Formálne jazyky a automaty

poznámky z prednášok

verzia 0.98

Predhovor

Tento text sú prednášky z predmetu 'Formálne jazyky a automaty', ktoré prednášal doc. RNDr. Branislav Rovan, CSc. v šk.r. 1998/1999. Autormi textu sú študenti, preto sa s otázkami, návrhmi a pod. prosím obracajte na nich mailom na adresu foja@redbull.dcs.fmph.uniba.sk. *Aktuálnu verziu textu spolu s ďalšími informáciami je možné nájsť na domovskej stránke textu -* http://redbull.dcs.fmph.uniba.sk/foja.

Autori neručia za pravdivosť a úplnosť informácií v texte uvedených.

Obsah

Kapitola 1

Gramatiky

1.1 Základné pojmy

Najskôr definujeme niektoré pojmy, ktoré sú veľmi dôležité, pretože sa intenzívne využívajú v celom texte. Ich pochopenie je nevyhnutné pre ďalšie pokračovanie v čítaní.

Definícia 1.1.1. *Abeceda (lingvisticky slovník) je konečná neprázdna množina symbolov (písmen). Označuje sa* Σ.

Definícia 1.1.2. *Slovo (lingvisticky veta) v abecede* Σ *je konečná postupnosť symbolov z* Σ. *Prázdnu postupnosť (prázdne slovo) označujeme* ε.

Poznámka 1.1.1. Pri zápise slova vynechávame čiarky oddelujúce jednotlivé členy postupnosti (píšeme $u = a_1a_2 \dots a_n$, nie $u = a_1, a_2, \dots, a_n$).

Definícia 1.1.3. *Dĺžka slova je dĺžka postupnosti, ktorá ho vytvára. Dĺžku slova* w *značíme* $|w|$.

Definícia 1.1.4. *Podslovo slova* $v = a_1a_2..a_n$ *je súvislá podpostupnosť* $a_ia_{i+1}..a_j$, *pričom platí* $1 \leq i \leq j \leq n$. *Špeciálne podslová sú prefix a sufix . Pre prefix platí* $i = 1$ *a pre sufix* $j = n$.

Príklad 1.1.1. Nech je daná abeceda $\Sigma = \{a, b, c\}$. Potom slová v abecede Σ sú napr. aa,$abac$,ε.

Definícia 1.1.5. *Jazyk v abecede* Σ *je ľubovoľná množina slov v abecede* Σ. *Označujeme* L.

Poznámka 1.1.2. Symboly označujeme malými písmenami zo začiatku abecedy $(a, b, c...)$, slová písmenami z konca abecedy $(v, w, u...)$. Zápis Σ^* značí množinu všetkých slov v abecede Σ, $\#_a w$ označuje počet písmen a v slove w. Mohutnosť množiny M budeme v prípade jej konečnosti značiť aj $\#M$.

Príklad 1.1.2. $L_1 = \{a, b, aa, aba\}$, $L_2 = \{w \in \Sigma^* \mid w \text{ obsahuje párny počet písmen } a\}$. L_1 a L_2 definujú vlastne pravidlá, kedy je veta lingvisticky syntakticky správna. Napr. L_2 obsahuje všetky vety, v ktorých sa vyskytuje párny počet písmena a.

1.2 Operácie na slovách a jazykoch

Pretože jazyky sú množiny, sú na nich definované rovnaké operácie ako na množinách – zjednotenie, prienik, rozdiel a v istom zmysle aj komplement. Pri komplemente je problém s tým, čo je univerzálna množina. Zväčša za univerzum pokladáme jazyk Σ^*, kde Σ je abeceda, nad ktorou je uvažovaný jazyk, nie nejaká pevná abeceda. Uvedomme si, že mohutnosť množiny rôznych abecied je $\aleph_0$.

zreťazenie

$$u = a_1..a_n, v = b_1..b_n : \quad uv = a_1..a_n b_1..b_n \quad L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$$

iterácia (uzáver, Kleeneho $*$)

$$L^* = \bigcup_{i=0}^{\infty} L^i, \quad \text{kde} \quad L^0 = \{\varepsilon\}, \quad L^{i+1} = LL^i \quad \forall i \geq 0$$

kladná iterácia (uzáver, Kleeneho $+$)

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

ľavý kvocient jazyka L_1 podľa L_2

$$L_2 \backslash L_1 = \{u \mid \exists v \in L_2;\ vu \in L_1\}$$

pravý kvocient jazyka L_1 podľa L_2

$$L_1 / L_2 = \{u \mid \exists v \in L_2;\ uv \in L_1\}$$

zrkadlový obraz

$$u = a_1..a_n : \quad u^R = a_n..a_1 \quad L^R = \{u^R \mid u \in L\}$$

homomorfizmus

$$h : \Sigma_1^* \to \Sigma_2^* \quad h(uv) = h(u)h(v) \quad (\forall u, v \in \Sigma_1^*)$$

resp. pre jazyky

$$h(L) = \{h(u) \mid u \in L\}$$

inverzný homomorfizmus

$$h^{-1} : \Sigma_2^* \to 2^{\Sigma_1^*} \quad h^{-1}(v) = \{w \mid h(w) = v\}$$

resp. pre jazyky

$$h^{-1}(L) = \{w \in \Sigma_1^* \mid h(w) \in L\}$$

Poznámka 1.2.1. Ľavý kvocient má meno podľa toho, že zo slov jazyka L_1 odstraňuje ľavú časť, pravý kvocient odsekáva pravú časť.

Príklad 1.2.1. Nech $L_1 = \{fabcc, faabbc, gaabc, gabbcc\}$, $L_2 = \{f, g\}$. Potom $L_2 \backslash L_1 = \{abcc, aabbc, aabc, abbcc\}$.

Príklad 1.2.2. Nech $\Sigma_1 = \{a, b\}$, $\Sigma_2 = \{0, 1\}$, $h(a) = 00$, $h(b) = 010$ je homomorfizmus, potom $h^{-1}(01000) = \{ba\}$.

Príklad 1.2.3. Nech $\Sigma_1 = \{a, b, c\}$, $\Sigma_2 = \{0, 1\}$, $h(a) = 0$, $h(b) = 00$, $h(c) = 01$, potom $h^{-1}(0001) = \{bc, aac\}$.

1.3 Gramatiky Chomského hierarchie

Na jazyky sa nemusíme pozerať len ako na akýsi pokus modelovať prirodzené jazyky, ale neskôr uvidíme, že v podstate každý rozumne popísaný problém je vlastne jazykom. Gramatiky tvoria aparát, ktorý je veľmi podobný deklaratívnym programovacím jazykom, a ktorým možno tieto jazyky špecifikovať, čo znamená zároveň riešiť príslušné problémy.

Frázové gramatiky sú najvšeobecnejším modelom, ktorým možno popisovať jazyky. Neskôr uvidíme, že neexistuje žiadny silnejší výpočtový model ako je frázová gramatika.

Definícia 1.3.1. *Frázová gramatika je štvorica $G = (N, T, P, \sigma)$, kde N a T sú abecedy neterminálnych a terminálnych symbolov ($N \cap T = \emptyset$), $\sigma \in N$ je počiatočný neterminál a $P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ je konečná množina pravidiel.*

Poznámka 1.3.1. Fakt, že $(u, v) \in P$ zapisujeme $u \underset{P}{\to} v$, prípadne len $u \to v$ ak vieme o akú množinu pravidiel P ide. Pravidlá $u \to v_1, u \to v_2, \ldots, u \to v_n$ s rovnakou ľavou stranou zapisujeme skrátene $u \to v_1 | v_2 | \ldots | v_n$.

Definícia 1.3.2. *Krok odvodenia v gramatike G je binárna relácia $\underset{G}{\Longrightarrow}$ na $(N \cup T)^*$ definovaná takto: $x \underset{G}{\Longrightarrow} y$ práve vtedy, keď existujú slová w_1,w_2 a pravidlo $u \to v \in P$ také, že $x = w_1 u w_2$, $y = w_1 v w_2$.*

Definícia 1.3.3. *Jazyk generovaný gramatikou G je množina $L(G) = \{w \in T^* \mid \sigma \overset{*}{\underset{G}{\Longrightarrow}} w\}$, kde symbol $\overset{*}{\underset{G}{\Longrightarrow}}$ značí reflexívny a tranzitívny uzáver relácie $\underset{G}{\Longrightarrow}$.*

Poznámka 1.3.2. Ak bude jasné o akú gramatiku ide, budeme namiesto symbolu $\underset{G}{\Longrightarrow}$ písať $\Rightarrow$.

Definícia 1.3.4. *Odvodenie $\sigma \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$ je postupnosť krokov odvodení. Dĺžka odvodenia je počet takýchto krokov.*

Definícia 1.3.5. *Vetná forma je slovo z $(N \cup T)^*$, ktoré môžeme získať odvodením (odvodzovaním).*

Definícia 1.3.6. *Hovoríme, že gramatiky G_1, G_2 sú ekvivalentné, ak $L(G_1) = L(G_2)$.*

Príklad 1.3.1. Majme nasledovnú gramatiku: $G = (\{\sigma\}, \{a, b\}, \{\sigma \to a\sigma b, \sigma \to \varepsilon\}, \sigma)$. Ak začneme z počiatočného symbolu σ a postupne prepisujeme podľa zadaných pravidiel, dostaneme nasledovné odvodenie: $\sigma \Rightarrow a\sigma b \Rightarrow aa\sigma bb \Rightarrow aaa\sigma bbb \Rightarrow \ldots$. V každom kroku sme mohli pomocou pravidla $\sigma \to \varepsilon$ ukončiť prepisovanie, čiže je zrejmé, že $L(G) = \{a^n b^n \mid n \geq 0\}$. Tento predpoklad je potrebné dokázať:

$\subseteq$: Treba ukázať, že každé $w \in L(G)$ je tvaru $a^n b^n$. Použijeme matematickú indukciu (MI) vzhľadom na dĺžku odvodenia. Indukčný predpoklad (IP) je: každé slovo (vetná forma) odvodená na m krokov je tvaru $a^i b^i$ alebo $a^i \sigma b^i$.

1° $m = 0 \ldots$ triviálne platí

2° Predpokladajme, že indukčný predpoklad platí pre $m \leq k$. Dokážeme, že platí pre $m = k + 1$.

$$\sigma \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{k+1}$$

Podľa indukčného predpokladu je w_k tvaru $a^i b^i$ alebo $a^i \sigma b^i$, teda $w_k = a^i \sigma b^i$. Teda w_{k+1} je alebo tvaru $a^i b^i$, ak sa použilo pravidlo $\sigma \to \varepsilon$, alebo $a^i \sigma b^i$ ak sa použilo pravidlo $\sigma \to a\sigma b$.

$\supseteq$: Potrebujeme dokázať, že pre každé n vieme vyrobiť slovo tvaru $a^n b^n$. Použijeme indukciu vzhľadom na n. Indukčný predpoklad je: na k krokov vieme odvodit slovo tvaru $a^k b^k$ ($\sigma \Rightarrow \ldots \Rightarrow a^k b^k$).

1° $n = 0$ ... triviálne platí

2° $n = k + 1$. Vieme, že $\sigma \Rightarrow a\sigma b$, použijeme IP a σ nahradíme $a^k b^k$, a teda $aa^k b^k b = a^{n+1} b^{n+1}$.

Predtým, ako definujeme špeciálne triedy gramatík, spomeňme jednu dôležitú skutočnosť. Keďže gramatika je objekt, ktorý možno zapísať na konečný kus papiera (lebo všetky množiny tvoriace gramatiku, hlavne δ-funkcia, sú konečné), možno gramatikám jedno-jednoznačne priradiť prirodzené čísla, a teda mohutnosť množiny všetkých gramatík je $\aleph_0$. Naproti tomu, jazyk je potenciálne nekonečný objekt, a teda mohutnosť množiny všetkých jazykov nad danou abecedou je $\aleph_1$. To značí, že k veľkému množstvu jazykov neexistuje gramatika, ktorá ich generuje. Avšak, také jazyky sú veľmi zložité a je veľmi ťažké ich definovať. Neskôr však niekoľko takýchto jazykov nájdeme.

Pozrime sa teraz na rôzne obmedzenia, ktoré môžeme klásť na gramatiky.

Definícia 1.3.7. *Kontextová gramatika je taká frázová gramatika, v ktorej pre každé pravidlo $u \to v$ platí $|u| \leq |v|$.*

Poznámka 1.3.3. V pôvodnej Chomského definícii mali pravidlá tvar $u\xi v \to uwv$, kde $u, v \in (N \cup T)^*$, $\xi \in N$, $w \in (N \cup T)^+$.

Definícia 1.3.8. *Bezkontextová gramatika je taká frázová gramatika, v ktorej $P \subseteq N \times (N \cup T)^*$.*

Definícia 1.3.9. *Regulárna gramatika je taká frázová gramatika, v ktorej $P \subseteq N \times (T^* \cup T^* N)$.*

1.4 Triedy jazykov

Jazyky delíme na viaceré triedy podľa gramatík, ktoré ich generujú. V zátvorkách sú uvedené anglické ekvivalenty slovenských názvov.

rekurzívne vyčíslitelné (frázové) *(recursively enumerated)*
$\mathcal{L}_{RE} = \{L \mid$ existuje frázová gramatika G taká, že $L = L(G)\}$

kontextové *(context sensitive)*
$\mathcal{L}_{CS} = \{L \mid$ existuje kontextová gramatika G taká, že $L = L(G)\}$

rozšírené kontextové *(extended context sensitive)*
$\mathcal{L}_{ECS} = \{L \mid$ existuje rozšírená kontextová gramatika G taká, že $L = L(G)\}$

bezkontextové *(context free)*
$\mathcal{L}_{CF} = \{L \mid$ existuje bezkontextová gramatika G taká, že $L = L(G)\}$

regulárne *(regular)*
$\mathcal{R} = \{L \mid$ existuje regulárna gramatika G taká, že $L = L(G)\}$

1.5 Úlohy

1. Je operácia zreťazenia komutatívna? Je asociatívna?
2. Nájdite jazyky L_1, L_2 také, že L_1^* je nekonečný a L_2^* konečný jazyk.
3. Platí rovnosť $\{\varepsilon\} = \emptyset$?

Kapitola 2

Normálne (normované) tvary bezkontextových gramatík

Ako názov kapitoly napovedá, budeme sa v nej zaoberať výlučne bezkontextovými gramatikami, aj keď to nebude explicitne uvedené.

2.1 Motivácia

Ak máme nájsť gramatiku pre konkrétny jazyk, zväčša sa uspokojíme s nájdením jednej. Ku každému jednoduchému jazyku existuje celá trieda gramatík, ktoré ho generujú. Keď chceme pracovať s gramatikami, často požadujeme, aby boli akosi jednoduché. Táto kapitola pojednáva o tom, ako k nejakej gramatike nájsť k nej ekvivalentnú gramatiku, ale takú, aby mala isté pekné vlastnosti.

Jednou z pekných vlastností môže byť odstránenie pravidiel $\xi \to \xi$ z gramatiky. Tieto pravidlá určite neposunú výpočet nijako dopredu, ba ani iným smerom. Každým použitím takéhoto pravidla dostaneme v odvodení tú istú vetnú formu. Môžeme teda tvrdiť nasledovné.

Lema 2.1.1. *Nech G je bezkontextová gramatika a nech $G' = (N', T', P', \sigma')$ je gramatika, pre ktorú platí $N' = N, T' = T, \sigma' = \sigma$ a $P' = P - \{\xi \to \xi \mid \xi \in N\}$. Potom $L(G') = L(G)$.*

Dôkaz. Dokážeme dve inklúzie rovnosti $L(G') = L(G)$[1]

$\subseteq$:

Z predpokladu $w \in L(G')$, vyplýva, že $\exists$ odvodenie slova w v G' $\sigma' \underset{G'}{\Longrightarrow} u_1 \underset{G'}{\Longrightarrow} u_2 \underset{G'}{\Longrightarrow} \ldots \underset{G'}{\Longrightarrow} u_n = w$. Treba nájsť odvodenie slova w v G. Zrejme ak $u_i \underset{G'}{\Longrightarrow} u_{i+1}$, tak platí aj $u_i \underset{G}{\Longrightarrow} u_{i+1}$, pretože $P' \subseteq P$. A keďže platí $\sigma' = \sigma$, tak $\sigma \underset{G}{\Longrightarrow} u_1 \underset{G}{\Longrightarrow} u_2 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} u_n = w$, teda $w \in L(G)$.

$\supseteq$:

Keďže $w \in L(G)$, tak existuje odvodenie $\sigma \underset{G}{\Longrightarrow} v_1 \underset{G}{\Longrightarrow} v_2 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} v_m = w$. Treba nájsť odvodenie slova w v G'. Použijeme na to takýto trik. To, že $w \in G$,

[1] čo vo veľkej väčšine bude štandardný postup našich dôkazov

znamená, že existuje odvodenie tohto slova minimálnej dĺžky[2]. Vzhľadom na tento predpoklad sa v tomto odvodení slova w nepoužíva pravidlo $\xi \to \xi$, teda každé použité pravidlo je aj v P' a teda $\sigma = \sigma' \underset{G'}{\Longrightarrow} v_1 \underset{G'}{\Longrightarrow} \ldots \underset{G'}{\Longrightarrow} v_n = w$, čiže $w \in L(G')$.

□

2.2 Redukované gramatiky

Definícia 2.2.1. *Gramatika G je v redukovanom tvare, ak pre všetky neterminály ξ platí*

$$(\exists u, v \in (N \cup T)^* \ \sigma \overset{*}{\underset{G}{\Longrightarrow}} u\xi v) \wedge (\exists w \in T^* \ \xi \overset{*}{\underset{G}{\Longrightarrow}} w)$$

(: V preklade do ľudskej reči táto definícia hovorí to, že každý neterminál je dosiahnuteľný[3] a že z každého neterminálu vieme v tejto gramatike odvodiť nejaké terminálne slovo. :)

Príklad 2.2.1. Nech je daná gramatika $G = (N = \{\sigma, A, B\}, T = \{a, b\}, P, \sigma)$, kde

$P = \{\sigma \to \sigma A \mid \sigma b$
$\quad A \to aA$
$\quad B \to aB \mid bb\}$

Potom neterminál B aj so všetkými svojimi pravidlami je úplne zbytočný (lebo žiadna vetná forma odvoditeľná v G neobsahuje B) a môžeme ho z gramatiky odstrániť, pritom však dostaneme ekvivalentnú gramatiku. Nasledujúca lema hovorí, ako nájsť tie neterminály z gramatiky, ktoré sa nikdy nepoužijú.

Lema 2.2.1. *Nech G je bezkontextová gramatika, H je množina neterminálov, $H = \{\xi \in N \mid \exists u, v \in (N \cup T)^*, \sigma \overset{*}{\underset{G}{\Longrightarrow}} u\xi v\}$. Potom*

1. $$H = \bigcup_{i=0}^{\#N} H_i$$

   kde $H_0 =_{df} \{\sigma\}$, $H_{i+1} = H_i \cup \{\xi \in N \mid (\exists \eta \in H_i)\eta \to u\xi v \in P\}$

2. $$L(G) = L(G')$$

   kde $G' = (H, T, P', \sigma)$ a $P' = P \cap (H \times (H \cup T)^*)$.

Dôkaz. 1. Indukciou by sme poľahky dokázali, že $H = \bigcup_{i=0}^{\infty} H_i$. Musíme však ukázať, že nemusíme robiť zjednotenie donekonečna, ale stačí po celkový počet neterminálov. Uvedomme si, že akonáhle $H_i = H_{i+1}$, potom už pre všetky $j > i$ bude platiť $H_j = H_i$. To znamená, že mohutnosť množín H_i rastie. Ale rásť môže len po číslo $\#N$. Keďže nemôže existovať rastúca postupnosť čísel $1, 2, ..., n$ dĺžky väčšej ako n, je tvrdenie dokázané.

[2] každá zdola ohraničná podmnožina celých čísel má minimum; my si môžeme vytvoriť množinu $\heartsuit_w = \{n \mid \exists$ odvodenie slova w, pričom odvodenie má veľkosť $n\}$; tá je zrejme zdola ohraničená

[3] t.j. že existuje vetná forma, ktorá ho obsahuje a je odvoditeľná v tejto gramatike z počiat. neterminálu

2. Na dôkaz inklúzie $L(G') \subseteq L(G)$ si stačí uvedomiť, že každé pravidlo z P' je zároveň pravidlom z P a teda každý krok odvodenia v G' je krokom odvodenia v G. Indukciou k dĺžke odvodenia slova $w \in L(G')$ by sme poľahky túto inklúziu dokázali.

   Inklúzia $L(G) \subseteq L(G')$ je trochu ťažšia. Našťastie, opäť vieme overiť, že každé pravidlo z P použité pri odvodení slova $w \in L(G)$ je zároveň pravidlom z P'. Totiž, ak sme použili pravidlo $\xi \to X$ pri odvodení slova $w \in L(G)$, znamená to, že existujú slová w_i, w_{i+1}, u, v také, že $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w_i$, $w_i \underset{G}{\Longrightarrow} w_{i+1}$, $w_i = u\xi v$, $w_{i+1} = uXv$. Ale podľa definície množiny H potom $\xi \in H$, a tak $(\xi \to X) \in P'$.

□

Príklad 2.2.2. Nech $G = (N = \{\sigma, A, B\}, T = \{a, b\}, P, \sigma)$,

$P = \{\sigma \to aA \mid \varepsilon$
$A \to aB|b\sigma$
$B \to bB\}$.

Ľahko sa možno presvedčiť, že žiadne terminálne slovo sa nedá odvodiť z neterminálu B. To značí, že tento symbol môžeme z gramatiky odstrániť spolu so všetkými pravidlami, ktoré majú naľavo alebo napravo slovo, ktorého je súčasťou. Nasledovná lema hovorí o tom, ako nájsť všetky takéto zbytočné neterminály.

Lema 2.2.2. *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech $w \overset{*}{\underset{G}{\Longrightarrow}} t \in T^*$. Potom $w \in (H \cup T)^*$, kde H je množina definovaná analogicky ako v prechádzajucej leme.*[4]

Dôkaz. Tvrdenie dokážeme indukciou vzhľadom na počet krokov odvodenia terminálneho slova t z vetnej formy w.
Bázu indukcie by mohlo tvoriť číslo 0 ako počet krokov odvodenia. Triviálne totiž platí $t \overset{*}{\underset{G}{\Longrightarrow}} t$ a $t \in T^* \subseteq (H \cup T)^*$. Nech $w \underset{G}{\Longrightarrow} w_1 \overset{*}{\underset{G}{\Longrightarrow}} t$. Podľa indukčného predpokladu $w_1 \in (H \cup T)^*$. Z definície relácie $\underset{G}{\Longrightarrow}$ vyplýva, že existujú vetné formy u, v, X, neterminál ξ a pravidlo $(\xi \to X) \in P$ také, že $w = u\xi v$, $w_1 = uXv$. Keďže platí $w_1 = uXv \in (H \cup T)^*$, patrí aj vetná forma X množine $(H \cup T)^*$. Z definície množiny H plynie, že potom aj $\xi \in H$, a teda z doterajších predpokladov $u, v \in (H \cup T)^*$, $\xi \in H$ vyplýva dokazované tvrdenie $w = u\xi v \in (H \cup T)^*$. □

Lema 2.2.3. *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech*

$$H = \{\xi \in N \mid (\exists w \in T^*) \xi \overset{*}{\underset{G}{\Longrightarrow}} w\}$$

Potom platí

1. *$H = \bigcup_{i=0}^{\#N} H_i$, kde $H_0 = \{\xi \in N \mid (\exists t \in T^*)\ \xi \to t \in P\}$, $H_{i+1} = H_i \cup \{\xi \in N \mid \xi \to t \in P, t \in (T \cup H_i)^*\}$*
2. *$L(G) = L(G')$, kde $G' = (H, T, P \cap H \times (H \cup T)^*, \sigma)$.*

Dôkaz. 1. Dokazuje sa úplne analogicky ako prvá časť 2.2.1.

2. Dokazuje sa analogicky ako v 2.2.1, len si treba uvedomiť, že ak sme pri odvodení $w \in L(G)$ použili pravidlo $\xi \to X$, potom existujú vetné formy w_1, w_2, u, v také, že $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w_1 \to_G$

[4] Tu bude $H = \{\xi \in N \mid (\exists w \in T^*)\ \xi \overset{*}{\underset{G}{\Longrightarrow}} w\ \}$

$w_2 \overset{*}{\underset{G}{\Longrightarrow}} w \in T^*$, $w_1 = u\xi v$, $w_2 = uXv$. Potom ale $\xi \in H$ (podľa lemy 2.2.2), a tak pravidlo $(\xi \to X) \in P'$.

□

Veta 2.2.1. *K ľubovoľnej bezkontextovej gramatike $G = (N, T, P, \sigma)$ existuje redukovaná gramatika G' s ňou ekvivalentná.*

Dôkaz. Ak ku gramatike G nájdeme s ňou ekvivalentnú gramatiku obsahujúce len také neterminály, z ktorých vieme odvodiť terminálne slovo (podľa lemy 2.2.3) a potom túto gramatiku upravíme na G_2 tak, aby neobsahovala nedosiahnuteľné neterminály (podľa lemy 2.2.1), sme hotoví.
Dokážeme, že táto gramatika už neobsahuje ani neterminály, z ktorých sa nedá odvodiť terminálne slovo. Totiž, z každého neterminálu $\xi \in N_2$ sa dá v G_1 odvodiť nejaké terminálne slovo w odvodením $\sigma \Rightarrow w_1 \Rightarrow ... \Rightarrow w_n = w$. Ale každé takéto odvodenie v G_1 je aj odvodením v G_2, lebo slová w_i obsahujú len dosiahnuteľné neterminály z G_1, ktoré tým pádom patria gramatike G_2.

(: Keby sme poradie krokov vymenili, nemuseli by sme dospieť k cieľu :) □

2.3 Stromy odvodenia

Táto kapitolka bola odprednášaná neformálne, ľudsky. Nasledujúce definície vznikli v hlavách autorov tohto textu ako akési riešenie úlohy formalizovať pojem stromu odvodenia, ktorú zadal prednášajuci.

Definícia 2.3.1. *Exstrom je dvojica strom, funkcia f, kde f priraďuje každej orientovanej hrane stromu nejaké prirodzené číslo a pre každý vrchol v a množinu všetkých jeho priamych nasledovníkov $\{v_1, ..., v_n\}$ platí $\{f(vv_i) \mid i = 1, 2, ..., n\} = \{1, 2, ..., n\}$.*

Definícia 2.3.2. *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, a nech $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w$. Stromom odvodenia slova w je dvojica exstrom (S, f), funkcia $F : V(S) \to N \cup T$, pre ktorú platí*

1. *Koreňu stromu S je priradená σ $(F(v_0) = \sigma)$*
2. *Funkcia F priraďuje listom výlučne terminály a vnútorným vrcholom neterminály*
3. *Ak množina všetkých nasledovníkov vrchola v je $\{v_1, v_2, ..., v_m\}$ tak platí implikácia*

$$(\forall i) f(vv_i) = i \implies (\ F(v) \to F(v_1)F(v_2)...F(v_m)) \in P$$

4. *Ak zavedieme funkciu g z množiny vrcholov do množiny postupností prir. čísel tak, že koreňu priradíme prázdnu postupnosť a každému ďalšiemu vrcholu v postupnosť priradenú jeho otcovi u predĺženú o $f(uv)$, potom ak množina všetkých listov je $\{l_1, l_2, ..., l_k\}$ a platí $g(l_i) <_* g(l_j)$ pre všetky $i < j$, kde $<_*$ je relácia lexikografického usporiadania, potom*

$$F(l_1)F(l_2)...F(l_k) = w$$

Poznámka 2.3.1. Jednotlivé body okrem posledného v definícii hovoria, že dvojica (T, f) je stromom odvodenia a posledný, že tento strom odvodenia je stromom odvodenia slova w.

Poznámka 2.3.2. Pojem z definície 2.3.1 sa zrejme v žiadnej literatúre nenachádza. Exstrom je vlastne strom, v ktorom platí, že deti každého otca sú *usporiadané*.

Ako si pozorný čitateľ už iste uvedomil, strom odvodenia nedefinuje gramatiku. Strom odvodenia *konkrétneho slova* v *konkrétnej gramatike* určuje *konkrétne odvodenie* tohto slova v tejto gramatike.
V [1] je cvičenie, ktoré požaduje, aby riešiteľ zistil, či isté slovo w patrí do jazyka generovaného istou gramatikou G, pričom je uvedený strom odvodenia iného slova v G. To však neznamená, že strom odvodenia definuje gramatiku. Znamená to len to, že strom odvodenia definuje istú podmnožinu gramatiky. Ak zistíme, že w patrí jazyku generovaného touto podgramatikou, potom patrí aj jazyku generovaného celou gramatikou. Ak nie, potom odpoveďou na toto cvičenie je : *Nevieme.*

Definícia 2.3.3. *Hovoríme, že bezkontextová gramatika G je viacznačná, ak v $L(G)$ existuje slovo s aspoň dvoma rôznymi stromami odvodenia. Inak hovoríme, že G je jednoznačná.*

Definícia 2.3.4. *Hovoríme, že bezkontextový jazyk L je jednoznačný, ak existuje jednoznačná bezkontextová gramatika G, taká, že $L = L(G)$. Inak* [5] *hovoríme, že L je vnútorne viacznačný.*

Príklad 2.3.1. Príkladom vnútorne viacznačného jazyka je jazyk
$L = \{a^n b^n c^k \parallel n, k \geq 1\} \cup \{a^k b^n c^n \parallel n, k \geq 1\}$.
(:Idea dôkazu, že tento jazyk je naozaj vnútorne viacznačný: Každá z tých dvoch polovičiek jazyka si vyžaduje rovnaký spôsob odvodzovania. Ale slová $a^n b^n c^n$ sa potom dajú odvodiť dvoma spôsobmi. :)

Definícia 2.3.5. *Hovoríme, že odvodenie $w_0 = \sigma \Rightarrow w_1 \Rightarrow ... \Rightarrow w_n = w$ slova w je ľavým krajným (pravým krajným) odvodením, ak platí*

$$(\forall i)(\exists u_1 \in T^*, \xi \in N, u_2 \in (N \cup T)^*, (\xi \to \pi) \in P)\ w_i = u_1 \xi u_2 \ \wedge \ w_{i+1} = u_1 \pi u_2$$

$$(\ (\forall i)(\exists u_1 \in (N \cup T)^*, \xi \in N, u_2 \in T^*, (\xi \to \pi) \in P)\ w_i = u_1 \xi u_2 \wedge w_{i+1} = u_1 \pi u_2)$$

(: Ľavým krajným, resp. pravým krajným odvodením teda rozumieme také odvodenie, v ktorom v každom kroku odvodenia nahradzáme ten neterminál, ktorý je najľavejší (najpravejší) zo všetkých neterminálov danej vetnej formy. :)

Lema 2.3.1. *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech $w \in L(G)$. Potom existuje ľavé krajné aj pravé krajné odvodenie slova w v G.*

2.4 Chomského normálny tvar

Definícia 2.4.1. *Hovoríme, že gramatika $G = (N, T, P, \sigma)$ je v Chomského normálnom tvare, ak $P \subseteq N \times (T \cup \{\varepsilon\} \cup NN)$.*

Veta 2.4.1. *Ku každej bezkontextovej gramatike G existuje gramatika G' s ňou ekvivalentná, ktorá je v Chomského normálnom tvare.*

Dôkaz. *(: Najprv ukážeme, ako takúto gramatiku G' vytvoríme, a potom dokážeme, že je naozaj s ňou ekvivalentná. :)*
Nech $N = \{A_1, A_2, ..., A_m\}$, $T = \{a_1, a_2, ..., a_n\}$. Potom definujeme množinu nových neterminálov $N_1 = \{\xi_i \mid i = 1, 2, .., n\} \cup \{\xi_\varepsilon\}$ a množinu nových pravidiel pre tieto neterminály $P' = \{\xi_i \to a_i \mid i = 1, 2, ..., n\} \cup \{\xi_\varepsilon \to \varepsilon\}$. Zaveďme homomorfizmus $h : (N \cup T) \to (N \cup N_1)$ takto: $h(x) = x$ pre $x \in N$ a $h(a_i) = \xi_i$ pre $a_i \in T$. Pristúpme k definícii gramatiky G_1 ekvivalentnej s G, ktorá už je bližšie k Chomského normálnemu tvaru ako pôvodná G. Nech

[5] t.j. ak každá gramatika generujúca L je viacznačná

$G_1 = (N \cup N_1, T, P_1 \cup P', \sigma)$, kde $P_1 = \{\xi \to h(\alpha) \mid \xi \to \alpha \in P\}$. Gramatika G_1 nie je v Chomského normálnom tvare už len z toho dôvodu, že pravé strany pravidiel sú dlhé, prípadne krátke. To napravíme nasledovnou konštrukciou: Nech d_π je dĺžka pravej strany pravidla π[6].

$G_2 = (N_2, T, P_2, \sigma)$, kde
$N_2 = N_1 \cup \{\zeta_{\pi i} \mid \pi_i \in P_1 \wedge d_\pi > 2,\ i = 1, 2, ..., d_\pi - 2\}$ a
$P_2 = \{\pi \mid \pi \in P_1 \wedge (d_\pi = 2 \vee \pi \in N \times \{\varepsilon\}\} \cup$
$\{\alpha \to \beta\xi_\varepsilon \mid (\alpha \to \beta) \in P_1 \wedge |\beta| = 1\} \cup$
$\{\alpha \to \beta_1\zeta_{\pi 1} \mid \pi = \alpha \to \beta_1\beta_2...\beta_n \wedge d_\pi = n > 2\} \cup$
$\{\zeta_{\pi i} \to \beta_{i+1}\zeta_{\pi i+1} \mid \pi = \alpha \to \beta_1\beta_2...\beta_n \wedge d_\pi = n > 2 \ \wedge \ i = 1, 2, ..., n-3\} \cup$
$\{\zeta_{\pi d(\pi)-2} \to \beta_{n-1}\beta_n \mid \pi = \alpha \to \beta_1...\beta_n \in N_1\} \cup P'$

Zostáva dokázať, že G_2 je v Chomského normálnom tvare a že $L(G) = L(G_2)$.

1. Poľahky nahliadneme, že G_2 je v Chomského normálnom tvare, ak si všimneme, aké pravidlá sme do P_2 dávali.

2. Ponechávame ako cvičenie čitateľovi. [7]. Podotýkame len, že pri dôkaze inklúzie $L(G_2) \subseteq L(G)$ sa nám bude ľahko postupovať, ak sa oprieme o lemu 2.3.1.

□

2.5 Greibachovej normálny tvar

Definícia 2.5.1. *Hovoríme, že bezkontextová gramatika $G = (N, T, P, \sigma)$ je v Greibachovej normálnom tvare, ak platí $P \subseteq N \times T(N \cup T)^*$.*

Poznámka 2.5.1. V knihe [1] je trochu iná definícia Greibachovej normálneho tvaru. My budeme používať túto, trochu zjednodušenú.

Definícia 2.5.2. *Ak A je neterminálnym sybolom gramatiky $G = (N, T, P, \sigma)$, potom A-pravidlom nazveme každé pravidlo $A \to \xi \in P$.*

Lema 2.5.1. *Nech $G = (N, T, P, \sigma)$, $\pi = A \to B\xi \in P$. Nech $B \to \xi_i (i = 1, 2, ..., m)$ sú všetky B-pravidlá. Potom $G' = (N, T, P', \sigma)$, kde $P' = P - \{\pi\} \cup \{A \to \xi_i\xi \mid i = 1, 2, ..., m\}$ je ekvivalentná gramatike G.*

Lema 2.5.2. *Nech $G = (N, T, P, \sigma)$ je gramatika, $A \in N$, $Z \notin N$. Nech $P_A = \{A \to A\xi_i \mid i = 1, 2, ..., m\}$ je množina všetkých A-pravidiel, ktoré majú na začiatku pravej strany neterminál A. Ďalej nech $A \to \beta_i$, $i = 1, ..., n$ sú všetky zvyšné A-pravidlá. Nech gramatika $G' = (N \cup \{Z\}, T, P', \sigma)$, ktorej množina pravidiel $P' = P - P_A \cup \{A \to \beta_i Z \mid i = 1, 2, ..., n\} \cup \{Z \to \xi_i, Z \to \xi_i Z \mid i = 1, 2, ..., m\}$. Potom $L(G) = L(G')$*

Veta 2.5.1. *Ku každej bezkontextovej gramatike G existuje ekvivalentná gramatika G' v Greibachovej normálnom tvare.*

Dôkaz. Nech $G = (N = \{A_1, A_2, ..., A_n\}, T, P, \sigma)$[8].

V prvom kroku nájdeme gramatiku $G_1 = (N_1, T, P_1, \sigma)$ ekvivalentnú s G takú, že $A_i \to A_j\xi \in P_1 \Longrightarrow i < j$. Budeme postupovať indukciou. Predpokladajme, že pre $i = 1, 2, ..., k$

[6] Formálne: nech $d_{\alpha\to\beta} = |\beta|$
[7] Trvajúce dlhé hodiny počas dlhých zimných večerov
[8] Treba si vždy zvoliť čo najvhodnejšie úplné usporiadanie množiny neterminálov

už platí $A_i \to A_j\xi$ je pravidlom[9], potom $i < j$. Upravme teraz A_{k+1}-pravidlá nasledovne: Podľa lemy 2.5.1 a indukčného predpokladu sa nám každé A_{k+1}-pravidlo podarí na najviac k krokov previesť na skupinu A_{k+1} pravidiel, ktorých pravá strana začína terminálom alebo neterminálom s indexom väčším alebo rovným $k+1$. Potom podľa lemy 2.5.2 zavedením nového neterminálu dokážeme odstrániť pravidlá s pravou stranou začínajucou symbolom A_{k+1}. Uvedomme si, že pravidlá pre tieto nové neterminály začínajú nejakým terminálom alebo neterminálom z pôvodnej množiny N. Po úprave všetkých pravidiel z P potom ďalším použitím lemy 2.5.1 budú aj pravidlá pre nové neterminály v požadovanom tvare. □

2.6 Bezepsilonové gramatiky

Definícia 2.6.1. *Bezkontexotvá gramatika G je v bezepsilonovom tvare (is ε-free), ak* $P \subseteq N \times (N \cup T)^+$

Poznámka 2.6.1. Pravidlu $\xi \to \varepsilon$ hovoríme tiež epsilonové.

V nasledujúcom odseku sa dozvieme, že trieda bezepsilonových CF-gramatíky je akosi takmer ekvivalentná s triedou všetkých CF-gramatík, čiže ak zakážeme používanie epsilonových pravidiel, v princípe sme toho veľa nezakázali.

Veta 2.6.1. *K ľubovoľnej bezkontextovej gramatike G existuje bezepsilonová bezkontextová gramatika G' taká, že $L(G) - \{\varepsilon\} = L(G')$.*

Dôkaz. Nech $H = \{\xi \in N \mid \xi \overset{*}{\underset{G}{\Longrightarrow}} \varepsilon\}$. Nech $G' = (N, T, P', \sigma)$, kde
$P' = \{\alpha \to \beta_1\beta_2...\beta_m \mid (\beta_i \in (N \cup T)^*) \wedge$
$(\exists\zeta_1, \zeta_2, ..., \zeta_{m-1} \in H)(\alpha \to \beta_1\zeta_1\beta_2\zeta_2...\beta_{m-1}\zeta_{m-1}\beta_m) \in P\} \cap N \times (N \cup T)^+$.
Gramatika G' je iste v bezepsilonovom tvare. Dôkaz rovnosti $L(G) - \{\varepsilon\} = L(G')$ pre nedostatok miesta neuvádzame. □

Príklad 2.6.1. Daná je gramatika $G = (N, T, P, \sigma)$, $N = \{\sigma, A, B, C, D, E\}$, $T = \{a, b, c\}$,
$P = \{\sigma \to\ CDE \mid aAb$,
$A \to\ aB \mid c$,
$B \to\ cEA \mid b$,
$C \to\ EE \mid BcB$,
$D \to\ \varepsilon \mid d$,
$E \to\ D \mid B\ \}$.
Štandarným spôsobom zostrojte gramatiku G' takú, že $L(G) - \{\varepsilon\} = L(G')$ a G' neobsahuje epsilonové pravdilá.

Riešenie: $H_0 = \{D\}$, $H_1 = \{D, E\}$, $H_2 = \{D, E, C\}$, $H_3 = \{D, E, C, \sigma\}$, $H_4 = \{D, E, C, \sigma\} = H_3 = H$.
$P' = \{\sigma \to\ CDE|CD|CE|DE|C|D|E \mid aAb$,
$A \to\ aB \mid c$,
$B \to\ cEA|cA|\ b$,
$C \to\ EE|E|\ BcB$,
$D \to\ d$,
$E \to\ D \mid B\}$

[9] Formuláciu *je pravidlom* volíme preto, lebo formálne zapísať, do ktorej množiny pravidiel ktorej gramatiky patrí by bolo trestom aj pre silnejšie povahy

Poznámka 2.6.2. Z faktu, že $\sigma \in H$ vyplýva, že $\varepsilon \in L(G)$ a keďže prázdne slovo nepatrí do žiadneho jazyka generovaného bezepsilonovou gramatikou, neplatí tu $L(G) = L(G')$, ale len $L(G) - \{\varepsilon\} = L(G')$.

2.7 Reťazcové pravidlá (Chain Rules)

Definícia 2.7.1. *Reťazcovým pravidlom nazývame každé pravidlo $\pi \in N \times N$. Takéto pravidlá tiež nazývame chain rules.*

(:Ide teda o pravidlá typu $\xi \to \theta$, kde ξ, θ sú neterminály. :)

Lema 2.7.1. *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech $\xi \to \eta \in P$, $\eta \in N$. Označme $N_\eta = \{w \in (N \cup T)^* \mid (\eta \to w) \in P\}$. Potom $L(G) = L(G')$, kde $G' = (N, T, P', \sigma)$, $P' = P - \{\xi \to \eta \mid \xi, \eta \in N\} \cup \{\xi \to w \mid w \in N_\eta\}$.*

Dôkaz. Dôkaz sa vo veľkej miere opiera o lemu 2.5.1. □

Nasledovná veta hovorí o tom, ako k danej bezkontextovej gramatike nájsť ekvivalentnú bezkontextovú gramatiku bez chain rules. Lema 2.7.1 obsahuje len základnú myšlienku. Keby sme však odstraňovali chain rules z gramatiky v ľubovoľnom poradí bez rozmyslu, mohli by sme sa zacykliť. Pre nedostatok miesta uvádzame aj nasledovnú vetu bez formálneho dôkazu. Namiesto neho uvádzame algoritmus, podľa ktorého ekvivalentnú gramatiku bez chain rules zostrojíme.

Veta 2.7.1. *K ľubovoľnej bezkontextovej gramatike G existuje ekvivalentná bezkontextová gramatika G, ktorá neobsahuje chain rules.*

Algoritmus zostrojenia ekvivalentnej gramatiky bez chain rules.

Nech $G = (N, T, P, \sigma)$, $N = \{N_1, N_2, ..., N_n\}$.

1. Pre $i = n, n-1, ..., 1$ rob nasledovné: Aplikovaním lemy 2.7.1 na N_i-pravidlá zostroj ekvivalentnú gramatiku G_1, pre ktorú platí, že ak reťazcové pravidlo $(N_k \to N_l) \in P_1$, tak $k > l$.

2. Pre $i = 1, 2, ..., n$ rob nasledovné: Aplikovaním lemy 2.7.1 na N_i-pravidlá zostroj ekvivalentnú gramatiku G_2, ktorá neobsahuje chain rules.

Z lemy 2.7.1 vyplýva, že po každom kroku tohto algoritmu dostaneme gramatiku ekvivalentnú s pôvodnou. Jediné, čo treba overiť, je to, že tento proces je konečný. Dôkaz konečnosti by sa dal uskutočniť dôkazom nasledujúcich tvrdení, ktoré sú jednoduchšie ako celé naše tvrdenie, sú však všeobecnejšie, a teda sa ľahšie dokážu matematickou indukciou. Tvrdenia, ktoré hovoria veľa, sa matematickou indukciou dokazujú síce dlho, lebo treba veľa dokázať, no však ľahšie, **lebo máme silný indukčný predpoklad a teda v indukčnom kroku sa môžme oprieť o celý rad predpokladov.** Nasledujú tvrdenia, z ktorých vyplýva konečnosť algoritmu:

1. Nahradením reťazcového pravidla $N_j \to N_k$, kde $j < k$ v j-tom kroku prvej fázy nepribudlo žiadne také reťazcové pravidlo $N_l \to N_m$, že $l < m$.

2. Po i-tom kroku prvej fázy neexistuje $j \geq (n - i + 1)$, $k \geq j$, také, že $N_j \to N_k$ je pravidlom.

3. Nahradením reťazcového pravidla $N_j \to N_k$ $(j > k)$ v i-tom kroku druhej fázy nepribudlo žiadne reťazcové pravidlo.

4. Po i-tom kroku druhej fázy neexistuje žiadne reťazcové N_j-pravidlo pre $j \leq i$.

5. Každá klesajúca postupnosť prirodzených čísel je konečná.

Indukciou by sme poľahky dokázali konjunkciu prvého a druhého tvrdenia, resp. konjunkciu tretieho a štvrtého tvrdenia.

2.8 Úlohy

1. Nájdite bezkontextovú gramatiku G, v ktorej bude existovať neterminál, ktorý nemôže byť súčasťou vetnej formy, ktorú v G vieme odvodiť na menej ako $n-1$ krokov.

2. Dokážte vetu o Chomského normálnom tvare.

3. Zostrojte gramatiku v Chomského normálnom tvare a v Greibachovej normálnom tvare pre jazyk $L = \{a^i b^i a^j b^{2j}\}$.

Kapitola 3

Konečné automaty

3.1 Deterministický konečný automat

Najskôr si zavedieme najjednoduchší druh automatu pomocou štyroch nasledujúcich definícií. Podľa tejto schémy budú definované všetky druhy automatov, čiže tieto definície majú zásadný význam.

Definícia 3.1.1. *Deterministický konečný automat A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je vstupná abeceda, $q_0 \in K$ je počiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times \Sigma \to K$ je prechodová funkcia.*

Definícia 3.1.2. *Konfigurácia deterministického konečného automatu je prvok $(q, w) \in K \times \Sigma^*$, kde q je stav automatu a w je zvyšok vstupného slova.*

Definícia 3.1.3. *Krok výpočtu deterministického konečného automatu A je relácia $\underset{A}{\vdash}$ na konfiguráciách definovaná $(q, av) \underset{A}{\vdash} (p, v) \Leftrightarrow p = \delta(q, a)$*

Definícia 3.1.4. *Jazyk akceptovaný deterministickým konečným automatom A je množina $L(A) = \{w \in \Sigma^* \mid (q_0, w) \overset{*}{\underset{A}{\vdash}} (q, \varepsilon),\ q \in F\}$*

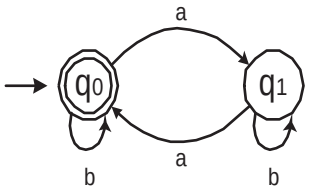
Poznámka 3.1.1. Deterministický konečný automat sa označuje skrátene ako DKA, prípadne z anglického jazyka DFA (*deterministic finite automaton*).

Poznámka 3.1.2. δ funkciu zvykneme zapisovať aj prechodovou tabuľkou (tabuľka 3.1) alebo prechodovým diagramom (obrázok 3.1). Prechodová tabuľka by mala byť intuitívne jasná, zastavme sa pri prechodovom diagrame. Jednotlivé stavy automatu sa kreslia kružnicou, do ktorej sa píšu názvy stavov. Ak sa jedná o koncový stav, kreslíme ho dvojitou kružnicou. Na počiatočný stav v diagrame ukazuje šípka. Ak v automate platí $\delta(q, a) = p$, potom stavy q a p v diagrame spojíme orientovanou hranou smerujúcou k vrcholu p ohodnotenou symbolom a.

Príklad 3.1.1. $A = (\{q_o, q_1\}, \{a, b\}, \delta, q_0, \{q_0\})$
$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_0 \quad \delta(q_1, a) = q_0 \quad \delta(q_1, b) = q_1$
Tento automat zisťuje, či má vstupné slovo párny počet písmen a, čiže
$L(A) = \{w \in \{a, b\}^* \mid \#_a w \, je \, párny\}$. Pre vstupné slovo $abaaba$ bude vyzerať výpočet takto: $(q_0, abaaba) \vdash (q_1, baaba) \vdash (q_1, aaba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_1, a) \vdash (q_0, \varepsilon)$

δ	a	b
q_0	q_1	q_0
q_1	q_0	q_1

Tabuľka 3.1: prechodová tabuľka



Obrázok 3.1: prechodový diagram

3.2 Nedeterministický konečný automat

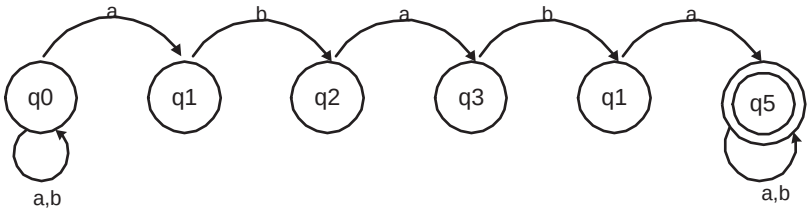
Definícia *nedeterministického konečného automatu* pozostáva z rovnakých štyroch častí ako pri DKA, pričom sa mení len definícia automatu a kroku výpočtu.

Definícia 3.2.1. *Nedeterministický konečný automat A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je vstupná abeceda, $q_0 \in K$ je počiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times (\Sigma \cup \{\varepsilon\}) \to 2^K$ je prechodová funkcia.*

Definícia 3.2.2. *Krok výpočtu nedeterministického konečného automatu A je relácia $\underset{A}{\vdash}$ na konfiguráciách definovaná $(q, an) \underset{A}{\vdash} (p, n) \Leftrightarrow p \in \delta(q, a)$*

Poznámka 3.2.1. Nedeterministický konečný automat sa označuje skrátene ako NKA, prípadne z anglického jazyka $NDFA$ (*non-deterministic finite automaton*).

Poznámka 3.2.2. Hlavný rozdieľ medzi deterinistickým a nedeterministickým automatom je, že $\delta(q, a)$ je množina stavov (aj prázdna) a nie jeden stav. Výraz $\delta(q, a) = \{p_1, p_2, \ldots, p_n\}$ potom chápeme tak, že automat sa v stave q pri čítaní symbolu a môže rozhodnúť do akého stavu sa dostane po posune čítacej hlavy. Na danom slove môže teda prebiehať viacero rôznych výpočtov, pričom niektoré sú akceptujúce, niektoré neakceptujúce.



Obrázok 3.2: NKA pre slová obsahujúce podslovo *ababa*

Príklad 3.2.1. Nájsť automat, ktorý akceptuje slová obsahujúce *ababa*. Jeho prechodový diagram je na obrázku 3.2.

3.3 Ekvivalencia DKA a NKA

O automatoch sa dajú vysloviť a dokázať podobné vety o redukcii ako o gramatikách. My sa tým ale nebudeme zaoberať. Jedným z ďalších normálnych tvarov pre automaty je aj ten nasledujúci.

Veta 3.3.1. *K ľubovolnému nedeterministickému konečnému automatu A existuje nedeterministický automat A' bez ε (t.j. $\delta(q,\varepsilon) \neq \emptyset \vee q$) tak, že $L(A) = L(A')$*

TODO: obrazok

Obrázok 3.3: konštrukcia A'

Dôkaz. Skúsme formalizovať konštrukciu automatu A' z obrázku 3.3.
$A' = (K', \Sigma', \delta', q_0', F')$: $K' = K$, $\Sigma' = \Sigma$, $q_0' = q_0$, $F' = F \cup \{q_0 \,|\, \varepsilon \in L(A)\}$[1], $[q]_\varepsilon$[2] $= \{p \,|\, (q,\varepsilon) \overset{*}{\vdash} (p,\varepsilon)\}$
$\delta'(q,a) = \{p \,|\, p \in [s]_\varepsilon, s \in \delta(q,a)\} \,\forall\, q \neq q_0$
$\delta'(q_0,a) = \{p \,|\, p \in [s]_\varepsilon, s \in \delta(r,a),\ r \in [q_0]_\varepsilon\}$

Treba ukázať, že $L(A') = L(A)$

$\subseteq$:

$w = a_1 \dots a_n \in L(a)') \rightsquigarrow$ existuje akceptujúci výpočet w v A $(q_0', a_1 \dots a_n) \underset{A'}{\vdash} (q_1, a_2 \dots a_n) \underset{A'}{\vdash} \cdots \underset{A'}{\vdash} (q_n, \varepsilon)$ $q_n \in F', \forall a_i \in Z$ Treba nájsť akceptujúci výpočet na w v A. Tvrdíme, že pre každé i $(q_i, a_{i+1} \dots a_n) \overset{*}{\underset{A}{\vdash}} (q_{i+1}, a_{i+2} \dots a_n)$

- $i = 0$ ak $\delta(q_i, a_i) = q_{i+1}$ (dá sa to urobiť priamo) ak $\delta(q_i, a_i) \neq q_{i+1}$, potom z definície δ' existuje $r \in [q_0]_\varepsilon | s \in \delta(r, a_{i+1})$ $q_{i+1} \in [s]_\varepsilon \rightsquigarrow (q_i, a_{i+1}) \overset{*}{\underset{A}{\vdash}} (q_{i+1}, \varepsilon)$
- $i \neq 0$ podobne ako pre $i = 0$, teda $(q_0', a_1 \dots a_n) \overset{*}{\underset{A}{\vdash}} (q_i, a_2 \dots a_n) \overset{*}{\underset{A}{\vdash}} \cdots \overset{*}{\underset{A}{\vdash}} (q_n, \varepsilon)$; $q_n \in F' \rightsquigarrow q_n \in F \vee q_n = q_0$ $\varepsilon \in L(A) \rightsquigarrow (q_0, \varepsilon) \overset{*}{\underset{A'}{\vdash}} (q, \varepsilon)$, $q \in F$

$\supseteq$:

$w = a_1 \dots a_n \in L(A)$, $a_i \in \Sigma \cup \{\varepsilon\}$ $(q_0, a_1 \dots a_n)$ $\underset{A}{\vdash} (q_1, a_2 \dots a_n) \underset{A}{\vdash} \cdots \underset{A}{\vdash} (q_n, \varepsilon)$, $q_n \in F$. „Význačné" sú tie konfigurácie $(q_i, a_{i+1} \dots a_n)$, kde $a_{i+1} \in \Sigma$. Nech $i_1, i_2, \dots, i_k$ sú všetky indexy „význačných" konfigurácií. Nech $k \geq 1$. Potom $w = a_{i_1} a_{i_2} \dots a_{i_k}$ a $\delta'(q_0, a_{i_j}) \ni q_{i_2 - 1}, \delta'(q_{i_j - 1}, a_{i_j}) \ni q_{i_{j+1} - 1}$. Teda $(q_0, a_{i_1} \dots a_{i_k}) \underset{A'}{\vdash} (q_{i_2 - 1}, a_{1_2} \dots a_{i_k}) \underset{A'}{\vdash} \cdots \underset{A'}{\vdash} (q_n, \varepsilon)$. Ak $k = 0 \rightsquigarrow w \in \varepsilon$ a $\varepsilon \in L(A')$ podľa def. F'.

□

Príklad 3.3.1. TODO: obrazky

[1] podmienená množina TODO: vysvetliť
[2] epsilonový chvost

Veta 3.3.2. *K ľubovolnému nedeterministickému konečnému automatu A existuje deterministický konečný automat A' taký, že $L(A) = L(A')$.*

Dôkaz. Bez straty na všeobecnosti predpokladajme, že A je bez ε. Zostrojme A' takto: $K' = 2^K = \{q_B \mid B \subset K\}$, $\Sigma' = \Sigma$, $q'_0 = q_{\{q_0\}}$, $F' = \{q_B \mid B \cap F \neq \emptyset\}$, $\delta'(q_B, a) = q_C$, kde $C = \{q \in K \mid q \in \delta(S, a), S \in B\}$. Treba ukázať, že $L(A') = L(A), \quad \forall a_i \in \Sigma$.

$\subseteq$:

$w = a_1 \dots a_n \in L(A')$, t.j. existuje výpočet $(q_{B_0}, a_1 \dots a_n) \underset{A'}{\vdash} (q_{B_1}, a_2 \dots a_n) \underset{A'}{\vdash} \cdots \underset{A'}{\vdash} (q_{b_n}, \varepsilon)$, $q_{b_n} \in F'$. Treba nájsť výpočet v A: $q_{B_n} \in F' \rightsquigarrow B_n \cap F \neq \emptyset$, t.j. existuje $q_n \in F \cap B_n$. Výpočet v A zostrojíme nasledovne: v každom B_i existuje stav q_i taký, že $\delta(q_{i+1}, a_i) \ni q_i$, $(q_0, a_1 \dots a_n) \underset{A}{\vdash} (q_1, a_2 \dots a_n) \underset{A}{\vdash} \cdots \underset{A}{\vdash} (q_n, \varepsilon)$

$\supseteq$:

$w = a_1 \dots a_n \in L(A)\, a_i \in \Sigma \forall i$
$(q_0, a_1 \dots a_n) \underset{A}{\vdash} (q_1, a_2 \dots a_n) \underset{A}{\vdash} \cdots \underset{A}{\vdash} (q_n, \varepsilon)\, q_n \in F$. Nech $(q_{B_0}, a_1 \dots a_n) \underset{A'}{\vdash} (q_{B_1}, a_2 \dots a_n) \underset{A'}{\vdash} \cdots \underset{A'}{\vdash} (q_{B_n}, \varepsilon)$ je výpočet A' na w. Treba ukázať, že je akceptujúci. Indukciou ľakho dokážeme, že $\forall i\, q_i \in B_i$. Teda $q_n \in B_n \rightsquigarrow q_{B_n} \in F$.

□

Poznámka 3.3.1. Touto vetou sme dokázali, že $\mathcal{L}_{NKA} \subseteq \mathcal{L}_{DKA}$. Obrátená inklúzia triviálne vyplýva z definícií NKA a DKA, čiže $\underline{\mathcal{L}_{NKA} = \mathcal{L}_{DKA}}$.

3.4 Úlohy

1. Nech A je nedeterministický konečný automat s n stavmi. Je možné, aby akceptoval práve všetky slová, ktorých dĺžka nepresahuje n?
2. Nájdite DKA pre jazyk z príkladu 3.2.1.
3. Pokúste sa navrhnúť algoritmus, ktorý pre dva dané konečné automaty A_1, A_2 zistí, či $L(A_1) = L(A_2)$.
4. Zostrojte konečný automat, ktorý rozpoznáva čísla deliteľné siedmimi. Predpokladajte, že sú zapísané v desiatkovej sústave.
5. Ku každému konečnému automatu existuje s ním ekvivalentný nedeterministický konečný automat, ktorý má najviac jeden akceptujúci stav. Dokážte.

Kapitola 4

Regulárne jazyky

4.1 Ekvivalencia tried jazykov $\mathcal{L}_{NKA}$ a $\mathcal{R}$

Zadefinovaním konečných automatov sme získali len iný pohľad na triedu jazykov a nie novú triedu jazykov, ako ukazujú nasledujúce tvrdenia o vzťahu medzi regulárnymi gramatikami a konečnými automatmi.

Veta 4.1.1. *K ľubovoľnému nedeterministickému konečnému automatu A existuje regulárna gramatika G taká, že $L(G) = L(A)$.*

Dôkaz. K danému konečnému automatu A zostrojíme regulárnu gramatiku G takto:
$N = K$, $T = \Sigma$, $\sigma = q_0$
$P = \{q \to ap \mid p \in \delta(q, a)\} \cup \{q \to \varepsilon \mid q \in F\}$

Treba ukázať, že $L(G) = L(A)$.

$\subseteq$**:** Nech slovo $w = a_1a_2a_3 \dots a_n$ patrí do jazyka $L(G)$. Musí preto existovať nejaké odvodenie

$$q_0 \underset{G}{\Longrightarrow} a_1q_1 \underset{G}{\Longrightarrow} \dots \underset{G}{\Longrightarrow} a_1a_2 \dots a_nq_n \underset{G}{\Longrightarrow} a_1a_2 \dots a_n$$

Z definície gramatiky G vieme, že $q_1 \in \delta(q_0, a_1)$, $q_2 \in \delta(q_1, a_2)$, $\dots$, $q_n \in \delta(q_{n-1}, a_n)$ a $q_n \in F$. Akceptujúci výpočet na slove w pre automat A je

$$(q_0, a_1a_2a_3 \dots a_n) \underset{A}{\vdash} (q_1, a_2a_3 \dots a_n) \underset{A}{\vdash} \dots \underset{A}{\vdash} (q_n, \varepsilon)$$

a teda $w \in L(A)$.

$\supseteq$**:** Ak slovo $w = a_1a_2a_3 \dots a_n$ je z $L(A)$, potom existuje akceptujúci výpočet

$$(q_0, a_1a_2a_3 \dots a_n) \underset{A}{\vdash} (q_1, a_2a_3 \dots a_n) \underset{A}{\vdash} \dots \underset{A}{\vdash} (q_n, \varepsilon)$$

Potom $q_1 \in \delta(q_0, a_1)$, $q_2 \in \delta(q_1, a_2)$ atď. Podľa konštrukcie gramatiky G tá musí obsahovať pravidlá $q_0 \to a_1q_1$, $q_1 \to a_2q_2$, $\dots$, $q_{n-1} \to a_nq_n$, $q_n \to \varepsilon$. Pomocou týchto pravidiel môžeme v gramatike G vytvoriť slovo w takto

$$q_0 \underset{G}{\Longrightarrow} a_1q_1 \underset{G}{\Longrightarrow} \dots \underset{G}{\Longrightarrow} a_1a_2 \dots a_nq_n \underset{G}{\Longrightarrow} a_1a_2 \dots a_n$$

□

Poznámka 4.1.1. V celom texte predpokladáme štandardné označenie pre automaty a gramatiky. Teda ak sa povie, že A_i je automat, myslíme tým, že množina stavov je K_i, počiatočný stav je q_{0i} a tak ďalej. Pre automat typu $A^x \quad x \in \{', \heartsuit, \star, \dots\}$ platí to isté. Gramatiky uvažujeme analogicky.

Podľa práve dokázaného tvrdenia platí $\mathcal{L}_{NKA} \subseteq \mathcal{R}$. Na dôkaz opačnej inklúzie $\mathcal{R} \subseteq \mathcal{L}_{NKA}$ budeme potrebovať nasledujúcu lemu o normálnom tvare regulárnych gramatík.

Lema 4.1.1. *(O normálnom tvare regulárnych gramatík) K ľubovoľnej regulárnej gramatike G existuje regulárna gramatika G' taká, že $P' \subseteq N' \times (T' \cup \{\varepsilon\})(N' \cup \{\varepsilon\})$ a $L(G') = L(G)$.*

Dôkaz. Myšlienka je identická s dôkazom vety 2.4.1 o Chomského normálnom tvare a s upravovaním pravidiel, ktorých dĺžka pravej strany je väčšia ako dva. □

Veta 4.1.2. *K ľubovoľnej regulárnej gramatike G existuje nedeterministický konečný automat A taký, že $L(A) = L(G)$.*

Dôkaz. Podľa lemy 4.1.1 môžeme predpokladať, že $P \subseteq N \times (T \cup \{\varepsilon\})(N \cup \{\varepsilon\})$. Zkonštruujme NKA A takto:
$q_0 = \sigma$
$K = N \cup \{q_f\}$
$\Sigma = T$, $F = \{q_f\}$
$\delta(\xi, a) = \{\eta \in N \mid \xi \to a\eta \in P\} \cup \{q_f \mid \xi \to a \in P\} \qquad \forall \xi \in N, \forall a \in T \cup \{\varepsilon\}$

Opäť treba ukázať, že $L(A) = L(G)$. Obe inklúzie sú pomerne jednoduché a vo veľkej miere sa dá použiť myšlienka dôkazu vety 4.1.1, preto dôkaz ich platnosti prenechávame čitateľovi ako ľahké domáce cvičenie. □

Veta 4.1.3. $\mathcal{L}_{NKA} = \mathcal{R}$

Dôkaz. Vyplýva z viet 4.1.1 a 4.1.2. □

4.2 Myhill-Nerodova veta

Poukážeme tu na vzťahy medzi konečnými automatmi, jazykmi nimi generovanými a reláciami ekvivalencie. Najskôr ale zadefinujeme potrebné pojmy.

Definícia 4.2.1. *Binárna relácia R nad Σ sa nazýva sprava invariantná (vzhľadom na operáciu zreťazenie), ak platí*

$$xRy \Rightarrow \forall v \in \Sigma^* \; xvRyv$$

Definícia 4.2.2. *Počet tried relácie ekvivalencie nazývame index. Ak je tento počet konečný, hovoríme, že relácia je konečného indexu.*

No a teraz už sľubovaná veta.

Veta 4.2.1. *(Myhill-Nerod) Majme jazyk $L \subseteq \Sigma^*$. Nasledujúce tvrdenia sú ekvivalentné*

1. *L je akceptovaný konečným automatom*
2. *L je zjednotením niekoľkých tried sprava invariantnej relácie ekvivalencie konečného indexu.*
3. *relácia R_L definovaná*

$$uR_Lv \Leftrightarrow_{df} \forall x \in \Sigma^* \; (ux \in L \Leftrightarrow vx \in L)$$

je reláciou ekvivalencie konečného indexu.

Dôkaz. Dokážme tvrdenia 1.⇒2., 2.⇒3. a 3.⇒1.

1.⇒2. Nech $L = L(A)$, kde A je deterministický konečný automat. Definujme reláciu R takto:

$$uRv \Leftrightarrow_{df} (q_0, u) \overset{*}{\underset{A}{\vdash}} (p, \varepsilon), (q_0, v) \overset{*}{\underset{A}{\vdash}} (q, \varepsilon) \text{ a } p = q$$

Čo všetko musíme ukázať ? Že táto relácia je

- reláciou ekvivalencie. Ale všetky potrebné vlastnosti (teda reflexívnosť, symetrickosť a tranzitívnosť) sa zdedia z rovnosti $p = q$ (overte).
- sprava invariantná. Ale ak nejaké slová u, v dovedú automat pri výpočte do rovnakého stavu q, tak potom sa zrejme z tohto stavu automat na slove x dostane do nejakého stavu p a teda slová ux a vx opäť privedú automat do rovnakého stavu.
- konečného indexu. Zrejmé, počet tried ekvivalencie je nanajvýš $|K|$.
- a že L je zjednotením niekoľkých tried tejto relácie. Budú to tie triedy, kde sú slová, ktoré privedú automat do koncového stavu. Označme $[q] = \{w \in \Sigma^* \mid (q_0, w) \overset{*}{\vdash} (q, \varepsilon)\}$. Vidíme, že $L = \bigcup_{q \in F} [q]$.

  $\subseteq$**:** ak $w \in L$, tak $w \in L(A)$, teda existuje výpočet, ktorý privedie automat do koncového stavu q_f. Teda $w \in [q_f]$ a preto $w \in \bigcup_{q \in F} [q]$.

  $\supseteq$**:** ak $w \in \bigcup_{q \in F} [q]$, potom zrejme $w \in [q]$, $q \in F$ pre nejaké q, teda $(q_0, w) \overset{*}{\vdash} (q, \varepsilon)$ a teda $w \in L(A) = L$.

2.⇒3. Prv, než pristúpime k dôkazu je potrebné si uvedomiť, že táto implikácia sa musí dať dokázať bez toho, že by sme čosi vedeli o konečných automatoch. A teraz už k samotnému dôkazu. Ukážeme, že R je relácia ekvivalencie a že je zjemnením relácie R_L. Teda

$$uRv \Rightarrow uR_Lv$$

Dôkaz toho, že R je relácia ekvivalencie prenechávame opäť na čitateľa. Rozoberme si to zjemenenie. Teda nech $(u, v) \in R$. Keďže R je sprava invariantná, platí $\forall x\ uxRvx$. Jazyk L je zjednotením niekoľkých tried tejto relácie, znamená to, že slová ux,vx buď obe patria do L, ak príslušná trieda relácie ekvivalencie R patrí do L, alebo obe slová do L nepatria z rovnakej príčiny. Toto platí pre všetky x. Ale toto je presne definícia relácie R_L, teda máme uR_Lv. Dokázali sme, že R je zjemnením R_L. Preto R_L nemôže mať viacej tried ako relácia R. Keďže R je konečného indexu, musí takou byť aj relácia R_L.

3.⇒1. Na zákalade vlastnosti R_L treba zostrojiť konečný automat A pre jazyk L. Automat bude vyzerať takto.
$K = \{[x] \mid x \in \Sigma^*\}$, kde $[x]$ je trieda ekvivalencie obsahujúca x. K bude konečná, lebo R_L je konečného indexu.
$\delta([x], a) = [xa] \qquad \forall x \forall a$
$q_0 = [\varepsilon]$
$F = \{[x] \mid x \in L\}$

Musíme ukázať o našej δ funkcii, že je to skutočne funkcia, teda že jednému prvku je priradený práve jeden prvok. Ulážme, že našu δ funkciu sme zadefinovali korektne a že funkčná hodnota nezávisí od voľby reprezentanta, teda že

$$[x] = [y] \Rightarrow xaR_Lya, \text{ t.j. } [xa] = [ya]$$

$[x] = [y] \Rightarrow xR_L y \Rightarrow \forall z(xz \in L \Leftrightarrow yz \in L) \Rightarrow \forall az(x(az) \in L \Leftrightarrow y(az) \in L) \Rightarrow \forall z((xa)z \in L \Leftrightarrow (ya)z \in L) \Rightarrow [xa] = [ya]$

Zrejme platí tiež $([\varepsilon], w) \overset{*}{\vdash} ([w], \varepsilon)$a $[w] \in F \Leftrightarrow w \in L$. Z daného máme $L = L(A)$.

□

4.3 Uzáverové vlastnosti

Definícia 4.3.1. *Trieda jazykov $\mathcal{L}$ je uzavretá na binárnu operáciu $\diamond$, ak pre všetky jazyky $L_1, L_2 \in \mathcal{L}$ platí, že $L_1 \diamond L_2 \in \mathcal{L}$.*

Poznámka 4.3.1. Definícia sa dá rozšíriť aj na n-árne operácie, $n \geq 1$ (teda vrátane unárnych, vtedy by sme zrejme hovorili o zúžení definície).

Veta 4.3.1. *$\mathcal{R}$ je uzavretá na zjednotenie.*

Dôkaz. Majme $L_1, L_2 \in \mathcal{R}$ a regulárne gramatiky G_1, G_2 také, že $L_1 = L(G_1)$, $L_2 = L(G_2)$ a $N_1 \cap (N_2 \cup T_2) = N_2 \cap (N_1 \cup T_1) = \emptyset$. Čitateľ sa na tomto mieste isto rád zamyslí nad tým, že také dve gramatiky vždy exitujú. Zostrojme gramatiku G_3 s vlastnosťou $L(G_3) = L(G_1) \cup L(G_2)$.

$$G_3 = (N_1 \cup N_2 \cup \{\sigma_3\}, T_1 \cup T_2, \{\sigma_3 \to \sigma_1 | \sigma_2\} \cup P_1 \cup P_2, \sigma_3) \quad \text{kde } \sigma_3 \text{ je nový neterminál}$$

Dokážme, že $L(G_3) = L(G_1) \cup L(G_2)$.

$\subseteq$: Majme odvodenie slova w v G_3 $\sigma_3 \underset{G_3}{\Longrightarrow} w_1 \underset{G_3}{\Longrightarrow} w_2 \underset{G_3}{\Longrightarrow} \ldots \underset{G_3}{\Longrightarrow} w_n = w$. Na začiatku sa muselo použiť pravidlo $\sigma_3 \to \sigma_1$ alebo $\sigma_3 \to \sigma_2$. Predpoladajme, že sa použilo to prvé. Potom $w_1 = \sigma_1$ a máme odvodenie $\sigma_3 \underset{G_3}{\Longrightarrow} \sigma_1 \underset{G_3}{\Longrightarrow} w_2 \underset{G_3}{\Longrightarrow} \ldots \underset{G_3}{\Longrightarrow} w$ v ktorom pre vetné formy $w_i, w_{i+1}, 1 \leq i \leq n-1$, platí $w_i \underset{G_1}{\Longrightarrow} w_{i+1}$, teda že sa používali pravidlá len z množiny P_1. To sa dá ukázať indukciou na i. Z tohto dôvodu máme $\sigma_1 \overset{*}{\underset{G_1}{\Longrightarrow}} w_n = w$, z čoho plynie $w \in L(G_1)$ a z toho $w \in L(G_1) \cup L(G_2)$. Prípad použitia pravidla $\sigma_3 \to \sigma_2$ sa dokazuje úplne analogicky.

$\supseteq$: Majme $w \in L(G_1) \cup L(G_2)$. Nech slovo w je z množiny $L(G_i)$, $i \in \{1, 2\}$. Teda existuje odvodenie $\sigma_i \underset{G_i}{\Longrightarrow} w_1 \underset{G_i}{\Longrightarrow} w_2 \underset{G_i}{\Longrightarrow} \ldots \underset{G_i}{\Longrightarrow} w_n = w$. Keďže neterminály σ_3, σ_i sú v relácii krok odvodenia $\underset{G_3}{\Longrightarrow}$ a každé pravidlo z P_i je aj pravidlo z P_3, dostávame tak odvodenie slova w gramatikou G_3 $\sigma_3 \underset{G_3}{\Longrightarrow} \sigma_i \underset{G_3}{\Longrightarrow} w_1 \underset{G_3}{\Longrightarrow} w_2 \underset{G_3}{\Longrightarrow} \ldots \underset{G_3}{\Longrightarrow} w_n = w$.

□

Veta 4.3.2. *$\mathcal{R}$ je uzavretá na zreťazenie.*

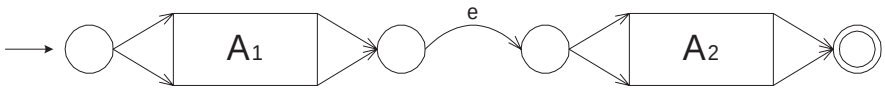
Dôkaz. Urobme teraz dôkaz pomocou automatov. Majme teda jazyky $L_1, L_2 \in \mathcal{R}$, pre ne konečné automaty A_1, A_2 a máme ukázať, že jazyk $L_1 L_2 \in \mathcal{R}$. Bez straty na všeobecnosti môžeme predpokladať

- *$K_1 \cap K_2 = \emptyset$ (: to, ako sa nazývajú stavy, nemá žiaden vplyv na jazyk, ktorý automaty akceptujú, preto si ich jednoduchou substitúciou môžeme premenovať :)*
- *$|F_1| = |F_2| = 1$ (: dorobí sa nový koncový stav, kam sa bude skákať na ε zo starých koncových stavov :)*

- $q_{0i} \notin \delta_i(q,a) \qquad \forall q \forall a,\ i \in \{1,2\}$, kde q_{0i} je počiatočný stav automatu i *(: teda že sa automat nikdy nevráti do počiatočného stavu; to sa zabezpečí zavedením nového počiatočného stavu :)*
- $\delta_i(q_{f_i},a) = \emptyset \qquad \forall a,\ i = 1,2 \qquad$ kde $F_1 = \{q_{f_1}\}$, $F_2 = \{q_{f_2}\}$

Ak automat v takomto normálnom tvare zakreslíme

náš automat A_3 potom zostrojíme takto



Formálne zapísané
$K_3 = K_1 \cup K_2$
$\Sigma_3 = \Sigma_1 \cup \Sigma_2$
$q_{03} = q_{01}$
$F_3 = F_2$

$$\delta_3(q,a) = \begin{cases} \delta_1(q,a) & q \in (K_1 - F_1), a \in \Sigma_1 \cup \{\varepsilon\} \\ \delta_2(q,a) & q \in K_2, a \in \Sigma_2 \cup \{\varepsilon\} \\ q_{02} & q \in F_1, a = \varepsilon \end{cases}$$

Treba dokázať, že $L(A_3) = L_1 L_2$.

$\supseteq$: Táto inklúzia je ľahšia. Ak $w \in L_1 L_2$, potom $\exists u_1, u_2$ také, že $u_1 \in L_1, u_2 \in L_2$ a $w = u_1 u_2$. Keďže platí $(q_{01}, u_1) \overset{*}{\underset{A_1}{\vdash}} (q_{f_1}, \varepsilon)$ a $(q_{02}, u_2) \overset{*}{\underset{A_2}{\vdash}} (q_{f_2}, \varepsilon)$, z definície δ_3 funkcie vyplýva, že $(q_{01}, u_1 u_2) \overset{*}{\underset{A_3}{\vdash}} (q_{f_1}, u_2) \underset{A_3}{\vdash} (q_{02}, u_2) \overset{*}{\underset{A_3}{\vdash}} (q_{f_2}, \varepsilon)$.

$\subseteq$: Ak $w \in L(A_3)$, tak máme nejaký akceptujúci výpočet $(q_{01}, a_1 a_2 \ldots a_n) \underset{A_3}{\vdash} (q_1, a_2 \ldots a_n) \underset{A_3}{\vdash} \ldots \underset{A_3}{\vdash} (q_{f_2}, \varepsilon)$. Vzhľadom na disjunktnosť množín K_1 a K_2 a definíciu δ_3 funkcie musí existovať nejaké i také, že $q_i = q_{02}$. Zrejme potom $q_{i-1} = q_{f_1}$ a $a_i = \varepsilon$. Teda

$(q_{01}, a_1 a_2 \ldots a_{i-1}) \overset{*}{\underset{A_1}{\vdash}} (q_{f_1}, \varepsilon)$

$(q_{02}, a_{i+1} a_{i+2} \ldots a_n) \overset{*}{\underset{A_2}{\vdash}} (q_{f_2}, \varepsilon)$

Našli sme slová $w_1 \in L(A_1)$ a $w_2 \in L(A_2)$ také, že $w = w_1 w_2$.

□

Veta 4.3.3. *$\mathcal{R}$ je uzavretá na prienik.*

Dôkaz. Nech $L_1, L_2 \in \mathcal{R}$ a nech A_1, A_2 sú konečné automaty, pre ktoré $L(A_1) = L_1$ a $L(A_2) = L_2$. Zostrojme nový automat A_3, ktorý bude simulovať paralelnú prácu oboch automatov. Bude si pamätať, v akom stave by bol prvý automat a taktiež v ktorom stave by bol druhý automat po prečítaní písmenka. Toto sa dá ľahko dosiahnúť karteziánskym súčinom

množín K_1, K_2. A_3 akceptuje dané slovo, len ak by sa automat A_1 aj automat A_2 dostali do akceptačných stavov. Konštrukcia A_3:
$K_3 = K_1 \times K_2$
$\Sigma_3 = \Sigma_1 \cap \Sigma_2$[1].
$F_3 = F_1 \times F_2$
$\delta_3((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$

Ľahko vidieť, že $L(A_3) = L(A_1) \cap L(A_2)$. Formálny dôkaz sa prenecháva čitateľovi ako domáce cvičenie. □

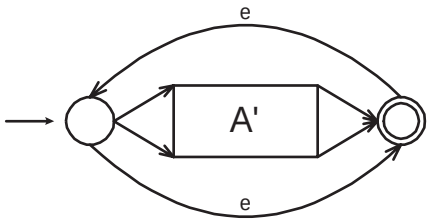
Veta 4.3.4. *$\mathcal{R}$ je uzavretá na komplement.*

Dôkaz. Ukážeme jednoduchý dôkaz pomocou Myhill-Nerodovej vety. Podľa nej je jazyk $L \in \mathcal{R}$ zjednotením niekoľkých tried sprava invariantnej relácie ekvivalencie konečného indexu. Komplement tohto jazyka zrejme bude zjednotením tých druhých tried. □

Poznámka 4.3.2. Toto tvrdenie sa dá dokázať aj iným spôsobom. Ak si uvedomíme, že deterministické konečné automaty sú deterministické a vždy musia dočítať vstupné slovo, môžeme jednoducho zostrojiť automat A' pre komplement jazyka $L(A)$ tak, že zameníme akceptačné stavy za neakceptačné a naopak. Formálne $F' = \{q \in K \mid q \notin F\}$.

Veta 4.3.5. *$\mathcal{R}$ je uzavretá na iteráciu*[2].

Dôkaz. Nech A je automat pre jazyk $L \in \mathcal{R}$ v normálnom tvare podľa vety4.3.2. Potom automat A' pre jazyk L^* bude vyzerať takto.



Automat vždy, keď akceptuje slovo $u_i \in L$, má možnosť sa vrátiť a akceptovať ďalšie slovo. Taktiež je vidieť, že automat A' akceptuje ε. Dokážme teda $L(A') = L^*$. Ak máme slovo $w \in L^*$, tak z vlastnosti $L^* = \{u_1u_2u_3 \dots u_k \mid k \in N, u_i \in L, 1 \leq i \leq k\} \cup \{\varepsilon\}$ máme $w = u_1u_2 \dots u_k$ pre nejaké k alebo $w = \varepsilon$. Druhý prípad je hneď vybavený a prvý vlastne tiež, lebo potom máme k akceptačných výpočtov automatu A na slovách $u_1, u_2, \dots, u_k$, ktoré môžeme prechodmi na epsilon z q_f do q_0 pospájať do jedného výpočtu automatu A'. Obrátene, ak máme slovo $w \in L(A')$, tak si budeme všímať konfigurácie, ktoré obsahujú stav q_f. Potom zrejme nasledujúci stav môže byť len q_0 (z definície δ funkcie) a teda zrejme postupnosť medzi dvomi význačnými konfiguráciami je jeden výpočet automatu A a teda dané slovo w sa dá rozdeliť na viacero slov u_i, z ktorých každé je akceptované konečným automatom A. □

Veta 4.3.6. *Každý konečný jazyk je v $\mathcal{R}$.*

[1] ale mohli sme kľudne dať aj $\Sigma_1 \cup \Sigma_2$
[2] Kleeneho $*$

Dôkaz. Nech $L = \{w_1, w_2, w_3, \ldots, w_n\}$ je konečný jazyk nad nejakou abecedou Σ. Zostrojíme regulárnu gramatiku G pre tento jazyk nasledovne. $G = (\{\sigma\}, \Sigma, P, \sigma)$, kde

$$P = \{\sigma \to w_1 | w_2 | w_3 | \ldots | w_n\}$$

□

4.4 Kleeneho veta

Veta 4.4.1. *R je najmenšia trieda jazykov obsahujúca všetky konečné jazyky a uzavretá na zjednotenie, zreťazenie a uzáver.*

Dôkaz. Podľa predchádzajucich odsekov R naozaj obsahuje všetky konečné jazyky a na horeuvedené vlastnosti je uzavretá.

Na to, aby sme dokázali, že R je najmenšou takouto triedou, musíme ukázať, že každý regulárny jazyk vznikne z konečných jazykov aplikovaním konečného počtu horeuvedených operácií. Nech teda $R \ni L = L(A)$, kde A je konečný deterministický automat, ktorý akceptuje L.
Označme $K = \{p_1, p_2, ..., p_n\}$, $q_0 = p_1$.
Ďalej označme

$$R_{ij}^k = \{a_1 a_2 ... a_l \mid (q_1, a_1 a_2 ... a_l) \underset{A}{\vdash} (q_2, a_2 ... a_l) \underset{A}{\vdash} ... \underset{A}{\vdash} (q_{l+1}, \varepsilon) \wedge$$

$$(q_1 = p_i) \wedge (q_{l+1} = p_j) \wedge (\forall s)(1 < s < l+1)(\exists t \leq k) q_s = p_t\}$$

(: To znamená, že R_{ij}^k je množina takých slov, ktoré znamenajú prechod zo stavu p_i do stavu p_j, pričom nijaký stav okrem p_i, p_j, cez ktorý prechádzame, nemá index väčší než k. :)
Zrejme platí

$$L = \bigcup_{p_i \in F} R_{1i}^n$$

Stačí teda dokázať, že $(\forall i, j) R_{ij}^n$ vznikne z konečných jazykov konečným počtom horeuvedených operácií. Toto dokážeme matematickou indukciou k n.

$R_{ij}^0 = \{a \in \Sigma \mid \delta(p_i, a) = p_j\}$. Tento jazyk je konečný a teda vyhovuje.
Nech tejto podmienke vyhovujú všetky jazyky R_{ij}^k, $k < k_0$ a dokážeme, že potom vyhovuje aj jazyk $R_{ij}^{k_0}$.
Ľahko vidno, že $R_{ij}^{k_0} = R_{ij}^{k_0 - 1} \cup R_{i(k_0)}^{k_0 - 1} (R_{(k_0)(k_0)}^{k_0 - 1})_{kon}^* R_{(k_0)j}^{k_0 - 1}$, a teda tiež vyhovuje. □

4.5 Regulárne výrazy

Definícia 4.5.1. *Regulárne výrazy sú:*

1. *$\underline{\varepsilon}$, $\underline{\emptyset}$, $\underline{a}$ $\forall a \in \Sigma$ sú regulárne výrazy.*
2. *Ak r_1, r_2 sú regulárne výrazy, potom aj $(r_1 + r_2)$, $r_1 r_2$, $(r_1)^*$ sú regulárne výrazy.*
3. *Nič iné nie je regulárnym výrazom.*

Definícia 4.5.2. *Jazyk popísaný regulárnym výrazom r je jazyk $L(r)$ definovaný takto:*

1. *$L(\underline{\varepsilon}) = \{\varepsilon\}$, $L(\underline{\emptyset}) = \emptyset$, $L(\underline{a}) = \{a\}$.*

2. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.

3. $L(r_1 r_2) = L(r_1)L(r_2)$

4. $L((r)^*) = (L(r))^*$

5. $L((r)) = L(r)$.

4.6 Pumpovacia lema pre regulárne jazyky

Nasledujúca veta je po Nerodovej vete ďalším nástrojom na zistenie, že nejaký jazyk nie je regulárny.

Veta 4.6.1. *Ku každému regulárnemu jazyku L existuje číslo p také, že pre každé slovo $w \in L$, $|w| > p$ platí $\exists x, u, y \in \Sigma^*$ také, že platí*

1. $w = xuy$

2. $u \neq \varepsilon$.

3. $|xu| \leq p$

4. $(\forall i \geq 0) xu^i y \in R$

Podľa vlastnosti (4) sa táto veta nazýva pumovacou, lebo táto vlastnosť pripomína akési napumpovávanie slova.

Dôkaz. Nech L je regulárny jazyk a A je jemu prislúchajuci deterministický automat. Označme $p = |K(A)|$ a dokážme, že toto p vyhovuje tvrdeniu vety. Nech $a_1 a_2 ... a_l = w \in L$, $l > p$ a nech $(q_0, a_1 a_2 ... a_l) \vdash (q_1, a_2 ... a_l) \vdash ... \vdash (q_l, \varepsilon)$ je akceptujúci výpočet slova w. Podľa Dirichletovho princípu musia existovať čísla i, j také že $0 \leq i < j \leq p$ a $q_i = q_j$. Nech $x = a_1 a_2 ... a_i$, $u = a_{i+1} a_{i+2} ... a_j$, $y = a_{j+1} a_{j+2} ... a_l$. Poľahky nahliadneme, že xu^2y, xu^3y,... sú týmto automatom takisto akceptované, a tak patria L, čo bolo treba dokázať. □

Príklad 4.6.1. Dokážte, že jazyk $L = \{a^n b^n \mid n \geq 0 \}$ nie je regulárny.

Riešenie:

Postupujme sporom. Nech L je regulárny. Potom podľa pumpovacej lemy existuje isté p s istými vlastnosťami. Uvažujme slovo $a^p b^p$, ktoré definitoricky patrí L. Musia existovať slová u, v, x také, že

1. $a^p b^p = uvx$

2. $v \neq \varepsilon$

3. $|uv| \leq p$

4. $uv^k x \in L$ pre $\forall k$

Ale pre $k = 2$ dostávame, že slovo $a^{p+j} b^p$, kde $j > 0$ patrí L, čo je spor.

4.7 Úlohy

1. Dokážte uzavretosť $\mathcal{R}$ na zreťazenie pomocou gramatík.
2. Ukážte, prečo vo všeobecnosti neplatí $L(A)^c = L(A')$, kde A je NKA, $A' = (K, \Sigma, \delta, F')$ a $F' = (q \in K \mid q \notin F)$.
3. Popíšte algoritmus, ktorý k danému konečnému automatu A nájde regulárny výraz V taký, že $L(A) = L(V)$.
4. Nech L je jazyk nad jednopísmenovou abecedou. Potom L^* je regulárny. Dokážte.
5. Nájdite jazyk, ktorý nie je regulárny, no platí preň pumpovacia lema.

Kapitola 5

Zásobníkové automaty

5.1 Definície

V popisovaní zariadení a jazykov sa dostávame opäť k vyšším métam. Dosiaľ sme pracovali iba s konečnými automatmi, ktoré boli obmedzené tým, že rozpoznávali len regulárne jazyky, pričom nedeterminizmus na tejto skutočnosti nič nemenil. V nasledujúcich odsekoch sa budeme venovať zásobníkovým automatom.

Definícia 5.1.1. *Nedeterministickým zásobníkovým automatom nazývame 7-ticu* $(K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde* K *je konečná množina stavov,* Σ *je abeceda vstupných symbolov,* Γ *je abeceda zásobníkových symbolov,* $Z_0 \in \Gamma$ *je symbol, ktorý je v zásobníku na začiatku výpočtu,* $F \subseteq K$ *je množina akceptačných stavov a* $\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{K \times \Gamma^*}_{kon}$ *je prechodová funkcia.*

Definícia 5.1.2. *Konfiguráciou nedeterministického zásobníkoveho automatu nazývame trojicu* $\eta \in K \times \Sigma^* \times \Gamma^*$.

Definícia 5.1.3. *Krokom výpočtu nedeterministického zásobníkového automatu* A *nazývame reláciu* $\vdash_A$ *na množine konfigurácií definovanú takto:*

$$(q, au, \gamma z) \vdash_A (p, u, \gamma\beta) \;\Leftrightarrow\; (p, \beta) \in \delta(q, a, z) \tag{5.1}$$

Poznámka 5.1.1. Hovoríme o deštrukčnom čítaní. To znamená, že automat na jeden krok môže prečítať vst. symbol, zo zásobníka vyberie (vždy) jeden symbol, prejde do iného stavu a poprípade na vrch zásobníka vloží zopár symbolov.

Poznámka 5.1.2. V literatúre vládne nejednotnosť ohľadom toho, či vrch zásobníka písať naľavo alebo napravo. My budeme používať notáciu, v ktorej vrch zásobníka je vpravo.

Na rozdiel od konečných automatov máme na tomto mieste dve možnosti, ako definovať jazyk akceptovaný zásobníkovým automatom. Prvá z nich je rovnaká ako u konečných automatov, t.j. považovať za akceptované slovo také, po prečítaní ktorého sa automat ocitne v aspoň jednom z akceptujúcich stavov. Druhá možnosť je taká, že za akceptované slovo považujeme také, po prečítaní ktorého sa vyprázdni zásobník. Neskôr dokážeme, že obe tieto možnosti sú z hľadiska svojej popisnej sily ekvivalentné.

Definícia 5.1.4. *Jazyk akceptovaný zásobníkovým automatom* A **akceptujúcim stavom** *je množina* $L(A) =_{df} \{w \mid (\exists q \in F)(q_0, w, Z_0) \vdash^*_A (q, \varepsilon, \gamma)\}$.

Definícia 5.1.5. *Jazyk akceptovaný zásobníkovým automatom* A **prázdnou pamäťou** *je množina* $N(A) =_{df} \{w \mid \exists q\ (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \varepsilon)\}$.

Poznámka 5.1.3. Ak chceme zdôrazniť, že používame druhú definíciu, pri definícii automatu kladieme $F = \emptyset$. Nie žeby na tom záležalo, ale buchne to do očí.

Uvedomme si, že δ-funkcia je akýsi primitívny programovací jazyk. Cieľom tejto prednášky je práve naučiť študentov pochopiť princípy akéhokoľvek programovacieho jazyka.

Z definície zásobníkových automatov priamo vyplýva, že ich popisná sila nie je menšia ako popisná sila konečných automatov. Zatiaľ však ešte nevieme ani to, či je väčšia. Nasledujúci príklad ukazuje, že áno (pretože vieme, že $\{a^n b^n \mid n \in N\}$ nie je regulárny jazyk).

Príklad 5.1.1. Navrhnite zásobníkový automat, ktorý akceptuje jazyk $L = \{a^n b^n \mid n \in N\}$ akceptujúcim stavom.

Riešenie. $A = (\Sigma = \{a, b\}, K = \{q_0, q_1, q_2\}, \Gamma = \{Z, a\}, \delta, Z, q_0, F = \{q_2\})$, kde
$\delta(q_0, a, Z) = (q_0, Za)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \varepsilon)$
$\delta(q_1, b, a) = (q_1, \varepsilon\}$
$\delta(q_1, \varepsilon, Z) = (q_2, Z)$

V tejto chvíli sa dostávame k miestu, kde, ako sme sľúbili, ukážeme, že je v princípe jedno, či akceptujeme prázdnou pamäťou alebo akceptujúcim stavom.

5.2 Ekvivalencia medzi akceptáciou prázdnou pamäťou a akceptačným stavom

Veta 5.2.1. *K ľubovoľnému zásobníkovemu automatu* A *existuje zásobníkový automat* A' *taký, že* $N(A') = L(A)$.

Dôkaz. Skonštruujeme automat $A' = (K', \Sigma', \Gamma', \delta', Z_0', q_0', F' = \emptyset)$ nasledovne: $K' = K \cup \{q_v\}$[1], $\Sigma' = \Sigma$, $\Gamma' = \Gamma \cup \{Z_0'\}$[2], $q_0' = q_0$. Funkciu δ' definujeme nasledovne:
$\delta'(q, a, Z)$ obsahuje $\delta(q, a, Z)$ pre $q \in K$, $Z \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$
$\delta'(q, \varepsilon, Z)$ obsahuje naviac $\{(q_v, \varepsilon)\}$ pre $q \in F$, $Z \in \Gamma$,
$\delta'(q_v, \varepsilon, Z) = \{(q_v, \varepsilon)\}$ pre $Z \in \Gamma'$
$\delta'(q_0', \varepsilon, Z_0') = \{(q_0, Z_0' Z_0)\}$.

$\subseteq$: Ak je slovo w akceptované prázdnou pamäťou, znamená to, že ani symbol Z_0' v zásobníku po jeho dočítaní nezostane. Ten sa ale zo zásobníka odstráni jedine, ak sme vo vymazávacom stave a z pásky už nič nečítame. Ale do stavu q_v sa dostaneme jedine z nejakého stavu, ktorý je pre automat A akceptačným. Preto doteraz prečítané slovo $w \in L(A)$.

$\supseteq$: Ak je slovo w akceptované automatom A akceptačným stavom, potom existuje jeho akceptačný výpočet $(q_0, w, Z_0) \vdash_A^* (q_F, \varepsilon, \gamma)$. Potom ale v A' tiež existuje výpočet $(q_0', w, Z_0') \vdash_{A'} (q_0', w, Z_0' Z_0) \vdash_{A'}^* (q_F, \varepsilon, Z_0'\gamma) \vdash_{A'}^* (q_v, \varepsilon, \varepsilon)$, a tak $w \in N(A')$ □

Veta 5.2.2. *K ľubovoľnému zásobníkovemu automatu* A *existuje zásobníkový automat* A' *taký, že* $N(A) = L(A')$.

[1] Bez ujmy na všeobecnosti predpokladáme, že $q_v \notin K$; poznamenajme, že q_v dostal meno od **v**ymazávací
[2] Bez ujmy na všeobecnosti predpokladáme, že $Z_0' \notin \Gamma$

Dôkaz. Popíšeme len myšlienku konštrukcie vhodného automatu A'. Od A sa bude líšiť tým, že bude obsahovať jeden akceptačný stav, a o jeden zásobníkový symbol Z_0' viac. Na začiatku výpočtu sa postaráme, aby v zásobníku bolo slovo $Z_0'Z_0$. Potom bude automat počítať ako pôvodný automat A, až kým navrchu zásobníka neuvidí symbol Z_0'. Keď sa to stane, prejdeme do akceptačného stavu, lebo doteraz prečítané slovo je akceptované automatom A prázdnou pamäťou a teda automatom A' akceptačným stavom. □

V nasledovnom vyriešime otázku sily zásobníkových automatov v porovnaní s bezkontextovými gramatikami. Prvá z viet ukazuje, že zásobníkové automaty sú minimálne tak silné ako bezkontextové gramatiky. Ďalšia veta, ktorej dôkaz je mierne rafinovanejší nám povie, že nie sú silnejšie.

5.3 Bezkontextové gramatiky a zásobníkové automaty

Veta 5.3.1. *K ľubovoľnej bezkontextovej gramatike G existuje zásobníkový automat A, ktorý akceptuje $L(G)$ prázdnou pamäťou.*

Dôkaz. *(:Podáme myšlienku konštrukcie tohto automatu. Bude obsahovať iba jeden stav, množina zásobníkových symbolov obsahovať množinu neterminálov ako aj množinu terminálov v bezkontextovej gramatike. Zásobníkový automat bude svojim výpočtom nápadne pripomínať ľavé krajné odvodenie slova v danej gramatike. Na začiatku výpočtu bude v zásobníku neterminál σ. Automat bude pracovať nasledovne:*

1. *Ak na vrchu zásobníka uvidí neterminál A, vyberie ho a nahradí ho reverzom pravej strany nejakého A-pravidla.*
2. *Ak na vrchu zásobníka vidíme terminál, skontrolujeme, či sa rovná vstupnému symbolu na páske. Ak nie, automat sa zasekne.*

:)

Nech $A = (K = \{q_0\}, \Sigma = T, \Gamma = N \cup T, \delta, Z_0 = \sigma, F = \emptyset)$, kde $\delta(q_0, \varepsilon, A) = \{(q_0, \gamma^R) \mid (A \to \gamma) \in P\}$ pre $\forall A \in N$
$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$ pre $\forall a \in T$.

Dokážeme $N(A) = L(G)$:

$\subseteq$: Nech $w \in N(A)$. Indukciou k počtu krokov výpočtu by sme poľahky ukázali, že $(q_0, w_1w_2, \sigma) \vdash^* (q_0, w_2, \gamma) \Rightarrow \sigma \overset{*}{\underset{G}{\Longrightarrow}} w_1\gamma$, čo implikuje dokazovanú inklúziu.

$\supseteq$: Opäť by sa ľahko indukciou k dĺžke ľavého krajného odvodenia ľahko dalo dokázať, že $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w_1w_2 \Rightarrow (q_0, w_1w_2, \sigma) \vdash_A^* (q_0, w_2, \gamma) \wedge \gamma \overset{*}{\underset{G}{\Longrightarrow}} w_2$

□

Veta 5.3.2. *Nech A je zásobníkový automat. Potom existuje bezkontextová gramatika G taká, že $L(G) = N(A)$*

Dôkaz. *(: Tento dôkaz je trochu zložitejší, preto uvádzame len jeho myšlienku. Uvedomme si však najprv, aký úžasny rozdiel je, keď k bezkontextovej gramatike vieme zostrojiť ekvivalentný zásobníkový automat s jedným jediným stavom a teraz zrazu musíme zostrojiť ekvivalentnú gramatiku k zásobníkovemu automatu s k stavmi. Uvedomme si, že z týchto dvoch viet vyplýva, že keby sme boli definovali zásobníkový automat tak, že môže mať len jeden stav, neubudlo by z jeho výpočtovej sily. :)*

Skonštruujeme bezkontextovú gramatiku G takú, že $L(G) = N(A)$:
$G = (N = K \times \Gamma \times K \cup \{\sigma\}, T = \Sigma, P, \sigma)$, kde

$$P = \{\sigma \to (q_0, Z_0, q) \mid q \in F\} \cup$$

$$\{(q, Z, p) \to a(s, Z_k, s_k)(s_k, Z_{k-1}, s_{k-1})...(s_2, Z_1, s_1 = p) \mid (s, Z_1...Z_k) \in \delta(q, a, Z)\}$$

$$\cup \{(q, Z, s) \to a \mid (s, \varepsilon) \in \delta(q, a, Z)\} \tag{5.2}$$

Aj trpezlivému čitateľovi zrejme na tomto mieste došiel dych. Nič to, ponúkame vysvetlenie tejto konštrukcie: Ide o to, že sa snažíme, aby platilo

$$(p, Z, q) \overset{*}{\underset{G}{\Longrightarrow}} w \iff (p, w, Z) \vdash_A^* (q, \varepsilon, \varepsilon) \tag{5.3}$$

□

5.4 Vlastnosti bezkontextových jazykov

V tomto odseku sa budeme venovať otázkam uzavretosti triedy bezkontextových jazykov vzhľadom na rôzne operácie. Uvedomme si, že pri dôkazoch môžeme využívať ako model bezkontextového jazyka jeho gramatiku alebo zásobníkový automat, ktorý ho akceptuje. Zopakujme si, že triedu všetkých bezkonextových jazykov označujeme L_{CF}.

Veta 5.4.1. *L_{CF} je uzavretá na zjednotenie.*

Dôkaz. Dôkaz pomocou gramatík. Nech G_1 a G_2 sú dve bezkontextové gramatiky. Bez ujmy na všeobecnosti predpokladajme, že množiny $N_1 \cup T_1$ a N_2,resp. $N_2 \cup T_2$ a N_1 sú disjunktné. Uvažujme gramatiku $G = (N, T, P, \sigma)$, kde $N = \{\sigma\} \cup N_1 \cup N_2$, $T = T_1 \cup T_2$, $P = \{\sigma \to \sigma_1 | \sigma_2\} \cup P_1 \cup P_2$. Poľahky nahliadneme, že $L(G) = L(G_1) \cup L(G_2)$. □

Veta 5.4.2. *L_{CF} je uzavretá na zreťazenie.*

Dôkaz. Postupovali by sme rovnako ako v dôkaze vety 5.4.1, len by sme pridali pravidlo $\sigma \to \sigma_1\sigma_2$ □

Veta 5.4.3. *L_{CF} je uzavretá na *.*

Dôkaz. Opäť budeme postupovať analogicky ako v predchádzajucich prípadoch, len pridáme neterminál σ', ktorý bude počiatočným neterminálom a pravidlá preň budú nasledovné: $\sigma' \to \sigma\sigma' | \varepsilon$ □

Nasledovná veta konečne hovorí aj o neuzavretosti.

Veta 5.4.4. *L_{CF} nie je uzavretá na $\cap$.*

Dôkaz. Dopustíme sa istej neporiadnosti, a to síce v tom, že odkážeme čitateľa dopredu. Neskôr totiž dokážeme, že jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$ nie je bezkontextový. Zatiaľ považujme toto (aj keď ešte nedokázané) tvrdenie za dogmu. Ale tento jazyk je prienikom zrejme bezkontextových jazykov $L_1 = \{a^n b^n c^k \mid n \geq 0, k \geq 0\}$ a $L_2 = \{a^k b^n c^n \mid n \geq 0, k \geq 0\}$. Tým je dôkaz podaný. □

Napriek tomuto smutnému zisteniu to s tým prienikom nie až také kritické. Totiž, platí, že prienikom bezkontextového jazyka a regulárneho jazyka je bezkontextový jazyk.

Uvedomme si, o čo tu vlastne ide. Ak máme dva bezkontextové jazyky (ekvivalentne: dva zásobníkové automaty), nemôžeme zapezpečiť, aby nejaký zásobníkový automat simuloval prácu oboch, pretože jedným zásobníkom nemôžeme nahradiť dva zásobníky.
Avšak, ako ukáže dôkaz nasledujúcej vety, môžeme jedným zásobníkovým automatom simulovať prácu iného zásobníkového automatu a ďalšieho konečného automatu.

Veta 5.4.5. *Nech L_1 je bezkontextový jazyk, L_2 regulárny jazyk. Potom $L_1 \cap L_2$ je bezkontextový jazyk.*

Dôkaz. Budeme postupovať nasledovne: K zásobníkovému automatu A_1 a deterministickému konečnému automatu A_2 zostrojíme zásobníkový automat A, pre ktorý platí $L(A_1) \cap L(A_2) = L(A)$.
Skúsme $A = (K = K_1 \times K_2, \Sigma = \Sigma_1 \cap \Sigma_2, \Gamma = \Gamma_1, \delta, Z_0 = Z_{01}, F = F_1 \times F_2)$, kde

$$\delta((q_1,q_2),a,A) = \left\langle \begin{array}{ll} \{((r,s),B) \mid (r,B) \in \delta_1(q_1,a,A) \wedge s = \delta_2(q_2,a)\} & \text{pre } a \in \Sigma \\ \{((r,q_2),B) \mid (r,B) \in \delta_1(q_1,a,A)\} & \text{pre } a = \varepsilon \end{array} \right.$$

Poľahky nahliadneme, že platí

$$(q_{z_1},w,\gamma_z) \overset{*}{\underset{A_1}{\vdash}} (q_{k_1},\varepsilon,\gamma_k) \wedge (q_{z_2},w) \overset{*}{\underset{A_2}{\vdash}} (q_{k_2},\varepsilon) \Leftrightarrow (\ (q_{z_1},q_{z_2}),w,\gamma_z) \overset{*}{\underset{A}{\vdash}} (\ (q_{k_1},q_{k_2}),\varepsilon,\gamma_k)$$

z čoho vyplýva dokazované tvrdenie. □

Podobne platí, že ak L_1 je bezkontextový jazyk, L_2 regulárny jazyk, potom Shuffle(L_1,L_2) je bezkontextový jazyk. Dôkaz by sa dal urobiť konštrukciou zásobnikového automatu, ktorého množina stavov je kartézskym súčinom množín stavov zásobníkového automatu A_1, resp. konečného automatu A_2, ktoré akceptujú L_1, resp. L_2. So zásobníkom pracuje len automat A_1 (podobne ako v predch. vete) a v stave sa mení vždy len jedna zložka (raz pracuje jeden automat, raz druhý - podobne ako pri kartézskom súčine dvoch konečných automatov).

5.5 Pumpovacia lema pre bezkontextové jazyky

Táto veta sa v literatúre tiež označuje ako *$p-q$-lema,* alebo tiež *lema o vkladaní.* Hovorí napríklad o tom, že jazyk $\{a^n b^n c^n \mid n \in N\}$ nie je bezkontextový. Tento fakt sme už spomenuli v dôkaze vety 5.4.4.

Veta 5.5.1. *K ľubovoľnému $L \in L_{CF}$ nad abecedou Σ existujú čísla p,q také, že*

$$(\forall w \in L,\ |w| > p)\ \ \exists u,x,v,y,z \in \Sigma^*$$

také, že

1. $w = uxvyz$
2. $xy \neq \varepsilon$
3. $|xvy| \leq q$
4. $\forall i \geq 0 \quad ux^i vy^i z \in L$

Dôkaz. Bez ujmy na všeobecnosti predpokladajme, že L je bezepsilonový, nech G je gramatika v bezepsilonovom tvare generujúca L, ktorá neobsahuje chain rules. Nech $m = \max\{|w| \mid \exists \xi\ \xi \to w \in P\}$, $p = m^{|N|+1}$. Nejmenší strom odvodenia slova w, $|w| > p$, má výšku aspoň $|N| + 1$. Zvoľme najdlhšiu cestu a na nej prvý opakujúci sa neterminál ξ zospodu (podľa Dirichletovho princípu sa tam nejaký neterminál opakovať musí). Dostali sme

$$\sigma \overset{*}{\underset{G}{\Longrightarrow}} u\xi z$$

a z neterminálu ξ sa neskôr odvodilo slovo $x\xi y$. Ak sme teda slovo w odvodili nasledovne:

$$\sigma \overset{*}{\underset{G}{\Longrightarrow}} u\xi z \overset{*}{\underset{G}{\Longrightarrow}} ux\xi yz \overset{*}{\underset{G}{\Longrightarrow}} uxvyz$$

môžeme odvodiť tiež slovo $w_i = ux^ivy^iz$ nasledovne:

$$\sigma \overset{*}{\underset{G}{\Longrightarrow}} u\xi z \overset{*}{\underset{G}{\Longrightarrow}} ux\xi yz \overset{*}{\underset{G}{\Longrightarrow}} uxx\xi yyz \overset{*}{\underset{.}{\Longrightarrow}} .. \overset{*}{\underset{u}{\Longrightarrow}} x^i\xi y^iz \overset{*}{\underset{u}{\Longrightarrow}} x^ivy^iz$$

Zrejme $q = m|N|$. Uvedomme si, že takto možno odvodiť i slovo $ux^0vy^0z = uvz$. □

Príklad 5.5.1. Na základe pumpovacej lemy možno dokázať, že jazyk $a^nb^nc^nd^+$ nie je bezkontextový. Dôkaz prenechávame na čitateľa.

Bohužiaľ, pumpovacia lema nie je postačujúcou podmienkou bezkontextovosti jazyka. Preto vznikla ďalšia veta, ktorá je zosilnením pumpovacej lemy.

Veta 5.5.2. *(Ogdenova lema)*
Pre ľubovoľný bezkontextový jazyk L nad abecedou Σ existuje číslo n také, že pre každé $w \in L$ a $|w| > n$ a ľubovoľné označenie n alebo viac symbolov vo w existujú $u, x, v, y, z \in \Sigma^*$, také, že

1. *$w = uxvyz$*
2. *xy obsahuje aspoň jeden označený symbol*
3. *xvy obsahuje najviac n označených symbolov*
4. *$(\forall i \geq 0)ux^ivy^iz \in L$*

Dôkaz. Uvedieme len myšlienku dôkazu s dôrazom na tie jeho body, v ktorých sa líši od dôkazu obyčajnej pumpovacej lemy.
Keďže L je bezkontextový jazyk, existuje gramatika G v Chomského normálnom tvare, ktorá ho generuje. Opäť si všimnime strom odvodenia slova w. V ňom nazveme vrchol *vetviacim bodom*, ak obe jeho podslová obsahujú označené písmená. Keďže w obsahuje aspoň n označených bodov a uvažovaný strom odvodenia je stromom odvodenia pre gramatiku v Chomského normálnom tvare, existuje na istej jeho ceste aspoň $\lg_2 n$ vetviacich bodov. Ak by sme zvolili $n = 2^{|N|+1}$, potom opäť na tejto ceste budú existovať dva rovnaké neterminály. Ďalej postupujeme rovnako ako v dôkaze 5.5.1. □

5.6 Úlohy

1. Ukážte, že každý bezkontextový jazyk nad jednopísmenkovou abecedou je regulárny.
2. Dokážte, že jazyk $a^nb^nc^nd^+$ nie je bezkontextový.

Kapitola 6

Zovšeobecnenia konečných automatov

Ako vieme, konečné automaty akceptujú práve regulárne jazyky. V nasledujúcom texte sa budeme venovať ich rôznym zovšeobecneniam a pozrieme sa na ich výpočtovú silu.

6.1 Viachlavé automaty

Ako sa hovorí, viac hláv– viac rozumu. V nasledovnom túto ľudovú múdrosť formálne dokážeme.

Definícia 6.1.1. *Viachlavým konečným automatom rozumieme 6-icu* $(K, \Sigma, \delta, q_0, F, k)$*, kde* K*,* Σ*,* q_0*,* F *je ako pri DKA,* k *je počet hláv a* $\delta : K \times \Sigma^k \rightarrow 2^{K \times \{0,1\}^k}$

Definícia 6.1.2. *Konfiguráciou viachlavého konečného automatu nazveme* $k + 2$*-ticu* $(q, w, l_1, l_2, ..., l_k)$*, kde* k *je počet hláv automatu,* w *čítané slovo,* l_i *pozícia* i*-tej hlavy,* q[1] *aktuálny stav.*

Definícia 6.1.3. *Krokom výpočtu viachlavého konečného automatu* A *nazveme binárnu reláciu* $\underset{A}{\vdash}$ *definovanú na množine konfigurácií nasledovne:*

$$(q_1, w, l_1, l_2, ..., l_k) \underset{A}{\vdash} (q_2, w, l'_1, l'_2, ..., l'_k) \overset{df}{\iff} (q_1, l'_1 - l_1, l'_2 - l_2, ..., l'_k - l_k) \in \delta(q, w_{l_1}, w_{l_2}, ..., w_{l_k})$$

Definícia 6.1.4. *Jazykom* L *akceptovaným viachlavým konečným automatom* A *nazveme jazyk* $L \overset{df}{=} \{w \mid (q_0, w, 1, 1, ..., 1) \overset{*}{\underset{A}{\vdash}} (q_f, w, n, n, ..., n),\ q_f \in F,\ n = |w|\}$

Poznámka 6.1.1. Predchádzajucou definíciou sme akceptáciu viachlavým zaviedli tak, že automat sa dostane do akceptačného stavu a všetky hlavy slovo dočítajú. Čitateľ by mal vedieť poľahky dokázať, že alternatívnou definíciou, kedy by sme na akceptáciu nevyžadovali dočítanie slova, by sa výpočtová sila triedy viachlavých konečných automatov nezmenila.

Poďme sa teraz zaoberať otázkou, aká je výpočtová sila viachlavých konečných automatov vo vzťahu k iným triedam automatov. Predovšetkým si uvedomme, že ich sila nie je menšia, ako sila obyčajných konečných automatov (s jednou hlavou). Naopak, napríklad jazyky

- $L_1 = \{ww \mid w \in \Sigma^*\}$

[1] na veľké prekvapenie čitateľskej obce

- $L_2 = \{(w\#)^k \mid w \in L_R\}$
- $L_3 = \{a^n b^n \mid n \in \mathbb{N}\}$

sú akceptovateľné dvojhlavým, resp. k-hlavým, resp. dvojhlavým konečným automatom. Nijaký týchto jazykov však nie je regulárny a prvé dva dokonca nie sú ani bezkontextové.

Teda, trieda viachlavých konečných automatov je ostro silnejšia ako trieda $\mathcal{R}$. Dokonca obsahuje aj nejaké bezkontextové jazyky. Avšak, neobsahuje všetky bezkontextové jazyky, napríklad $\{ww^R \mid w \in \Sigma^*\}$, a tak je táto trieda s triedou bezkontextových jazykov neporovnateľná.

6.2 Dvojsmerné automaty

Na rozdiel od viachlavých automatov, dvojsmerné automaty nebudú znamenať nárast výpočtovej sily automatov. Dôkaz tohto tvrdenia nebude patriť k najtriviálnejším. [2]

Základnou myšlienkou dvojsmerných deterministických automatov je, že na základe informácie o stave, v ktorom sa nachádza a prečítaného symbolu prejde do iného stavu a s hlavou sa posunie doprava alebo doľava. Aby sme sa nedostali pred začiatok slova a aby sme si označili koniec slova, budeme používať zarážky ¢, resp. a \$ znamenajúce začiatok, resp. koniec slova.

Definícia 6.2.1. *Dvojsmerným (konečným deterministickým) automatom (2DKA) nazývame 5-icu* $A = \{K, \Sigma, \delta, q_0, F\}$, *kde* K, Σ, q_0, F *je ako pri DKA,*
$\delta : K \times \Sigma \cup \{¢, \$\} \to K \times \{-1, 0, 1\}$
pričom platí $\delta(q, \$) \subseteq \{0, 1\}$ *a* $\delta(q, ¢) \subseteq K \times \{-1, 0\}$ *pre všetky* $q \in K$.

Definícia 6.2.2. *Nech* $\# \notin \Sigma$. *Konfiguráciou dvojsmerného deterministického automatu* A *nazývame dvojicu* $(q, ¢\Sigma^*\#\Sigma^*\$)$

Definícia 6.2.3. *Krokom výpočtu* $\underset{A}{\vdash}$ *dvojsmerného konečného deterministického automatu* A *nazývame binárnu reláciu na množine konfigurácií definovanú nasledovne:*

$$\begin{aligned}
(p, u\#av) &\underset{A}{\vdash} (q, ua\#v) \quad ak \quad (q, 1) \in \delta(p, a) \\
(p, u\#av) &\underset{A}{\vdash} (q, u\#av) \quad ak \quad (q, 0) \in \delta(p, a) \\
(p, ua\#v) &\underset{A}{\vdash} (q, u\#av) \quad ak \quad (q, -1) \in \delta(p, a)
\end{aligned}$$

Definícia 6.2.4. *Jazyk akceptovaný 2DKA* A *definujeme*

$$L(A) = \{w \mid (q_0, ¢\#w\$) \overset{*}{\underset{A}{\vdash}} (q_f, ¢w\#\$),\ q_f \in F\}$$

Veta 6.2.1. *Ku každému 2DKA* A *existuje konečný nedeterministický automat* A' *taký, že* $L(A) = L(A')$. *(Dvojsmerné konečné automaty nie sú silnejšie ako jednosmerné)*

Dôkaz. Všimnime si, ako môže potenciálne konečný automat A využiť svoju nedeterministickosť. Koľkokrát sa môže vrátiť na tú istú pozíciu na páske? Samozrejme, že potenciálne nekonečne mnoho, ale pokiaľ sa nejedná o nekonečný, a teda neakceptujúci výpočet, tak len toľkokrát, koľko existuje stavov automatu A. Preto možno ku každej pozícii na páske priradiť konečnú postupnosť stavov (dlhú max. $|K|$). Túto postupnosť nazveme prechodovou postupnosťou.

Nedeterministický konečný automat A' si ku každému políčku tipne prechodovú postupnosť a pre každé dve prechodové postupnosti, ktoré prislúchajú susedným políčkam overí, či môžu na seba navädzovať.

[2] Poučenie, ktoré z toho plynie je, že i človek by možno mal každý text čítať raz, ale poriadne

Tipnutú prechodovú postupnosť si možno pamätať v stave automatu A',hoci pre počet stavov tohto automatu bude platiť $|K'| > |K|^{|K|}$.

Náväznosť prechodových postupností zaručí, že pôvodný dvojsmerný automat mohol pri výpočte produkovať tipnuté prechodové postupnosti. Na záver musí automat A' zistiť, či prechodová postupnosť prislúchajuca poslednému políčku slova obsahuje stav, ktorý bol akceptujúcim stavom automatu A. Ak áno, automat A' akceptuje. Toto sa dá jednoducho zabezpečiť tak, že ak $K' = (K \cup \{\epsilon\})^{|K|} \times K_n$, teda že stavy automatu A' nesú informáciu o prechodových postupnostiach a ďalšiu kontrolnú informáciu, potom $F' = K' - ((K - F^C) \cup \{\epsilon\})^{|K|} \times K_n$ (kde F^C je množina neakceptujúcich stavov automatu A).

Na čitateľa nechávame dôkaz tvrdenia, že pre danú dvojicu prechodových postupností a symbolov na tých dvoch susedných políčkach pásky, ktorým tieto prechodové postupnosti prislúchajú, je odpoveď na otázku, či na seba naväzujú bez ohľadu na zvyšok pásky, jednoznačná. □

Kapitola 7

Prekladače

7.1 A-prekladač

V tejto časti sa zoznámime so zariadením, ktoré umožní transformovať jazyky. Teda už nepôjde len o automat, ktorý nečinne pozerá pásku a potom sa ocitne v nejakom stave, ale budeme mať zariadenie, ktoré bude schopné výstupu na pásku druhú.

Definícia 7.1.1. *A-prekladač (konečne stavový prekladač s akceptačnými*[1] *stavmi) je 6-tica* $M = (K, \Sigma_1, \Sigma_2, H, q_0, F)$*, kde* K *je konečná množina stavov,* Σ_1 *je vstupná abeceda,* Σ_2 *je výstupná abeceda,* q_0 *je počiatočný stav automatu,* F *je množina akceptačných stavov a* $H \subseteq_{kon} K \times \Sigma_1^* \times \Sigma_2^* \times K$

Vidíme, že tento automat bude mať okrem vstupnej pásky, aj pásku výstupnú. Automat na základe vstupu dá nejaký výstup. A-prekladač zadefinujeme ďalej rovnakým spôsobom, ako sme boli zvyknutí pri automatoch. Ukážeme potom iný spôsob definície, pomocou homomorfizmov, inverzných homomorfizmov a prienikov s regulárnymi jazykmi.

Definícia 7.1.2. *Konfiguráciou a-prekladača nazývame usporiadanú trojicu* $(q, u, v) \in K \times \Sigma_1^* \times \Sigma_2^*$ *(kde* q *znamená momentálny stav riadiacej jednotky,* u *je zvyšok vstupu a* v *je doteraz vygenerovaný výstup).*

Definícia 7.1.3. *Krokom výpočtu a-prekladača nazývame reláciu* $\vdash$ *definovanú na konfiguráciách takto:*

$$(q, xu, v) \vdash (p, u, vy) \qquad , \; ak \; (q, x, y, p) \in H \tag{7.1}$$

Definícia 7.1.4. *Obrazom jazyka* L *a-prekladačom* M *nazývame množinu*

$$M(L) = \{v \in \Sigma_2^* \mid \exists w \in L; (q_0, w, \varepsilon) \overset{*}{\vdash} (q, \varepsilon, v); q \in F\} \tag{7.2}$$

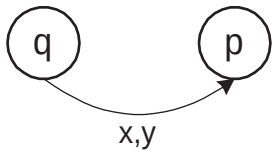
Teda výstup[2] a-prekladača je určený vstupom w. Tiež si môžeme všimnúť, že čítanie vstupu je podobné ako pri automate s pružnou hlavou. A-prekladač dokáže naraz prečítať celý úsek slova. Teraz prichádza ten sľubovaný okamih, keď si ukážeme inú definíciu výpočtu a obrazu a-prekladača, pomocou už spomínaných homomorfizmov, ktoré nám uľahčia niektoré tvrdenia o a-prekladačoch. Čitateľ môže pouvažovať, prečo sú dané definície ekvivalentné.

Krok výpočtu a-prekladača znázorňujeme na diagrame takto (prečíta slovo x a na výstup dá slovo y):

[1] preto v názve **a**-prekladača to **a**

[2] voľne povedané, výstup je to, čo automat prežuje z L

!ht



Poznámka 7.1.1. V nasledujúcich úvahách budeme niekedy chápať prvok $h \in H$ ako nejaký symbol (písmeno) abecedy H. Veríme, že to čitateľovi nebude robiť problémy, veď s podobným príkladom sme sa už stretli napríklad v dôkaze 3.3.2, kde sme stotožňovali stavy automatu s podmnožinami nejakej množiny stavov.

Definícia 7.1.5. *Homomorfizmy pr_i, $i \in \{1,2,3,4\}$ z množiny H^* také, že*

$$pr_i((x_1,x_2,x_3,x_4)) = x_i \tag{7.3}$$

nazývame projekcie. Jednotlivé projekcie majú svoje obrazy postupne v množinách $K^*, \Sigma_1^*, \Sigma_2^*, K^*$.

Definícia 7.1.6. *Výpočtom a-prekladača M nazývame postupnosť $h_1, h_2, h_3, \ldots, h_k$ štvoríc z H^3 takú, že platí*

1. $pr_1(h_1) = q_0$
2. $pr_4(h_k) \in F$
3. $pr_4(h_i) = pr_1(h_{i+1})$ *pre všetky* $i \in \{1,2,\ldots,k-1\}$

Symbolom $\prod_M$ označujeme množinu všetkých výpočtov a-prekladača M [4].

Definícia 7.1.7. *Zobrazením a-prekladača M nazývame množinu*

$$M(L) = pr_3(pr_2^{-1}(L) \cap \prod_M) \tag{7.4}$$

Na prvý pohľad sa môže zdať, že táto definícia je hora nič nehovoriacich matematických symbolov, ale keď sa na to pozrieme bližšie, uvidíme, že je to presne to isté, ako v pôvodnej definícii. Množina $pr_2^{-1}(L)$ obsahuje všetky možné postupnosti prvkov $h \in H$ také, že druhá projekcia tejto postupnosti (slova z H^*) je slovo z L. Pozor! Tieto postupnosti nemusia byť ešte výpočtom. Potom $pr_2^{-1} \cap \prod_M$ je zrejme množina legálnych výpočtov na slovách z L. Na tieto výpočty aplikujeme tretiu projekciu, ktorou dostaneme množinu výstupných slov a-prekladača.

Objavuje sa nám otázka, či je trieda bezkontextových jazykov uzavretá na zobrazenie a-prekladača. Vieme, že $\mathcal{L}_{CF}$ je uzavretá na homomorfizmus aj na inverzný homomorfizmus, čo sú zobrazenia pr_3 a pr_2^{-1}. Ak by sme vedeli, že $\prod_M$ je regulárny jazyk, mohli by sme použiť vetu o uzavretosti triedy $\mathcal{L}_{CF}$ s triedou R vzhľadom na $\cap$, teda $M(L)$ by bol bezkontextový jazyk.

Lema 7.1.1. $\prod_M \in R$ *pre každý a-prekladač* M.

Dôkaz. M je automat, zostrojme preto konečný automat A, ktorý bude akceptovať slová z $\prod_M$ [5] takto:

$$A = (K, H, \delta, q_0, F)$$

[3] ak berieme prvok množiny H ako symbol abecedy, potom táto postupnosť predstavuje slovo z H^+

[4] alebo množinu všetkých slov, ak postupnosť prvkov $h \in H$ považujeme za slovo

[5] uvedomme si, že tento jazyk je jazykom nad abecedou H

kde $\delta(q,(q,x,y,p)) = p$.

Zrejme platí $L(A) = \prod_M$. □

Veta 7.1.1. *Trieda bezkontextových jazykov je uzavretá na zobrazenie a-prekladačom.*

Dôkaz. Vyplýva z definície $M(L)$, predchádzajúcej lemy a uzavretosti $\mathcal{L}_{CF}$ na homomorfizmus a inverzný homomorfizmus. □

Veta 7.1.2. *Trieda regulárnych jazykov je uzavretá na zobrazenie a-prekladačom.*

Dôkaz. Dá sa použiť analogický dôkaz ako v predchádzajucom prípade. □

Užitočnosť tohto automatu je pomerne veľká. A-prekladač sa dá použiť na dokazovnaie, že niečo je z $\mathcal{L}_{CF}$(resp. R), alebo nie je. Býva prirodzenejšie zostrojiť nejakú mašinku, ako dokazovať bezkontextovosť pomocou prienikov, homomorfizmov, alebo iných vlastností.

1. Chceme ukázať, že nejaký jazyk L patrí do $\mathcal{L}_{CF}$. Tak si zoberieme nejaký jazyk $L' \in \mathcal{L}_{CF}$ a hodíme ho na vstup a-prekladača. Ak nám ten vyhodí ako výstup jazyk L, tak na základe vety o uzavretosti, bude aj $L \in \mathcal{L}_{CF}$.

2. Chceme ukázať, že nejaký jazyk L nepatrí do $\mathcal{L}_{CF}$. Snažíme sa zostrojiť a-prekladač tak, aby jeho výstup po prežutí jazyka L nebol z $\mathcal{L}_{CF}$. Tak dostaneme, že ani pôvodný jazyk L nemohol byť z $\mathcal{L}_{CF}$ (ak by bol, tak z vety o uzavretosti na obraz a-prekladača by musel byť aj výstup z $\mathcal{L}_{CF}$).

A-prekladačom napríklad môžeme ukázať, že Pascal nie je bezkontextový jazyk[6]. Všimnime si jednu vlastnosť Pascalovských procedúr. Každé volanie nejakej procedúry musí obsahovať rovnaký počet parametrov ako pri jej definovaní. Teda kompilátor musí kontrolovať, či je tam správny počet argumentov. Náš a-prekladač narazí na prvú definíciu nejakej procedúry a za každú čiarku, ktorá oddeluje argumenty, dá na výstup symbol a. Potom čaká na ďalšie volanie tej istej procedúry a za každú čiarku dá tentoraz b. Tak isto pridáme aj symboly c. Ostatné veci ignorujeme (pošleme výstup do ε). Takto získame slová tvaru $a^n b^n c^n$ (a len tie), ktoré, ako vieme, netvoria bezkontextový jazyk.

Ohľadne definícií a modifikácií a-prekladača sa vynárajú rozmanité otázky:

1. Ako prerobiť a-prekladač tak, aby $H \subseteq K \times (\Sigma_1 \cup \{\varepsilon\}) \times \Sigma_2^* \times K$ a či sa o niečo ochudobní takýto a-prekladač.

2. Ako prerobiť a-prekladač tak, aby $H \subseteq K \times \Sigma_1^* \times (\Sigma_2 \cup \{\varepsilon\}) \times K$ a či sa o niečo ochudobní takýto a-prekladač.

3. Ako prerobiť a-prekladač tak, aby $H \subseteq K \times (\Sigma_1 \cup \{\varepsilon\}) \times (\Sigma_2 \cup \{\varepsilon\}) \times K$ a či sa o niečo ochudobní takýto a-prekladač.

4. Ako prerobiť a-prekladač tak, aby jeho výstup bol taký istý, ako obraz daného homomorfizmu (ľubovoľného).

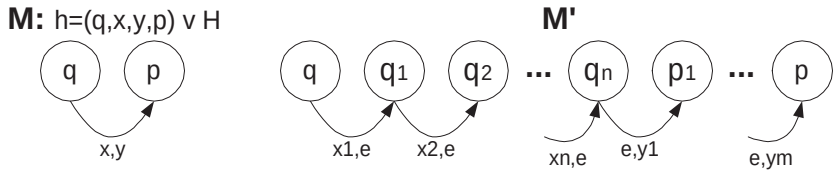
5. Ako zoslabíme a-prekladač, ak mu odstránime akceptačné stavy.

6. Aký silný nástroj dostaneme, ak výstup pridáme zásobníkovému automatu.

[6] pričom pod Pascalom rozumieme množinu všetkých dobre napísaných programov (skompilovateľných), kde 1 slovo = 1 program

Úvahy 1., 2. a 3. sa dajú riešiť podobným spôsobom. Naznačíme tento spôsob na príklade 3:

Majme v a-prekladači M prechod zo stavu q do stavu p, pričom M prečíta slovo $x = x_1x_2 \dots x_n \in \Sigma_1^*$ a na výstup dá slovo $y = y_1y_2 \dots y_m \in \Sigma_2^*$. Na úpravu tohto prechodu zavedieme nové stavy $q_1, q_2, \dots, q_n, p_1, \dots, p_{m-1}, p_m = p$. Stavy $q_1, \dots, q_n$ využijeme



na prečítanie slova x po písmenku, pričom ako výstup bude ε. Takže budeme mať štvorice $(q_j, x_{j+1}, \varepsilon, q_{j+1})$, pričom $j \in \{0, 1, 2, \dots, n-1\}$ a stav q_0 stotožňujeme so stavom q. Podobne, stavy $p_1, \dots, p_m$ využijeme na výpis slova y na pásku. Čítať budeme na ε a výpis bude po jednom písmenku. Príslušné štvorice si už čitateľ istotne vie predstaviť sám.

Poznámka 7.1.2. S využitím normálneho tvaru a-prekladača, ktorý sme spomenuli v príklade 3 sa dá dokázať, že zobrazenia a-prekladačmi sú uzavreté na skladanie. Toto tvrdenie ale presahuje rámec nášho textu, preto ho tu nebudeme uvádzať.

Kapitola 8

Deterministické bezkontextové jazyky

Ako už názov napovedá, pôjde o triedu jazykov rozpoznávanú deterministickými zásobníkovými automatmi. Kým v prípade konečných automatov nedeterminizmus nepridal žiadnu silu, tu uvidíme, že nederminizmus akúsi silu pridá.

8.1 Základné pojmy

Definícia 8.1.1. *Deterministickým zásobníkovým automatom A nazveme 7-icu $(K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde K je konečná množina stavov, Σ je abeceda vstupných symbolov, Γ je pracovná abeceda zásobníkových symbolov,*

$$\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \ \rightarrow \ (K \times \Gamma^*) \cup \{\emptyset\}^1$$

je funkcia spĺňajuca

$$(\forall q \in K)(\forall z \in \Gamma)\ (\delta(q, \varepsilon, z) \neq \emptyset \ \Rightarrow \ (\forall a \in \Sigma)\ \delta(q, a, \gamma) = \emptyset) \tag{8.1}$$

$Z_0 \in \Gamma$ je počiatočný symbol zásobníka a $F \subseteq K$ je množina akceptujúcich stavov.

Prechodová funkcia nám teda v každom stave určuje, ako ďalej. Buď je pre istý vrchný zásobníkový symbol a stav definovaná buď tak, že čítať pásku nemá, alebo tak, že čítať pásku má(ale nemôže si vybrať).

Definícia 8.1.2. *Konfiguráciou deterministického zásobníkového automatu A rozumieme trojicu $(q, w, \gamma) \ \in K \times \Sigma^* \times \Gamma^*$*

Definícia 8.1.3. *Krokom výpočtu deterministického zásobníkového automatu A nazveme reláciu $\underset{A}{\vdash}$ definovanú nasledovne:*

$$(q_z, aw_1, \gamma_1 g) \underset{A}{\vdash} (q_k, w_1\gamma_1 h) \quad \underset{def}{\Longleftrightarrow} \quad \delta(q_z, a, g) = (q_k, h) \quad kde\ a \in \Sigma \cup \{\varepsilon\} \tag{8.2}$$

Definícia 8.1.4. *Jazykom rozpoznávaným deterministickým zásobnikovým automatom A nazývame množinu*

$$L(A) = \{w \mid (q_0, w, Z_0) \overset{*}{\underset{A}{\vdash}} (q, \varepsilon, \gamma) \wedge q \in F\}$$

Triedu deterministických zásobníkových automatov označujeme $\mathcal{L}_{DCF}$.

[1] δ-funkcia teda môže vrátiť prázdnu množinu ako identifikáciu zaseknutia sa

8.2 Rozdiel v akceptácii prázdnou pamäťou a koncovým stavom

Na tomto mieste nebudeme definovať det. zás. automat rozpoznávajuci prázdnou pamäťou, lebo predpokladáme, že čitateľ si tento pojem sám ľahko definuje. Triedu jazykov rozpoznávaných det. zás. automatmi prázdnou pamäťou budeme označovať $\mathcal{L}_{DPCF}$. V nasledujúcom texte ukážeme, že $\mathcal{L}_{DPCF} \subsetneqq \mathcal{L}_{DCF}$.

Lema 8.2.1. *Každý deterministický zásobníkovým automat rozpoznávajuci prázdnou pamäťou možno simulovať deterministickým zásobníkovým automatom rozpoznávajucim akceptujúcim stavom.*

Dôkaz. Dôkaz je analógiou dôkazu 5.2.2 □

Veta 8.2.1.

$$\mathcal{L}_{DPCF} \subsetneqq \mathcal{L}_{DCF}$$

Dôkaz. To, že ide o podmnožinu je jasné z predchádzajucej lemy. Zostáva dokázať, že ide o pravú podmnožinu. Toto ukážeme na príklade. Uvažujme jazyk $L = \{w \in \{0,1\}^* \mid \#_0 w = \#_1 w\}$.

Ľahko by sme zostrojili deterministický zásobníkový automat rozpoznávajuci tento jazyk akceptujúcim stavom. Pozor! Nepôjde presne o kópiu nedeterministického zásobníkového automatu. V tomto prípade musíme zaviesť až 5 zásobnikových symbolov: $Z_0, 0, 1, 0', 1'$. V prípade, že vidíme na vrchu zásobníka Z_0, pridáme podľa vst. symbolu na páske do zásobníka symbol $0'$, resp. $1'$. Ak vidíme na vrchu zásobníka symbol $0'$, resp. $1'$ a zo vstupu čítame 1, resp. 0, prejdeme do akceptujúceho stavu. Inokedy do akceptujúceho stavu neprechádzame (iba na začiatku).

Avšak, nijako sa nám nepodarí zostrojiť deterministický zás. automat rozpoznávajuci L prázdnou pamäťou. Keby tento existoval, potom by musel akceptovať slovo 01. To značí, že po prečítaní 01 sa vyprázdni zásobník. Ale keby čítal zo vstupu slovo $0101 \in L$, zásobník sa mu vyprázdni ešte skôr, než dočíta celé slovo a ďalší výpočet teda nie je definovaný (automat sa zasekne), a teda takéto slovo neakceptuje. Problém je teda v tom, že det. zás. automat nemôže rozpoznávať(akceptovať) prázdnou pamäťou slová, ktorých vlastný prefix má byť tiež rozpoznaný a akceptovaný. □

8.3 Vlastnosti triedy $\mathcal{L}_{DCF}$

Niektoré vlastnosti tejto triedy jazykov sú rovnaké ako v prípade triedy $\mathcal{L}_{CF}$, v niektorých sa však budú líšiť. Práve v tom, že tieto triedy nemajú rovnaké vlastnosti dokazuje, že nie sú totožné. Keďže triviálne platí $\mathcal{L}_{DCF} \subseteq \mathcal{L}_{CF}$, týmto dokážeme, že platí

$$\mathcal{L}_{DCF} \subsetneqq \mathcal{L}_{CF}$$

Veta 8.3.1. *$\mathcal{L}_{DCF}$ nie je uzavretá na prienik.*

Dôkaz. Poľahky možno zostrojiť deterministický zás. automat rozpoznávajuci jazyk $L_1 = \{a^n b^n c^i \mid i, n \geq 1\}$, ako aj $L_2 = \{a^i b^n c^n \mid i, n \geq 1\}$. Ich prienik však nepatrí ani len do $\mathcal{L}_{CF}$, a preto ani do $\mathcal{L}_{DCF}$. □

Veta 8.3.2. *$\mathcal{L}_{DCF}$ je uzavretá na prienik s $\mathcal{R}$.*

Dôkaz. Dokazuje sa analogicky ako v prípade nedeterministických zásobníkových automatov. □

Nasledovná vlastnosť je tá, ktorá implikuje rôznosť tried bezkontextových a deterministicky bezkontextových jazykov. Ideme ukázať, že trieda $\mathcal{L}_{DCF}$ je uzavretá na komplement. [2]

Lema 8.3.1. *Ku každému deterministickému zásobníkovému automatu A existuje automat A' taký, že $L(A) = L(A')$ a A' vždy dočíta vstupné slovo.*

Dôkaz. Princíp práce automatu A' bude spočívať v tom, že bude mať akési **countery (počítadlá)**, ktoré mu budú počítať, koľko ε-ových prechodov robí a aký veľký už má zásobník. Zamyslime sa najprv nad tým, čo sa stane, keď pôvodný automat nedočíta celé vstupné slovo. Zrejme sú dve možnosti:

1. Automat sa zasekne, keďže má prázdny zásobník.
2. Automat sa zacyklí, t.j. donekonečna vykonáva ε-ové prechody. Tu sú opäť dve možnosti.
   (a) Zásobník neobmedzene rastie.
   (b) Výška zásobníka nikdy neprekročí istú hranicu.

Všetky tieto možnosti teraz musíme vyriešiť.

1. Na dno zásobníka vložíme špeciálny symbol, a keď ho opäť uvidíme (čo sa stane práve vtedy keď pôvodný automat sa zasekne), na ε-ové prejdeme do špeciálneho neakceptačného stavu, z ktorého zas definujeme prechody na symboly Σ opäť do toho istého stavu, aby nový automat slovo dočítal a zostal v neakceptačnom stave.
2. Označme $t = |\Gamma|$, $s = |K|$, $r = \max\{|w| \mid (q', w) = \delta(q, a, Z)\}$.
   (a) Našťastie, ľahko možno túto situáciu objaviť. Totiž, platí tvrdenie

   Ak uvažujeme len ε-ové kroky, potom výška zásobníka narastie na hodnotu väčšiu ako rst
   práve vtedy keď
   zásobník neobmedzene rastie

   Dôkaz implikácie $\Leftarrow$ je triviálny, pozrime sa na implikáciu $\Rightarrow$. Ak je výška zásobníka rst, potom automat počas plnenia zásobníka týmito symbolmi prešiel aspoň st konfiguráciami takými, že od doby tejto konfigurácie sa nikdy nestalo, aby hĺbka zásobníka klesla pod úroveň, ktorú mala v dobe tejto konfigurácie. [3] Potom ale z Dirichletovho princípu plynie, že medzi týmito st konfiguráciami existovali také konfigurácie k_1, k_2, ktoré sa zhodovali vrchným symbolom zásobníka a stavom automatu. Z predchádzajuceho vieme, že hĺbka zásobníka počas k_2 je väčšia ako v čase k_1 a že sa medzitým

[2] Čitateľ by si mal hneď na základe deMorganových zákonov uvedomiť, že to znamená, že nie je uzavretá na operáciu zjednotenia, čo je tiež vlastnosť, ktorou sa líši od triedy bezkontextových jazykov

[3] Pre čitateľov dôsledných, no lenivých ponúkame plné znenie lemy, o ktorú sa tu opierame aj s dôkazom.

Lema 8.3.2. *Nech $a = a_1, a_2, a_3, ..., a_n$, $a_1 = 0$, $a_n > rk$ je postupnosť prirodzených čísel taká, že platí $|a_i - a_{i+1}| < r$. Potom existuje aspoň k členov postupnosti $a_{i_1}, a_{i_2}, ..., a_{i_k}$ takých, že $(\forall j > i)$ $(a_i < a_j)$.*

Dôkaz. Uvedieme konštrukčný dôkaz indukciou. Týchto k členov $a_{i_1}, a_{i_2}, ..., a_{i_k}$ nájdeme takto:
$i_k = \max\{j \leq n \mid a_j \leq rk - r\}$,
$i_{k-1} = \max\{j \leq i_k - 1 \mid a_j \leq rk - 2r\}$,
...
Je jasné, že keby napríklad a_{i_k} nespĺňala požadovanú vlastnosť, potom by existovalo nejaké m také, že $i_k < m < n$ a $a_m < a_{i_k} < a_n$, čo je ale spor s maximalitou i_k. □

robili výhradne ε-ové kroky. Ale keďže sa jedná a deterministický automat, je jasné, akým smerom sa ďalej bude výpočet uberať. Ďalej sa bude automat na ε-ové kroky správať rovnako ako predtým - bude rásť zásobník a z času načas sa objaví opäť konfigurácia s rovnakým stavom automatu a rovnakým vrchným symbolom zásobníka ako k_1, resp. k_2.

(b) Čo ale robiť v prípade, že sa automat zacyklí, no hĺbka zásobníka nikdy nepresiahne magickú hodnotu rst? Našťastie, všetky abecedy sú konečné, takže rôznych konfigurácii s rovnakým zvyškom slova a takto obmedzeným zásobníkom je $x = s.(t+1)^{rst}$. Ak teda zistíme, že sme urobili po sebe viac než x ε-ových prechodov po sebe, môžeme si byť istí, že chodíme do kruhu a determinizmus nám nedáva šancu z neho vyjsť.

To značí, že automat A stačí rozšíriť o dva countery, a to rozšírením množiny stavov na $K' = K \times \{0, 1, ..., rst\} \times \{0, 1, ..., s(t+1)^{rst}\} \cup \{q_s\}$. Prvý counter bude počítať ako sa zväčšila hĺbka zásobníka odkedy robíme ε-ové kroky, druhý bude počítať počet týchto ε-ových krokov. Vždy, keď prečítame vstupný symbol z pásky, oba countery vynulujeme. Druhý counter vynulujeme i v prípade, že hodnota prvého klesne na nulu, t.j. keď sa hĺbka zásobníka zmenší pod pôvodnú hodnotu, aká bola na začiatku ε-ových krokov. Po ε-ovom prechode druhý counter zväčšíme o jedna a prvý zmeníme o hodnotu, o ktorú sa zmení hĺbka zásobníka. Ak by jeden z counterov mal prekročiť povolenú hodnotu, prejdeme do špeciálneho neakceptačného stavu q_s, v ktorom čítame vstupné symboly z pásky, aby sme splnili náš účel dočítať vstupné slovo a zostať v neakceptačno stave.

□

Veta 8.3.3. *Nech A je deterministický zásobníkový automat. Potom existuje taký deterministický zásobníkový automat A', že $L(A) = L(A')^{C}$*[4].

Dôkaz. Na základe predošlej lemy možno bez ujmy na všeobecnosti predpokladať, že A vždy dočíta vstupné slovo. Kvôli tomu, že deterministický zásobníkový automat môže robiť ε-ové prechody, môže sa stať, že deterministický zásobníkový automat po prečítaní vstupného slova prejde do akceptačného stavu (akceptuje ho), ale potom urobí niekoľko ε-ových prechodov, po ktorých prejde do neakceptujúceho stavu. To znamená, že keby sme A' skonštruovali tak, že by sa od A líšil len tým, že $F' = K - F$, potom by automat A' mohol akceptovať aj tie slová, ktoré akceptuje aj automat A, čo nechceme. Preto musí nastúpiť akýsi fígeľ.

Množina stavov nového automatu $K' = K \times \{p, n, f\}$. V druhej zložke si budeme ukladať informáciu o tom, či sme od posledného prečítania vstupného symbolu boli v nejakom akceptujúcom stave alebo nie. Hodnota p bude značiť, že áno, n nie a f znamená, že už nemôžme robiť ani ε-ové prechody. Konkrétne teda,

1. Nech $\delta(q, a, Z) = (q', \gamma)$. Potom

$$\delta'((q, f), a, Z) = \delta'((q, p), a, Z) =_{df} \begin{cases} ((q', p), \gamma) & \text{ak } q' \in F \\ ((q', n), \gamma) & \text{ak } q' \notin F \end{cases}$$

$$\delta'((q, n), \varepsilon, Z) =_{df} ((q', f), \gamma)$$

2. Nech $\delta(q, \varepsilon, Z) = (q', \gamma)$. Potom

$$\delta'((q, p), \varepsilon, Z) =_{df} ((q', p), \gamma)$$

[4] Pre zábudlivých čitateľov dodávame, že operácia komplementu je definovaná vzhľadom na jazyk Σ^* určený abecedou vstupných symbolov automatu A

$$\delta'((q,m),\varepsilon,Z) =_{df} \begin{cases} ((q',p),\gamma) & \text{ak } q' \in F \\ ((q',m),\gamma) & \text{ak } q' \notin F \end{cases} \quad \text{pre } m = n, f$$

Množina akceptačných stavov bude $F' = K \times \{f\}$, t.j. akceptujeme tie slová, na ktoré sa pôvodný automat od posledného čítania vstupného symbolu iste nedostal do žiadneho akceptujúceho stavu, a to ani s využitím ε-ových prechodov.

□

Kapitola 9

Lineárne ohraničené automaty

9.1 Definície

Tento krát posunieme latku opäť o niečo vyššie. Postúpime v Chomského hierarchii a zadefinujeme si automaty, ktoré dokážu rozpoznávať kontextové jazyky. Dostatočne silným nástrojom sa ukáže byť, ak povolíme hlave čítať aj zapisovať na pásku, pričom množstvo pásky zostane ohraničené vstupným slovom.

Definícia 9.1.1. *Nedeterministickým lineárne ohraničeným automatom nazývame 6-ticu* $(K, \Sigma, \Gamma, \delta, q_0, F)$*, kde* K *je konečná množina stavov,* Σ *je abeceda vstupných symbolov,* Γ *je abeceda pracovných symbolov* $(\Sigma \subseteq \Gamma)$*,* $q_0 \in K$ *je počiatočný stav,* $F \subseteq K$ *je množina akceptačných stavov a* $\delta : K \times (\Gamma \cup \{¢, \$\}) \to 2^{K \times (\Gamma \cup \{¢,\$\}) \times \{-1,0,1\}}$

Definícia 9.1.2. *Konfigurácia LBA je*

1. *trojica z* $K \times ¢\Gamma^*\$ \times \mathbb{N}$*, pričom* $(q, ¢a_1 \dots a_n\$, d)$ *znamená, že hlava je nastavená na* a_d*,* $d \in \langle 0, \dots, n+1 \rangle$
2. *dvojica z* $K \times (\restriction ¢\Gamma^*\$ \cup ¢\Gamma^* \restriction \Gamma^*\$)$
3. *slovo z* $K¢\Gamma^*\$ \cup ¢\Gamma^*K\Gamma^*\$$*,* $(K \cap \Gamma = \emptyset)$

Definícia 9.1.3. *Krok výpočtu LBA je relácia* $\vdash$ *na konfiguráciách definovaná nasledovne:*

1. $(q, ¢a_1 \dots a_n\$, k) \vdash (p, ¢b_1 \dots b_n\$, l) \Leftrightarrow (p, c, d) \ni \delta(q, a_k), l = k + d, b_k = c, b_i = a_i \forall i \neq k$
2. $(q, u \restriction av\$) \vdash (p, ub \restriction v) \Leftrightarrow (p, b) \ni \delta(q, a, 1)$
   $(q, ua \restriction bv) \vdash (p, u \restriction acv) \Leftrightarrow (p, c) \ni \delta(q, b, -1)$
   $(q, u \restriction av) \vdash (p, u \restriction bv) \Leftrightarrow (p, b) \ni (q, a, 0)$
3. $¢uqav\$ \vdash ¢upbv\$ \Leftrightarrow (p, b, 0) \ni \delta(q, a)$
   $¢uqav\$ \vdash ¢ubpv\$ \Leftrightarrow (p, b, 1) \ni \delta(q, a)$
   $¢ucqav\$ \vdash ¢upcbv\$ \Leftrightarrow (p, b, -1) \ni \delta(q, a)$

Definícia 9.1.4. *Jazyk akceptovaný LBA je* $L(A) = \{w \in \Sigma^* \mid (q_0, ¢ \restriction w\$) \vdash^* (q, ¢u \restriction \$); u \in \Gamma^*, q \in F\}$

Definícia 9.1.5. *Deterministický lineárne ohraničený automat je taký LBA, kde* $\#(\delta_k(q, a)) \leq 1; \forall q \in K, a \in \Gamma$

Príklad 9.1.1. Zostrojte LBA A, ktorý bude akceptovať jazyk $L = \{a^n b^n c^n \mid n \geq 1\}$.

Riešenie $A = (K, \Sigma, \Gamma, \delta, q_0, F)$, kde $K = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b, c, a^*, b^*, c^*\}$, $F = \{q_4\}$. Prechodovú funkciu definujeme nasledovne:

$\delta(q_0, a)$	=	$(q_1, a^*, 1)$	označíme si ďalšie písmenko, ktoré ideme kontrolovať
$\delta(q_1, a)$	=	$(q_1, a, 1)$	prejdeme cez všetky nasledujúce neoznačené a-čka
$\delta(q_1, b^*)$	=	$(q_1, b^*, 1)$	podobne prejdeme cez všetky označené bé-čka
$\delta(q_1, b)$	=	$(q_2, b^*, 1)$	nájdeme prvé neoznačené b a označíme ho
$\delta(q_2, b)$	=	$(q_2, b, 1)$	podobne prechádzame cez všetky neoznačené b-čka
$\delta(q_2, c^*)$	=	$(q_2, c^*, 1)$	prejdeme aj cez všetky označené c-čka
$\delta(q_2, c)$	=	$(q_3, c^*, -1)$	označíme prvé neoznačené c – teraz sa chceme vrátiť
$\delta(q_3, z)$	=	$(q_3, z, -1);\ z \in \{c^*, b, b^*, a\}$	vraciame sa na prvé neoznačené a
$\delta(q_3, a^*)$	=	$(q_0, a^*, 1)$	označíme ho a pokračujem opäť v stave q_0
$\delta(q_0, b^*)$	=	$(q_4, b^*, 1)$	zastavíme sa, ak už nebudeme mať neoznačené a
$\delta(q_4, x)$	=	$(q_4, x, 1);\ x \in \{b^*, c^*\}$	ešte musíme prejsť na koniec pásky a dostať sa pred \$ – tak bola definovaná akceptujúca konfigurácia

Dôkaz $L(A) = L$:
$\supseteq$: ľahší smer indukcie – tak sme robili konštrukciu – urobí sa indukciou vzhľadom na dĺžku slova w
$\subseteq$: Sú dve možnosti dôkazu:

1. Zoberieme si ľubovolný akceptujúci výpočet a začneme usudzovať, ako musí vyzerať : musel skončiť v stave q_4 a hlava musela skončiť na znaku \$, ale automat sa mohol dostať do stavu q_4 jedine cez stav q_0 a muselo sa čítať písmenko b ... takto si rozparcelujeme výpočet na úseky, z ktorých už jasne vyplýva ako muselo vyzerať slovo, ktoré automat akceptoval.

2. [1]Dokážeme pre všetky slová, ktoré nepatria do jazyka L, že automat A sa pri výpočte zasekne. Ako vyzerajú slová zlého tvaru? Majú buď rôzny počet písmeniek a,b,c alebo, ak aj počty sedia, slovo má zlý tvar (nemá tvar $a^n b^n c^n$).
   Čo sa stane ak počty nesedia :

   (a) a-čok je viac ako niektorých iných a vtedy sa to zasekne v stave q_1 – ak sa už minuli b-čka (nedostaneme sa do stavu q_2 a to spôsobí zaseknutie) alebo sa minuli c-čka a tu podobne nastane zaseknutie v stave q_2

   (b) a-čok je menej ako niektorých iných písmen a vtedy sa to zasekne v stave q_4, keďže tam budú existovať aj neoznačené b (resp. c), cez ktoré v stave q_4 automat A nevie prejsť a keďže sa nedostane na \$, automat nemôže akceptovať dané slovo.

♠

9.2 Ekvivalentnosť kontextových gramatík a lineárne ohraničených automatov.

Veta 9.2.1. *K ľubovoľnej kontextovej gramatike G existuje LBA A taký, $L(A) = L(G)$.*

[1]tento spôsob dokazovania sa všeobecne odporúča iba v prípade, ak je dokazovaný jazyk ľahkého tvaru a sme si istí, že nezabudneme na nijaké slová z jazyka

Dôkaz. Nech $G = (N, T, P, \sigma)$. Potom LBA $A = (K, \Sigma, \Gamma, \delta, q, F)$ bude pracovať tak, že bude simulovať odvodenie v gramatike G odzadu, pričom

$$K = \{q, q_k\} \cup \{q_{[a_1,\ldots,a_n;b_1,\ldots,b_m]}, q'_{[a_1,\ldots,a_n;b_1,\ldots,b_m]} \mid (a_1 \ldots a_n \to b_1 \ldots b_m) \in P\}$$

$$\Sigma = T,\ \Gamma = T \cup N,\ F = \{q_k\}$$

Prácu automatu rozdelíme na etapy :

1. Uhádne, ktoré pravidlo bolo použité ako posledné pri odvodení vetnej formy, ktorá je v danom momente na páske:

$$\delta(q, a) \ = \ \{(q_{[a_1,\ldots,a_n;b_1,\ldots,b_m]}, a, 0) \mid (a_1 \ldots a_n \to b_1 \ldots b_m) \in P\} \text{ pre } a \in \Gamma$$

2. Najskôr hľadá a potom uhádne správny výskyt pravej strany pravidla na páske:

$$\delta(q_{[u,v]}, a) \ = \ \{(q_{[u,v]}, a, 1), (q'_{[u,v]}, a, 0)\} \text{ pre } a \in \Gamma,\ q_{[u,v]} \in K$$

3. Kontroluje, či správne uhádol pravidlo. Ak áno, tak prepíše pravý výskyt pravidla ľavou stranou daného pravidla, ak nie, tak sa zasekne a nepokračuje ďalej vo výpočte.

$$\delta(q'_{[a\,u,bv]}, a) \ = \ \{(q'_{[u,v]}, b, 1)\} \ \text{ pre } u, v \in \Gamma^*; a \in \Gamma; b \in \Gamma \cup \{\varepsilon\}$$

   Epsilon je tu preto, že je potrebné čakať na stav $q_{[\varepsilon,\varepsilon]}$, ale treba počítať aj so stavmi $q_{[u,\varepsilon]}$ pre $u \in \Gamma^*$

   Ak v stave $q'_{[a\,u,bv]}$ nenasleduje na páske a, automat sa zle rozhodol, čiže nepokračuje. Ak nasleduje, môže začať s nahrádzaním pravej strany prepísaním a na b (resp. ak $b = \varepsilon$, tak iba dočítava vstup pre kontrolu).

4. Uprace pásku. Tým sa myslí to, že ak bola ľavá strana niektorého pravidla kratšia ako pravá strana, zmaže políčka, ktoré na páske vznikli ako prebytočné, presunie tieto prázdne políčka za posledné písmeno a ak sa to naozaj skrátilo, zapíše za koncom nového (kratšieho) slova falošný \$.

5. Skontroluje, či sa na páske nenachádza už iba ($\text{¢}\sigma\$$). Ak áno, tak zmení stav na akceptačný stav q_k a prejde hlavou na naozajstný \$, ak nie, pokračuje v práci od etapy 1.

□

Veta 9.2.2. *K ľubovoľnému LBA A existuje kontextová gramatika G taká, že $L(G) = L(A) - \{\varepsilon\}$.*

Dôkaz. Nech LBA $A = (K, \Sigma, \Gamma, \delta, q, F)$.
Pri dôkaze tejto vety sa nám hodí tretí variant konfigurácií: konfigurácia je slovo z $K\text{¢}\Gamma^*\$ \cup \text{¢}\Gamma^* K \Gamma^*\$$, pričom $K \cap \Gamma = \emptyset$.

Konštrukcia gramatiky G je založená na simulácii automatu A na vstupnom slove tak, že najskôr vygeneruje všetky možné vstupné konfigurácie a potom na nich začne simulovať automat A. Ako ho bude simulovať?

Najprv popis problémov v konštrukcii gramatiky. Ak pre δ funkciu automatu A platí : $(p, b, 0) \in \delta(q, a)$, potom na konfigurácii nastala zmena : $(\text{¢}uqav\$) \vdash (\text{¢}upbv\$)$, čiže sa zdá byť rozumné dať do konštruovanej gramatiky G pravidlo $qa \to pb$, pretože tak by mohla simulovať tento krok výpočtu automatu A.

Podobne ak $(p, b, 1) \in \delta(q, a)$ potom na konfigurácii nastala zmena $(\text{¢}uqav\$) \vdash (\text{¢}ubpv\$)$ a preto v gramatike G musí existovať pravidlo $qa \to bp$. Menší problém bude s posledným pravidlom,

pretože ak platilo $(p, b, -1) \ni \delta(q, a)$ potom sa mohla konfigurácia zmeniť $(\text{¢}ucqav\$) \vdash (\text{¢}upcbv\$)$ a to pre ľubovoľné $c \in \Gamma$, ktoré dopredu nevieme určiť. Preto do gramatiky G pridáme pravidlá $cqa \to pcb$ pre všetky $c \in \Gamma$.

Takže simuláciu máme zabezpečenú, vzniká však iný problém : simuláciou automatu na vstupnej konfigurácii sa pokazila vstupná konfigurácia. Automatu to nevadí, pretože jemu stačí odakceptovať, ale gramatika musí vrátiť slovo, preto si musíme na začiatku nejako vstupnú konfiguráciu zapamätať, aby sme po odakceptovaní dokázali „vypísať" slovo z nášho jazyka.

Riešením je použitie viacstopej pásky (konkrétne dvojstopej pásky)[2].

Gramatiku G teda konštruujeme nasledovne:

1. G generuje všetky možné počiatočné konfigurácie a zároveň zdvojuje pásku, (generované konfigurácie sú tvaru : $\text{¢}q_0\binom{a_1}{a_1}\binom{a_2}{a_2}...\binom{a_n}{a_n}\$$ pre ľubovoľné $a_0, a_1, ..., a_n \in \Sigma$).

$$\begin{array}{lcll} \sigma & \to & \sigma_1\$ & \\ \sigma_1 & \to & \sigma_1\binom{a}{a} & \forall a \in \Sigma \\ \sigma_1 & \to & \text{¢}q_0 & \end{array}$$

2. G simuluje výpočet na hornej stope:

$$\begin{array}{rcll} q\binom{a}{x} & \to & p\binom{b}{x} & \forall x \in \Sigma \quad \text{pre} \quad (p, b, 0) \in \delta(q, a) \\ q\binom{a}{x} & \to & \binom{b}{x}p & \forall x \in \Sigma \quad \text{pre} \quad (p, b, 1) \in \delta(q, a) \\ \binom{c}{x}q\binom{a}{y} & \to & p\binom{c}{x}\binom{b}{y} & \forall x \in \Sigma \quad \text{pre} \quad (p, b, -1) \in \delta(q, a) \end{array}$$

3. V tejto etape generuje gramatika G terminálne slovo:

$$q\$ \to Q\$ \text{ pre } \forall q \in F$$

Symbol Q je akýmsi signálom, aby sa začalo prepisovať slovo zo spodnej stopy.

$$\binom{a}{x}Q \to Qx$$

Teraz použijeme malý podvod spočívajuci v tom, že v gramatike G vztvoríme pravidlá, ktoré nie sú kontextové.

$$\begin{array}{rcl} \text{¢}Q & \to & \varepsilon \\ \$ \to \varepsilon & & \end{array}$$

Ako bude možné vysporiadať sa s poslednými dvoma nekontextovými pravidlami?
Opäť nám pomôžu viacposchodové vetné formy, budeme mať dokonca päť poschodí : spodné dve poschodia budú doterajšie dve poschodia a keďže potrebujeme schovať tri znaky, ktoré nám kazia kontextovosť, bude prvé poschodie pre stav, druhé pre $ a tretie pre ¢. Tieto vrchné tri poschodia budú obsahovať symboly : $q \in K$ na zaznačenie stavu (resp. $\$, \text{¢}$), zaznačenie prázdneho miesta $-$, ďalej dolárové a centové poschodie budú ešte potrebovať špeciálne endmarkre $\$'$ resp. $\text{¢}'$, ktoré nám umožnia rozlíšiť, či sa hlava nachádza na znaku v danom stĺpci alebo na $ resp. ¢ [3].
Bude nutné mierne modifikovať pravidlá gramatiky, čo prenechávame len najextrémnejšie oriemtovaným čitateľom.

[2] Nejde o technické vylepšenie automatu, pretože takáto páska sa dá zabezpečiť napríklad zmenou pracovnej abecedy Γ na $\Gamma \times \Gamma$.

[3] Napríklad ak sa hlava nachádza naozaj na ¢, tak je na tretej stope pásky zapísaný endmarker $\text{¢}'$, ak je hlava na a_0, potom je na páske zapísaný ¢.

Dôkaz: $L(A) = L(G)$:
$\subseteq$: máme slovo $w \in L(A)$ a aj jeho odvodenie. Potrebujeme dokázať, že ho vieme vygenerovať gramatikou G. V prvej fáze vieme vygenerovať ľubovoľné slovo (ľahko dokázateľné MI), ďalej v druhej fáze existuje práve jedno jednoznačné priradenie medzi konfiguráciami automatu a vetnými formami, čiže vieme prejsť z vetnej formy, ktorá zodpovedá prvej konfigurácii, k vetnej forme [4] zodpovedajúcej nasledujúcej konfigurácii[5]. No a prepísanie slova z dolnej pásky na výstup je opäť zrejmé.

$\supseteq$: Potrebujeme dokázať, že čokoľvek gramatika G vygeneruje je slovo, na ktorom zbehne akceptujúci výpočet automatu A. Zoberieme si ľubovoľné odvodenie terminálneho slova v gramatike, v tomto odvodení sa dajú nájsť dve význačné miesta:

1. Prvý krát sa objaví symbol stavu, v danej chvíli vetná forma zodpovedá konfigurácii počiatočného stavu automatu.

2. Jediný spôsob, ako dosiahnuť sterminálňovanie vetnej formy je objavenie sa akceptačného stavu a endmarkra $\$'$ v jednom stĺpci pásky.

S využitím týchto dvoch bodov a vedomosti, že každému kroku v gramatike zodpovedá pravidlo výpočtu v automate opäť nie je problém pre vášnivého formalistu dokončiť dôkaz. □

9.3 Uzáverové vlastnosti L_{cs}

Veta 9.3.1. *Trieda jazykov akceptovaných nedeterministickými LBA je uzavretá na prienik.*

Dôkaz. Najskôr sa trochu zamyslime: nemôžeme spojiť dva automaty spôsobom, akým sme to urobili v prípade konečných automatov, pretože máme iba jednu hlavu a tie sa pohybujú každá potenciálne iným smerom.

Preto opäť použijeme trik s dvojstpu páskou (kartézsky súčin množín pôvodných páskových symbolov).
Nech máme dané dva LBA automaty A_1, A_2, ktoré akceptujú jazyky z triedy jazykov akceptovaných $NLBA$, zostrojíme automat A_3, ktorý bude akceptovať $L(A_1) \cap L(A_2)$.
Automat A_3 bude pracovať v nasledovných etapách:

1. vyrobí na páske dve stopy, skopíruje w na hornú aj na spodnú stopu

2. simuluje automat A_1 na hornej stope a ak akceptuje prejde do nasledujúcej etapy

3. simuluje automat A_2 na spodnej stope a akceptuje, ak automat A_2 akceptuje.

predpokladáme : $K_1 \cap K_2 = \emptyset$
definujeme automat $A_3 : (K_3, \Sigma_3, \Gamma_3, \delta_3, q_{03}, F_3)$

$K_3 = \{q_{01}, q^*, q^{**}\} \cup K_1 \cup K_2; \Sigma_3 = \Sigma_1 = \Sigma_2 = \Sigma; \Gamma_3 = \Sigma \cup \Gamma_1 \times \Gamma_2; F_3 = F_2$

1.
$$\begin{array}{rcll} \delta_3(q_{03}, a) & = & \{(q_{03}, \binom{a}{a}, 1)\} & \text{pre } \forall a \in \Sigma \\ \delta_3(q_{03}, \$) & = & \{(q^*, \$, -1)\} & \\ \delta_3(q^*, \binom{a}{a}) & = & \{(q^*, \binom{a}{a}, -1)\} & \text{pre } \binom{a}{a} \in \Sigma \times \Sigma \\ \delta_3(q^*, ¢) & = & \{(q_{01}, ¢, 1)\} & \end{array}$$

[4] na základe delta funkcie automatu A
[5] A kedže poznáme odvodenie v automate A, vieme povedať ako bude vyzerať.

2.

$$\begin{array}{rcll}
\delta_3(q,\binom{a}{b}) & = & \{(p,\binom{a'}{b},d) \mid (p,a',d) \in \delta_1(q,a))\} & \text{pre } q \in K_1 \\
\delta_3(q,\not\in) & = & \delta_1(q,\not\in) & \text{pre } q \in K_1 \cup K_2 \\
\delta_3(q,\$) & = & \delta_1(q,\$) & \text{pre } q \notin F_1 \\
\delta_3(q,\$) & = & \{(q^{**},\$,-1)\} & \text{pre } q \in F_1 \\
\delta_3(q^{**},\binom{a}{b}) & = & \{(q^{**},\binom{a}{b},-1)\} & \text{pre } \binom{a}{b} \in \Gamma_1 \times \Gamma_2 \\
\delta_3(q^{**},\not\in) & = & \{(q_{02},\not\in,1)\} &
\end{array}$$

3.

$$\begin{array}{rcll}
\delta_3(q,\binom{a}{b}) & = & \{(p,\binom{a}{b'},d) \mid (p,b',d) \in \delta_2(q,a))\} & \text{pre } p,q \in K_2 \\
\delta_3(q,\not\in) & = & \delta_2(q,\not\in) & \text{pre } q \in K_2 \\
\delta_3(q,\$) & = & \delta_2(q,\$) & \text{pre } \forall q \in K_2
\end{array}$$

Tvrdenie: $L(A_3) = L(A_1) \cap L(A_2)$
$\supseteq$: nech $(q_{01}, \not\in \restriction w\$) \vdash^* (q_{n1}, \not\in u \restriction \$)$ je akceptujúci výpočet na slove w v automate A_1 a $(q_{02}, \not\in \restriction w\$) \vdash^* (q_{m2}, \not\in u \restriction \$)$ je akceptujúci výpočet na slove w v automate A_2, hľadáme akceptujúci výpočet na slove w v automate A_3.
Tvrdíme, že to bude nasledujúci výpočet : $(q_{03}, \not\in \restriction w\$) \vdash^* (q^*, \restriction \not\in\binom{w}{w}\$) \vdash (q_{01}, \not\in \restriction \binom{w}{w}\$) \vdash^* (q_{n1}, \not\in\binom{w'}{w} \restriction \$) \vdash^* (q^{**}, \restriction \not\in\binom{w'}{w}\$) \vdash (q_{02}, \not\in \restriction \binom{w'}{w}\$) \vdash^* (q_{m2}, \not\in\binom{w'}{w''} \restriction \$)$ je to veľmi ľahko vidieť aj preto, že sme tak robili konštrukciu.

$\subseteq$ máme akceptujúci výpočet v automate A_3 na slove w, opäť sa ho snažíme rozparcelovať na úseky : akceptovali sme v stave $q \in F_2$, ale do tohto stavu sme sa moholi dostať jedine v prípade, keď sme postupovali podľa δ_2 funkcie – a teda sa nám niekde musel vyskytnúť stav q_{02} (tu dostaneme akceptujúci výpočet v automate A_2 na slove w, ak zoberieme všetky konfigurácie $(q_{02}, \not\in \restriction \binom{w'}{w}\$) \vdash^* (q_{m2}, \not\in\binom{w'}{w''} \restriction \$), q_{m2} \in F_2$ a zabudneme na vrchnú stopu pásky), do stavu q_2 sme sa moholi dostať jedine cez stav q^{**}, do neho jedine cez stav $q_{n1} \in F_1$ a v konečnom dôsledku cez q_{01} (tu podobne zoberieme výpočet medzi stavmi q_{01} a q_{0n}, zabudneme na spodnú časť pásky a máme akceptujúci výpočet na slove w v automate A_1). □

Veta 9.3.2. *L_{cs} je uzavretá na inverzný homomorfizmus.*

Dôkaz. Budeme to dokazovať cez automaty. Nech $A = (K, \Sigma, \Gamma, \delta, q, F)$ je LBA, ktorý akceptuje jazyk L, treba zostrojiť LBA, ktorý bude akceptovať jazyk $h^{-1}(L)$. Nech je dané slovo w, potrebujeme zistiť, či slovo $h(w) \in L$, pretože to je ekvivalentné s $w \in h^{-1}(L)$.

Pri dokazovaní uzavretosti na inv. homomorfizmus $\mathcal{L}_{cf}$ sme používali buffer, ktorý sme dokázali simulovať stavom. To nám však teraz nepomôže, pretože slovo na páske sa môže meniť a LBA môže pohybovať hlavou na vstupnej páske. Preto budeme musieť zmeny, ktoré sme vykonali na páske, zaznačiť opäť niekde na páske. Ibaže narážame na ohraničenosť pásky. Tento problém je možné vyriešiť hrebeňovou páskou[6]. Táto páska sa bude vyznačovať tým, že počet stôp nad jednotlivými znakmi vstupného slova nemusí byť nutne rovnaký, dokonca nad niektorými znakmi môže byť prázdne políčko (bude to akýsi štrbavý hrebeň).

Automat A' dostane na vstup slovo w, vytvorí si hrebeňovú pásku a zároveň si na ňu bude zapisovať slovo $h(w)$. Potom sa posunie na začiatok slova $h(w)$ a začne na ňom simulovať prácu automatu A. Tu treba klasicky prepísať delta funkciu automatu A tak, aby vedel chodiť po viacstopej páske nahor a dole, aby si pri tom pamätal smer, od kadiaľ prišiel, mal ošetrené kraje pásky a vedel riešiť aj problém "štrbavého hrebňa", t.j. ošetriť problém prázdneho políčka na vstupnej páske. □

Veta 9.3.3. *L_{cs} je uzavretá na zjednotenie.*

[6] Ide vlastne o viacstopú pásku, ktorej počet stôp bude maximálne $m = max\{h(x), x \in \Sigma\}$.

Dôkaz.

1. cez gramatiky : tak isto ako pre bezkontextové, zavedením nového neterminálu, ktorý neskôr prepíšeme na každý z pôvodných, musíme opäť zabezpečiť prázdnosť prieniku neterminálnych symbolov jedného automatu so zjednotením terminálnych a neterminálnych symbolov druhého automatu a naopak ...

2. cez automaty : zavedieme nový počiatočný stav a v ňom sa nedeterministicky rozhodneme, ktorý automat sa spustí ...

□

Veta 9.3.4. *L_{cs} je uzavretá na zreťazenie.*

Dôkaz.

1. cez gramatiky : to čo zbehlo pre bezkontextové jazyky ($\sigma \to \sigma_1\sigma_2$) nám teraz naráža na problém, pretože v kontextových jazykoch môžem mať pravidlá typu $x\xi \to ...$, teda môže nastať kolízia na rozhraní dvoch slov: toho, ktoré vzniklo zo σ_1 a toho, ktoré je práve vyrábané zo σ_2. Mohlo by sa to riešiť disjunktnosťou terminálov, ale to je nepraktické a škaredé.
   Dá sa to riešiť :

   (a) niekde si zapíšem zarážku – pred chvíľou pri odstraňovaní kontextových pravidiel viacstopou páskou pri tvorbe gramatiky ku automatu sme videli, ako je možné tento znak naviac zapísať na páske a pri tom neporušiť kontextovosť.

   (b) normálnym tvarom gramatiky, ktorá bude mať na ľavej strane iba neterminály : $P \subseteq N^+ \times (N \cup T)^+$. Potom bude ďalej možné použiť princíp dôkazu uzavretosti bezkontextových gramatík na zreťazenie.

2. cez automaty : máme k dispozícii dva automaty pre jazyk L_1 a L_2. Máme zostrojiť automat, ktorý dostane na vstupe slovo w a má zistiť, či $w \in L_1.L_2$. Bude pracovať nasledovne : vyrobí si dvojstopú pásku, potom si niekde nedeterministicky na jednej stope zapíše falošný dolár ($\binom{\$}{a}$, $a \in \Sigma$), vráti sa na začiatok a spustí výpočet prvého automatu po $\binom{\$}{a}$, keď bude v akceptačnom stave, zruší falošný dolár, vyrobí falošný cent a spustí simuláciu druhého automatu od falošného centa. Ak sa dostane do akceptačného stavu, tak slovo w akceptuje.

□

Veta 9.3.5. *L_{cs} je uzavretá na kladnú iteráciu.*

Dôkaz. Budeme využívať uzavretosť na zreťazenie pre L_{cs}.

1. Gramatika generujúca L^+ najprv vygeneruje vetnú formu pozostávajucu z oddelených počiatočných neterminálov. Avšak, môžu nastať technické problémy spojené s vymazaním veľkého počtu uddeľovačov (uvedome si, že vetná forma sa nikdy nesmie skrátiť). Preto oddeľovače nepoužijeme, ale namiesto nich vygenerujeme vetnú formu v tvare $\sigma'\sigma''\sigma'\sigma''$....
   Potom použijeme grmatiku zyvlášť v čiarkovanom a dvojčiarkovanom tvare.

   Budeme striedať párne a nepárne verzie gramatiky, ktoré budú robiť to isté ako pôvodná gramatika, iba budú mať disjunktné neterminály a nebudú sa navzájom ovplyvňovať.

2. Automat akceptujúci iteráciu pôvodného jazyka sa bude správať nasledovne:
keď vyrobí falošný dolár, bude simulovať pôvodný automat na slove pred ním a ak sa dostanem do akceptačného stavu, tak zmaže falošný dolár, vyrobí falošný cent a buď umiestni nový falošný dolár alebo opäť bude simulovať po pravý dolár. Automat skončí iba ak sa bude nachádzať v akceptačnom stave a na pravom dolári.

□

9.4 Szelepczényiho veta

Matematici zaoberajúci sa teóriou formálnych jazykov po desaťročia hľadali odpoveď na otázku, či je trieda jazykov rozpoznávaných LBA, teda kontextových jazykov rozšírených o ε[7] uzavretá na komplement[8]. Väčšina z týchto bádateľov si totiž myslela, že práve neuzavretosť na komplement je vlastnosť, ktorá odlišuje triedu kontextových jazykov od triedy rekurzívne enumerovaných jazykov, t.j. jazykov akceptovaných Turingovým strojom[9]. Preto hľadali dôkazy toho, že trieda $\mathcal{L}_{\mathcal{ECS}}$ nie je uzavretá na komplement. Ukázalo sa však, že opak je pravdou. Za tento veľký výsledok vďačíme slovenskému matematikovi[10] Róbertovi Szelepczényimu. Ešte predtým, ako sa pustíme do dôkazu, poznamenajme, že inšpiráciou k dôkazu mu bola práca s nekvalitnými harddiskami. Ak totiž chcel, aby jeho program klasickým spôsobom čítať údaje z pevného disku, často sa stávalo, že harddisk neprečítal všetko. Bola teda potrebná rutina, ktorá porovná objem skutočne prečítaných dát s objemom dát, ktoré mali byť prečítané. Čitateľ, ktorý preštuduje nasledujúci dôkaz iste nájde paralelu medzi týmto problémom a myšlienkou, ktorá sa v dôkaze niekoľkokrát objaví.

Veta 9.4.1. *(Innermann, Szelepczényi) Trieda $\mathcal{L}_{ECS}$ je uzavretá na komplement.*

Dôkaz. Nech $L \in \mathcal{L}_{ECS}$ je rozpoznávaný lineárne ohraničeným automatom $A = (K, \Sigma, \Gamma, \delta, q_0, F)$. Skonštruujme lineárne ohraničený automat A', ktorý akceptuje L^C. Uvedomme si najprv, kde je problém. Keby sme jednoducho zamenili akceptujúce a neakceptujúce stavy, potom by sme dostali LBA, ktorý akceptuje práve tie slová, pre ktoré sa pôvodný LBA vedel dostať do neakceptujúceho stavu s hlavou na dolári. Budeme teda postupovať iným spôsobom. Základnou myšlienkou je, že A' bude generovať konfigurácie A pre vstupné slovo a zisťovať o nich, či sú dosiahnuteľné. Ak narazí na dosiahnuteľnú konfiguráciu takú, ktorá má hlavu na konci vstupného slova a stav akceptačný, slovo neakceptuje[11]. Ak zistíme, že sme už prešli všetky dosiahnuteľné konfigurácie a žiadna nebola akceptovateľná pôvodným A, potom slovo akceptujeme.

Uvedomme si, že pre dané vstupné slovo w vieme zhora ohraničiť počet rôznych dosiahnuteľných konfigurácii LBA A pre vstupné slovo w. Všetkých možných konfigurácii totiž je $|\Gamma|^{|w|}.|K|.(|w|+2)$[12].

Ďalej bude náš dôkaz pokračovať svojím koncom. Predpokladajme, že poznáme počet dosiahnuteľných konfigurácií Q. Neskôr ukážeme, že toto číslo vieme naozaj vypočítať. Postupne

[7] Ďalej v tomto paragrafe nebudeme rozlišovať medzi kontextovými jazykmi a kontextovými jazykmi rozšírenými o ε, ak to nebude nutné

[8] Teda či k ľubovoľnému kontextovému jazyku L nad abecedou Σ existuje jazyk nad tou istou abecedou L' taký, že $L \cap L' = \emptyset \wedge L \cup L' = \Sigma^*$. Jazyk L' v tom prípade označujeme L^C

[9] Pripomíname, že Turingov stroj je najsilnejšie výpočtové zariadenie, ekvivalentné s počítačmi tak, ako ich poznáme (ak predpokladáme potenciálne nekonečný pevný disk, či inú pamäť), trieda jazykov akceptovaných TS neobsahuje všetky možné jazyky. Totiž, mohutnosť množiny všetkých jazykov nad Σ je $|2^{\Sigma^{\mathbb{N}}}| = \aleph_1$, ale mohutnosť množiny všetkých Turingových strojov je len $|2^{(N)}| = \aleph_0$

[10] ktorý v tom čase študoval na MFF UK Bratislava

[11] Napríklad zacyklením sa

[12] Číslo $|\Gamma|^{|w|}$ vyjadruje počet rôznych slov zapísaných na páske a $|w|+2$ pozíciu hlavy

na páske automat A' generuje všetky možné konfigurácie. Pre každú znich nedeterministicky uhádne, či je dosiahnuteľná automatom A. V prípade, že háda áno, skontroluje, či naozaj. V prípade, že táto konfigurácia je naozaj dosiahnuteľná, zistí to v konečnom čase. Ak nie, táto výpočtová vetva sa zacyklí, no možnosť nedeterministického „nie" dáva možnosť zistenia jeho nedosiahnuteľnosti. Týmto spôsobom dosiahneme, že automat A' nás môže oklamať len jedným smerom. Nedosiahnuteľnú konfiguráciu nikdy neoznačí za dosiahnuteľnú, no niektoré dosiahnuteľné prehlási za nedosiahnuteľné. Nato sme práve potrebovali vedieť Q. Ak po poslednej vygenerovanej konfigurácii zistíme, že sme dosiahnuteľnými vyhlásili menej ako Q konfigurácii, celý proces opakujeme.

Dôležité na tomto mieste je uvedomiť si, že logaritmus čísla Q je nanajvýš $|w|.|\Gamma| + c$, a teda si ho lineárne ohraničený automat dokáže pamätať.

Poďme teraz ukázať, že LBA si naozaj dokáže vypočítať Q[13]. Postupne bude generovať čísla $Q_1, Q_2, ...$, ktoré budú vyjadrovať počet dosiahnuteľných konfigurácií automatu A na vstupnom slove w, do ktorých sa vie dostať na maximálne i krokov. Keďže do všetkých dosiahnuteľných konfigurácií, ktorých je konečne veľa, sa vie A dostať na konečne veľa krokov, existuje konečné i také, že $Q_i = Q$. Zrejme platí $Q_i = Q_{i+1} \Rightarrow Q_i = Q$. Číslo Q_1 poznáme, je rovné jednej. Ukážme, že na základe poznania čísla Q_i si dokáže LBA vypočítať Q_{i+1}. Naozaj to musí zvládnuť len na základe poznania čísla Q_i a nie zapamätania si všetkých Q_i konfigurácii, nato totiž LBA nemá na páske priestor. Má však priestor nato, aby dokázal postupne generovať všetkých Q_i dosiahnuteľných konfigurácií. Za tohto predpokladu ukážeme, že potom vie postupne vygenerovať aj všetkých Q_{i+1} konfigurácií. Uvedomme si najprv, že pre dané vstupné slovo w, konfiguráciu k a číslo i dokáže LBA nedeterministicky v konečnom čase zistiť, či sa A asi dokáže na maximálne i krokov dostať do tejto konfigurácie. „Asi" znamená, že jeho „nie" v skutočnosti znamená „možno" a jeho „áno" znamená naozaj „áno". LBA bude postupovať nasledovne:

1 Nastaví counter Q_{i+1} na hodnotu Q_i.

2 Postupne generuje niektoré z Q_i dosiahnuteľných konfigurácií $k_1, k_2, ..., k_m$ na max. i krokov.

3 Pre každú k_i urobí jeden krok výpočtu automatu A a opäť postupne generuje niektoré z Q_i dosiahnuteľných konfigurácií $k'_1, k'_2, ..., k'_n$ aby ich porovnal s k_i. Ak je porovnanie pre nejaké z nich úspešné, potom Q_i nezväčšuje. Ak na konci generovania $k'_1, ..., k'_n$ zistí, že $n < Q_i$, potom tento krok opakuje.

4 Ak $m < Q_i$, potom sa vráť na krok [2]. Ak $m = Q_i$ a všetky prechádzajuce porovnania boli negatívne, inkrementuj counter Q_{i+1} o jedna.

Tým sme dokázali, že Q_i dokážeme vypočítať.

Tým je dôkaz podaný. Poznamenajme na záver, že podrobný dôkaz vyžadoval niekoľko desiatok strán a skonštruovaný LBA A' mal niekoľko desiatok stôp. Myšlienka použitá pri dôkaze, ktorá využije nedeterminizmus na generovanie všetkých možností, pričom ich vygenerujeme určite nie viac, no možno menej, a tak tento podvod odhalíme, dostala názov **inductive counting**. □

[13] Logicky presnejšie: Pre daný LBA M vieme vyrobiť LBA taký, ktorý pre každé vstupné slovo zistí počet dosiahnuteľných konfigurácií automatu M.

Kapitola 10

Turingove stroje

10.1 Definície

V popisovaní zariadení na rozpoznávanie jazykov sme sa dostali k poslednej abstrakcii, ku stroju, ktorý dokáže zastúpiť každý doposiaľ zmienený automat, dokáže simulovať ľubovolný iný Turingov stroj, ba dokonca všetko, čo je algoritmické (teda naprogramovatelné).
V tejto časti sa budeme zamýšlať aj nad vlastnosťami TS, ktoré budú väčšinou len zovšeobecnením LBA. Turingove stroje sú vlastne LBA, ale s nekonečnou páskou.

Najprv definujeme Turingove stroje neformálne, s istým stupňom abstrakcie a neskôr si ukážeme aj názornú formálnu definíciu.

Turingov stroj je stroj, s nekonečnou páskou, jednou hlavou pohybujúcou sa doprava, doľava, alebo stojacou na mieste. Hlava načíta písmeno na páske, prepíše ho novým znakom[1] a pohne sa doprava, doľava, alebo nikde.

Na začiatku predpokladáme, že sa hlava nachádza na prvom písmenku vstupného slova, pričom na miestach, kde sa vstupné slovo nenachádza, je na páske zapísané písmeno '**B**' (ako Blank). Platí tu malá dohoda: **B** sa nemôže zapisovať. Spôsobuje to iba malé problémy - napr. keď čítame za vstupom, prečítame **B**lank a musíme tu niečo zapísať. Riešenie je buď zapisovať falošné **B**lank-y, alebo jednoducho zrušiť dohodu.

Konfigurácie môžeme zapisovať podobne ako u LBA s tým, že zapisujeme iba neprázdne časti (bez Blank-ov). To znamená, že máme na výber viacero možností :

1. Konfigurácia so stavom :
   (wqv), $v, w \in \Gamma^*$ pričom konfigurácie s hlavou na začiatku, resp. na konci zapisujeme ako (wq) , resp. ($q\mathbf{B}w$)

2. Konfigurácia s bičíkom $\restriction$:
   (q , $w \restriction v$), $v, w \in \Gamma^*$ pričom konfigurácie s hlavou na začiatku, resp. na konci zapisujeme ako (q , $w \restriction$) , resp. (q , $\restriction \mathbf{B}w$)

3. Konfigurácia s udaním pozície od začiatku slova :
   (q , $a_1a_2 \ldots a_n, i$), $w = a_1a_2 \ldots a_n$, $w \in \Gamma^*$ pričom je výhodnejšie používať Turingov stroj s jednosmernou páskou[2] a potom sa hlava môže nachádzať na miestach 1 - začiatok, až n+1 - koniec (vtedy čítame Blank)

[1] písmená sú z pracovnej abecedy Γ
[2] dôkaz ekvivalencie jednosmerne nekonečnej pásky a obojsmerne nekonečnej pásky bude neskôr

Akceptovanie Turingovým strojom zabezpečíme iba akceptačným stavom, a teda všetky kroky v akceptačnom stave sú nepotrebné.[3]

Definícia 10.1.1. *Nedeterministickým Turingovym strojom nazývame 6-ticu* $(K, \Sigma, \Gamma, \delta, q_0, F)$. *Kde* K *je konečná množina stavov,* Σ *je abeceda vstupných symbolov,* Γ *je abeceda pracovných symbolov* $(\Sigma \subseteq \Gamma)$, $q_0 \in K$ *je počiatočný stav,* $F \subseteq K$ *je množina akceptačných stavov,*

$$\delta : K \times (\Gamma \cup \{\mathbf{B}\}) \to 2^{K \times (\Gamma) \times \{-1,0,1\}}$$

pričom platí

$$\delta(q, \mathbf{B}) \subseteq K \times \{\mathbf{B}\} \times \{-1, 0, 1\}$$

Definícia 10.1.2. *Konfigurácia TS je prvok z* $K\mathbf{B}\Gamma^* \cup \Gamma^* K \Gamma^* \cup \Gamma^* K \mathbf{B}$, $(K \cap \Gamma = \emptyset)$

Definícia 10.1.3. *Krok výpočtu TS je relácia* $\vdash$ *na konfiguráciách definovaná nasledovne:* $\forall a, b, c \in \Gamma \cup \{\mathbf{B}\}$, $p, q \in K$*:*

- $\Gamma^* q a \Gamma^* \vdash \Gamma^* p b \Gamma^* \Leftrightarrow (p, b, 0) \ni \delta(q, a)$
- $\Gamma^* q a \Gamma^* \vdash \Gamma^* b p \Gamma^* \Leftrightarrow (p, b, 1) \ni \delta(q, a)$
- $\Gamma^* c q a \Gamma^* \vdash \Gamma^* p c b \Gamma^* \Leftrightarrow (p, b, -1) \ni \delta(q, a)$

Definícia 10.1.4. *Jazyk akceptovaný TS je*

$$L(A) = \{\ w \in \Sigma^* \mid (q_0 w) \vdash^* (uqv);\ u, v \in \Gamma^*,\ q \in F\ \}$$

Definícia 10.1.5. *deterministický Turingov stroj je taký TS, kde:*

$$\#(\delta_k(q, a)) \leq 1;\ \forall q \in K,\ a \in \Gamma \cup \{\mathbf{B}\}$$

10.2 Varianty Turingových strojov

- Nedeterministické a deterministické Turingové stroje
- TS s jednosmerne (zprava) nekonečnou páskou
  Riešenie je buť špeciálnymi stavmi, ktoré pri pokuse zapisovať doľava najprv celé slovo posunú doprava a nadstavia hlavu na začiatok - úplne doľava, kde sa dané písmeno zapíše. Alebo je možné normálny TS simulovať pomocou pásky zohnutej na dvojstopú, pričom znaky na hornej stope by boli vlastne znaky, ktoré mali byť naľavo od začiatku, a znaky na spodnej stope by boli na svojích miestach. V tomto prípade je nutné si pásku na dvojstopú najprv upraviť (doplnením falošných Blank-ov nad vstupné slovo)
- Viacpáskový s hlavami na každej páske
  Pomocou jednej hlavy by sme takýto automat mohli simulovať s viacerými stopami. Pohyby jednotlivých hláv by sa prejavovali zodpovedajúcimi pohybmi slov na jednotlivých páskach nad určitým miestom. Automat by iba zaisťoval prečítanie poschodového znaku nad týmto miestom na základe čoho by posunul slová na jednotlivých stopách prípadne tam zapísal dané znaky.
  Taktiež by sme to mohli vyriešiť dvojnásobným počtom stôp, ako máme pások. Na každej druhej stope by sme si evidovali pozíciu hlavy na jej páske (stopa pod ňou). Treba si uvedomiť, že to, čo číta K - hláv, si vieme zapamätať v stave.

[3] z toho je napríklad vidieť, že Turingov stroj je schopný akceptovať aj slovo, ktoré neprečítal

- Viachlavé TS
  Dohoda: Keď chce viac hláv zapisovať na to isté políčko, zapíšeme znak hlavy s najväčším číslom.
  Riešenie: Na 1 páske máme viac stôp, pre každú hlavu jednu, kde si zapisujeme pozície hláv. Znaky načítané jednotlivými hlavami si môžme zapamätať v stave, a potom postupne robiť to, čo pôvodný automat s viacerými hlavami.

- 2D páska
  Treba vymysliť iba vhodnú bijekciu medzi políčkami na 2D páske a políčkami na normálnej páske. Taktiež je nutné simulovať každý pohyb uskutočnený na 2D páske aj na normálnej (napríklad pohyb z [i, j] hore na [i, j+1], ...).[4]

10.3 Kódovanie TS

(: V tejto časti si povieme o zakódovaní δ-funkcie do postupností $\{0,1\}^+$. Keď budeme mať daný kód δ-funkcie (kód TS A označujme <A>), budeme môcť zostrojiť tzv. Univerzálny TS, ktorý dostane na vstup kód nejakého TS a vstupné slovo w, pričom tento univerzálny stroj bude simulovať stroj A na vstupe w. :)

10.3.1 Jednoduhší kód pre TS

Berme do úvahy DTS.

Stavy K	Pracovná abeceda $\Gamma \supseteq \Sigma$	Pohyb hlavy
q_0 10	s_0 10	-1 10
q_1 110	s_1 110	0 110
q_2 1110	s_2 1110	1 1110

Na oddelenie jednotlivých častí zakódovanej δ-funkcie budeme používať znak #.
Napríklad $\delta(q_1, s_0) = (q_2, s_3, 1)$ zakódujeme ako "#110101110111101110#".
Zaiste si vieme predstavť, že môžme zakódovať aj znak # (pričom trochu zmeníme naše kódovanie) a teda celý kód TS bude iba slovo z $\{0,1\}^+$.

Poznámka 10.3.1. Každý reťazec $\{0,1\}^+$ kóduje nejaký TS. Ak by tu boli nejaké nezrovnalosti, tak je to taký TS A pre ktorý platí $L(A) = \emptyset$ [5]

10.4 Vlastnosti Turingových strojov

10.4.1 Uzavretosť jazykov rozpoznávaných TS

Veta 10.4.1. *Jazyky rozpoznávané Turingovými strojmi sú uzavreté na prienik.*

Dôkaz. *(: Podobne ako u LBA :)* Pomocou dvojstopej pásky. Na vrchnej stope budeme dané slovo kontrolovať jedným Turingovým strojom a po akceptácii budeme simulovať druhý Turingov stroj na spodnej stope. Ak by prvý Turingov stroj potreboval dlhšiu pásku ako je vstupné slovo zapisovali by sme na spodnú stopu falošné Blank-y. Po akceptácii prvým strojom by sme však

[4] na 2D páske môže byť písmenami vyplnený ľubovolný súvislý útvar

[5] Všetko nasledujúce sa dá aj pri prísnej definícii kódu TS. Dôležité je len aby sme vedeli algoritmicky nájsť k číslu i kód i-teho TS a obrátene ku kódu jeho poradie v zozname kódov TS (to obnáša generovať poporadí reťazce a iba skontrolovať či korektne kódujú δ-funkciu)

s hlavou skončili za (alebo aj pred)slovom na spodnej stope a teda by sme najprv museli nájsť začiatok slova na spodnej páske.
Riešením by bolo aj používať Turingove stroje s jednosmerne nekonečnou páskou a potom by sme si mohli tieto slová zapísať aj v tvare $w\#w$. Prvý automat by pracoval na prvom slove w (na zľava nekonečnej páske), druhý na druhom w (na zprava nekonečnej páske) □

10.4.2 Ekvivalencia medzi TS a frázovými jazykmi

Veta 10.4.2. *K ľubovolnej frázovej gramatike G existuje NTS A taký, že $L(A) = L(G)$.*

(: V tomto prípade by sme mohli uviesť podobný algoritmus ako pri LBA, so zmenou, keď je ľavá časť pravidla dlhšia ako pravá,[6] *musíme do slova napísať celú ľavú časť, čo obnáša posunutie zvyšku slova doprava.*
My však uvedieme aj iný prístup k zisteniu, či je dané slovo patrí do jazyka generovaného gramatikou G. Zatiaľčo u LBA sme sa z daného sova w nedeterministicky dostali až k počiatočnému neterminálu gramatiky G (tipnutím si naposledy použité pravidlo použité na nejakom mieste pri odvodzovaní vstupného slova), teraz z počiatočného neterminálu na základe pravidiel vyrobíme nedeterministicky nejaké slovo a skontrolujeme so vstupom, či sa rovná. (vstup sme si uchovali na druhej stope) :)

Dôkaz. Náš TS bude pracovať nasledovne:

1. Prekopíruje vstup na hornú stopu a na dolnej nechá počiatočný neterminál gramatiky G.[7]

$$\begin{aligned}\delta(q_0, a) &= \{(q_0', \tfrac{a}{\sigma})\}\\ \delta(q_0', a) &= \{(q_0', \tfrac{a}{\mathbf{B}})\}\\ \delta(q_0', \mathbf{B}) &= \{(q_1, \mathbf{B})\}\end{aligned}$$

2. Tipne si pravidlo, ktoré ide použiť. Nech je tvaru $a_1a_2\ldots a_n \to b_1\ldots b_k$, automat prejde do stavu $\delta(q_1, \frac{x}{a}) = \{q_{[\ a_1a_2\ldots a_n,\ b_1\ldots b_k]}, \frac{x}{a}, 0\}$

3. Tipuje si miesto, kde možno dané pravidlo použiť.
$\delta(q_{[u,v]}, \frac{x}{a}) = \{(q_{[u,v]}, \frac{x}{a}, -1), \delta(q_{[u,v]}, \frac{x}{a}, 0), (q_{[u,v]}, \frac{x}{a}, 1)\}$

4. Ide prepísať písmenka podľa tipnutého pravidla. $(q_{[u,v]}, \frac{x}{a}) = \{(q'_{[u,v]}, \frac{x}{a}, 0)\}$

5. Prepisuje a upravuje pásku. $\delta(q'_{[au,bv]}, \frac{x}{a}) = \{(q'_{[u,v]}, \frac{x}{b}, 1)\}$
Keď skončila ľavá strana pravidla posunie vstup doprava a pripíše zostatok pravej strany pravidla do vyrobeného priestoru.
Keď skončila pravá strana pravidla kontroluje ešte ľavú stranu a postupne maže znaky, teda posúva zostatok doľava. $\delta(q'_{[au,\varepsilon]}, \frac{x}{a}) = \{(q^Z_{[u,\varepsilon]}, \frac{x}{a}, 0)\}$ $\delta(q'_{[au,\varepsilon]}, \frac{x}{b}) = \emptyset$
Pričom $(q^Z_{[u,v]}, \frac{x}{a}, 0)$ posunie vstup na spodnej páske o 1 doľava, čím prepíše písmeno a, a skončí na tom istom mieste kde začal v stave $(q'_{[u,\varepsilon]})$

6. Ak prepísal pravidlo, rozhodne sa, či skontroluje vygenerované slovo so vstupom na hornej páske, alebo či generuje ďalej. $(q'_{[\varepsilon,\varepsilon]}, \frac{x}{a}) = \{(q_1, \frac{x}{a}, 0), (q_{end}, \frac{x}{a}, 0)\}$

7. Ak bol v stave q_{end} iba skontroluje či sa stopy rovnajú, a podľa toho ide do akceptačného stavu.

[6] Toto u L_{CS} nie je možné
[7] Prípadne si dáme lepší pozor na prázdne slovo

Je zrejmé, že ak dané slovo patrí do L(G), potom existuje odvodenie v G a teda aj akceptujúci výpočet v A (automat robí to, čo gramatika). Naopak ak existuje akceptujúci výpočet v A, potom nám tento ponúka aj odvodenie slova v G. Záverom teda platí $L(A) = L(G)$. □

Veta 10.4.3. *K ľubovolnému TS A existuje frázová gramatika G taká, že* $L(G) = L(A)$.

Dôkaz. Vytvoríme gramatiku G, ktorá bude simulovať TS A:

1. G vygeneruje konfiguráciu $Zq_0 \begin{smallmatrix} a_1 \\ a_1 \end{smallmatrix} \dots \begin{smallmatrix} a_n \\ a_n \end{smallmatrix}$ pre ľubovolné $a_1 \dots a_n$[8]
   $\sigma \to Z\sigma' \begin{smallmatrix} a \\ a \end{smallmatrix}$; $\sigma' \to \sigma' \begin{smallmatrix} a \\ a \end{smallmatrix}$; $\forall a \in \Sigma$; $\sigma' \to q_0$
   $N = K \bigcup \Gamma \times \Gamma$; $T = \Sigma$

2. G simuluje výpočet na spodnej stope:
   Ak $(p, b, 0) \in \delta(q, a)$, potom $q \begin{smallmatrix} x \\ a \end{smallmatrix} \to p \begin{smallmatrix} x \\ b \end{smallmatrix}$; $\forall x \in \Sigma$
   Ak $(p, b, 1) \in \delta(q, a)$, potom $\begin{smallmatrix} x \\ a \end{smallmatrix} q \to p \begin{smallmatrix} x \\ b \end{smallmatrix}$; $\forall x \in \Sigma$
   Ak $(p, b, -1) \in \delta(q, a)$, potom $q \begin{smallmatrix} y & x \\ c & a \end{smallmatrix} \to \begin{smallmatrix} y \\ c \end{smallmatrix} p \begin{smallmatrix} x \\ b \end{smallmatrix}$; $\forall x, y \in \Sigma$; $\forall c \in \Gamma$

3. Generujeme terminálne slovo $q \to Q$; $q \in F$. Všetko sa má prepísať na hornú stopu, keď už raz TS akceptoval.
   $\begin{smallmatrix} x \\ y \end{smallmatrix} Q \to Qx$, $ZQ \to \varepsilon$

□

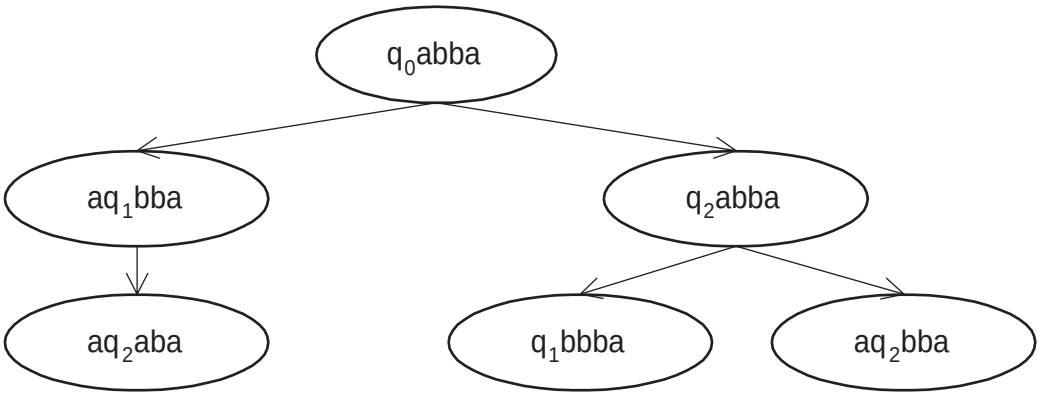
10.4.3 Ekvivalencia deterministického a nedeterministického TS

Veta 10.4.4. $L_{DTS} = L_{NTS}$

Dôkaz. Inklúzia $L_{DTS} \subseteq L_{NTS}$ je zrejmá.[9] Pozrime sa na opačnú inklúziu $L_{DTS} \supseteq L_{NTS}$:

(: Princíp bude vo vytváraní akéhosi stromu konfigurácii NTS A. To znamená, že budeme mať strom s uzlami znázorňujúcimi kofigurácie, do ktorých sa náš NTS A mohol počas výpočtu dostať. Koreň bude začiatočná konfigurácia q_0w. *Následníci každého uzlu sú také konfigurácie, do ktorých sa NTS A mohol dostať na uskutočnený 1 krok výpočtu.*

Príklad:



My budeme tento strom vytvárať postupne odvrchu, po jednotlivých úrovniach. To znamená,

[8] Z je tu ako zarážka pre začiatok
[9] determinizmus je len špeciálny prípad nedeterminizmu

že na páske budeme mať vždy iba jednu úroveň, ktorú postupným prejdením prepíšeme na nasledujúcu (na nižšie poschodie stromu). Keďže stavov, písmen na páske NTS A je konečný počet, a hlavne δ-funkcia NTS pre daný stav a symbol definuje iba konečnú množinu, prepis jednej konfigurácie na všetky možné jej nasledujúce konfigurácie je konečný[10]*, a teda aj prepis jednej úrovne stromu na nasledujúcu je konečný.*
Akceptujúcu konfiguráciu NTS A spoznáme jednoducho podľa stavu, ktorý je tiež zapísaný v konfigurácii na páske. Ak teda zbadáme na páske písmeno označujúce nejaký z akceptačných stavov NTS A, pôvodné slovo w akceptujeme aj my naším DTS A'. :) □

[10] Daná konfigurácia má konečnú dĺžku

Kapitola 11

Problémy riešiteľné a neriešiteľné

11.1 Zakódovanie TS nad abecedou {0,1}

Poznámka 11.1.1. Zopakujme si, čo sme sa dozvedeli a na čom sme sa dohodli ohľadom kódovania TS.

- Každý reťazec z $\{0,1\}^*$ je kód TS.
- Ak to nebude kód skutočného TS, tak to bude TS, ktorý akceptuje $\emptyset$.

Poznámka 11.1.2. Všetko nasledujúce sa dá uvažovať aj pri prísnej definícii kódu TS[1]. Dôležité je, aby sme vedeli algoritmicky k číslu i nájsť kód i-teho TS a obrátene, ku kódu TS jeho poradie v zozname kódov TS.

Zakódovanie TS do postupnosti núl a jednotiek využijeme v nasledujúcom tvrdení.

Veta 11.1.1. *Existuje jazyk, ktorý nepatrí do* $\mathcal{L}_{RE}$.

Dôkaz. Zostrojme si nekonečnú tabuľku, kde uvidíme, aké slová ktorý TS akceptuje:

	w_1	w_2	w_3	w_4	$\cdots$
A_1	$+$	$-$	$+$	$-$	$\cdots$
A_2	$-$	$-$	$+$	$-$	$\cdots$
A_3	$+$	$+$	$+$	$-$	$\cdots$
A_4	$+$	$-$	$-$	$+$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Do tabuľky T na miesto t_{ij} píšeme + a − takto:
$t_{ij} = +$, ak TS A_i akceptuje slovo w_j
$t_{ij} = -$, ak TS A_i neakceptuje slovo w_j

TS reprezentujeme slovom z $\{0,1\}^*$. Uvedomme si, že v tejto tabuľke sú všetky TS. Zostrojme jazyk, ktorý sa nebude vyskytovať v žiadnom riadku tabuľky (a teda žiadny TS nemôže tento jazyk akceptovať). Pôjdeme po diagonále a invertujeme znamienka. Najskôr ale zadefinujme diagonálny jazyk L_D (ten, ktorý chceme invertovať), ktorý ešte v ďalšom texte využijeme.

$$L_D := \{w \in \{0,1\}^* \mid w \in L(A_w)\}$$

[1] Žeby na skúške ?

Uvedomme si, že podľa našej dohody je kód TS A_i práve w_i. Potom zrejme $L_D^c \notin \mathcal{L}_{RE}$, lebo ak by existoval TS A_j, ktorý by tento jayzk akceptoval, tak sa pozrieme na slovo w_j a zistíme, že A_j toto slovo akceptuje, ale v jazyku L_D^c teda nie je, čo je spor, alebo obrátene, TS A_j toto slovo neakceptuje a v jazyku L_D^c práve preto je, čo je tiež spor. □

Definícia 11.1.1. *Symbolom $\mathcal{L}_{rec}$ označujeme množinu jazykov L, pre ktoré existuje deterministický TS A, ktorý zastaví [2] na každom vstupe a $L = L(A)$. Množina $\mathcal{L}_{rec}$ sa nazýva trieda rekurzívnych jazykov.*

Keďže existujú TS, ktoré sa zacyklia pre nejaký vstup, tak potom sa zrejme $\mathcal{L}_{rec}$ a $\mathcal{L}_{RE}$ budú líšiť.

Veta 11.1.2. *$\mathcal{L}_{rec}$ je uzavretá na komplement.*

Dôkaz. Máme TS A, ktorý sa zastaví na každom vstupe a $L(A) = L$. Zrejme môžme predpokladať normálny tvar automatu A, a to taký, že v akceptačnom stave sa TS už nehýbe. Zostrojme TS A' pre komplement jazyka L. Definujeme $F' = F^c$. Pozrieme sa na stavy a písmenká, pre ktoré TS stojí. Ak stojí na neakceptačnom stave, potom sa δ-funkciacia v novom TS A' dodefinuje na skok do nového akceptačného stavu. Staré akceptačné stavy do množiny F' nezahrnieme. Potom bude platiť $L(A') = L(A)^c$. □

Teraz ukážeme, že $\mathcal{L}_{RE}$ na komplement uzavretá nie je. Na to využijeme nasledujúcu lemu, ktorá hovorí, že pomocou TS (tzv. univerzálny TS) môžeme napodobňovať prácu iných TS. Pre daný vstup - kód TS A a slovo w nám tento univerzálny TS simuluje prácu TS A na slove w.

Lema 11.1.1. *Existuje UNIVERZÁLNY TS U taký, že vstup "<kód A> \$ w" akceptuje práve vtedy, ak $w \in L(A)$.*

Dôkaz. Myšlienka dôkazu nie je ťažká a formálne sa nám to tu nechce písať. Čitateľ si tento dôkaz môže napísať ako domáce cvičnie. Taktiež je možné nahliadnúť do literatúry ([1]). □

Veta 11.1.3. *$\mathcal{L}_{RE}$ nie je uzavretá na komplement.*

Dôkaz. Zatiaľ vieme len o jazyku L_D^c, že nie je z $\mathcal{L}_{RE}$. Teda ak ukážeme, že $L_D \in \mathcal{L}_{RE}$, tak máme, že $L_D^c \notin \mathcal{L}_{RE}$ a teda $\mathcal{L}_{RE}$ nie je uzavretá na komplement.
Ako zostrojiť TS A pre jazyk L_D ? My vieme, že o každom slove $w_i \in \{0,1\}^*$ vie rozhodnúť TS A_i. Tak sme ten jazyk L_D konštruovali. Teda podľa toho, aký vstup dostaneme, taký TS by sme mali spustiť. Zostrojme teda TS A pre L_D:

1. Pre vstup w vyrobíme na páske <kód A> \$ w. Čiže $w\$w$, lebo kód TS A_w je podľa našej dohody w.
2. Spustí na $w\$w$ stroj U z lemy 11.1.1.

Zrejme $L(A) = L_D$. Teda $L_D \in \mathcal{L}_{RE}$ a pretože $L_D^c \notin \mathcal{L}_{RE}$, tak $\mathcal{L}_{RE}$ nie je uzavretá na komplement. □

Praktický príklad univerzálneho TS je to, čo máme zadrátované v počítači. Tak ako počítač interpretuje programy, tak tento TS interpretuje iné TS, ktoré sme naprogramovali[3].
Zadefinujme teraz univerzálny jazyk, ktorý veľmi úzko súvisí s univerzálnym TS.

[2] δ -fcia nie je definovaná
[3] Napísať δ -fciu je vlastne programovanie TS

Definícia 11.1.2. *Univerzálny jazyk L_U je definovaný nasledovne:*

$$L_U = \{v\#w \mid w \in L(A_v)\}$$

Načrtneme tu teraz jednu predstavu o triedach $\mathcal{L}_{rec}$ a $\mathcal{L}_{RE}$. TS si môžeme predstaviť ako krabičku

1. s garanciou, že po určitom čase TS povie jednu z odpovedí $<^{\text{áno}}_{\text{nie}}$. Potom tento TS patrí do $\mathcal{L}_{rec}$.
2. bez vstupu, ktorá produkuje slová, v akom poradí nevedno, ale každé slovo z toho jazyka časom vyprodukuje. Takýto TS patrí do $\mathcal{L}_{RE}$. Preto aj ten názov REKURZÍVNE VYMENOVATEĽNÉ (rekurzívne vyčísliteľné).

Pomocou tohto intuitívneho rozdieľu veľmi ľahko dokážeme nasledujúce tvrdenie.

Veta 11.1.4. *Ak pre nejaký jazyk L platí $L \in \mathcal{L}_{RE}$ a $L^c \in \mathcal{L}_{RE}$, potom $L \in \mathcal{L}_{rec}$.*

Dôkaz. Opäť len naznačíme. Pre vstup w máme rozhodnút v konečnom čase, či $w \in L$, alebo nie. Spustíme obidva TS (pre L a pre L^c) naraz. Vieme, že po istom čase nám jeden z nich povie áno. Podľa toho, ktorý to bol, sa rozhodneme, či akceptujeme, alebo nie. Stroje musia robiť systémom: každý chvíľku ťahá píľku, čím odsimulujeme ich paralelný výpočet. □

11.2 Turingova téza

Definícia 11.2.1. *Hovoríme, že problém je turingovsky riešiteľný, ak existuje DTS, ktorý zastaví na každom vstupe a ktorý rieši daný problém.*

[teda ak vhodne zakódovaný problém je rekurzívny jazyk]

Téza 11.2.1. *Turingovská riešiteľnosť je ekvivalentná algoritmickej riešiteľnosti.*

(: Čo je algoritmicky riešiteľné, je riešiteľné aj na turingových strojoch. :)

Poznámka 11.2.1. Keďže prijmeme Turingovu tézu za platnú, odteraz budeme hovoriť zjednotene RIEŠITEĽNOSŤ nám ekvivalentným pojmom algoritmická a turingovská riešiteľnosť.

Definícia 11.2.2. *Hovoríme, že problém je rozhodnuteľný, ak je riešiteľný a riešenie je odpoveď $<^{áno}_{nie}$.*

Poznámka 11.2.2. Z týchto definícií nám vyplýva $L \in \mathcal{L}_{rec} \Leftrightarrow$ problém príslušnosti slova do jazyka je rozhodnuteľný.

11.3 Problém zastavenia pre TS

Predstavme si úlohu naprogramovať softvér, ktorý pre daný program a dáta rozhodne, či sa program na týchto dátach dopočíta, alebo nie. Ukážeme, že taký softvér nie sme schopní vo všeobecnosti napísať.

Sformulované do nášho jazyka: Určiť pre daný (kód) DTS A a slovo w, či stroj A spustený na w zastaví, alebo nie.

Veta 11.3.1. *Problém zastavenia pre TS je nerozhodnuteľný*[4].

[4] teda neexistuje DTS, ktorý zastaví na každom vstupe a pre ľubovoľný vstup (A, w) rozhodne, či A na w zastaví, alebo nie

[Vhodne zakódovaný problém $L_{HALT} := \{<$kód $A > \$\ w \mid A$ na w zastaví$\}$. Veta hovorí, že $L_{HALT} \notin \mathcal{L}_{rec}$. (Teda $\nexists$ DTS taký, ktorý zastaví na každom vstupe a ...)]

Dôkaz. Vetu dokážeme sporom. Ukážeme, že ak $L_{HALT} \in \mathcal{L}_{rec}$, potom aj $L_D \in \mathcal{L}_{rec}$. Nech A_{HALT} je DTS, ktory vždy zastaví pre L_{HALT}. Zostrojme A pre L_D takto:

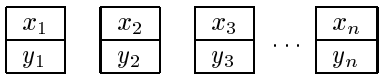
1. A na w vyrobí <kód $A_w > \$\ w$, kde TS A_w je v normálnom tvare (v akceptačnom sa nehýbe)
2. A simuluje na <kód $A_w > \$\ w$ stroj A_{HALT} a ak ten povie nezastaví, A zastaví a neakceptuje a ak povie zastaví, tak A simuluje A_w (môže, lebo A_w zastaví) a akceptuje ak A_w akceptuje.

Teda máme dôkaz, ktorý na každom vstupe zastaví a akceptuje L_D, čo je spor, lebo $L_D \notin \mathcal{L}_{rec}$. Všetko, čo sme robili, je ľahko konštruovateľné (čo sa týka algoritmickej stránky). Jediné, na čom to mohlo stroskotať, je zlý TS pre A_{HALT}. Teda taký stroj nemôže existovať. □

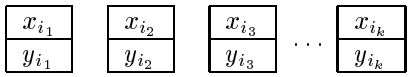
11.4 Postov korešpondenčný problém

Rozhodnúť pre dané n a dve n-tice neprázdnych slov $x = (x_1, x_2, \ldots, x_n)$, $y = (y_1, y_2, \ldots, y_n)$, či existuje postupnosť indexov $i_1, i_2, \ldots, i_k$, $k \geq 1$, taká, že $x_{i_1} x_{i_2} \ldots x_{i_k} = y_{i_1} y_{i_2} \ldots y_{i_k}$. Indexy i_j nemusia byť nutne rôzne[5].

Iná, názornejšia interpretácia tejto úlohy je **dominová verzia**:
Máme nejaké domináa



Každé domino máme v neobmedzenom množstve. Úlohou je poskladať tie domináa k sebe takým spôsobom, aby sa vrchné poschodie rovnalo spodnému:



Ukážeme, že tento problém nie je rozhodnuteľný, teda že nie je algoritmicky riešiteľný. Zdôrazňujeme, že riešením tohto problému nie je hľadanie indexov i_j, ale len odpoveď, či taká postupnosť indexov existuje. Modifikujeme trochu tento PKP a ten potom zredukujeme na problém zastavenia TS.

11.4.1 Modifikovaný Postov korešpondečný problém

Je rovnaký ako PKP, ale vyžadujeme, aby $i_1 = 1$. Teda chceme vedieť, či existuje postupnosť indexov $i_2, i_3, \ldots, i_k$ taká, že $x_1 x_{i_2} x_{i_3} \ldots x_{i_k} = y_1 y_{i_2} y_{i_3} \ldots y_{i_k}$.

Poznámka 11.4.1. Ak v PKP existuje riešenie, v modifikovanom PKP riešenie existovať nemusí. Napríklad:

[5] Môžu sa nám opakovať tie isté slová. Ak by sme to zakázali, jednochu by sme vyzkúšali všetky možnosti (bol by ich konečný počet) a vedeli by sme, či riešenie existuje

$\boxed{\begin{array}{|c|}\hline 11 \\ \hline 1 \\ \hline \end{array}}$ $\boxed{\begin{array}{|c|}\hline 2 \\ \hline 2 \\ \hline \end{array}}$

Obrátene to už platí, ak v modifikovanom PKP (MPKP) existuje riešenie, tak pre tie isté domináriviálne existuje aj v PKP. Platí však nasledujúce tvrdenie.

Veta 11.4.1. *PKP je rozhodnuteľný práve vtedy, ak je rozhodnuteľný MPKP.*

Poznámka 11.4.2. Táto veta hovorí o algoritmoch, nie o konkrétnych úlohách, kde implikácia medzi výrokmi neplatí.

Dôkaz. Dokážme obe implikácie.

$\Leftarrow$

Máme dokázať: ak je MPKP rozhodnuteľný, potom je rozhodnuteľný aj PKP. *(: t.j. ak máme algoritmus pre MPKP, potom existuje aj algoritmus pre PKP :)* Ak A je TS [6]pre rozhodnuteľnosť MPKP, potom TS A' pre rozhodnuteľnosť PKP zostrojíme takto: A' spustí v cykle od 1 po n pre dané x, y simuláciu stroja A na prípade $(x^1, y^1), \ldots, (x^n, y^n)$, ktoré získa z (x, y) cyklickým posunom komponent. Ak na nejakom takomto probléme povie A existuje, tak A' tiež povie existuje. Ak má PKP riešenie, tak musí začínať nejakým dominom, to sa mu skúšame podstrčiť v prípade MPKP, preto je náš dôkaz týmto podaný.

$\Rightarrow$

Teraz dokážme: ak je PKP rozhodnuteľný, tak je rozhodnuteľný aj MPKP. Majme teda MPKP a algoritmus A pre PKP. Ako to naňho našiť? Ako prinútiť A, aby začal prvým dominom? Myšlienka je nejak pozmeniť slová x_i a y_i, aby jediným možným počiatočným dominom bolo práve to prvé.
Zoberme si prvé domino $\boxed{\begin{array}{|c|}\hline x_1 \\ \hline y_1 \\ \hline \end{array}}$ a urobme z neho $\boxed{\begin{array}{|c|}\hline \#x_1\# \\ \hline \#y_1 \\ \hline \end{array}}$, kde znak $\#$ dáme medzi každé písmeno vnútri slov x_1 a y_1. Ostatné dominá (ale ešte aj prvé) prispôsobíme tak, aby sa dali za prvé pekne napájať. Teda domino $\boxed{\begin{array}{|c|}\hline x_i \\ \hline y_i \\ \hline \end{array}}$, $i \in \{1, 2, \ldots, n\}$ bude $\boxed{\begin{array}{|c|}\hline x_i\# \\ \hline \#y_i \\ \hline \end{array}}$, kde znak $\#$ dáme opäť medzi všetky písmená slov x_i a y_i. Na záver ešte vyrobíme domino $\boxed{\begin{array}{|c|}\hline \# \\ \hline \#\# \\ \hline \end{array}}$, ktorým môžeme ukončiť prikladanie domín. Formálne, k danému prípadu MPKP $(\overline{x}, \overline{y})$ zostrojíme prípad PKP (x', y') takto:

$$\begin{array}{llll} x'_1 = \#h_1(x_1) & x'_i = h_1(x_{i-1}) & i \in \{2, 3, 4, \ldots, n+1\} & x'_{n+2} = \# \\ y'_1 = h_2(y_1) & y'_i = h_2(y_{i-1}) & & y'_{n+2} = \#\# \end{array}$$

a pre homomorfizmy h_1, h_2 platí $\begin{array}{l} h_1(a) = a\# \\ h_2(a) = \#a \end{array}$ $\forall a \in \Sigma$

Ľahko vidno, že $i_1, i_2, \ldots, i_k$ je riešením MPKP pre $(\overline{x}, \overline{y})$ práve vtedy, keď $1, i_{2+1}, i_{3+1}, \ldots, i_{k+1}, n+2$ je riešením PKP pre (x', y').

□

Veta 11.4.2. *MPKP je nerozhodnuteľný.*

[6]rozumieme tým deterministický, ktorý zastaví na každom vstupe

Dôkaz. Sporom a redukciou na problém zastavenia. Predpokladajme, že je MPKP rozhodnuteľný, t.j. existuje algoritmus (TS A) a pre prípad MPKP (x,y) rozhodne, či existuje riešenie. Na základe tohto predpokladu zostrojíme algoritmus A' pre rozhodnutie problému zastavenia. To bude spor, lebo taký algoritmus, ako už dobre vieme, neexistuje.

Vstup pre TS A' bude kód stroja D a slovo w. Na základe vstupu si urobíme domináa, na ktoré spustíme algoritmus A. Tými dominami budeme simulovať výpočet automatu D na slove w (to jest postupnosť konfigurácií oddelených znakom #). Pričom chceme, aby keď sa TS zastaví, tak MPKP mal riešenie a obrátene, keď sa TS zacyklí, tak aby sme mohli prikladať domina tiež do nekonečna a bez úspechu.

Naším prvým dominom (ktorým podľa definície MPKP začneme) bude $\boxed{\begin{array}{|c|}\hline \#q_0w\# \\ \hline \# \\ \hline \end{array}}$, kde máme na hornom poschodí počiatočnú konfiguráciu TS D a na ktorom bude prebiehať simulácia výpočtu tohto stroja. Teraz musíme nájsť domino, ktoré priložíme a na hornom poschodí urobíme jeden krok výpočtu. Ak máme $\boxed{\begin{array}{|c|}\hline \#q_0aw_1\# \\ \hline \# \\ \hline \end{array}}$ a $\delta(q_0, a) = (p, b, 1)$ tak si definujeme domino $\boxed{\begin{array}{|c|}\hline bp \\ \hline q_0a \\ \hline \end{array}}$. Po priložení teda budeme mať $\boxed{\begin{array}{|c|}\hline \#q_0aw_1\#bp \\ \hline \#q_0a \\ \hline \end{array}}$. Zrejme budeme ďalej potrebovať dominá $\boxed{\begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array}}$ pre všetky $a \in \Gamma$ aby sme mohli ďalej prikladať[7]. Zhrňme si naše uvažovanie a ukážme dominá (n-tice slov x_i a y_i), ktoré budeme potrebovať:

$\overline{X}$	$\overline{Y}$	
$\#q_0w\#$	$\#$	
a	a	$\forall a \in \Gamma_D$
pb	qa	$(p, b, 0) = \delta_D(q, a)$
bp	qa	$(p, b, 1) = \delta_D(q, a)$
pcb	cqa	$(p, b, -1) = \delta_D(q, a)$
$\#pBb$	$\#qa$	$(p, b, -1) = \delta_D(q, a)$
$\#$	$\#$	

Pre situáciu, keď je TS D na BLANK-u, dostaneme nasledovné

$pb\#$	$q\#$	$(p, b, 0) = \delta_D(q, B)$
$bp\#$	$q\#$	$(p, b, 1) = \delta_D(q, B)$
$pcb\#$	$cq\#$	$(p, b, -1) = \delta_D(q, B)$

Ľavé krajné BLANK-y vybavíme podobne.

Týmito dominami vieme prikladaním simulovať výpočet TS D. A nič iné, ako výpočet tými dominami neurobíme.

- Ak sa výpočet TS D na w nezastaví, tak si dominá môžeme prikladať koľko len chcem, nikdy neskončím.
- Ak sa výpočet TS D ale zastaví, tak zrejme pre takú kombináciu stavu a písmenka nie je definovaná δ fcia. Teda nemáme ani žiadne domino, ktoré by sme priložil. Teraz ale chceme, aby MPKP mal riešenie. Preto zavedieme nejaké nové domino a s ním do výpočtu nový symbol Q, ktorým sa podarí vyrovnať rozdieľ v hornej a dolnej časti pospájaných

[7] do ohňa

domín (náskok, ktorý získalo poschodie $\overline{x}$ použitím prvého domina a ktorý sa zatiaľ $\overline{y}$ snaží márne dobehnúť).

Nový symbol Q na hornom poschodí bude postupne požierať symboly okolo seba. V každej konfigurácii (postupnosť symbolov oddelených znakom #) zožerie na hornom poschodí jeden symbol, čím sa rozdieľ medzi oboma poschodiami zemnší o 1. Teda nové domináa budú takéto:

$$\begin{array}{cc} \overline{X} & \overline{Y} \\ Q & qa \quad \text{ak } \delta(q,a) = \emptyset \\ Q & Qa \\ Q & aQ \end{array}$$

Týmto spôsobom Q okolo seba vymaže všetky symboly a skončí v takomto konci $\begin{array}{|c|}\hline \#Q\# \\ \hline \# \\ \hline \end{array}$.

Takže teraz už len zdárne dokončiť naše započaté dielo: $\begin{array}{|c|}\hline \# \\ \hline Q\#\# \\ \hline \end{array}$.

Zrejme prípad MPKP pre takúto sadu domín má riešenie práve vtedy, ak D na w zastaví. Vzhľadom na to, že konštrukcia prípadu MPKP zo vstupu D a w je algoritmická, jediná problematická časť pri A' je práve využitie algorimu A, preto A nemôže existovať. □

Veta 11.4.3. *PKP je nerozhodnuteľný.*

Dôkaz. Tvrdenie vyplýva z viet 11.4.1 a 11.4.2. □

11.5 Štandardné problémy pre bezkontextové jazyky

V tejto časti budeme zisťovať, či daná bezkontextová gramatika generuje prázdnu množinu, konečný resp. nekonečný jazyk, Σ^*, či prienik dvoch bezkontextových jazykov je prázdna množina a podobne. Kompletná tabuľka týchto problémov aj pre ostatné gramatika je uvedená v našom texte neskôr.

Vo všetkých prípadoch budeme uvažovať ako vstup pre problémy bezkontextové gramatiky[8].

Problém prázdnosti prieniku pre jazyky $\mathcal{L}_{CF}$

Zistiť predané L_1,L_2, či $L_1 \cap L_2 = \emptyset$. Niekedy tiež podávaný ako problém frustrovaného programátora [9] takto: Chudák programátor programuje v C-čku a každý kompilátor je iný a nie všetko mu ten či onen zožerie. Normy nie sú až tak jednotné a tak chce vedieť (ak budeme na chvíľku predpokladať, že jazyk C je bezkontextový), či existuje program, ktorý skompilujú dva kompilátory tohto jazyka od rôznych firiem.

Veta 11.5.1. *Problém prázdnosti prieniku bezkontextových jazykov je nerozhodnuteľný*[10].

Dôkaz. Sporom. Redukciou na PKP. Nech teda existuje algoritmus A na rozhodnutie prázdnosti prieniku $\mathcal{L}_{CF}$. Ukážme, že potom vieme zostrojiť algoritmus A' pre PKP. Majme na vstupe sadu domín $(\overline{x}, \overline{y})$ pre prípad PKP. Treba zostrojiť vstup pre A, t.j. dve bezkontextové gramatiky

[8] na výber boli aj zásobníkové automaty

[9] viď Rovanove prednášky (naživo)

[10] nevieme, či to frustrovanovému programátorovi pomôže, ale hovorí mu to, že je to ťažká vec, že na to nejaký univerzálny postup neexistuje

G_1, G_2 tak, aby ich slová simulovali nejak to prikladanie domín. Jazyk L_1 horné poschodie a jazyk L_2 dolné. Teda aby sme mali

$$L(G_1) \cap L(G_2) = \emptyset \Leftrightarrow \text{prípad } (\overline{x}, \overline{y}) \text{ PKP nemá riešenie}$$

Predpokladajme pre jednoduchosť, že $x_i, y_i \in \{a, b\}^+$. Nech $\overline{x} = (x_1, x_2, \ldots, x_n)$. Definujme jazyk $L(\overline{x})$ nad abecedou $\{1, 2, \ldots, n, a, b, \clubsuit\}$:

$$L(\overline{x}) := \{i_1 i_2 \ldots i_k \clubsuit x_{i_k}^R x_{i_{k-1}}^R \ldots x_{i_1}^R \mid k \geq 1,\ 1 \leq i \leq n\}$$

Zrejme $L(\overline{x}) \in \mathcal{L}_{CF}$ a platí $w \in L(\overline{x}) \cap L(\overline{y}))$ práve vtedy, keď existuje riešenie prípadu $(\overline{x}, \overline{y})$ PKP. Keďže PKP je nerozhodnuteľný, je nerozhodnuteľný aj problém prázdnosti prieniku bezkontextových jazykov. □

Zamyslime sa teraz nad niektorými problémami, ktoré sú pre $\mathcal{L}_{CF}$ rozhodnuteľné.

Problém prázdnosti jazyka $\mathcal{L}_{CF}$

Máme algoritmicky určiť, či $L(G) = \emptyset$.

Veta 11.5.2. *Problém prázdnosti pre bezkontexové jazyky je rozhodnuteľný.*

Dôkaz. Pôjdeme na to cez redukované gramatiky (kap. 2). Pre danú gramatiku G zostrojíme množinu neterminálov, z ktorých sa dá odvodiť terminálne slovo. Ak do tej množiny patrí aj σ, potom je $L(G) \neq \emptyset$, ináč $L(G) = \emptyset$. □

Problém konečnosti jazyka $\mathcal{L}_{CF}$

Máme nájsť algoritmus, ktorý pre danú bezkontextovú gramatiku G zistí, či $L(G)$ je konečná množina.

Veta 11.5.3. *Je rozhodnuteľné určiť pre ľubovoľnú bezkontextovú gramatiku G, či je jazyk $L(G)$ konečný.*

Dôkaz. Podľa p-q lemy (veta 5.5.1) existuje nejaké číslo p také, že pre všetky slová $w \in L(G)$; $|w| > p$ môžeme nejaký ich neprázdny úsek pumpovať. To ale zabezpečí nekonečnosť jazyka, my si môžeme takto napumpovať nekonečne veľa slov. Takže nám stačí vedieť, či v $L(G)$ existuje slovo dlhšie ako p :

$$L(G) \cap \Sigma^{p+1}\Sigma^* \neq \emptyset \Leftrightarrow L(G) \text{ je nekonečný}$$

Keď si uvedomíme, že jazyk $\Sigma^{p+1}\Sigma^*$ je regulárny, a že číslo p vieme algoritmicky zistiť (viď pupovaciu lemu pre bezkontextové jazyky), tak sa nám tento problém redukuje na problém prázdnosti bezkontextového jazyka. □

Problém príslušnosti slova do daného jazyka L

Pre danú gramatiku G nájsť algoritmus, ktorý by pre ľubovoľné slovo w určil, či $w \in L(G)$.

Veta 11.5.4. *Problém príslušnosti slova do daného jazyka L je pre $L \in \mathcal{L}_{CF}$ rozhodnuteľný*[11].

Dôkaz. Algoritmus bude pracovať takto:

1. ak $w = \varepsilon$, tak stačí zistť, či σ patrí do množiny vymazávajúcich neterminálov (kap. 2).

[11] automat na to rozhodne nemôžeme použiť, lebo je nedeterministický

2. ak $w \neq \varepsilon$, tak môžeme predpokladať, že gramatika G je v Greibachovej normálnom tvare (kap. 2), čo je úkon, ktorý vieme urobiť algoritmicky. Potom ak $w \in L(G)$, tak jeho odvodenie má najviac $|w|$ krokov. Takýchto odvodení je ale konečne veľa, tak ich všecky vyskúšajme . . . .

□

Toto boli asi tak všecky rozhodnuteľné problémy, ktoré nás zaujímajú, pre $\mathcal{L}_{CF}$. Dôkaz poslednej vety sa dá robiť oveľa elegantnejšie a to pomocou dynamického programovania. Uvedieme teraz daný algoritmus.

Algoritmus Cocke-Younger-Kasami

Predpokladajme, že gramatika G je ε-free a v Chomského normálnom tvare. Majme slovo $w = a_1 a_2 \dots a_n$. Algoritmus bude postupne konštruovať množiny neterminálov N_{ij} také, že

$$N_{ij} = \{\xi \in N \mid \xi \stackrel{*}{\Longrightarrow} a_i a_{i+1} \dots a_{i+j-1}\}$$

pričom bude využívať množiny skôr vytvorené.

N_{1n}				
$N_{1(n-1)}$	$N_{2(n-1)}$			
$\vdots$		$\ddots$		
N_{11}	N_{21}	N_{31}	$\cdots$	N_{n1}

Prvý riadok, teda množiny $N_{11}, N_{21}, \dots, N_{n1}$, vytvoríme vďaka Chomského normálnemu tvaru raz dva. Pozrieme sa, ktoré pravidlá nám dávajú daný symbol a_i a príslušnú ľavú stranu pravidla (neterminál ξ) dáme do množiny N_{i1}. Poďme teraz budovať množinu N_{ij}. V tej majú byť tie neterminály, z ktorých sa dá odvodiť slovo $a_i a_{i+1} \dots a_{i+j-1}$. Zrejme ak $j > 1$, tak také slovo sa dá odvodiť len z neterminálu, ktorý je ľavou stranou nejakého pravidla, ktoré má na pravej strane dva neterminály (pravidlo $\xi \to \eta\vartheta$). Z neterminálu η sa ďalej musí dať odvodiť slovo $a_1 \dots a_k$ (prefix slova $a_i a_{i+1} \dots a_{i+j-1}$) a z neterminálu ϑ zasa slovo $a_{k+1} \dots a_{i+j-1}$ (suffix slova $a_i a_{i+1} \dots a_{i+j-1}$). Budeme preto kombinovať neterminály týchto množín:

$$\begin{array}{ll} N_{i(j-1)} & N_{j1} \\ N_{i(j-2)} & N_{(j-1)2} \\ \vdots & \\ N_{i1} & N_{(i+1)(j-1)} \end{array}$$

Teda ak máme v gramatike G pravidlo $\xi \to \eta\vartheta$ také, že $\eta \in N_{i(j-k)}$ a $\vartheta \in N_{(j-k+1)k}$, potom neterminál ξ dáme do množiny N_{ij}. No a nakoniec sa pozrieme, či je σ v množine N_{1n}.

Poznámka 11.5.1. Zložitosť tohto algoritmu je $O(n^3)$, kým toho z dôkazu je exponeciálna.

Na ďalšie hókusy-pókusy budeme potrebovať nasledujúce jazyky. Pre dané postupnosti $(\overline{x}, \overline{y})$ slov nad abecedou $\{a, b\}$ definujeme:

$$L(x, y) := L(x)\mathbin{\text{¢}}L^R(y)$$

$$L_{SYM} := \{u\text{¢}v\text{¢}v^R\text{¢}u^R \mid u \in \{1, 2, \dots, n\}^*, v \in \{a, b\}^*\}$$

Poznámka 11.5.2. Dopustili sme sa tu malej nepresnosti – keďže nemáme fixnú abecedu $\{1, 2, \ldots, n\}$ (n je premnelivé), tak nemáme jeden jediný jazyk L_{SYM}. Ak by sme si ale zakódovali indexy $\{1, 2, \ldots, n\}$ pomocou abecedy $\{a, b\}$, tak by sme už mali jediný jazyk L_{SYM}.

Poznámka 11.5.3. $L(x, y) \cap L_{SYM} \neq \emptyset \iff$ PKP prípad (x, y) má riešenie

Je dôležité si uvedomiť, že jazyky $L(x, y), L_{SYM} \in \mathcal{L}_{CF}$. Ďalej potrebujeme, že $L^c(x, y), L^c_{SYM} \in \mathcal{L}_{CF}$. Lenže môžeme ukázať, že $L^c(x, y), L^c_{SYM} \in \mathcal{L}_{DCF}$ a teda aj z $\mathcal{L}_{CF}$. Teda platí

$$(L(x, y) \cap L_{SYM})^c = L^c(x, y) \cup L^c_{SYM} \in \mathcal{L}_{CF}$$

Takto podané máme vlastne nasledujúce tvrdenie.

Problém generovania Σ^*

Veta 11.5.5. *Problém zistiť, či CF gramatika G generuje Σ^* je nerozhodnuteľný.*

Dôkaz. Sporom. Redukciou na PKP. Nech pre každú gramatiku G vieme určiť, či generuje Σ^*. Tak zostrojme gramatiku G pre $(L(x, y) \cap L_{SYM})^c$ (je to predsa bezkontextový jazyk). Potom zrejme platí:

$$L(G) = \Sigma^* \Leftrightarrow L(x, y) \cap L_{SYM} = \emptyset \Leftrightarrow \text{PKP pre } (\overline{x}, \overline{y}) \text{ nemá riešenie}$$

Z daného už spor vyplýva až príliš okate. □

Problém regulárnosti daného jazyka

Veta 11.5.6. *Je nerozhodnuteľné určiť pre danú bezkontextovú gramatiku G, či $L(G) \in \mathcal{R}$.*

Dôkaz. Sporom. Redukciou na PKP. Majme preto nejaký prípad $(\overline{x}, \overline{y})$ PKP. K tomuto prípadu máme jazyky $L(x, y)$ a L_{SYM}. Uvedomme si, že $L(x, y) \cap L_{SYM}$ je buďto $\emptyset$ alebo ∞. To plynie z toho, že ak má PKP riešenie, potom ich už má nekonečne veľa. Ak ten prienik nie je prázdny, potom nie je bezkontexový. Neplatí totižto pumpovacia lema. Na druhú stranu ale vieme, že $(L(x, y) \cap L_{SYM})^c \in \mathcal{L}_{CF}$. Kedy môže byť tento jazyk regulárny ? Keďže $\mathcal{R}$ je uzavretá na komplement, tak len ak je ten prienik Σ^*[12]. Takže sme dospeli k niečomu takémuto:

$$(L(x, y) \cap L_{SYM})^c \in \mathcal{R} \Leftrightarrow L(x, y) \cap L_{SYM} \in \mathcal{R} \Leftrightarrow L(x, y) \cap L_{SYM} = \emptyset \Leftrightarrow \text{PKP nemá riešenie}$$

Čo k tomu dodať? Snáď už len dokresliť štvorček . . . . □

11.6 Štandardné problémy pre štandardné jazyky

Pýtame sa, či daný problém je rozhodnuteľný (R), nerozhodnuteľný (N), prípadne či vždy platí (T).

[12] ak by tak nebolo, musel by byť aj komplement regulárny, ale keďže ten by nebol prázdny, tak podľa nášho zistenia by nebol ani bezkontexový

Problém	$\mathcal{R}$		$\mathcal{L}_{CF}$		$\mathcal{L}_{(E)CS}$		$\mathcal{L}_{RE}$
1. $L(G) = \emptyset$, konečná, ∞	R	$\Leftarrow$	R		N	$\Rightarrow$	N
2. $L(G) = \Sigma^*$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
			$\Downarrow$				
3. $L(G_1) = L(G_2)$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
			$\Downarrow$				
4. $L(G_1) \subseteq L(G_2)$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
5. $L(G_1) \cap L(G_2) = \emptyset$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
6. $L(G) = R, R$ pevná z $\mathcal{R}$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
7. $L(G) \in \mathcal{R}$	T		N	$\Rightarrow$	N	$\Rightarrow$	N
8. $L(G_1) \cap L(G_2)$ je toho istého typu	T		N	$\not\Rightarrow$	T		T

Celú tabuľku T si môžeme predstaviť ako dvojrozmerné pole a indexovať ho $T[p, L]$, kde p je číslo problému, teda riadok a L je trieda jazykov, teda stĺpec. Doplňme ešte tabuľku o informáciu, že $T[2, L_{CF}] \Rightarrow T[6, L_{CF}]$. Pozrime sa teraz bližšie na jednotlivé tvrdenia[13]:

- $T[1, \mathcal{R}] \Leftarrow T[1, \mathcal{L}_{CF}]$. Ak je $G \in \mathcal{R}$, tak je aj z $\mathcal{L}_{CF}$ a teda môžeme použiť rovnaký algoritmus.
- $T[2, \mathcal{R}]$. $L(G) = \Sigma^* \Leftrightarrow L(G)^c = \emptyset$ a $\mathcal{R}$ je uzavretá na komplement.
- $T[3, \mathcal{R}]$. Z teórie množín vyplýva, že $A = B \Leftrightarrow A - B = \emptyset \wedge B - A = \emptyset \Leftrightarrow A \cap B^c = \emptyset \wedge B \cap A^c = \emptyset$. No a keďže $\mathcal{R}$ je uzavretá na komplement, tak ....
- $T[4, \mathcal{R}]$. Podobne, ako prípad $T[3, \mathcal{R}]$.
- $T[6, \mathcal{R}]$. Podobne, ako prípad $T[3, \mathcal{R}]$.
- $T[2, \mathcal{L}_{CF}] \Rightarrow T[3, \mathcal{L}_{CF}]$. Ak by bolo $T[3, \mathcal{L}_{CF}]$ rozhodnuteľné, tak za $L(G_2)$ dáme Σ^* a mali by sme to.
- $T[3, \mathcal{L}_{CF}] \Rightarrow T[4, \mathcal{L}_{CF}]$. Dve inklúzie znamenajú rovnosť. Takže by sme vedeli $T[3, \mathcal{L}_{CF}]$.
- $T[2, \mathcal{L}_{CF}] \Rightarrow T[6, \mathcal{L}_{CF}]$. Táto myšlienka už tu niekde bola spomenutá ....

Zostal už len problém $T[1, \mathcal{L}_{(E)CS}]$, ale ten by čitateľ mohol dokázať sám. Zrejme sporom a nejakou redukciou.

Poznámka 11.6.1. Problém $L(G_1) \subseteq L(G_2)$ pre $\mathcal{L}_{CF}$ je nerozhodnuteľný, avšak pre špeciálne prípady gramatík $R \in \mathcal{R}$, $G \in \mathcal{L}_{CF}$ dostávame:

- $R \subseteq L(G)$ ako nerozhodnuteľný problém (sporom: potom nech $R = \Sigma^*$ a vedeli by sme rozhodnúť generovanie Σ^*).
- $L(G) \subseteq R$ ako rozhodnuteľný problém ($L(G) \subseteq R \Leftrightarrow L(G) \cap R^c = \emptyset$ a problém prázdnosti jazyka pre $\mathcal{L}_{CF}$ je rozhodnuteľný).

11.7 Nejaké redukcie pre TS

Najskôr zopár zaujímavých a potrebných jazykov.

[13] tak totižto túto tabuľku chápeme, každé miesto $T[i, L]$ je samostatná veta, ktorú by sme mali byť schopní vysloviť a dokázať

Jazyk platných výpočtov TS

Daný je TS A, definujeme jazyk $L_{PV(A)}$:
$L_{PV(A)} := \{w_1 \# w_2^R \# \ldots \# w_k^{R^{(k+1) \bmod 2}} \mid \forall i\ w_i$ je konfigurácia A, w_1 je poč.konfig., w_k je akceptačná, $w_i \vdash w_{i+1}\}$

Jazyk neplatných výpočtov TS

$L_{NPV(A)} := L^c_{PV(A)}$, pričom komplement sa myslí vzhľadom na množinu všetkých reťazcov.

O týchto jazykoch platia nasledujúce tvrdenia:

1. $L_{NPV(A)} \in \mathcal{L}_{CF}$
2. K ľubovoľnému TS A existujú jazyky $L_{A_1}, L_{A_2} \in \mathcal{L}_{CF}$ také, že $L_{PV(A)} = L_{A_1} \cap L_{A_2}$
3. $L(A) = \emptyset \Leftrightarrow L_{PV(A)} = \emptyset$

K bodu 2) asi toľkoto: L_{A_1} bude kontrolovať nadväznosť $w_i \vdash w_{i+1}$, kde i je neparne a L_{A_2} bude kontrolovať nadväznosť $w_i \vdash w_{i+1}$, kde i je párne.

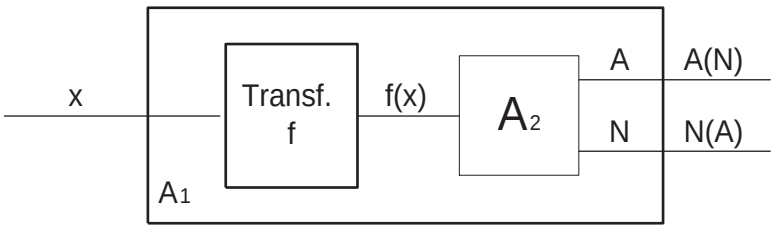
Vieme, že $L(G) = \emptyset$,resp. ∞ je pre $\mathcal{L}_{RE}$ nerozhodnuteľné a teda je to nerozhodnuteľné aj pre jazyk $L_{PV(A)}$. Tento fakt sa dá ďalej využiť pri rozhodovaní iných problémov.

Ako vlastne prebieha redukcia vo všeobecnosti si ukážeme teraz.

Redukcia L_2 na L_1

$$L_1 \leq L_2 : \exists f \quad x \in L_1 \Leftrightarrow f(x) \in L_2 \text{ (príp. } f(x) \notin L_2)$$

Graficky znázorňujeme diagramom takto:



Príklad 11.7.1. Dá sa ukázať, že $L_u \in \mathcal{L}_{RE}$, ale $L_u \notin \mathcal{L}_{rec}$. Požije sa redukcia $L_D \leq L_u$, z čoho už vyplýva $L_u \notin \mathcal{L}_{rec}$.

Definícia 11.7.1. *Nech $\mathcal{S}$ je vlastnosť jazyka v L_{RE}, potom definujeme jazyk*

$$L_{\mathcal{S}} := \{<A> \mid L(A) \textit{ má vlastnosť } \mathcal{S}\}$$

Poznámka 11.7.1. Ak je vlastnosť rozhodnuteľná, tak musí existovať DTS A, ktorý zastaví na každom vstupe a teda máme, že $L_{\mathcal{S}} \in \mathcal{L}_{rec}$.

Prázdnosť pre TS

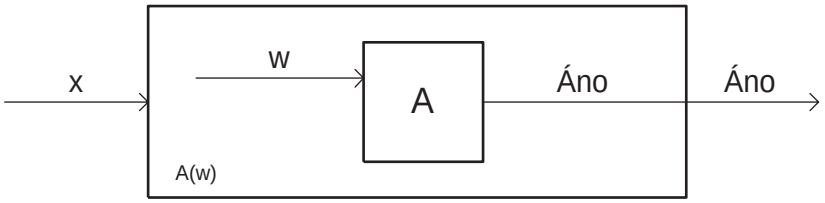
Našou vlastnosťou $\mathcal{S}$ je prázdnosť, preto dostávame jazyk $L_{pr} := \{<A> \mid L(A) = \emptyset\}$.

Veta 11.7.1. *Je nerozhodnuteľné určiť pre daný TS A, či $L(A) = \emptyset$, tj. $L_{pr} \notin \mathcal{L}_{rec}$*

Dôkaz. Sporom. Redukciou na L_u. Nech $L_{pr} \in \mathcal{L}_{rec}$, teda existuje DTS A_{pr}, ktorý zastaví na každom vstupe a $L(A_{pr}) = L_{pr}$. Zostrojíme DTS C, ktorý zastaví na každom vstupe a $L(C) = L_u$.

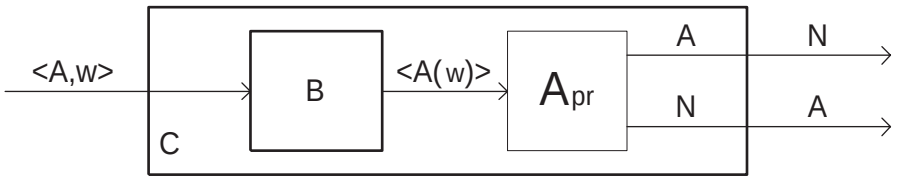
1. Nech B je DTS, ktorý pre každé $<A, w>$ zostrojí <A(w)> takého TS $A(w)$, že

   $L(A(w)) = \emptyset$ ak $w \notin L(A)$
   $L(A(w)) = \Sigma^*$ ak $w \in L(A)$

   Ako vyzerá TS $A(w)$? Vôbec si nebude všímať vstup, ale začne simulovať automat A na w. Ako ale skonštruovať TS B algoritmicky? Vieme vytvoriť TS X na zmazanie vstupu a skopírovanie slova w. To bude deterministický TS. Ďalej máme na vstupe kód TS A aj kód slova w. Ak TS X používal n stavov, tak ich všetky skopírujeme pred kód TS A spolu s δ-fciou a stavy TS A (ich kódy) inkrementneme o n. Toto všetko vieme algoritmicky uskutočniť, teda taký DTS B viem vytvoriť.

2. Zostrojíme TS C takto.

□

Poznámka 11.7.2. $L_{pr}^c = L_{nepr} \in \mathcal{L}_{RE}$. Prečo ?

- Nedeterministicky - uhádneme slovo a na tom spustíme A
- Deterministicky - budeme robiť výpočty na všetkých slovách, raz to jedno musíme akceptovať (v každom kroku budeme na pásku pridávať jedno nové slovo v lexikografickom usporiadaní, odsimulujeme na každom slove zopár krokov a zasa pridáme nové slovo a tak stále dokola).

Rekurzívnosť množiny $L(A)$ pre TS A

Pýtame sa, či pre množinu, ktorú vygeneruje nejaký TS A, existuje TS $A' \in \mathcal{L}_{rec}$ taký, že $L(A) = L(A')$.
$L_r := \{<A> \mid L(A) \in \mathcal{L}_{rec}\}$
$L_{nr} := \{<A> \mid L(A) \notin \mathcal{L}_{rec}\}$

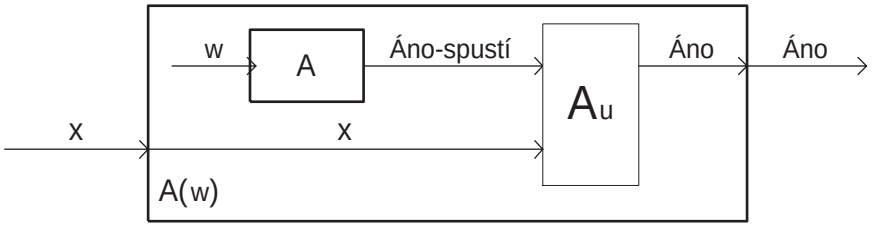
Veta 11.7.2. *$L_r \notin \mathcal{L}_{RE}$, $L_{nr} \notin \mathcal{L}_{RE}$.*

Dôkaz. Dokážme obe tvrdenia.

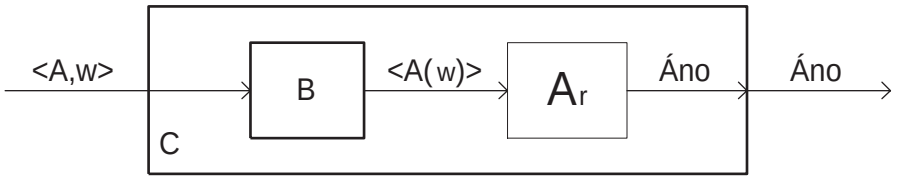
$L_r \notin \mathcal{L}_{RE}$ Predpokladajme, že $L_r \in \mathcal{L}_{RE}$, teda že existuje TS A_r taký, že $L(A_r) = L_r$. Ukážeme, že potom vieme zostrojiť TS C pre L_u^c.

1. Nech B je DTS, ktorý pre $<A, w>$ zostrojí <A(w)> takého TS $A(w)$, že

   $L(A(w)) = \emptyset$ ak $w \notin L(A)$
   $L(A(w)) = L_u$ ak $w \in L(A)$



2. Zostrojíme TS C pre L_u^c.



$L_{nr} \notin \mathcal{L}_{RE}$ Predpokladajme, že $L_{nr} \in \mathcal{L}_{RE}$, teda že existuje TS A_{nr} taký, že $L(A_{nr}) = L_r$. Ukážeme, že potom vieme zostrojiť TS C pre L_u^c.

1. Nech B je DTS, ktorý pre $<A, w>$ zostrojí <A(w)> takého TS, že

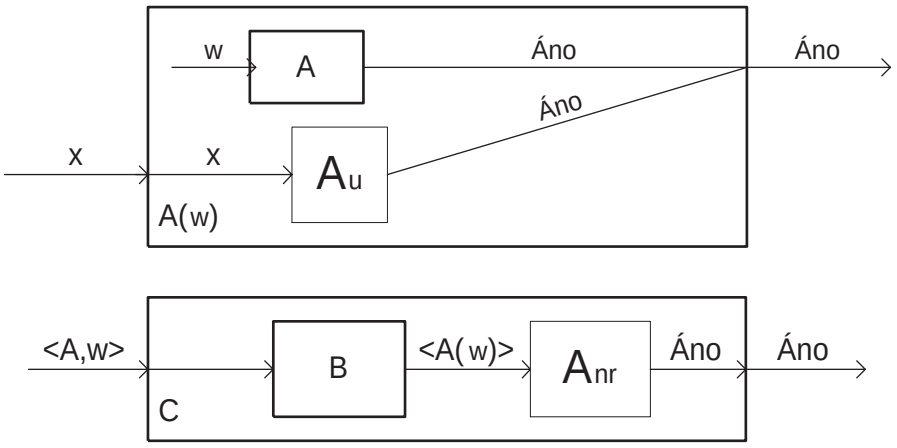
   $L(A(w)) = \Sigma^*$ ak $w \in L(A)$
   $L(A(w)) = L_u$ ak $w \notin L(A)$

2. Zostrojíme TS C pre L_u^c.

□

11.8 Riceho vety

Vieme, že každá vlastnosť rekurzívne vyčísliteľných jazykov (t.j. množina jazykov majúcich túto vlastnosť) je podmnožinou $\mathcal{L}_{RE}$. Otázkou zostáva, kedy je podmnožinou $\mathcal{L}_{rec}$. Teda kedy je tá vlastnosť rozhodnuteľná. Na túto otázku nám dáva odpoveď nasledujúca veta.



Definícia 11.8.1. *Vlastnosť $\mathcal{S}$ jazykov $\mathcal{L}_{RE}$ sa nazýva netriviálna, ak existujú jazyky $L_1, L_2 \in \mathcal{L}_{RE}$ také, že $L_1 \in \mathcal{S}$ a $L_2 \notin \mathcal{S}$. Vlastnosť $\mathcal{S}$ sa nazýva triviálna, ak $\mathcal{S} = \emptyset \vee \mathcal{S} = \mathcal{L}_{RE}$.*

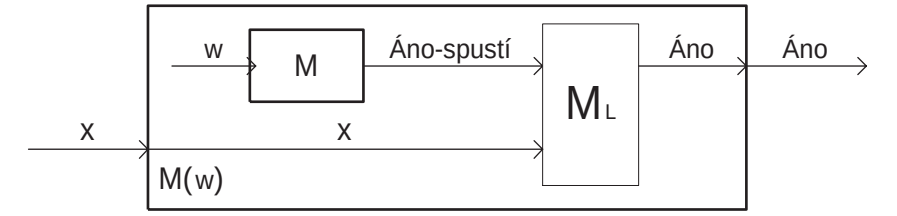
Veta 11.8.1. *(Rice) Každá netriviálna vlastnosť rekurzívne vyčísliteľných jazykov je nerozhodnuteľná.*

Dôkaz. Sporom. Redukciou na jazyk L_u. Nech $\mathcal{S}$ je netriviálna rozhodnuteľná vlastnosť a nech $\emptyset \notin \mathcal{S}$ (ak $\emptyset \in \mathcal{S}$, tak zoberieme komplement $\mathcal{S}$, ktorý musí byť tiež z $\mathcal{L}_{rec}$). Zostrojme teraz TS C, ktorý zastaví na každom vstupe a akceptuje L_u. Podľa predpokladov nech $L \in \mathcal{S}$, $L \neq \emptyset$. K nemu máme TS M_L s vlastnosťou $L(M_L) = L$.

(: K slovu w a Turingovmu stroju M vyrobíme TS $< M, w >$, o ktorom vieme len toľko, že buď akceptuje $\emptyset$ (v prípade, že $w \notin L(M)$), alebo akceptuje L (v prípade $w \in L(M)$), ale iná možnosť nie je. Jazyky $\emptyset$ a L sa líšia v tom, že $\emptyset$ má vlastnosť $\mathcal{S}$, ale L nemá vlastnosť $\mathcal{S}$. Keby sme teda vedeli rozhodnúť, či $L(< M, w >)$ má vlastnosť $\mathcal{S}$, vedeli by sme, či $L(< M, w >) = L$ alebo $L(< M, w >) = \emptyset$, a teda aj to, či $w \in L(M)$. :)

1. Nech B je DTS, ktorý pre každé $<M, w>$ zostrojí <M(w)> takého TS $M(w)$, že

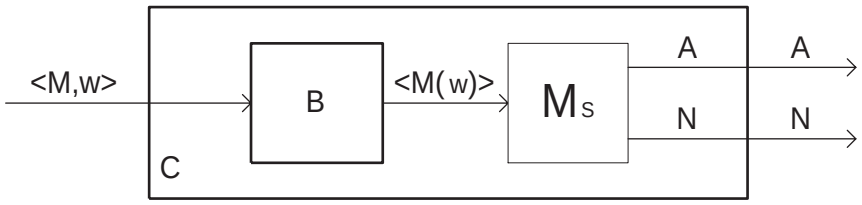
   $L(M(w)) = \emptyset$ ak $w \notin L(M)$ //tu sme využili $\emptyset \notin \mathcal{S}$
   $L(M(w)) = L$ ak $w \in L(M)$



2. Zostrojíme TS C pre L_u, ktorý zastaví na každom vsupe.

□

Dôsledok 11.8.1. Nasledujúce vlastnosti rekurzívnych jazykov sú nerozhodnuteľné.



1. Prázdnosť
2. Konečnosť
3. Regulárnosť
4. Bezkontextovosť
5. ...

Dôkaz. Každá z týchto vlastností je netriviálna (máme konečné aj nekonečné jazyky, prázdne aj neprázdne, ...). □

No a kedy je $L_{\mathcal{S}}$ rekurzívne vyčísliteľný jazyk ?

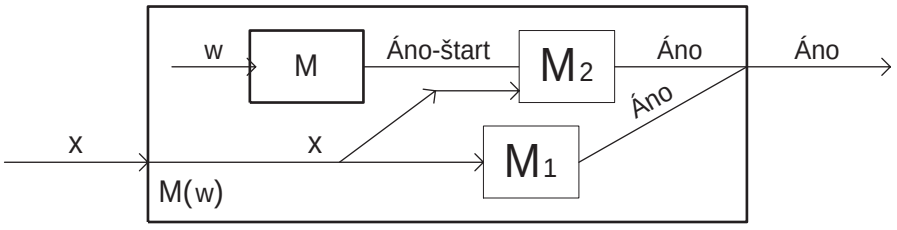
Veta 11.8.2. *(Rice)* $L_{\mathcal{S}} \in \mathcal{L}_{RE}$ *práve vtedy, keď sú splnené nasledovné tri vlastnosti:*

1. *Ak* $L \in \mathcal{S}$ *a* $L \subseteq L'$, $L' \in \mathcal{L}_{RE}$ *potom* $L' \in \mathcal{S}$.
2. *Ak* $L \in \mathcal{S}$ *je nekonečný jazyk potom existuje* $L' \subseteq L$, L' *je konečný a* $L' \in \mathcal{S}$.
3. *Množina všetkých konečných jazykov v* $\mathcal{S}$ *je rekurzívne vyčísliteľná.*

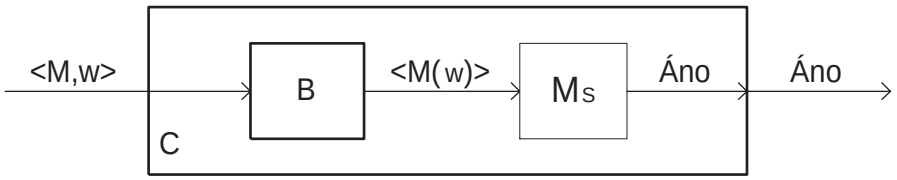
Dôkaz. Dokážeme obe implikácie.

$\Rightarrow$ Takže

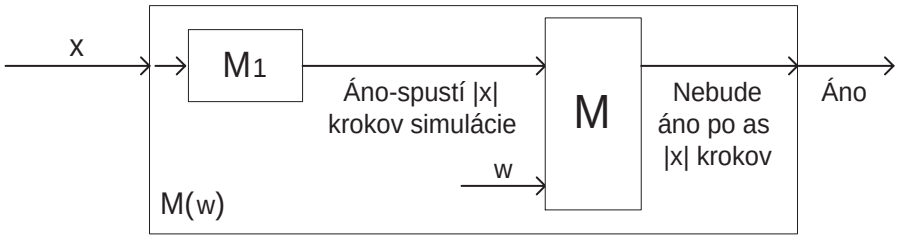
1. Ak neplatí 1, potom $L_{\mathcal{S}} \notin \mathcal{L}_{RE}$
   Sporom. Nech $L_{\mathcal{S}} \in \mathcal{L}_{RE}$ a nech $L_1 \in \mathcal{S}$, $L_1 \subseteq L_2$, $L_2 \in \mathcal{L}_{RE}$, $L_2 \notin \mathcal{S}$. Ukážeme, že potom vieme zostrojiť TS C pre L_u^c.
   (a) Nech B je DTS, ktorý pre každé $<M, w>$ zostrojí <M(w)> takého TS $M(w)$, že
   $L(M(w)) = L_2$ ak $w \in L(M)$
   $L(M(w)) = L_1$ ak $w \notin L(M)$



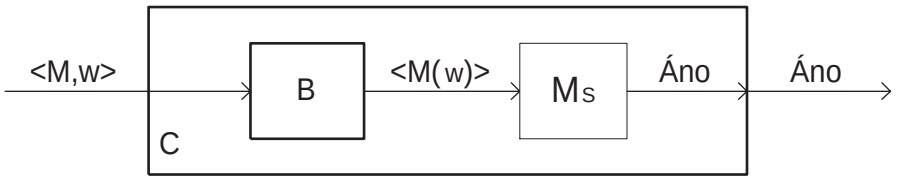
   (b) Zostrojíme TS C pre L_u^c.



2. Ak neplatí 2, potom $L_{\mathcal{S}} \notin \mathcal{L}_{RE}$
   Opäť sporom. Predpokladajme teda, že $L_{\mathcal{S}} \in \mathcal{L}_{RE}$ a že $L_1 \in \mathcal{S}$ je nekonečný jazyk a žiadna jeho konečná podmnožina L_1 nepatrí do $\mathcal{S}$. Ukážeme, že potom vieme zostrojiť TS pre jazyk L_u^c.
   (a) Nech B je DTS, ktorý pre každé $<M, w>$ zostrojí <M(w)> takého TS $M(w)$, že
   $L(M(w)) = L_1$ ak $w \notin L(M)$
   $L(M(w)) =$ nejaká konečná podmnožina L_1 ak $w \in L(M)$



   (b) Zostrojíme TS C pre L_u^c.



3. Nech $L_{\mathcal{S}} \in \mathcal{L}_{RE}$, potom platí 3
   Zostrojme TS A, ktorý postupne generuje všetky konečné prvky z $\mathcal{S}$[14]a bude pracovať takto:
   Postupne generuje dvojice čísel (i, j), kde i je kód konečnej množiny. Túto má dať na výstup, ak má vlastnosť $\mathcal{S}$. Máme TS $M_{\mathcal{S}}$ pre $L_{\mathcal{S}} = \{<M> \mid L(M) \in \mathcal{S}\}$, preto zostrojí kód TS M_i pre túto konečnú množinu i (to sa dá ľahko, napríklad konečný automat). Spustí j krokov stroja $M_{\mathcal{S}}$ na $<M_i>$. Ak $M_{\mathcal{S}}$ akceptuje, dá i na výstup. Generuje ďalšiu dvojicu (i, j). Pomocou j sme zabezpečili, že sa nám TS A nezacyklil na jednom vstupe i, čím by sme sa nedostali k všetkým konečným podmnožinám.

$\Leftarrow$ Nech M_1 je TS, ktorý generuje všetky konečné jazyky v $\mathcal{S}$. $M_{\mathcal{S}}$ zostrojíme takto:
Pre vstup $<M>$ robí $M_{\mathcal{S}}$ nasledovné:

1. $M_{\mathcal{S}}$ generuje všetky dvojice (i, j).
2. Pre každé (i, j) $M_{\mathcal{S}}$ simuluje M_1 j krokov. Ak M_1 v priebehu j krokov vygeneruje i, potom $M_{\mathcal{S}}$ simuluje M j krokov na každom slove z i. Ak behom j krokov M akceptuje každé slovo z i, potom $M_{\mathcal{S}}$ akceptuje $<M>$.

[14] konečné podmnožiny máme kódované do reťazca z $\{0, 1\}$

□

Dôsledok 11.8.2. Nasledujúce vlastnosti rekurzívne vyčísliteľných jazykov nie sú rekurzívne vyčísliteľné.

- $L = \emptyset$ porušené 1); nadmnožiny to nededia, už nie sú $\emptyset$
- $L = \Sigma^*$ porušené 2)
- $L \in \mathcal{L}_{rec}$
- $L \notin \mathcal{L}_{rec}$
- $Card(L)$[15] $= 1$
- $L \in \mathcal{R}$
- $L - L_u \neq \emptyset$ porušené 3)

Dôsledok 11.8.3. Nasledujúce vlastnosti sú rekurzívne vyčísliteľné.

$L \neq \emptyset$

$Card(L) \geq 10$

$w \in L$ pre pevné w

$L \cap L_u \neq \emptyset$

11.9 Úlohy

1. Dokážte lemu 11.1.1
2. Ak definujeme $L(U)$ ako univerzálny jazyk, kde U je univerzálny TS, bude potom komplement tohto jazyka z $\mathcal{L}_{RE}$?
3. Dopíšte dominá z dôkazu vety 11.4.2 pre prípad δ-fcie čítajúcej BLANK.
4. Dokážte, že jazyk $L(\overline{x})$ z vety11.5.1 je bezkontextový.
5. Dokážte, že jazyky $L(x,y)^c, L^c_{SYM}$ sú bezkontextové.
6. Dokážte, že problém prázdnosti, konečnosti a nekonečnosti (teda problém $T[1, \mathcal{L}_{(E)CS}]$) je pre triedu kontextových jazykov nerozhodnuteľný.

[15] Počet slov v jazyku

Kapitola 12

Syntaktická analýza

Syntaktická analýza je veľmi široká problematika, ktorá nachádza uplatnenie pri tvorbe kompilátorov alebo prekladačov prirodzených jazykov. Čitateľ si určite dobre uvedomuje, že na preklad viet z jedného jazyka do druhého nestačí mať dobrý slovník[1], ale ideálne by bolo mať v knižnici akýsi vetník, ktorý by napríklad obsahoval všetky anglické vety a ich preklady do slovenčiny. Problém by však spočíval predovšetkým v tom, že vetník by musel byť nekonečný. Práve preto treba hľadať prefíkané cesty, ako čo najjednoduchšie automatizovať preklad prirodzených jazykov[2].

Formálne možno preklad definovať ako reláciu $M \subseteq \Sigma_1^* \times \Sigma_2^*$, prípadne $M \subseteq L_1 \times L_2$. Uvedomme si, k akému posunu v terminológii tu dochádza. Totiž na to, čo v bežnom živote uznačujeme ako slová, sa v teórii formálnych jazykov pozeráme ako na písmená, veď počet slov každého prirodzeného jazyka je konečný. Ďalej na vety sa v teórii formálnych jazykov pozeráme ako na slová a jazyk je pre nás množina prirodzených viet. Podobne, syntakticky správne pascalské programy sú slová, kľúčové slová a použité symboly sú písmenami abecedy.

Pri štúdiu formálnych jazykov sme sa už s istými prekladmi stretli. Najjednoduchším prekladom bol homomorfizmus, inverzný homomorfizmus, neskôr sme sa zoznámili s A-prekladačmi. Aj na Turingov stroj sa možno pozrieť ako na prekladač.

12.1 Syntaxou riadené prekladové schémy

Syntaxou riadené prekladové schémy nápadne pripomínajú bezkontextové gramatiky. Formálne ich možno definovať nasledovne:[3]

Definícia 12.1.1. *Štoricu* $S = (N, T, P, \sigma)$, *kde* N, T *sú disjunktné, konečné množiny neterminálov, resp. terminálov,*

$$P = N \times \{u_1\xi_1u_2...\xi_ku_{k+1}; v_1\xi_1v_2...\xi_kv_{k+1} \mid \xi_1, ..., \xi_k \in N,\ u_1, u_2, ..., u_{k+1} \in T^*,$$

$$v_1, v_2, ..., v_{k+1} \in T^*\}$$

$\sigma \in N$ *nazývame prekladovou schémou.*

Definícia 12.1.2. *Prekladovou formou nazveme slovo*

$$u_1\xi_1u_2...\xi_ku_{k+1}; v_1\xi_1v_2...\xi_kv_{k+1}$$

[1] Mohlo by sa stať, že anglické *Are you off?* by takýto prekladač nepreložil správne *Odchádzaš?*, ale *Si vypnutý?*
[2] a z prekladateľov a tlmočníkov urobiť nezamestnaných
[3] Autormi formálnych definícií sú autori tohto textu, na prednáške bola táto časť odprednášaná voľne

kde

$$\xi_1, ..., \xi_k \in N,\ u_1, u_2, ..., u_{k+1}, v_1, v_2, ..., v_{k+1} \in T^*$$

Definícia 12.1.3. *Krok odvodenia prekladovej schémy S je relácia na množine prekladových foriem definovaná nasledovne:*

$$u_1\xi_1...u_i\xi_iu_{i+1}...\xi_ku_{k+1}; v_1\xi_1...v_i\xi_iv_{i+1}...\xi_kv_{k+1} \underset{S}{\Longrightarrow} u_1\xi_1...u_ixu_{i+1}\xi_ku_{k+1}; v_1\xi_1...v_iyv_{i+1}...\xi_kv_{k+1}$$

$$\overset{def}{\Longleftrightarrow}$$

$$(\xi_i \to x; y) \in P$$

Definícia 12.1.4. *Prekladom určeným programovou schémou S je množina* $\{u; v \in T^*; T^* \mid \sigma; \sigma \overset{*}{\underset{S}{\Longrightarrow}} u; v\}$

Zamyslime sa teraz voľne nad tým, čo teda prekladová schéma znamená. Ako vidíme, môžeme sa na ňu pozrieť ako na dvojicu veľmi podobných bezkontextových gramatík G_1, G_2 a na preklad ako na množinu slov $u; v$, kde slová u, v vznikli odvodením v gramatikách G_1, resp. G_2, pričom tieto odvodenia sú veľmi podobné, lebo sa používajú analogické pravidlá. Všimnime si totiž, že prekladová schéma obsahuje pravidlá, ktoré spolu s definíciou relácie kroku odvodenia spôsobujú, že každá prekladová forma je typu $u_1\xi_1...u_i\xi_iu_{i+1}...\xi_ku_{k+1}; v_1\xi_1...v_i\xi_iv_{i+1}...\xi_kv_{k+1}$.

Aj keď sme už ukázali, že Pascal nie je bezkontextový jazyk, všetky praktické prekladače kladú obmedzenia (napríklad na počet parametrov funkcie[4], ktoré zapezpečia bezkontextovosť uvažovaného programovacieho jazyka. Problémom teda je k slovu u nájsť slovo, prípadne slová v také, že $(u, v) \in M$. Pri tom potrebujeme pre dané slovo u nielen poznať odpoveď na otázku, či toto slovo patrí do jazyka, ale potrebujeme tiež poznať strom jeho odvodenia. Práve hľadanie stromu odvodenia tvorí jadro syntaktickej analýzy a preto sa v nasledujúcej časti pozrieme na základné prístupy používané pri riešení tehto úlohy.

12.2 Hľadanie stromu odvodenia

Aj keď sme sa už zmienili o algoritme CYK, ktorého rozšírená verzia umožňuje hľadať odvodenia slov, má zmysel sa tejto problematike ďalej venovať. Uvedomme si, že časová zložitosť tohto algoritmu závisí od dĺžky uvažovaného slova kubicky, čo by napríklad pri automatickom kompilovaní programov asi nikoho nepotešilo.

V zásade existujú dve možnosti, ako hľadať stromy odvodenia:

1. Zhora nadol: Nech vstupné slovo je $w = w_1w_2...w_n$. Postupne robíme nasledovné kroky: Počiatočnou formou F_1 je σ. Ak forma začína na terminál, kontrolujeme so začiatkom w. Ak sa nezhodujú, slovo sa nedá odvodiť (alebo sme v zlej nedeterministickej vetve). Ak začína na neterminál, nedeterministicky určíme pravidlo a upravíme formu. V podstate ide o podobný postup, ako pri konštruovaní zásobnikového automatu ekvivalentného s danou bezkontextovou gramatikou.

   Príklad 12.2.1. Vstupné slovo $w_0 = abbcb$. Začneme formou $F_0 = \sigma$, nedeterministicky sa rozhodneme pre pravidlo $\sigma \to aAB$, dostávame formu $F_1 = aAB$. Táto začína terminálom rovnakým ako w, preto modifikujeme formu aj w. Konkrétne, $w_1 = bbcb$, $F_2 = AB$. Nedeterministicky sa rozhodneme pre pravidlo $A \to bb$, dostávame $F_3 = bbB$. Kontrolujeme

[4] Napríklad známa implementácia Pascalu od firmy Borland , verzia 7, umožňuje maximálne 8329 parametrov procedúry

F_3 a w_1 a dostávame $F_4 = B$ a $w_2 = cb$. Ak sa teraz rozhodneme pre pravidlo $B \to cb$, dostaneme $F_5 = cb = w_2$, a teda slovo akceptujeme, pričom poznáme postupnosť použitých pravidiel.

Problémom je spomínaný nedeterminizmus. Keď chceme napísať deterministický algoritmus, musíme použiť nejakú rafinovanú techniku. Vo všeobecnsti by bol totiž problém s konečnosťou. Totiž, deterministický algoritmus si musí v každom kroku pamätať všetky možnosti, aké mohli nastať, teda množinu možných foriem spolu s postupnosťami použitých pravidiel. V prípade, že vstupné slovo nie je akceptovateľné, odvodzovanie foriem by nemuselo skončiť. Isté riešenie ponúka Greibachovej normálny tvar gramatík. Totiž, v každom kroku máme síce množinu možných foriem, ale o každej z nich platí, že

- začína terminálom, a teda v tomto kroku sa tejto formy zbavíme(ak tento terminál nesúhlasí s terminálom na začiatku zostatku vstupného slova) alebo sa táto forma skráti (ak tento terminál súhlasí so začiatkom zostatku vstupného slova)
- alebo začína neterminálom, ktorý však nahradíme pravou stranou pravidiel, z ktorých každá začína terminálom a teda v ďalšom kroku bude na začiatku formy opäť terminál

Je teda zrejmé, že po dvoch krokoch sa určite skráti dĺžka zostatku vstupného slova pre každú formu (aj keď ich počet sa zväčší).

2. Zdola nahor: Vstupné slovo čítame zľava, až kým neobjavíme takú jeho súvislú časť, ktorá je pravou stranou nejakého pravidla. Ak sme takú objavili, nahradíme ju príslušným neterminálom a opäť pokračujeme od začiatku vstupného slova. Ak sa nato pozrieme takto nedeterministicky, tak je nám ľahko, ale deterministický algoritmus sa nemá vo všeobecnosti podľa čoho rozhodnúť, ktorú takúto časť má použiť. Práve preto budeme v nasledujúcom skúmať isté triedy beykontextových gramatík, pre ktoré sa tento postup dá robiť deterministicky.

12.3 Algoritmus posuň a redukuj (Shift and reduce)

Definícia 12.3.1. *Ľubovoľnú súvislú časť vetnej formy, ktorá je pravou stranou nejakého pravidla gramatiky, nazývame rukoväť alebo handle.*

Tento algorimtus používa zásobník, do ktorého ukladá vstupné symboly (robí sa posun - shift). V momente, kedy sa na vrchu zásobníka objaví nejaká handle, nedeterministicky sa rozhodne, či bude redukovať. V prípade, že sa rozhodne redukovať, nahradí symboly zásobníka prislúchajuce tejto handle príslušným neterminálom a takto pokračuje ďalej. Každá redukcia znamená zapísanie použitého pravidla na výstup. Na konci budeme mať pravé krajné odvodenia yapísané odzadu.

Niekedy môžeme toto robiť deterministicky. V nasledujúcich častiach sa budeme venovať dvom typom gramatík, kedy sa nebudeme potrebovať nedeterministicky rozhodovať.

12.3.1 Precedenčné gramatiky

Pre každú dvojicu symbolov $s, t \in N \cup T$ môžu nastať tri možnosti:

- $s \approx t$ Ak s a t sa môžu nachádzať vedľa seba v najľavejšej handle. Túto možnosť nesmieme vylúčiť, ak existuje nejaké pravidlo typu $\xi \to \alpha s t \beta$

- $s \prec t$ Ak s nie je, ale t je v najľavejšej handle. Túto možnosť nevylučujeme, ak

$$(\exists A \in N)\ (A \to s\gamma) \in P$$

- $s \succ t$ Ak s je v najľavejšej handle, ale t nie je. Túto možnosť nemôžeme vylúčiť, ak

$$(\exists A \in N)\ (A \to \gamma s) \in P$$

Ak pre každú dvojicu symbolov z $N \cup T$ platí najviac jedna z horeuvedených možností, potom ide o precedenčnú gramatiku. Rozhodovanie, či sa má urobiť posun alebo redukcia je potom veľmi jednoduché. Redukcia sa robí v prípade, že nastala tretia možnosť.

Dá sa hovoriť aj o istých zovšeobecneniach precedenčných gramatík. Opisovaná verzia sa označuje aj $1-1$-precedenčná, sú však popísané aj rôzne $m-n$-precedenčné.

12.3.2 $LR0$-gramatiky

Pre $LR0$-gramatiky platí, že dokážeme deterministicky skonštruovať pravé krajné odvodenie slova iba na základe jedného symbolu na vrchu zásobníka.

Potom, čo sa prebrodíme záplavou pojmov, ukážeme vhodný príklad toho, ako pracuje $LR0$-analyzátor.

Definícia 12.3.2. *Životaschopným prefixom nazývame každý prefix vstupného slova, ktorého koniec nepresahuje koniec najľavejšej handle vstupného slova. (Najľavejšej v zmysle jej konca)*

Definícia 12.3.3. *$LR0$-element je pravidlo gramatiky s bodkou na pravej strane. Presnejšie, ak $\xi \to x_1x_2...x_m$ je pravidlo, potom*

$$\xi \to \cdot x_1x_2...x_m,\ \xi \to x_1 \cdot x_2...x_m, ..., \xi \to x_1x_2...x_m\cdot$$

sú $LR0$-elementy.

Definícia 12.3.4. *$LR0$-element $A \to \alpha \cdot \beta$ je platný pre životaschopný prefix γ, ak existuje odvodenie*

$$\sigma \underset{PK}{\overset{*}{\Longrightarrow}} \delta Aw \underset{PK}{\Longrightarrow} \delta\alpha\beta w\ \wedge \gamma = \delta\alpha$$

Definícia 12.3.5. *$LR0$-element $A \to R\cdot$ (s bodkou na konci) nazývame úplný $LR0$-element.*

V nasledujúcom príklade ukážeme, že vieme zostrojiť konečný automat, ktorý rozpozná maximálny životaschopný prefix. V prípade nedeterministického konečného automatu skončí v prípade akceptácie v stave, ktorý reprezentuje nejaký platný úplný $LR0$-element. Stavmi tohto konečného automatu budú $LR0$-elementy, akceptujúcimi stavmi budú platné $LR0$-elementy. Tento nedeterministický automat prevedieme na deterministický automat, ktorého stavmi budú množiny $LR0$-elementov, akceptujúcimi stavmi také množiny, ktoré obsahujú aspoň jeden platný $LR0$-element.

Príklad 12.3.1. Uvažujme gramatiku G definovanú týmito pravidlami:
$S' \to Sc$
$S \to SA \mid A$
$A \to aSb \mid ab$
Množina stavov, resp. množina $LR0$-elementov je
$K = \{S' \to \cdot Sc,\ S' \to S \cdot c,\ S' \to Sc\cdot,\ S \to \cdot SA,\ S \to S \cdot A,\ S \to SA\cdot,\ A \to \cdot aSb,\ A \to a \cdot Sb,\ A \to aS \cdot b,\ A \to aSb\cdot,\ A \to \cdot ab,\ A \to a \cdot b,\ A \to ab\cdot\}$

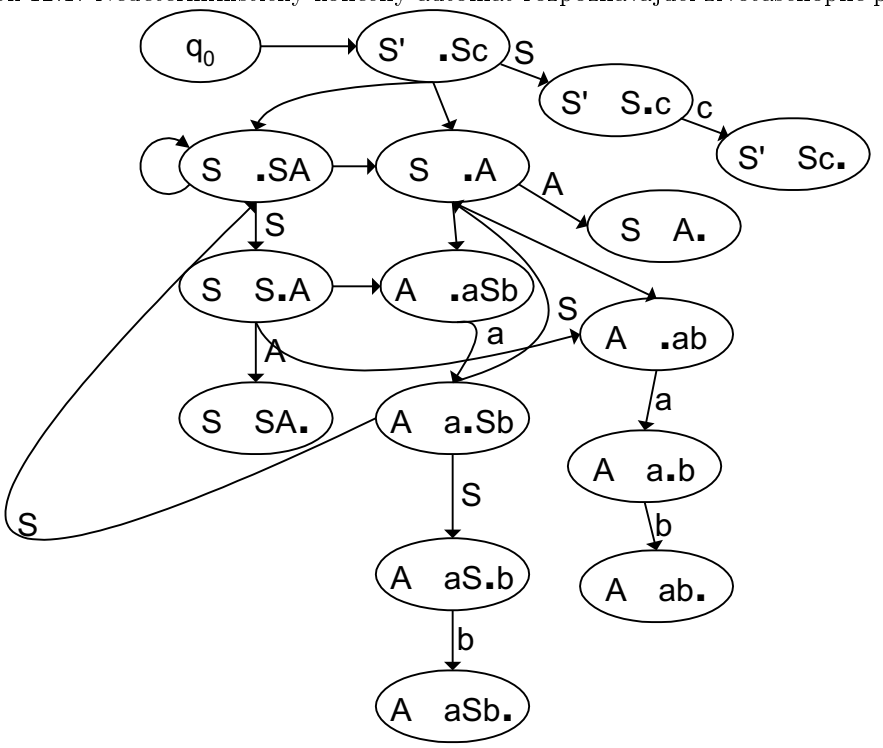
Uvažujme toto pravé krajné odvodenie:

$$S' \underset{G}{\Longrightarrow} Sc \underset{G}{\Longrightarrow} SAc \underset{G}{\Longrightarrow} S \underbrace{aSb}_{\text{handle}} c$$

Životaschonými prefixami tejto vetnej formy sú ε, S, Sa, SaS, $SaSb$. Pre životaschopný prefix SaS je jediným platným $LR0$-elementom $aS \cdot b$.

Schéma konečného nedeterministického automatu rozpoznávajuceho životaschopné prefixy.

Obrázok 12.1: Nedeterministický konečný automat rozpoznávajuci životaschopné prefixy



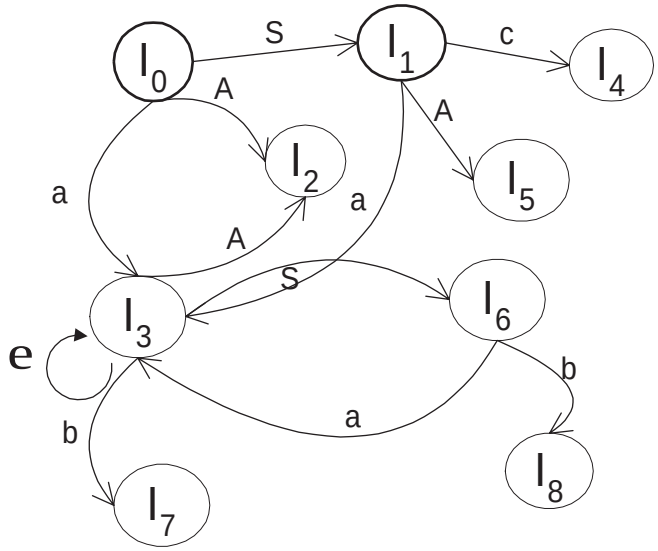
Prechodovú funkciu definujeme podľa nasledujúcich pravidiel:

- ε-ový prechod zo stavu, v ktorom je bodka pred neterminálom ξ spôsobí prechod do stavu reprezentujúceho $LR0$-element pravidla pre ξ s bodkou na začiatku
- V stave $\xi \to \alpha \cdot v\beta$ definujeme pri čítaní symbolu v prechod na stav $\xi \to \alpha v \cdot \beta$

To znamená, že vlastne začínajuc od koreňa (od σ) hľadáme najľavejšie najspodnejšie pravidlo, ktoré sme použili pri odvodení. Ak sa automat napríklad nachádza v stave $A \to a \cdot Sb$, znamená to, že ak najľavejšie pravidlo bolo $A \to a \cdot Sb$, potom ešte musíme prečítať Sb.

Takto získaný nedeterministický automat prevedieme na deterministický, nepodstatné stavy neuvažujeme Popis jednotlivých množín:
$I_0 = \{\ q_0, S' \to \cdot Sc, S \to \cdot SA, S \to \cdot A, A \to \cdot aSb, A \to \cdot ab\ \}$
$I_1 = \{\ S' \to S \cdot c, S \to S \cdot A, A \to \cdot aSb, A \to \cdot ab\ \}$
$I_2 = \{\ S \to A \cdot\ \}$
$I_3 = \{A \to A \cdot Sb, A \to A \cdot b, S \to \cdot SA, S \to \cdot A, A \to \cdot aSb, A \to \cdot ab\ \}$

Obrázok 12.2: Deterministický konečný automat rozpoznávajuci životaschopné prefixy (LR0-položkový automat)



$I_4 = \{\ S' \to Sc\cdot\ \}$
$I_5 = \{\ S \to SA\cdot\ \}$
$I_6 = \{\ A \to aS \cdot b, S \to S \cdot A, A \to \cdot aSb, A \to \cdot ab\ \}$
$I_7 = \{\ A \to ab\cdot\ \}$
$I_8 = \{\ A \to aSb\cdot\}$

Prácu $LR0$-analyzátora možno predbežne popísať tak, že v každom kroku si spustí na svojom zásobníku uvedený konečný automat. Ak skončí v stave, ktorý reprezentuje množinu s jedným platným úplným elementom, robí redukciu. Ak táto množina neobsahuje úplný platný element, robí ďalší posun. Ak obsahuje viac než jeden úplný platný element, potom nejde o $LR0$-gramatiku. Na príklade vstupného slova *aababbc* ukážeme, ako tento lexikálny analyzátor pracuje.

Zásobník	Stav	Zostatok vstupu	Akcia
0	I_0	$aababbc$	posun
$0a$	I_3	$ababbc$	posun
$0aa$	I_3	$babbc$	posun
$0aab$	I_7	$abbc$	redukuj pravidlo $A \to ab$
$0aA$	I_2	$abbc$	redukuj pravidlo $S \to A$
$0aS$	I_6	$abbc$	posun
aSa	I_3	bbc	posun
$aSab$	I_7	bc	redukuj pravidlo $A \to ab$
aSA	I_5	bc	redukuj pravidlo $S \to SA$
aS	I_6	bc	posun
aSb	I_8	c	redukuj pravidlo $A \to aSb$
A	I_2	c	redukuj pravidlo $S \to A$
S	I_1	c	posun
Sc	I_4	ε	redukuj pravidlo $S' \to Sc$

Programátorská implementácia tohto algoritmu nesmie byť taká ťažkopádna, ako sme ju kvôli jednoduchosti popísali. Kvôli zrýchleniu si na zásobníku môžeme pamätať za každým symbolom aj stav, v ktorom končil uvedený konečný automat, keby na vstup dostal časť zásobníka od spodu až po tento symbol. Po každom posune aj po každej redukcii potom stačí simulovať jeden krok automatu a netreba robiť celú simuláciu od začiatku.

Doteraz sme neformálne popisovali vlastnosti $LR0$-gramatík. Nasleduje teda jej definícia.

Definícia 12.3.6. *Bezkontextová gramatika G je $LR(0)$, ak sú splnené obe nasledujúce podmienky:*

- *σ sa nevyskytuje na pravej strane žiadneho pravidla*
- *Pre každý životaschopný prefix γ platí, že ak $A \to \alpha\cdot$ je platný pre γ, potom žiadny iný úplný $LR0$-element nie je platný pre γ a žiadny iný platný element nemá bodku pred terminálnym symbolom* [5]

Príklad konštrukcie automatu, ktorý sme uviedli a príklad práce analyzátora, ktorý je vlastne deterministickým zásobníkovým automatom nás núti vysloviť aj ďalšiu vetu.

Veta 12.3.1. *Ak $L = L(G)$ pre $LR(0)$ gramatiku G, potom $L \in \mathcal{L}_{detCF}$ a existuje deterministický zásobníkový automat A taký, že $L = N(A)$*[6]

Z praktického hľadiska sú zaujímavé aj $LR(k)$ gramatiky pre rôzne k. V nich sa vieme na základe k symbolov zo vstupu rozhodnúť, akú akciu zvolíme (posun alebo redukciu). Každá $LR(k)$ gramatika sa dá upraviť na ekvivalentnú $LR(1)$ gramatiku, ďalej na $LR(0)$ gramatiku už však nie.

Uvedomme si, že horeuvedený príklad hovorí o tom, že sa sá algoritmicky zistiť, či daná gramatika je $LR(0)$. Stačí zostrojiť konečný automat a nahliadnuť, či spĺňa potrebné vlastnosti. Uvádzame však aj príklad gramatiky, kde vieme ad hoc veľmi rýchlo rozodnúť, že nie je $LR(0)$:

Príklad 12.3.2. Gramatika G daná pravidlami
$S \to aAc \mid aBd$

[5] Uvedomme si, že ak nejaký platný element má bodku pred neterminálom (o tejto možnosti definícia zdanlivo mlčí), potom vďaka epsilonovým prechodom existuje platný element s bodkou pred terminálom, alebo by použitie iných pravidiel viedlo k zacykleniu

[6] Teda akceptuje prázdnou pamäťou. Pozor! Trieda jazykov rozpoznávaná deterministickým zásobníkovým automatom prázdnou pamäťou je menšia ako trieda jazykov rozpoznateľných deterministickým zásobníkových automatom (koncovým stavom)

$A \rightarrow aAc \mid b$
$B \rightarrow aBa \mid b$
nie je $LR(0)$. Uvažujme totiž slovo $aa...abb...b$. Je jasné, že najľavejšie handle je b. Nevieme však, či máme použiť pravidlo $A \rightarrow b$ alebo $B \rightarrow b$. Dokonca sa dá ukázať, že táto gramatika nie je $LR(k)$ pre žiadne k.

Kapitola 13

Úvod do teórie zložitosti

Čo sa týka Turingových strojov a vôbec algoritmov ako takých, skúmajú sa hlavne dve hľadiská zložitosti Turingových strojov, resp. algritmov. Ide o čas potrebný na výpočet a o priestor, t.j. o dĺžku potrebnej pásky (množstvo potrebnej pamäte). Zložitosť, ako časovú, tak aj priestorovú, definujeme ako funkciu závisiacu od dĺžky, resp. veľkosti vstupu.

13.1 Priestorová zložitosť

V prípade priestorovej zložitosti nejakého problému používame ako model Turingovho stroja Turingov stroj s jednou fixnou vstupnou páskou, na ktorú nemôžeme zapisovať a s k nekonečnými pracovnými páskami.

Definícia 13.1.1. *Hovoríme, že deterministický Turingov stroj A je $S(n)$ páskovo ohraničený, ak každý výpočet A na slove w dĺžky n použije na každej z k pracovných pások maximálne $S(n)$ políčok.*

Čo sa týka nedeterministických Turingových strojov, sú možné tri prístupy definície páskovej ohraničenosti.

Definícia 13.1.2. *Hovoríme, že nedeterministický Turingov stroj A je $S(n)$ páskovo ohraničený, ak*

Silná definícia: *Každý výpočet na vstupnom slove w dĺžky n použije maxmálne $S(n)$ políčok na každej so svojich k pracovných pások.*

Stredná definícia: *Každý výpočet na každom vstupnom slove $w \in L(A)$ dĺžky n použije maximálne $S(n)$ políčok na každej so svojich k pracovných pások.*

Slabá definícia: *Pre každé vstupné slovo $w \in L(A)$ dĺžky n existuje výpočet, ktorý použije maximálne $S(n)$ políčok na každej páske.*

V ďalšom texte budeme používať pojem slušnej funkcie. Pod slušnou funkciou rozumieme každú funkciu $f : \mathbb{N} \to \mathbb{N}$ takú, že existuje Turingov stroj, ktorý pre vstup dĺžky n označí (v čase $T(n)$) na páske $T(n)$ políčok.

Vráťme sa ešte k definícii 13.1.2. Pre slušné funkcie je jedno, ktorú z definícií používame. Pred samotným výpočtom si totiž môžeme na pomocnej páske vygenerovať $S(n)$ a potom každý výpočet, ktorý by potreboval viac ako $S(n)$, zabiť.

13.2 Časová zložitosť

Ako model používame Turingov stroj s k páskami, pričom vstup očakávame na prvej z nich.

Definícia 13.2.1. *Hovoríme, že deterministický Turingov stroj je $T(n)$ časovo ohraničený, ak pre každý vstup dĺžky n urobí Turingov stroj na tomto vstupe najviac $T(n)$ krokov.*

Pri definícii časovej zložitosti pre nedeterministické Turingove stroje máme opäť tri možnosti.

Definícia 13.2.2. *Hovoríme, že nedeterministický Turingov stroj A je $T(n)$ časovo ohraničený, ak*

Silná definícia: *Každý výpočet na vstupnom slove w dĺžky n urobí najviac $T(n)$ krokov.*

Stredná definícia: *Každý výpočet na každom vstupnom slove $w \in L(A)$ dĺžky n urobí maximálne $T(n)$ krokov*

Slabá definícia: *Pre každé vstupné slovo $w \in L(A)$ dĺžky n existuje výpočet, ktorý urobí maximálne $T(n)$ krokov.*

13.3 Triedy zložitosti

Na základe predchádzajucich definícií rozčleníme jazyky do tried zložitostí.

Definícia 13.3.1. *Triedy $DSPACE$, $NSPACE$, $DTIME$, $NTIME$ sú definované nasledovne:*

$$
\begin{aligned}
DSPACE(S(n)) &\underset{df}{=} \{L \mid \text{existuje det. TS } A, S(n)\text{-páskovo ohraničený, taký že } L = L(A)\} \\
NSPACE(S(n)) &\underset{df}{=} \{L \mid \text{existuje nedet. TS } A, S(n)\text{-páskovo ohraničený, taký že } L = L(A)\} \\
DTIME(T(n)) &\underset{df}{=} \{L \mid \text{existuje det. TS } A, T(n)\text{-časovo ohraničený, taký že } L = L(A)\} \\
NTIME(T(n)) &\underset{df}{=} \{L \mid \text{existuje nedet. TS } A, T(n)\text{-časovo ohraničený, taký že } L = L(A)\}
\end{aligned}
$$

13.4 Vety o kompresii, zrýchlení a redukcii

Čitateľ, ktorý už vie niečo o zložitosti algoritmov vie, že nie je zásadný rozdiel medzi algoritmom, ktorý potrebuje n^2 políčok a algoritmom, ktorý potrebuje $2n^2$ políčok. Z hľadiska Turingových strojov platí, že ak vieme nejaký problém riešiť Turingovym strojom s páskou dlhou $S(n)$, potom ho vieme riešiť aj Turingovým strojom s páskou dlhou $\frac{S(n)}{k}$, kde k je ľubovoľná konštanta.

Veta 13.4.1. *(O kompresii pásky:) K ľubovoľnému deterministickému Turingovmu stroju A, ktorý je $S(n)$-páskovo ohraničený a konštante k existuje deterministický Turingov stroj A', ktorý je $\frac{S(n)}{k}$-páskovo ohraničený a platí $L(A) = L(A')$.*

Dôkaz. Stačí použiť novú pracovnú abecedu $\Gamma' = \Gamma^k$ a reorganizáciu pásky urobiť takým spôsobom, že každý blok o k políčkach bude pre A' predstavovať jedno políčko. □

Podobná veta platí aj o zrýchlení. Musíme si však uvedomiť dva zásadné problémy. Prvý, že ak chceme na začiatku reorganizovať pásku, musíme ju celú prejsť, na čo budeme potrebovať n krokov. Preto budeme musieť predpokladať, že toto je zanedbateľne malý čas oproti času výpočtu automatu A.
Druhý problém spočíva v tom, že ak chceme simulovať v jednom kroku TS A' k krokov pôvodného

TS A', musíme mať informáciu o tom, čo je na páske o k políčok vľavo i vpravo od políčka, na ktorom je umiestnená hlava. Ak používame bloky o k pôvodných políčkach, musíme si v stave zapamätať, na ktorom z týchto k políčok by bola hlava pôvodného automatu. Ak je hlava kdesi na prvom políčku bloku, potom sa hlava vude musieť pozrieť na vedľajší blok.

Veta 13.4.2. *(O lineárnom zrýchlení:) Nech A je ľubovoľný deterministický Turingov stroj A, ktorý je $T(n)$-časovo ohraničený a nech*

$$\lim_{n\to\infty} \frac{T(n)}{n} \;=\; \infty$$

Nech k je ľubovoľná konštanta. Potom existuje deterministický Turingov stroj A', ktorý je $\frac{T(n)}{k}$-časovo ohraničený a platí $L(A) = L(A')$.

Dôkaz. Nech platia uvedené predpoklady. Turingov stroj A' bude pracovať nasledovne: Najprv reorganizuje pásku do blokov o m políčkach. To znamená, že každých m znakov prepíše na jeden znak (opäť totiž $\Gamma' = \Gamma^m$). Číslo m je na teraz neznáma hodnota, ale na záver ju vyčíslime. Potom simuluje prácu pôvodného automatu A s tým, že sa rozšíri množina stavov na $K \times \{1, 2, ..., m\}$, aby sme si pamätali číslo skutočného políčka, na ktorom by bol TS A v rámci aktuálneho bloku. TS A' urobí teraz štyri kroky: Prečíta políčko, resp. blok, na ktorom sa nachádza, posunie sa doľava a tiež toto políčko prečíta, potom urobí dva kroky doprava aby aj tento blok prečítal, vráti sa späť a na základe informácie, ktorú získal vykoná m krokov pôvodného automatu. Ten na m krokov určite nezapisoval do blokov vzdialených od aktuálneho viac ako o 1. Na tento zápis potrebujeme opäť 4 kroky.

To znamená, že pôvodných m krokov vieme simulovať ôsmimi krokmi. Máme teda dokázať, že platí

$$n + \frac{n}{m} + 8\frac{T(n)}{m} \leq \frac{T(n)}{k}$$

Na to ale stačí, aby platilo

$$m \geq \frac{8T(n)}{\frac{T(n)}{k} - n - \frac{n}{m}} = 8k + 8k\frac{n(1+\frac{1}{m})}{T(n)}$$

Ale limita $8k\frac{n(1+\frac{1}{m})}{T(n)}$ ide k nule, a teda existuje horné ohraničenie tejto postupnosti h. Potom stačí zvoliť $m = 8k + h$. □

Poznámka 13.4.1. To, že A' bude $T(n)/k$ časovo ohraničený, treba brať s rezervou. Pre $T(n) = n^2$ a $k = 100$ by to znamenalo, že A' bude pre slovo dĺžky 5 stačiť 1/4 kroku. V skutočnosti tvrdenie vo vete platí len pre slová dostatočnej dĺžky:

Nech A je ľubovoľný deterministický Turingov stroj A, ktorý je $T(n)$-časovo ohraničený a nech

$$\lim_{n\to\infty} \frac{T(n)}{n} \;=\; \infty$$

Nech k je ľubovoľná konštanta. Potom existuje deterministický Turingov stroj A', ktorý je $T'(n)$-časovo ohraničený, platí $L(A) = L(A')$ a $\exists N_0 \forall n > N_0 T'(n) \leq \frac{T(n)}{k}$.

V nasledujúcom texte ukážeme, že i počet použitých pások možno redukovať. Bohužiaľ, v niektorých prípadoch tým čosi stratíme.

Veta 13.4.3. *(O redukcii počtu pások z k na jednu pre priestor:) K ľubovoľnému k-páskovemu deterministickému TS A pracujúcom v priestore $S(n)$ existuje 1-páskový TS A' pracujúci tiež v priestore $S(n)$.*

Dôkaz. Pointa dôkazu spočíva v tom, že na pôvodný Turingov stroj A sa pozrieme tak, že namiesto toho, aby sa hýbali hlavy, budú sa hýbať jednotlivé pásky. Znamená to, že keď sa má pohnúť nejaká hlava doľava, namiesto toho vykoná zložitú procedúru[1], ktorá dokáža posunúť celé slovo na tejto páske doprava. Namiesto jedného kroku teda máme $S(n)$ krokov. Dostali sme sa teda k Turingovmu stroju, ktorého všetky hlavy sú na pevnej pozícii a pracuje tiež v priestore $S(n)$, iba čas sa nám zhoršil na $S(n)T(n) \leq (T(n))^2$. Takýto Turingov stroj môžeme poľahky prostredníctvom rozšírenia pracovnej abecedy na Γ^k previesť na hľadaný jednopáskový Turingov stroj. □

Z dôkazu predchádzajucej vety vyplýva ďalšia:

Veta 13.4.4. *(O redukcii počtu pások z k na jednu pre čas:) K ľubovoľnému deterministickému k-páskovemu Turingovmu stroju A pracujúcom v čase $T(n)$ existuje jednopáskový deterministický Turingov stroj A' pracujúci v čase $T(n)^2$ taký že $L(A) = L(A')$.*

Redukcia počtu pások na jednu nás síce stojí veľa, redukcia na dve pásky nás však stojí oveľa menej.

Veta 13.4.5. *(O redukcii počtu pások z k na dve pre čas) K ľubovoľnému deterministickému k-páskovemu Turingovmu stroju A pracujúcom v čase $T(n)$ existuje dvojpáskový deterministický Turingov stroj A' pracujúci v čase $T(n)\log(T(n))$ taký že $L(A) = L(A')$.*

Dôkaz. Použijeme druhú pásku ako akúsi cache pamäť. Opäť použijeme metódu posunu stopy pásky namiesto posunu hlavy. No nebudeme posúvať celú stopu, ale väčšinu len jej kúsok. Podstata použitej myšlienky spočíva v tom, že si pásku rozdelíme na myslené bloky o veľkostiach $1, 2, 4, 8, ..., 2^k$ políčok. Fungovanie automatu A' vysvetlíme na príklade.

							a_0							
a_7^-	a_6^-	a_5^-	a_4^-	a_3^-	a_2^-	a_1^-	x	a_1	a_2	a_3	a_4	a_5	a_6	a_7

kde x značí stred pásky, t.j. fixovanú polohu hlavy. Posun doprava:

							a_1^-	a_0						
a_7^-	a_6^-	a_5^-	a_4^-	a_3^-	a_2^-		x	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Ďalší posun doprava:

							a_2^-		a_0	a_1				
a_7^-	a_6^-	a_5^-	a_4^-		a_3^-	x	a_1^-	a_2	a_3	a_4	a_5	a_6	a_7	

Do tretice posun doprava:

							a_3^-	a_2^-	a_0	a_1				
a_7^-	a_6^-	a_5^-	a_4^-			x	a_1^-	a_2	a_3	a_4	a_5	a_6	a_7	

A na poslednom znázornenom posune doprava vidno silu tohto postupu:

							a_4^-				a_0	a_1	a_2	a_3
	a_7^-	a_6^-	a_5^-			x	a_3^-	a_1^-	a_2^-		a_4	a_5	a_6	a_7

O blokoch na páske platia vždy nesledujúce dve tvrdenia:

[1] Uvedomme si, že ona trvá rádovo S(n)

1. Ak sa k blokov napravo, resp. naľavo od hlavy zaplní a chceme do nich opäť pridať ďalší symbol, potom sa tieto bloky po nasledujúcom kroku do polovice vyprázdnia

2. O symetrických blobkoch platí, že ak je jeden plný, resp. poloplný, resp. prázdny, k nemu symetrický blok je prázdny, resp. poloplný, resp. plný.

To znamená, že ľahko možno odhadnúť, ako často sa bude pracovať s k-tym blokom. Zrejme až vtedy, keď sa predchádzajucich $k-1$ blokov zaplní. Keďže veľkosť prvých k blokov je zhruba 2^k, tak táto situácia nastane najviac raz za 2^k krokov. Práca s k-tym blokom trvá zhruba 2^k krokov, blokov je spolu najviac $\log_2 S(n) \leq \log_2 T_A(n)$, a preto

$$T_{A'}(n) \;=\; \sum_{k=1}^{\log_2(T_A(n))} \frac{T_A(n)}{2^k} 2^k \;\approx\; T_A(n)\log_2(T_A(n))$$

□

Keďže sme ukázali, že redukcia na dve pásky zhorší časovú zložitosť len pridaním logaritmu, na mieste je otázka, či by sa výsledok z vety 13.4.4 nedal zlepšiť. Nasledujúca lema ukazuje, že vo všeobecnosti sa tento výsledok nedá zlepšiť.

Lema 13.4.1. *Každý Turingov stroj pre $L = \{wcw^R \mid w \in \{a,b\}^*\}$ má časovú zložitosť aspoň $T(n) = n^2$.*

Dôkaz. Nech A je Turingov stroj akceptujúci tento jazyk. Uvažujme prechodové postupnosti $\{p^{x,i}\}_j$, pre $i = 1, 2, ..., n$, kde n je dĺžka vstupného slova x[2]. Zrejme platí

$$\sum_{i=1}^{n} |p^{x,i}| \;=\; T_A(n)$$

kde $|p^{x,i}|$ je dĺžka i-tej prechodovej postupnosti.
Uvažujme nasledovné podmnožiny jazyka L definované nasledovne:

$$L_m = \{yb^m cb^m y^R \mid y \in \{a,b\}^*,\ |y| = m\}$$

Ak sčítame časy potrebné na akceptáciu všetkých slov z jazyka L_m, dostávame

$$\sum_{x\in L_m} T_A(n) \geq \sum_{x\in L_m}\sum_{i=1}^{|x|} |p^{x,i}| \;\geq^{3} \sum_{x\in L_m}\sum_{i=m}^{2m-1} |p^{x,i}| \;=\; \sum_{i=m}^{2m-1}\sum_{x\in L_m} |p^{x,i}| \qquad (13.1)$$

Uvedomme si teraz, že musí platiť

1. $p^{x,i} \neq p^{y,i}$ pre $x \neq y$, $i = m..2m-1$. Keby totiž nastal opak, potom by TS A akceptoval aj slovo $yb^m cb^m x^R$.

2. $p^{x,i} \neq p^{x,j}$ pre $i, j = m..2m-1$, $i \neq j$. Keby nastal opak, potom by A akceptoval aj slovo $xb^{m-|j-i|}cb^m x^R$.

Z toho vyplýva, že uvažované postupnosti $p^{x,i}$ sú všetky rôzne. Formálne, aj skutočne ich teda pre fixné i je 2^m. Keby to všetko boli aj tie najkratšie postupnosti, dĺžka najdlhšieho z nich by

[2] Ide o postupnosti stavov, v ktorých sa A nachádzal, keď prechádzal i-tym políčok vstupného slova

bola aspoň $\log_s(2^m)$. Preto pre súčet dĺžok týchto postupností pre všetky slová z L_m $\sum_{x\in L_m}|p^{x,i}|$ platí

$$\sum_{x\in L_m}|p^{x,i}| \;\geq\; \sum_{d=1}^{\log_s(2^m)} d.s^d \tag{13.2}$$

Z kombinatorickej analýzy vieme, že

$$\sum_{d=1}^{n} ds^d \approx \frac{ns^n}{\log_2(s)} - \frac{s^n}{\log_2(s)} \tag{13.3}$$

Z 13.2 a 13.3 dostávame

$$\sum_{x\in L_m}|p^{x,i}| \;\geq\; \log_s(2^m).s^{\log_s(2^m)} = cm.2^m \tag{13.4}$$

Z toho a z 13.1 dostávame

$$\sum_{x\in L_m} T_A(n) \;\geq\; \sum_{i=m}^{2m-1}\sum_{x\in L_m}|p^{x,i}| \;\geq\; \sum_{i=m}^{2m-1} cm2^m \;=\; cm^2.2^m \tag{13.5}$$

Slov v jazyku L_m je presne 2^m, a preto pre jedno z týchto slov y musí platiť $T_{A;y} \geq cm^2$, čo bolo treba dokázať. □

13.5 Hierarchia tried zložitosti

Veta 13.5.1. *Nech $S_1(n)$, $S_2(n)$ sú slušné funkcie také, že*

$$\lim_{n\to\infty}\frac{S_1(n)}{S_2(n)} = 0$$

a obe sú aspoň $\log(n)$.
Potom DSPACE($S_1(n)$) $\subsetneq$ DSPACE($S_2(n)$)

Dôkaz. Celý dôkaz možno nájsť v [1] na strane 217. Uvádzame preto len myšlienku tohto dôkazu. Základný princíp je opäť v akejsi diagonalizácii, ale oveľa prefíkanejšej ako v prípade dôkazu vety 11.1.1.

Zostrojíme $S_2(n)$ páskovo ohraničený TS A taký, že bude akceptovať taký jazyk, ktorý určite neakceptuje žiadny $S_1(n)$-páskovo ohraničený TS.
TS A bude pracovať nasledovne: Nech $f : \Sigma \to \{0,1\}$ je surjektívna funkcia

1. Pre svoj vstup w dĺžky n označí na páske $S_1(n)$ políčok, aby pri ďalšej práci vedel, kedy skončiť.

2. Na vstup w sa pozrie ako na binárne číslo i (spočíta $f(w)$) a na páske vyrobí kód i-teho Turingovho stroja. Ak by jeho kód mal byť dlhší ako $S_2(n)$, musíme sa samozrejme zastaviť bez akceptovania. Avšak, pre každý Turingov stroj T_i je jeho kód konečný a teda existuje vstupné slovo w s veľkým počtom núl na začiatku, a teda s dostatočnou dĺžkou n takou, že priestor $S_2(n)$ stačí na vygenerovanie kódu $T_i(n)$.

3. Simuluje prácu T_i na vstupe w. Ak T_i akceptuje w, potom ho T neakceptuje. Ak T_i zastaví a neakceptuje, potom ho T akceptuje. Ak je T_i S_1-páskovo ohraničený a zacyklí sa[4] potom ho T_i akceptuje.

Ťažko explicitne definovať jazyk akceptovaný TS T. Avšak, môžeme dokázať, že žiadny TS T', ktorý je $S_1(n)$-páskovo ohraničený neakceptuje tento jazyk. Keby ho totiž akceptoval, bol by to nejaký T_i spomedzi Turingových strojov generovaných Turingovým strojom T v kroku 2. Ale pre vstupné slovo x také, že T v druhom kroku vygeneruje T_i platí, že T ho akceptuje práve vtedy, keď T_i ho neakceptuje, čo je spor. □

Veta 13.5.2. *Nech $T_1(n)$, $T_2(n)$ sú slušné funkcie také, že*

$$\lim_{n\to\infty} \frac{T_1(n).\log(T_1(n))}{T_2(n)} = 0$$

a obe sú aspoň $\log(n)$.
Potom $DTIME(T_1(n)) \subsetneq DTIME(S_2(n))$

Dôkaz. Dôkaz možno nájsť v [1] na strane 222, je podobný dôkazu vety 13.5.1. □

13.5.1 Elementárne súvislosti medzi triedami zložitosti

Všetky uvažované funkcie sú slušné.

Veta 13.5.3. *Platí*

1.

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \tag{13.6}$$

2.

$$\forall f(n) \geq \log(n)\ \forall L \in DSPACE(f(n))\ \exists c : L \in DTIME(c^{f(n)}) \tag{13.7}$$

3.

$$\forall f(n) \geq n\ \forall L \in NTIME(f(n))\ \exists c : L \in DTIME(c^{f(n)}) \tag{13.8}$$

4.

$$\forall f(n) \geq \lg(n) :\ NSPACE(f(n)) \subseteq DSPACE(f^2(n))\ \textit{[Savitch]} \tag{13.9}$$

Dôkaz. V prípade prvých troch tvrdení uvádzame len myšlienky dôkazov.

1. Za čas $T(n)$ aj tak nedokážeme navštíviť viac než $T(n)$ políčok
2. Počet konfigurácií, ktoré môžu s použitím $f(n)$ políčok nastať, je $k.|\Gamma|^{f(n)} = c^{f(n)}$ Môžeme počítať počet konfigurácií, v ktorých sa TS ocitol. Ak sa už ocitol vo viac konfiguráciach ako $c^{f(n)}$, potom slovo nemôže byť akceptované, lebo TS sa zacyklil.
3. Pomocou prehľadávania do šírky. Konštanta c je maximum z mohutností množín $\delta(q, a)$.

[4] Uvedomme si, že toto $S_2(n)$ páskovo ohraničený TS dokáže zistiť. Totiž, počet konfigurácií $S_1(n)$-páskovo ohraničeného TS je $S_1(n).q.k^{S_1(n)}$, logaritmus tohto čísla, čiže počet políčok potrebných na jeho zápis je $S_1(n)k + k_2$, čo máme k dispozícii, lebo $S_2(n) \gg S_1(n)$.

4. Ak existuje akceptujúci výpočet, tak existuje aj necykliaci sa akceptujúci výpočet. Takýto akceptujúci výpočet môže obsahovať maximálne $c^{f(n)}$ konfigurácií. Každá konfigurácia sa dá napísať na maximálne $f(n)$ políčok.

   Použijeme Savitchovu simuláciu nedeterministického TS A deterministickým TS A'. Bez ujmy na všeobecnosti predpokladáme, že TS A má jedinú akceptujúcu konfiguráciu. Nech existuje akceptujúci výpočet, t.j. postupnosť konfigurácií $k_1, k_2, ..., k_{c^{f(n)}}$. TS A' má zistiť, či existuje takáto postupnosť. Postupne skúša všetky možné konfigurácie a zisťuje, či mohli byť v strede tohto výpočtu. Problém zistenia existencie postupnosti konfigurácií teda redukuje na dva problémy zistenia existencie postupnosti dvoch kratších postupností. Na tento postup sa možno pozrieť ako na takúto rekurzívnu funkciu moderného imperatívneho programovacieho jazyka:

```
bool exists(CONFIG k1,CONFIG k2,int maxlength)
{
  if (maxlength == 1) return (k1 == k2);
  else {
    //generate all possible configurations
    CONFIG k=0
    while(getnextconfiguration(k)){
       //recursively test if k may be the midconfiguration
       return exists(k1,k,maxlength/2+maxlength %2 ) && exists(k,k2,maxlength/2);
    }
  }
}
```

   Uvedomme, si že hĺbka rekurzie je $\log(c^{f(n)}) = c'.f(n)$ a v každé volanie potrebuje priestor na zapamätanie si jednej konfigurácie (jediná premenná deklarovaná vo funkcii `exists` je premenná `k`, ktorá zastupuje akúsi konfiguráciu. Na jej uchovanie potrebujeme priestor $f(n)$. Celkovo budeme teda potrebovať na beh programu

```
exists(poc_konf,posl_konf,c^f(n))
```

   pamäť $c'.f(n).f(n) = c'f^2(n)$, čo bolo treba dokázať.

□

13.5.2 Dôležité triedy zložitosti

Triedu	DSPACE($\log(n)$)	označujeme	LOG
Triedu	DSPACE($\log(n)$)	označujeme	NLOG
Triedu	$\cup_k$DTIME(n^k)	označujeme	P
Triedu	$\cup_k$NTIME(n^k)	označujeme	NP
Triedu	$\cup_k$DSPACE(n^k)	označujeme	PSPACE

Poznámka 13.5.1. Zo Savitchovej vety vyplýva, že pri definícii triedy PSPACE je jedno, či uvažujeme zjednotenie DSPACE(n^k), alebo NSPACE(n^k).

Veta 13.5.4. *Platí*

$$LOG \subseteq NLOG \subseteq P \subseteq NP \subseteq PSPACE$$

a

$$LOG \subsetneq PSPACE$$

Dôkaz. Prvá inklúzia platí triviálne.
Druhá inklúzia: ???
Tretia inklúzia platí triviálne.
Štvrtá inklúzia je dôsledkom 13.7.

Druhé tvrdenie vyplýva z 13.5.1, ak dosadíme za $S_1(n)$ funkciu $\lg(n)$ a $S_2(n)$ ľubovoľný polynóm. Keďže teda

$$\text{LOG} \subsetneq \text{DSPACE}(n^k) \quad \text{pre všetky } k$$

platí aj

$$\text{LOG} \subsetneq \text{PSPACE}$$

□

Dodatok A

Vybrané riešené úlohy

V tejto časti sa nachádzajú niektoré riešené úlohy, ktoré sa vyskytli na domáce úlohy, alebo na písomkách v šk. roku 1998/99. Riešenia týchto úloh sú čím ďalej, tým voľnejšie a menej formálne. Čitateľ by však nemal strácať schopnosť ktorúkoľvek úlohu dokázať i čisto formálne.

A.1 Úvod

Príklad A.1.1. Nech L_1 a L_2 sú jazyky. Rozhodnite, či platí

$$(L_1 \cup L_2)^* \;=\; L_2(L_1L_2)^*$$

Riešenie *(: Zjavne platí inklúzia $L_2(L_1L_2)^* \subseteq (L_1 \bigcup L_2)^*$, lebo jazyk naľavo obsahuje niektoré slová, ktoré vznikli zreťazením slov z jazykov L_1 a L_2, zatiaľ čo jazyk napravo obsahuje všetky možné také slová. :)*

Platí $L_2(L_1L_2)^* \subseteq (L_1 \bigcup L_2)^*$.
Dôkaz: Nech $w \in L_2(L_1L_2)^*$. Potom existujú slová $u_1, u_2, ..., u_n \in L_1$, $v_1, v_2, ..., v_n, v_{n+1} \in L_2$ také, že $w = v_1u_1v_2u_2v_3...u_nv_{n+1}$ a $n \geq 0$. Ale keďže platí $u_i \in L_1$, $v_i \in L_2$ pre všetky duchaplné i, platí $u_i \in L_1 \bigcup L_2$, ako aj $v_i \in L_1 \bigcup L_2$, a tak $w = v_1u_1v_2u_2v_3...u_nv_{n+1} \in (L_1 \bigcup L_2)^*$.

Neplatí však obrátená inklúzia.
Kontrapríklad: Ak $L_1 = \{a\}$ a jazyk L_2 je ľubovoľný, potom prázdne slovo ε patrí do jazyka $(L_1 \bigcup L_2)^*$, ale iste nepatrí do jazyka $L_2(L_1L_2)^*$, lebo všetky slová z tohto jazyka vzniknú zreťazením slova a a nejakého slova z $(L_1L_2)^*$, a teda majú dĺžku aspoň 1. ♠

A.2 Bezkontextové gramatiky

Príklad A.2.1. Zostrojte gramatiku pre jazyk

$$L \;=\; \{w \mid w \text{ obsahuje podslovo } abba \} \text{ nad abecedou } \{a,b\}$$

Riešenie $G = (\{\sigma, C\}, \{a, b\}, P, \sigma)$, kde
$P = \{\sigma \to a\sigma | b\sigma | abbaC,$
$C \to aC | bC | \varepsilon\}$

Tvrdenie: Nech $L = \{v_1abbav_2 | v_1, v_2 \in \{a,b\}^*\}$. Potom $L = L(G)$.
Dôkaz:

1. $L \subseteq L(G)$:
   (: slovo v_1abbav_2, kde $v_1, v_2 \in \{a,b\}^*$ vygenerujeme tak, že $|v_1|$-krát použijeme pravidlo $\sigma \to a\sigma$, resp. pravidlo $\sigma \to b\sigma$, potom pravidlo $\sigma \to abbaC$, ďalej $|v_2|$-krát pravidlo $C \to aC$, resp. $C \to bC$ a nakoniec pravidlo $C \to \varepsilon$. :)
   Formálne: Najprv zavedieme dve výrokové formy:
   $F_1(n) :=$ Ak $v_1 \in \{a,b\}^*$ a $|v_1| = n$, potom $\sigma \stackrel{*}{\Longrightarrow} v_1abbaC$.
   $F_2(n) :=$ Ak $\sigma \stackrel{*}{\Longrightarrow} vC$ a $w \in \{a,b\}^*$ a $|w| = n$, potom $\sigma \stackrel{*}{\Longrightarrow} vwC$.

   Tvrdenie 1: $(\forall n \in N_0)F_1(n)$.
   Dôkaz: Indukciou cez n. Pre $n = 0$ stačí použiť pravidlo $\sigma \stackrel{*}{\Longrightarrow} abbaC$. Nech platí F_n pre všetky $n \leq k$. Dokážeme, že potom platí aj F_{k+1}. Nech $|v_1| = k+1$. Sú dve možnosti: Prvá, že $v_1 = av_1'$ a druhá, že $v_2 = bv_1'$. Bez ujmy na všeobecnosti predpokladajme, že platí prvá. Keďže platí $\sigma \stackrel{*}{\Longrightarrow} a\sigma$(aplikovaním jedného z pravidiel) a $\sigma \stackrel{*}{\Longrightarrow} v_1'abbaC$ (indukčný predpoklad), tak platí aj $\sigma \stackrel{*}{\Longrightarrow} av_1'abbaC = v_1abbaC$, čo bolo treba dokázať.

   Tvrdenie 2: $(\forall n \in N_0)F_2(n)$.
   Dôkaz: Tento dôkaz urobíme tiež indukciou cez n. Pre $n = 0$ je tvrdenie triviálnou implikáciou. Nech platí toto tvrdenie pre n. Dokážem, že platí aj pre $n+1$. Nech $|w| = n+1$. Sú dve možnosti: $w = w_1a$, alebo $w = w_1b$. Bez ujmy na všeobecnosti prepokladám prvú. Podľa indukčného predpokladu platí $\sigma \stackrel{*}{\Longrightarrow} vw_1C$. Použitím pravidla $C \to aC$ dostaneme, že $\stackrel{*}{\Longrightarrow} vwC$, čím je tvrdenie dokázané.

   Teraz už môžeme dokázať prvú inklúziu. Nech $v \in L$. Nech $v = v_1abbav_2$. Podľa tvrdenia 1 platí $\sigma \stackrel{*}{\Longrightarrow} v_1abbaC$. Ďalej podľa tvrdenia 2 platí $\sigma \stackrel{*}{\Longrightarrow} v_1abbav_2C$. Keďže platí $C \stackrel{*}{\Longrightarrow} \varepsilon$, platí aj $\sigma \stackrel{*}{\Longrightarrow} v_1abbav_2 = v$, a tým je dôkaz podaný.

2. Dokážeme obrátenú inklúziu $L(G) \subseteq L$:
   Dokážeme dokonca všeobecnejšie tvrdenie[1], z ktorého toto vyplýva:

   Tvrdenie 3. Nech v je vetná forma taká, že $\sigma \stackrel{*}{\Longrightarrow} v$. Potom buď obsahuje neterminál σ alebo podslovo $abba$, čiže
   $v \in \{C,\sigma,a,b\}^*\sigma\{C,\sigma,a,b\}^* \bigcup \{C,\sigma,a,b\}^*abba\{C,\sigma,a,b\}^* =_{def} L'$.
   Dôkaz: Dokážem to indukciou cez počet krokov odvodenia d vetnej formy v. Zrejme tvrdenie platí pre $d = 0$. Predpokladajme, že tvrdenie platí pre nejaké d a dokážme, že platí aj pre $d+1$. Nech $\sigma \Rightarrow v_1 \Rightarrow v_2 \Rightarrow ... \Rightarrow v_d \Rightarrow v_{d+1} = v$. Pre v_d podľa indukčného predpokladu tvrdenie platí, t.j. patrí do L'. Sú dve možnosti: Prvá, že vetná forma v vznikla z v_d nahradením neterminálu C, alebo druhá, že terminálu σ. Poľahky nahliadneme, že v oboch prípadoch bude tvrdenie pre v_{d+1} platiť.

♠

Príklad A.2.2. Zostrojte bezkontextovú gramatiku pre jazyk (nad abecedou $\{a,b\}$)

$$L = \{w \mid \#_a w = \#_b w\}$$

Nezabudnite formálne dokázať, že navrhnutá gramatika je gramatikou pre jazyk L.
Poznámka: $\#_a w$, resp. $\#_b w$ znamená počet písmen a, resp. b v slove w.

[1] Je veľmi častým javom, že pri dôkaze indukciou nedokazujeme tvrdenie, ktoré potrebujeme, ale také jeho zovšeobecnenie, ktoré je uzavreté na indukciu

Riešenie $G = (N = \{\sigma, A, B\}, T = \{a, b\}, P, \sigma)$, kde
$P = \{\sigma \to \varepsilon | aB | bA$
$A \to a\sigma | bAA$
$B \to b\sigma | aBB$
$\}$

Dôkaz, že $L(G) = L$, kde $L = \{w | w \in \{a, b\}^* \wedge \#_a w = \#_b w\}$:

1. $L(G) \subseteq L$:
   Dokážeme trošku silnejšie tvrdenie, a to nasledovné:

   Tvrdenie: Nech $w \in (N \bigcup T)^*$. Ak $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w$, potom

   $$\#_a w + \#_A w = \#_b w + \#_B w$$

   Dôkaz: Dôkaz urobíme indukciou cez dĺžku odvodenia n. Pre $n = 0$ je tvrdenie jasné. Nech teraz

   $$\sigma \Rightarrow w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_n \Rightarrow w_{n+1} = w$$

   Predpokladáme, že pre vetnú formu w_n toto tvrdenie platí a chceme dokázať, že platí aj pre w_{n+1}. Toto slovo mohlo vzniknúť použitím jedného z 7 pravidiel. Ani jedno z týchto pravidiel nemení hodnotu $\#_a w + \#_A w - \#_b w + \#_B w$. Napríklad, ak použijeme pravidlo $A \to bAA$, potom vlastne slovo w_{n+1} sa od w bude líšiť tak, že odbudne jedno A, pribudnú dve A a jedno b. Tým je táto časť dôkazu podaná.

2. $L \subseteq L(G)$:
   Opäť bude potrebné dokázať všeobecnejie tvrdenie, z ktorého toto vyplýva:

   Tvrdenie: Nech $w \in (N \bigcup T)^*$ a nech $|w| = n$.
   Potom platí:
   Ak $\#_a w + \#_A w = \#_b w + \#_B w$, potom $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w$.
   Ak $\#_a w + \#_A w = \#_b w + \#_B w + 1$, potom $A \overset{*}{\underset{G}{\Longrightarrow}} w$.
   Ak $\#_a w + \#_A w = \#_b w + \#_B w - 1$, potom $B \overset{*}{\underset{G}{\Longrightarrow}} w$.

   Dôkaz: Tentokrát by sme použili dôkaz indukciou cez dĺžku vetnej formy w. Pre $|w| = 0$ je z ľavých strán splnená jedine prvá a triviálne platí aj jej pravá strana. Nech toto tvrdenie platí pre všetky formy, ktorých dĺžka neprevyšuje n a overme, či je splnená aj pre $|w| = n + 1$.

   Nech vyzerá slovo w akokoľvek, vždy môže byť splnená najviac jedna ľavá strana dokazovaných implikácií. Sú 4 možnosti:

   (a) $\#_a w + \#_A w = \#_b w + \#_B w$ Musíme dokázať, že platí $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w$.
   Teraz máme 2 možnosti:

   i. $w = aw_1$. Potom slovo w_1 obsahuje o jedno béčko viac ako áčok. Jeho dĺžka je najviac n, a tak podľa indukčného predpokladu $B \overset{*}{\underset{G}{\Longrightarrow}} w_1$, a tak $\sigma \overset{*}{\underset{G}{\Longrightarrow}} aB \overset{*}{\underset{G}{\Longrightarrow}} aw_1 = w$.

   ii. $w = bw_1$. Dokazuje sa nalogicky ako v predchádzajucom prípade.

   (b) $\#_a w + \#_A w = \#_b w + \#_B w + 1$ Musíme dokázať, že platí $A \overset{*}{\underset{G}{\Longrightarrow}} w$.
   Opäť máme dve možnosti.

i. $w = bw_1$. Potom slovo w_1 obsahuje o dve viac áčok ako béčok. Existuje však jeho prefix w_1', ktorý obsahuje o jedno viac áčok ako béčok. Nech $w_1 = w_1'w_1''$. Potom slovo w_1'' obsahuje tiež o jedno viac áčok ako béčok. Dĺžky slov w_1', w_1'' sú nanajvýš n, tak6e podľa indukčného predpokladu $A \overset{*}{\underset{G}{\Longrightarrow}} w_1'$, ako aj $A \overset{*}{\underset{G}{\Longrightarrow}} w_1''$. Triviálne platí $A \overset{*}{\underset{G}{\Longrightarrow}} bAA$, a tak platí aj dokazované $A \overset{*}{\underset{G}{\Longrightarrow}} bw_1'w_1'' = bw_1 = w$

ii. $w = aw_1$. V tomto prípade slovo w_1 obsahuje rovnako áčok ako béčok. Podľa indukčného predpokladu teda $\sigma \overset{*}{\underset{G}{\Longrightarrow}} w_1$, a keďže triviálne platí aj $A \overset{*}{\underset{a}{\Longrightarrow}} \sigma$, platí aj $A \overset{*}{\underset{G}{\Longrightarrow}} aw_1 = w$, čo bolo treba dokázať.

(c) $\#_a w + \#_A w = \#_b w + \#_B w + 1$ Dokazované tvrdenie, že $B \overset{*}{\underset{G}{\Longrightarrow}} w$ sa dokáže ako v predchádzajucom prípade.

(d) $|\#_a w + \#_A w - \#_b w + \#_B w + 1| > 1$ V tomto prípade nie je splnená ani jedna ľavá strana imoplikácií, a tak sú všetky pravdivé.

Tým je dôkaz podaný. ♠

A.3 Konečné automaty

Príklad A.3.1. Zostrojte deterministický konečný automat (reprezentujte ho prechodovým diagramom, prechodovou tabuľkou a aj delta funkciou) pre jazyk:

$$L = \{w \mid w \text{ je číslo v dvojkovej sústave a po delení 3 dáva zvyšok 2}\}$$

(w je zapísané obvyklým spôsobom, teda najľavejšia cifra je najvýznamnejšia. Napr. 110 je 6.)

Riešenie *(:Riešenie tejto úlohy bude založené na tom, budeme mať 3 stavy, každý stav reprezentuje zvyšok po delení časti vstupného reťazca, ktorá už je prečítaná, tromi. Zrejme napríklad pri prečítaní znaku 0 v stave R_1, ktorý vyjadruje, že dosiaľ prečítaný reťazec je kongruentný s 1, prejdeme do stavu R_2. :)*
$A = (K = \{R_0, R_1, R_2\}, \Sigma = \{0,1\}, \delta, R_0, F = \{R_2\})$,
kde

$$\delta = \{((R_0,0),R_0),((R_0,1),R_1),((R_1,0),R_2),((R_1,1),R_0),((R_2,0),R_1),((R_2,1),R_2)\}$$

Dôkaz $L(A) = \{w \mid w \equiv 2 \pmod 3\}$:
Zrejme ak $c = 3k + r$, potom $2c + 0 = 3.2k + 2r$, a teda $2c + 0 \equiv 2r \pmod 3$. Ďalej $2c + 1 \equiv 3.2k + 2r + 1$, a teda $2c + 1 \equiv 2r + 1 (mod 3)$. Z toho vyplýva prechodová tabuľka

	0	1
R_0	R_0	R_1
R_1	R_2	R_0
R_2	R_1	R_2

Dokazovaná rovnosť je zrejmá. ♠

Príklad A.3.2. Dokážte, že ku každému nedeterministickému konečnému automatu A existuje nedeterministický konečný automat A', ktorý má práve jeden akceptujúci stav a pritom $L(A) = L(A')$.

Riešenie Nech $A = (K, \Sigma, \delta, q_0, F)$. Bez ujmy na všeobecnosti predpokladám $z \notin K$. *(:Využijeme jednu krásnu vlastnosť nedeterministických automatov, a to že môžeme z nejakého stavu do iného prejsť aj bez prečítania vstupného symbolu.:)* Definujeme nový automat $A' = (K \cup \{z\}, \Sigma, \delta \cup \{(f, \varepsilon) \to z \mid f \in F\}, q_0, \{z\})$.

Dokážeme, že $L(A) = L(A')$.

1. $L(A) \subseteq L(A')$ Ak $w \in L(A)$, potom $(q_0, w) \overset{*}{\underset{A}{\vdash}} (f, \varepsilon)$, ale každý krok odvodenia v A je aj krokom odvodenia v A', takže platí $(q_0, w) \overset{*}{\underset{A'}{\vdash}} (f, \varepsilon)$, ale triviálne platí $(f, \varepsilon) \vdash_{A'} (z, \varepsilon)$ a teda z tranzitívnosti $(q_0, w) \overset{*}{\underset{A'}{\vdash}} (z, \varepsilon)$, a teda $w \in L(A')$.
2. $L(A') \subseteq L(A)$ Táto časť je ešte elementárnejšia. Najprv si uvedomme, že neexistujú také slová u, v a stav q, že $(z, u) \overset{*}{\underset{A'}{\vdash}} (q, v)$, lebo zo stavu z sa nikam už nedá dostať.

   Keď slovo w patrí $L(A')$, potom $(q_0, w) \overset{*}{\underset{A'}{\vdash}} (f, \varepsilon) \vdash_{A'} (z, \varepsilon)$ pre nejaké $f \in F$. Ale všetky kroky odvodenia od počiatočnej konfigurácie (q_0, w) až po (f, ε) sú aj krokmi odvodenia v A, a teda platí $(q_0, w) \overset{*}{\underset{A}{\vdash}} (f, \varepsilon) \Rightarrow w \in L(A)$

♠

A.4 Neregulárne jazyky a zásobníkové automaty

Príklad A.4.1. Pomocou Myhill-Nerodovej vety dokážte, že $L = \{a^n b^n \mid n > 0\}$ nie je regulárny jazyk.

Riešenie Budeme postupovať sporom. Nech jazyk $L = \{a^n b^n\} \in R$. Z Myhill-Nerodovej vety potom vyplýva, že existuje relácia ekvivalencie E taká, že

1. Je konečného indexu
2. Je sprava invariantná
3. L je zjednotením niekoľkých jej tried rozkladu.

Označme index tejto relácie ekvivalencie k. Skúmajme slová z množiny $\{a, a^2, a^3, ..., a^{k+1}\}$. Z toho, že index E je k a Dirichletovho princípu vyplýva, že existujú prirodzené čísla i, j také že

$$1 \leq i < j \leq k + 1 \tag{A.1}$$

$$a^i \; E \; a^j \tag{A.2}$$

Z faktu, že E je sprava invariantná a z (A.2) vyplýva, že

$$a^i b^i \; E \; a^j b^i \tag{A.3}$$

Jazyk L má byť zjednotením niekoľkých tried rozkladu indukovaného reláciou E. Podľa (A.3) slová $a^i b^i$ a $a^j b^i$ patria tej istej triede rozkladu. Ale $a^i b^i \in L$, $a^j b^i \notin L$, čo je spor. ♠

Príklad A.4.2. Dokážte, že jazyk $L = \{w \mid \#_a w > \#_b w\}$ nie je regulárny.

Riešenie Budeme postupovať sporom. Nech jazyk L je regulárny. Podľa pumpovacej lemy pre regulárne jazyky existuje číslo p také, že pre všetky slová w z tohto jazyka dlhšie než p platí

$$(\exists u_1, u_2, v)(\forall k): \ (w = u_1 u_2 v) \ \wedge (|u_1 u_2| \leq p) \ \wedge (u_2 \neq \varepsilon) \ \wedge \ (u_1 u_2^k v \in L) \qquad \text{(A.4)}$$

Slovo $b^p a^{2p}$ zrejme patrí jazyku L, teda preň tiež platí (A.4). Slovo $u_1 u_2$, ak jeho dĺžka nemá presahovať p, musí byť zložené iba zo symbolov b. To ale znamená, že $w' = u_1 u_2^{3p} v$ má na začiatku aspoň $3p$ symbolov b a na konci presne $2p$ symbolov a.[2] Toto slovo ale definitoricky nemôže patriť jazyku L. ♠

Príklad A.4.3. Nech je daný (nedeterministický) konečný automat A s množinou stavov $K = \{q_0, q_1, ..., q_m\}$. Nech L je jazyk nad abecedou $\Sigma = K$ definovaný

$$L =_{df} \{q_1 q_2 ... q_n \mid \exists v \text{ , akceptujúci výpočet automatu } A \text{ na slove } v$$

$$\text{prechádzajúci po rade stavmi } q_1, q_2, ..., q_n\}$$

Dokážte, že L je regulárny jazyk a popíšte konštrukciu automatu rozpoznávajuceho jazyk L.

Riešenie Nech je daný NKA A s množinou stavov $K = \{q_0, q_1, ..., q_m\}$. Nech L je jazyk nad abecedou $\Sigma = K$ obsahujúci všetky také slová $p_0 p_1 ... p_n$, že existuje nejaké slovo w a akceptujúci výpočet automatu A na slove w prechádzajúci po rade stavmi $p_1, p_2, ..., p_n$. Dokážte, že L je regulárny jazyk.

Dokázať, že L je regulárny jazyk je ekvivalentné napríklad s existenciou konečného automatu akceptujúceho tento jazyk L. Skonštruujme preto nedeterministický konečný automat A' takto:

$A' := (K', \Sigma', \delta', q_0', F)$, kde
$K' = K \cup \{q_0'\}$
$\Sigma' = K$
$F' = F$
q_0' je nový symbol
a funkcia δ' je definovaná nasledovne:
$\delta'(q_0', q_0) = \{q_0\}$
$\delta'(q_i, q_j) = \{q_j \mid \exists s \in \Sigma; \ \delta(q_i, s) = q_j\}$

Automat A' skúma, či dané vstupné slovo u (postupnosť stavov) môže byť akceptujúcim výpočtom automatu A. Zrejme u môže byť výpočtom, len ak začína počiatočným stavom A. To je zabezpečené definíciou fcie δ'. Potom už automat kontroluje, či z daného stavu q_i môže automat A prejsť do stavu q_j. Ak áno, potom existuje symbol s, ktorý automat A prečítal v stave q_i. Takto sa dá konštruovať vstupné slovo w automatu A. Slovo w je akceptované, ak je automat v koncovom stave. Teda vtedy má byť v koncovom stave aj automat A', ale tak je predsa definovaná prechodová funkcia automatu A'. Automat A' je v koncovom stave, ak nejaký symbol s dostal automat A do koncového stavu, to jest ak nami konštruované slovo w je momentálne akceptované automatom A. Obrátene, ak máme automat A' v koncovom stave, tak je určite v koncovom aj automat A (je to rovnaký stav) a nami konštruované slovo w ho do tohto stavu priviedlo. ♠

[2] Sme naozaj veľkorysí

Príklad A.4.4. Nech L_1 je bezkontextový a L_2 regulárny jazyk. Potom Shuffle(L_1, L_2) je bezkontextový jazyk. Dokážte.

Riešenie Majme daný *zásobníkový automat*

$$A_z = (K_z, \Sigma_z, \Gamma, \delta_z, q_{0z}, Z_0, F_z)$$

ktorý rozpoznáva daný bezkontextový jazyk L_1 akceptačným stavom, a nedeterministický konečný automat

$$A_k = (K_k, \Sigma_k, \delta_k, q_{0k}, F_k)$$

rozpoznávajúci regulárny jazyk L_2.

Zostrojíme zásobníkový automat A rozpoznávajúci Shuffle(L_1, L_2)

$A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde:

$K = K_z \times K_k$ – nové stavy budú dvojice
$\Sigma = \Sigma_z \bigcup \Sigma_k$ – vstupná abeceda
$q_0 = [q_{0z}, q_{0k}]$ – počiatočný stav
$F = F_z \times F_k$ – akceptačné stavy budú tiež dvojice

A prechodovú funkciu δ definujeme:

$$\begin{aligned} \delta([q_z, q_k], a, Z) \quad = \quad & \{\ ([p_z, q_k], Z\gamma) \mid (p_z, Z\gamma) \ \in \ \delta_z(q_z, a, Z)\} \\ & \cup\{\ ([q_z, p_k], Z) \mid p_k \ \in \ \delta_k(q_k, a)\}\ pre\ a \in \Sigma \cup \{\varepsilon\};\ Z, \gamma \in \Gamma \end{aligned}$$

Z konštrukcie δ-funkcie je zrejmé, že $L(A) =$ Shuffle(L_1, L_2). Vidíme, že náš nedeterministický automat sa môže hocikedy rozhodnúť, či číta slovo z jazyka L_1 alebo L_2, pričom simuluje prechod automatom A_z alebo A_k. Náš automat sa nachádza v akceptačnom stave práve vtedy, kedy by sa nachádzali v akceptačnom stave aj simulované automaty A_z a A_k. ♠

Príklad A.4.5. Dokážte, že existujú bezkontextové jazyky L_1, L_2 také, že Shuffle(L_1, L_2) nie je bezkontextový.

Riešenie Príkladom takýchto jazykov by mohli byť $L_1 = \{a^n b^n\}$, $L_2 = \{c^n d^n\}$. Dokážem, že $L =_{df}$ Shuffle(L_1, L_2) nie je bezkontextový jazyk, a to sporom pomocou $p-q$-lemy. Nech teda L je bezkontextový jazyk. Potom existujú čísla p, q také, že

$$(\forall w \in L,\ |w| > p)\ \ \exists u, x, v, y, z \in \Sigma^*$$

také, že

1. $w = uxvyz$
2. $xy \neq \varepsilon$
3. $|xvy| \leq q$
4. $(\forall i \geq 0)\ ux^i vy^i z \in L$

Čo keby sme skúmali slovo $a^l c^l b^l d^l$, kde $l > q$, $l > p$. Toto slovo zrejme patrí jazyku L. Tretí bod však hovorí, že napumpovávané časti, x a y, musia byť k sebe „blízko".

$$aa...a\ cc...c\ bb...b\ dd...d$$

Kde teda tieto u, v budú? V každom prípade, nemôžeme zabezpečiť, ak sa týmto napumpovávaním x a y zväčšovuje počet povedzme *áčok*, aby sa zväčšoval i počet *béčok*, lebo tieto sú od seba ďaleko. Bohužiaľ, napumpovaním získame slovo, v ktorom počet *áčok* bude iný ako počet *béčok*, alebo počet *céčok* iný ako počet *déčok*. Ale pre všetky slová z L platí $\#_a w = \#_b w$ ako aj $\#_c w = \#_d w$, čo je vytúžený spor. ♠

Príklad A.4.6. Dokážte, že jazyk $L = \{u\#v^R \mid u, v$ sú konfigurácie daného Turingovho stroja také, že z u sa jedným krokom dostaneme do $v\}$ je bezkontextový.

Riešenie Nech $M = (K_M, \Sigma, \Gamma, \delta_M, q_{M0})$ je daný Turingov stroj. Ako vieme, jazyk

$$L' = \{w\#w^R \mid w \text{ je konfigurácia } M\}$$

je bezkontextový.

Stačí teda zostrojiť A-prekladač A, ktorý jazyk L' transformuje na jazyk L.

Kvôli jednoduchosti predpokladajme, že konfigurácie M sú v tvare

$$a_0 a_1 ... (a_i, q) a_{i+1} ... a_n$$

kde q je aktuálny stav a i je pozícia hlavy. Ďalej predpokladajme, že množiny stavov M a jeho pracovnej abecedy sú disjunktné. Zostrojme takýto A-prekladač:

$$A = (K, \Sigma_1 = \Gamma \cup \Gamma \times K_M, \Sigma_2 = \Sigma_1, H, q_0, F = \{q_{konc}\})$$

kde $K = \{q_0, q_{konc}\} \cup \{q_a \mid a \in \Gamma \cup \{\$, \S, \varepsilon\}\ \} \cup \{q_p \mid p \in K_M\}$ a δ funkciu definujeme nasledovne:

$H(q_0, \S) = (q_0, \S)$	prepíš počiatočný cent
$H(q_0, a) = (q_0, a)\ \forall a \neq \#$	kým nie je mriežka, iba prepisuj
$H(q_0, \#) = (q_\varepsilon, \#)$	mriežku vypíš, zapamätaj si nič-t.j. ε
$H(q_a, b) = (q_b, a)$ pre $\forall a, b \in \Gamma \cup \{\$, \S, \varepsilon\}$	zapíš zapamätané a, zapamätaj nové b
$H(q_a, (b, q)) = \{(q_c, (a, r)) \mid \delta_M(q, b) \ni (r, c, -1)\} \cup$	A je na pozícii hlavy, M robil krok vľavo
$\cup \{(q_\varepsilon, a(c, r)) \mid \delta_M(q, b) \ni (r, c, 0)\}$	A je na pozícii hlavy, M stojí na mieste
$\cup \{(q_r, ac) \mid \delta_M(q, b) \ni (r, c, 1)\}$	A je na pozícii hlavy, M robil krok vpravo
$H(q_p, b) = (q_\varepsilon, (b, p))$	vypíš zapamätaný stav a prečítaný symbol
$H(q_a, \$) = (q_{konc}, a\$)$	skonči

A-prekladač A simuluje jeden krok Turingovho stroja, keď narazí na pozíciu hlavy. V indexe stavu si A-prekladač pamätá posledne prečítaný symbol, aby v prípade, že Turingov stroj šiel vľavo, vedel toto zabezpečiť.

Z konštrukcie vyplýva, že $H(L') = L$, a teda jazyk L je bezkontextový, čo bolo treba dokázať. ♠

A.5 Lineárne ohraničené automaty a kontextové jazyky

Príklad A.5.1. Zostrojte lineárne ohraničený automat (LBA) pre nasledujúci jazyk:

$$L_1 = \{ww \mid w \in \{a, b\}^*\}$$

Riešenie Automat môže pracovať napríklad takto: Číta vstupné slovo (napr. ww), pričom sa kdesi nedeterministicky rozhodne, že momentálne sa nachádza na poslednom písmene slova w, t.j. na poslednom písmene prvej polovičky vstupného slova. Potom cyklicky vykonáva nasledovné:

1. Prepíše písmeno na páske znakom 0 a prejde do stavu s_a, resp. s_b podľa toho, aké písmeno to bolo
2. V stave s_a, resp. s_b číta vstupné symboly zľava doprava. Kým sú týmito symbolmi nuly, zostáva v stave, v ktorom je. Keď však z pásky prečíta iný znak, prejde do stavu s'_a, resp. s'_b a opäť číta vstupné symboly a keď narazí na znak 0, alebo \$, prejde do stavu s''_a, resp. s''_b.
3. Pohne sa o 1 pozíciu doľava a v prípade, že znak na páske sa nezhoduje s indexom stavu, potom prejde do neakceptačného stavu, v ktorom sa môže napríklad zaseknúť. Ak sa zhoduje, nahradí ho znakom 0 a prejde do stavu q_L, ktorý indikuje prechod doľava. V tomto stave zostáva a robí pohyb doľava, až kým nenarazí na znak 0. V ňom prejde do stavu q'_L a opäť číta vstupné symboly a prechádza páskou doľava. V momente, keď narazí na znak rôzny od nuly, prejde do stavu s_a, resp. s_b.
4. Kedy skončiť? Uvedomme si, že skončiť môžeme vtedy, ak v stave q'_L narazíme na znak ¢.

Formálne:
$A = (K, \Sigma, \Gamma = \Sigma \cup \{0\}, \delta, q_0 = z, F = \{q_{akc}\})$
$K = \{q_0, q'_0, s_a, s_b, s'_a, s'_b, s''_a, s''_b, q_L, q'_L, q_{akc}\}$
δ funkcia je definovaná nasledovne: $\delta(q_0, \#) = \{(q_0, \#, 1)\}$ [3]
$\delta(q_0, \$) = \{(q_{akc}, \$, 0)\}$
$\delta(q_0, a) = \{(q'_0, a, 1), (s_a, 0, 1)\}$
$\delta(q_0, b) = \{(q'_0, b, 1), (s_b, 0, 1)\}$

$\delta(q'_0, a) = \{(q'_0, a, 1), (s_a, 0, 1)\}$
$\delta(q'_0, b) = \{(q'_0, b, 1), (s_b, 0, 1)\}$

$\delta(s_a, 0) = \{(s_a, 0, 1)\}$
$\delta(s_a, a) = \{(s'_a, a, 1)\}$
$\delta(s_a, b) = \{(s'_a, b, 1)\}$
analogicky pre $\delta(s_b, ...)$

$\delta(s'_a, 0) = \{(s''_a, 0, -1)\}$
$\delta(s'_a, \$) = \{(s''_a, \$, -1)\}$
analogicky pre $\delta(s'_b, ...)$

$\delta(s''_a, a) = \{(q_L, 0, -1)$
$\delta(s''_b, b) = \{(q_L, 0, -1)$

$\delta(q_L, a) = \{(q_L, a, -1)\}$
$\delta(q_L, b) = \{(q_L, b, -1)\}$
$\delta(q_L, 0) = \{(q'_L, 0, -1)\}$

$\delta(q'_0, 0) = \{(q'_0, 0, -1)\}$
$\delta(q'_L, a) = \{(s_a, 0, 1)\}$

[3] V stave q_0 sa nachádza, ak ešte neprečítal žiadny vst. symbol okrem počiatočného ¢. Ak prečíta koncový \$, potom ho akceptujeme, inak prechádzame do stavu q'_0-stavu hľadania stredu

$\delta(q'_L, b) = \{(s_b, 0, 1)\}$
$\delta(q'_L, ¢) = \{(q_{akc}, ¢, 0)\}$

Dôležité je, že tento automat bude akceptovať akceptujúcim stavom bez uohľadu na to, kde sa nachádza hlava. T.j., v momente, keď sa ocitne v akceptujúcom stave, je slovo akceptované. ♠

A.6 Rozhodnuteľnosť/nerozhodnuteľnosť

Príklad A.6.1. Je rozhodnuteľné, či pre dané homomorfizmy $h_1, h_2 : \Sigma_1 \to \Sigma_2$ existuje slovo w také, že $h_1(w) = h_2(w)$?

Riešenie Ukážeme, že tento problém nie je rozhodnuteľný, lebo keby bol rozhodnuteľný, potom by aj PKP bol rozhodnuteľný.

Nech je daný PKP problém $P = \{(u_i, v_i) \mid i = 1, 2, ...n\}$ nad abecedou Σ. Definujme abecedy $\Sigma_1 = \{1, 2, ..., n\}$, $\Sigma_2 = \Sigma$ a homomorfizmy h_1 a h_2 nasledovne: $h_1(i) = u_i$, $h_2(i) = v_i$. Zrejme platí

$$(\exists w)\ (h_1(w) = h_2(w)) \iff \text{PKP } P \text{ má riešenie}$$

Totiž, ak takéto slovo $w = i_1 i_2 ... i_k$ existuje, potom priložením dominových kociek číslo $i_1, i_2, ..., i_k$ dostaneme riešenie PKP. Opačne platí analogická úvaha.

Keby sme teda vedeli rozhodnúť problém zo zadania, potom by sme vedeli rozhodnúť ľubovoľný PKP, čo je spor. ♠

Príklad A.6.2. Majme triedu Turingových strojov takých, ktoré nemôžu zapisovať na pásku blank. Je pre takéto stroje rozhodnuteľný problém

Pre daný TS A, vstupné slovo w a konfiguráciu k, existuje výpočet A na w, ktorý prechádza cez konfiguráciu k?

Riešenie Uvedomme si, že takýto Turingov stroj nikdy nedokáže skrátiť slovo zapísané na páske. Preto by stačilo overiť pre konečne veľa vstupných slov w (všetkých možných, ktorých dĺžka nepresahuje dĺžku slova v konfigurácii k), či $w \overset{*}{\underset{A}{\vdash}} k$. A pritom sa opäť stačí obmedziť na také výpočty, v ktorých žiadna konfigurácia svojou dĺžkou nepresahuje dĺžku slova v konfigurácii k. Ak si uvedomíme, že takýchto konfigurácií je konečne veľa, a že vieme detekovať zacyklenie sa daného Turingovho stroja A, odpoveď znie, že tento problém je riešiteľný. ♠

Index

Literatúra

[1] John E. Hopcroft, Jeffrey D. Ullman: Formálne jazyky a automaty. (Alfa SNTL, 1978)

[2] S. Ginsburg: The Mathematical Theory of Context-Free Languages. (McGrew-Hill, 1966)

[3] Michal Chytil: Teorie automatů a formálních jazyků. (Státní Pedagogické Nakladatelství Praha, 1978)

[4] Michal Chytil: Automaty a gramatiky. (Matematický seminář SNTL, 1984)