se tretji parameter zmanjšuje za 1 ob vsakem klicu procedure TEST, ker velja pri začetnem klicu i = mS(n) in, ker se klicanje neha, ko dobi i vrednost 0, je največje število naborov lokalnih podatkov O(S(n)). Torej je celoten uporabljeni prostor $O(S^2(n))$, na podlagi izreka 10.2 pa lahko M_2 spremenimo tako, da uporablja natančno prostor $S^2(n)$. |⁻|

10.22. PRIMER.

$NSPACE(\log n) \subseteq DSPACE(\log^2 n)$

$NSPACE(N^2) \subseteq DSPACE(n^4)$ in $NSPACE(2^n) \subseteq DSPACE(4^n)$.

Bodimo pozorni na to, da pri $S(n) \geq n$ Savitchev izrek velja tudi v primeru, ko je S(n) le prostorsko verna in ne popolnoma prostorsko verna. M_2 prične s simulacijo nekega TS M, ki računa S(n)pri vseh vhodih dolžine n in kot S(n) uporabi največji prostor, ki je bil uporabljen (torej ta prostor uporablja za odmero traku, namenjenega lokalnim podatkom. Potrebno je tudi poudariti dejstvo, da, če ne moremo izračunati S(n) v prostoru $S^2(n)$, potem ne moremo preveriti vseh vrednosti I_f ali I' ne, da bi naleteli na nekatere, ki zahtevajo preveč prostora.

10.5 Izrek o vrzelih

V tem razdelku obravnavamo neko na prvi pogled nenavadno lastnost prostorske mere kompleksnosti. Izkaže se, da ta lastnost in še mnoge druge, ki so na prvi pogled prav tako nenavadne, držijo tudi za druge mere kompleksnosti, in da pravzaprav izhajajo iz nekaterih splošnih lastnosti mer kompleksnosti, ki se dajo obravnavati aksiomatično. Vendar v to se v okviru tega dela ne bomo spuščali.

Izreka 10.17 in 10.18 razodevata, da sta prostorska in časovna hierarhija zelo gosti. Vendar v obeh primerih je prisoten pogoj, da mora biti funkcija verna. Ali je možno ta pogoj odpraviti? Odgovor je ne: deterministični prostorski in časovni hierarhiji vsebujejo poljubne vrzeli.

Za neko trditev, ki vsebuje parameter n, pravimo, da je resnična skoraj povsod (s.p.), kadar ne drži kvečjemu za končno mnogo vrednosti n. Kadar pa trditev drži pri neskončno vrednostih n, pravimo, da je resnična v neskončno primerih (v n.p.).

10.23. LEMA. Ko L sprejema neki TS M, ki ima prostorsko omejitev S(n) s.p., potem L sprejema tudi neki TS, ki ima prostorsko omejitev S(n).

Dokaz. Za končno mnogo vhodnih zaporedij, ki jih M ne sprejema s prostorsko omejitvijo S(n) (n je dolžina zaporedja), uporabimo nadzorno enoto. Poudariti pa je potrebno, da opisana konstrukcija ni efektivna, kajti glede na to, da nimamo na voljo neko časovno mejo, ne moremo vedeti, ali tako besedo M sprejme ali ne. $\square$

10.24. LEMA. Obstaja algoritem, ki pri podanih TS M, dolžini n vhodnega zaporedja in celemu številu m ugotavlja, ali je m največje število tračnih celic, ki jih M uporablja na nekem vhodu dolžine n.

Dokaz. Pri poljubnem m in n obstaja meja t za število korakov stroja M, ki jih lahko le-ta napravi, pri pogoju, da ne uporablja več kot m celic na vsakem traku in da ne ponavlja TO. Potemtakem je potrebno le simulirati vsa zaporedja največ t korakov, ki se začenjajo z vsemi možnimi vhodi dolžine n. |¯|

10.25. IZREK. (Borodinov izrek o vrzelih) Pri poljubni totalno rekurzivni funkciji $g(n) \geq n$ obstaja neka totalno rekurzivna funkcija S(n) z lastnostjo DSPACE(S(n)) = DSPACE(g(S(n))). Drugače povedano, med prostorsko mejo S(n) in g(S(n)) obstaja "vrzel", znotraj katere ne najdemo minimalne prostorske kompleksnosti nobenega jezika.

Dokaz. Naj bo $M_1, M_2, \ldots$ neko oštevilčenje TS. Naj bo $S_i(n)$ največje število tračnih celic, ki ga porabi M_i na nekem vhodu dolžine n. V primeru, ko se M_i vedno ustavi, je $S_i(n)$ totalna funkcija in predstavlja prostorsko kompleksnost M_i. Če pa se M_i pri nekem vhodu dolžine n ne ustavi, je $S_i(n)$ nedefinirano.[5] S(n) potem konstruiramo tako, da pri vsakem k bodisi

---

[5] Nedefinirano vrednost identificiramo z neskončnostjo, torej je večja od vsake definirane vrednosti.

(1) $S_k(n) \leq S(n)$ s.p. ali pa

(2) $S_k(n) \geq g(S(n))$ v n.p.

Z drugimi besedami nobena $S_k(n)$ ne leži med S(n) in g(S(n)) pri skoraj vseh n.

Pri konstrukciji S(n) pri neki podani vrednosti n se omejimo na končno množico TS $M_1, M_2, \ldots, M_n$. Vrednost S(n) izberemo tako, da pri vseh i med 1 in n $S_i(n)$ ne leži med S(n) in g(S(n)). Če bi lahko izračunali največjo končno vrednost $S_i(n)$ pri $1 \leq i \leq n$, potem bi lahko to vrednost priredili S(n). Vendar, ker je pri nekaterih n $S_i(n)$ nedefinirano, tega ne moremo storiti. Zato postopamo tako, da najprej postavimo j = 1 in preverimo, ali obstaja neki M_i v naši končni množici, za katerega $S_i(n)$ leži med j + 1 in g(j). Če tak i obstaja, potem naj j postane $S_i(n)$ in postopek ponovimo. Če takega i ni, postane S(n) enak j in končamo. Ker imamo opravka le s končnim številom TS in ker lahko na podlagi leme 10.24 preverimo, ali velja $S_i(n) = m$ pri nekem podanem m, bo opisani postopek na koncu našel neko tako vrednost j, da bo veljalo pri vseh $1 \leq i \leq n$ bodisi $S_i(n) \leq j$ ali pa $S_i(n) > g(j)$. To vrednost priredimo S(n).

Sedaj predpostavimo, da obstaja neki jezik L v razredu DSPACE(g(S(n)), ki ni v DSPACE(S(n)). Torej je L = L(M_k) pri nekem k, pri čemer je $S_k(n) \leq g(S(n))$ pri vseh n. Vendar smo S(n) tako konstruirali, da velja pri vseh $n \geq k$ $S_k(n) \leq S(n)$. Z drugimi besedami $S_k(n) \leq S(n)$ s.p. in je torej L na podlagi leme 10.23 v DSPACE(S(n)), kar je protislovje. Sklepamo torej, da velja DSPACE(S(n)) = DSPACE(g(S(n))). |_|

Izrek 10.25, kakor tudi ustrezni izreki za ostale mere kompleksnosti, imajo nekaj zelo nenavadnih posledic, kot na primer:

10.26. PRIMER. Eksistira totalna rekurzivna funkcija f(n) z naslednjo lastnostjo:

DTIME(f(n)) = NTIME(f(n)) = DSPACE(f(n)) = NSPACE(f(n)).

Očitno je DTIME(f(n)) vsebovano v NTIME(f(n)) ter DSPACE(f(n)). Podobno sta NTIME(f(n)) ter DSPACE(f(n)) vsebovana v NSPACE(f(n)). Na podlagi izreka 10.20 pri vseh $f(n) \geq \log_2 n$, če je L v NSPACE(f(n)), potem eksistira taka konstanta c, ki je odvisna le od L, da je L v $DTIME(c^{f(n)})$. Torej je L = L(M) za neki TS M, katerega časovna kompleksnost je navzgor omejena z $f(n)^{f(n)}$ s.p. Na podlagi leme, ki je podobna lemi 10.23, a se nanaša na DTIME, pripada L razredu $DTIME(f(n)^{f(n)})$. Končno, na

podlagi izreka, ki je podoben izreku 10.25 pri $g(x) = x^x$, a se nanaša na DTIME, obstaja neka f(n) z lastnostjo $DTIME(f(n)) = DTIME(f(n)^{f(n)})$, kar dokazuje zapisani rezultat.

Podobno lahko pokažemo, da, če imamo dva univerzalna razreda računalnikov, pri čemer je eden od njih zelo preprost in počasen (n.pr. Turingov stroj, ki opravi en korak na stoletje), drugi pa izredno hiter (n.pr. stroj z naslovljivim pomnilnikom, ki ima zelo močne vgrajene ukaze za množenje, eksponenciranje i.pd. in opravi miljon operacij na sekundo), potem eksistira taka totalna rekurzivna funkcija T(n), da je vsaka funkcija, ki je izračunljiva v času T(n) v enem modelu, tudi izračunljiva v času T(n) v drugem modelu.

10.6 Naloge

10.1. Pojem mejnih zaporedij -- zaporedij stanj, v katerih se opravijo prehodi med sosednimi celicami -- smo definirali v zvezi z dvosmernimi končnimi avtomati. Vendar ta pojem je prav tako uporaben pri enotračnih TS. Dokaži naslednje osnovne lastnosti mejnih zaporedij:

(a) Število korakov, ki ga porabi neki enotračni TS M pri vhodu w je vsota dolžin mejnih zaporedij med vsemi sosednimi celicami traku stroja M.
(b) Naj bo M enotračni TS, ki, če sprejme svoj vhod, potem ga sprejme tako, da je na koncu okno desno od celic, ki jih je uvodoma zasedal vhod. Dokaži, da v primeru, ko M sprejme vhod $w_1 w_2$, in je mejno zaporedje med w_1 in w_2 isto kot med x_1 in x_2 (pri vhodu $x_1 x_2$), potem M sprejme $x_1 w_2$.

10.2. Uporabi nalogo 10.1 za dokaz dejstva, da jezika

$$\text{(a) } \{wcw^R \mid w \in (a + b)^*\} \quad \text{(b) } \{wcw \mid w \in (a + b)^*\}$$

oba zahtevata kn^2 korakov pri nekem zadosti dolgem vhodu lihe dolžine, pri neki konstanti $k > 0$.

DODATEK A

SEZNAM DODATNE LITERATURE

[DM] M. Davis: COMPUTABILITY AND UNSOLVABILITY, McGraw-Hill, New York, 1958.

[JNI] N. Jacobson: BASIC ALGEBRA I, W. H. Freeman and Co., 1974.

[JNII] N. Jacobson: BASIC ALGEBRA II, W. H. Freeman and Co., 1980.

[MLS] S. MacLane, G. Birkhoff: ALGEBRA, Macmillan Publišing Co., Inc., 1979.

[PN] N. Prijatelj: MATEMATIČNE STRUKTURE I (MNOŽICE - RELACIJE - FUNKCIJE), Knjižnica Sigma, Mladinska knjiga, 1971.

[HPR] Paul R. Halmos: NAIVE SET THEORY, Springer 1960, 1974

[ME] Elliot Mendelson: INTRODUCTION TO MATHEMATICAL LOGIC, D. Van Nostrand 1964).

DODATEK B

SEZNAM NEKATERIH SIMBOLOV

alf	malo alfa
bta	malo beta
gma	malo gama
dlt	malo delta
lmd	malo lambda
u /	malo mi
sgm	malo sigma
GAMA	vel. gama
dlt	vel. delta
SIGMA	vel. sigma
|_|	unija
|‾|	presek
or	disjunkcija
and	konjunkcija
POT(X)	potenčna množica X

|‾	podmnožica
DOM(f)	domena
CODOM(f)	kodomena
∃|	eksistenčni kvantifikator
\∀	univerzalnostni kvantifikator
oo	neskončno
|_n_|	celi del n
|‾n‾|	"pokrov" od n (= |_n_| pri n celo število in |_n_| + 1 sicer).
|-	neposredno daje
|‾|	konec definicije ali dokaza.
NN	množica naravnih števil.
u o v	stik besed u in v
√n	kv. koren n