Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Formálne Jazyky a Automaty

(skriptá)

prof. RNDr. Branislav Rovan, PhD.

RNDr. Michal Forišek, PhD.

verzia zo dňa **8. marca 2011**

Úvod

Tieto skriptá obsahujú študijné materiály k predmetu *Formálne jazyky a automaty* na Fakulte matematiky, fyziky a informatiky UK. Veľká väčšina použitých materiálov pochádza z prednášok prof. RNDr. Branislava Rovana, PhD., ktoré odzneli v rámci uvedeného predmetu v akademických rokoch 1998/99 až 2007/08. Skriptá vlastnými materiálmi doplnil a spísal RNDr. Michal Forišek.

Niektoré časti týchto skrípt sú prevzaté z poznámok z týchto prednášok, ktoré anonymne spísali v akademickom roku 1998/99 študenti. Okrem nich patrí poďakovanie viacerým študentom v neskorších akademických rokoch, ktorí sa pričinili o hľadanie a opravu chýb v týchto skriptách. V roku 2011 opravil a doplnil niektoré pasáže Marek Zeman.

Obsah

0 Chýbajúce časti **6**

1 Základné pojmy **7**
1.1 Definície . . . 7
1.2 Operácie na slovách a jazykoch . . . 8
1.3 Automaty . . . 9
1.4 Gramatiky . . . 9
1.5 Triedy jazykov . . . 11
1.6 Jazyk ako problém . . . 11

2 Regulárne jazyky **13**
2.1 Deterministický konečný automat . . . 13
2.2 Nedeterministický konečný automat . . . 15
2.3 Ekvivalencia DKA a NKA . . . 16
2.4 Normálne tvary konečných automatov a regulárnych gramatík . . . 18
2.5 Ekvivalencia konečných automatov a regulárnych gramatík . . . 19
2.6 Uzáverové vlastnosti triedy regulárnych jazykov . . . 20
2.7 Kleeneho veta . . . 23
2.8 Pumpovacia lema pre regulárne jazyky . . . 24
2.9 Myhill-Nerodova veta . . . 25
2.10 Regulárne výrazy . . . 28
2.11 Zovšeobecnenia konečných automatov . . . 29
2.11.1 Automaty s pružnou hlavou . . . 29
2.11.2 Viachlavé automaty . . . 29
2.11.3 Dvojsmerné automaty . . . 30

3 Bezkontextové jazyky **34**
3.1 Redukované bezkontextové gramatiky . . . 34
3.2 Stromy odvodenia . . . 36
3.3 Chomského normálny tvar . . . 38
3.4 Bezepsilonové gramatiky . . . 39
3.5 Greibachovej normálny tvar . . . 40
3.6 Reťazcové pravidlá (Chain rules) . . . 41
3.7 Zásobníkové automaty . . . 42
3.8 Ekvivalencia medzi akceptáciou prázdnou pamäťou a akceptačným stavom . . . 44
3.9 Ekvivalencia bezkontextových gramatík a zásobníkových automatov . . . 44
3.10 Uzáverové vlastnosti bezkontextových jazykov . . . 46
3.11 Pumpovacia lema pre bezkontextové jazyky . . . 47
3.12 Zistenie príslušnosti slova do bezkontextového jazyka . . . 50

4 Zložitejšie modely **51**
4.1 A-prekladače . . . 51
4.1.1 Zobrazenia a-prekladačom . . . 52
4.1.2 Normálny tvar a-prekladača . . . 53
4.1.3 Použitie a-prekladačov . . . 53
4.2 Deterministické zásobníkové automaty . . . 53
4.2.1 Normálny tvar pre DPDA . . . 54
4.2.2 Akceptovanie stavom, pamäťou a jednoznačné gramatiky . . . 56
4.2.3 Uzáverové vlastnosti triedy $\mathscr{L}_{DPDA}$ . . . 57

5 Kontextové jazyky **60**
5.1 Rozšírené kontextové gramatiky . . . 60
5.2 Lineárne ohraničené automaty . . . 61
5.3 Ekvivalentnosť kontextových gramatík a lineárne ohraničených automatov . . . 62
5.4 Normálne tvary kontextových gramatík . . . 64
5.5 Uzáverové vlastnosti kontextových jazykov . . . 65
5.6 Szelepczényiho veta . . . 68

6 Rekurzívne vyčísliteľné a rekurzívne jazyky **70**
6.1 Turingove stroje . . . 70
6.2 Ekvivalencia det. a nedet. Turingovych strojov . . . 71
6.3 Varianty Turingovych strojov . . . 72
6.4 Ekvivalencia Turingovych strojov a frázových gramatík . . . 74
6.5 Uzáverové vlastnosti rekurzívne vyčísliteľných jazykov. . . . 75
6.6 Kódovanie jazykov a TS . . . 75
6.7 Diagonálny a univerzálny jazyk . . . 76
6.8 Rekurzívne jazyky, riešiteľnosť, rozhodnuteľnosť . . . 78
6.9 Základné nerozhodnuteľné problémy . . . 80
6.9.1 Turingovská a many-to-one redukcia . . . 80
6.9.2 Problém zastavenia . . . 81
6.9.3 Postov korešpondenčný problém . . . 82
6.10 Rozhodnuteľnosť otázok o jazykoch známych tried . . . 84
6.11 Riceove vety . . . 87

7 Úvod do teórie zložitosti **91**
7.1 Model TS a definície zložitosti . . . 91
7.2 Vety o kompresii, zrýchlení a redukcii počtu pások . . . 92
7.3 Triedy zložitosti . . . 94
7.4 Uzáverové vlastnosti tried zložitosti . . . 95
7.5 Vzťahy medzi deterministickým a nedeterministickým časom a priestorom . . . 96
7.6 Redukcie, ťažké a úplné problémy . . . 98

8 Syntaktická analýza **100**
8.1 Všeobecné bezkontextové gramatiky . . . 100
8.2 LR(0) gramatiky . . . 101
8.2.1 Motivácia a pomocné definície . . . 101
8.2.2 Definícia LR(0) gramatiky . . . 103
8.2.3 Vzťah LR(0) gramatík a DPDA . . . 103

9 Riešené úlohy **104**
9.1 Základné pojmy . . . 104
9.2 Regulárne jazyky . . . 104
9.3 Bezkontextové jazyky . . . 106
9.4 Zložitejšie modely . . . 108

9.5 Kontextové jazyky 109
9.6 Rekurzívne vyčísliteľné a rekurzívne jazyky 110
9.7 Úvod do teórie zložitosti 113

Kapitola 0

Chýbajúce časti

Tu je zoznam chýbajúcich častí, ktoré treba dokončiť.

- Kapitola 7 (zloz): redukcia k pasok na 2 pre čas
- Kapitola 7 (zloz): vztah P a NSPACE(o(log N))
- Kapitola 7 (zloz): vztah NLOG a P
- Kapitola 7 (zloz): pokec o P=NP
- Kapitola 7 (zloz): pokec o rozhodnuteľnosti, rekurzívnosti, λ-kalkule a Gödelovi
- Kapitola 8 (syntax): priklad polozkoveho automatu
- Kapitola 8 (syntax): pokec o top-down vs bottom-up parseroch?
- Kapitola 8 (syntax): lrk, precedencne gramatiky
- Kapitola 8 (syntax): prekladove schemy
- INE: opravit velkost fontu v obrazkoch, nech to nehadze tie blbe warningy
- INE: po definicii konfiguracie vzdy uviest, ako vyzera zaciatocna konfig. na slove w

Kapitola 1

Základné pojmy

1.1 Definície

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*** *je konečná neprázdna množina symbolov (písmen). Označuje sa* Σ.

Definícia 1.1.2 ***Slovo*** *nad abecedou* Σ *je konečná postupnosť symbolov z* Σ. *Prázdnu postupnosť (prázdne slovo) označujeme* ε.

Poznámka 1.1.1 Pri zápise slova vynechávame čiarky oddeľujúce 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* $v = a_1a_2 \dots a_n$ *je súvislá podpostupnosť* $a_ia_{i+1} \dots a_j$, *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 nad abecedou Σ sú napr. aa, cab, ε.

Definícia 1.1.5 ***Jazyk*** *nad abecedou* Σ *je ľubovoľná množina slov nad abecedou* Σ.

Poznámka 1.1.2 Symboly označujeme malými písmenami zo začiatku abecedy $(a, b, c, \dots)$, slová písmenami z konca abecedy $(u, v, w, \dots)$, jazyky najčastejšie písmenom L s vhodným indexom. Zápis Σ^* značí množinu všetkých slov nad abecedou Σ. Znakom $\#_a(w)$ označujeme počet písmen a v slove w.

Príklad 1.1.2 Množiny $L_1 = \{a, b, aa, aba\}$, $L_2 = \{w \in \{a, b\}^* \mid \#_a(w) \text{ je párny}\}$ sú jazykmi nad abecedou $\{a, b\}$. Jazyk L_1 je konečný, L_2 je nekonečný. Do L_2 patria napr. slová ε, bbb, aab, $aaaa$, ale nepatria tam slová a ani $abaa$.

Poznámka 1.1.3 Na jazyk $L = \{a, ab\}$ sa môžeme dívať ako na jazyk nad abecedou $\{a, b\}$, ale aj napr. ako na jazyk nad abecedou $\{a, b, c\}$.[1] Niekedy má zmysel uvažovať najmenšiu abecedu, nad ktorou je L. Takúto abecedu budeme značiť Σ_L.

[1] Keby sme chceli zájsť do extrému, mohli by sme definovať jazyk ako dvojicu $L = (\Sigma, W)$, kde Σ je konečná abeceda jazyka L a $W \subseteq \Sigma^*$ množina jeho slov.

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 povedať, čo vlastne je univerzálna množina. Zväčša za ňu pokladáme jazyk Σ_L^*. Okrem týchto operácií definujeme niekoľko ďalších, ktoré sa nám neskôr budú hodiť.

zreťazenie slov

Nech $u = u_1u_2 \dots u_n$, $v = v_1v_2 \dots v_m$, potom $u \cdot v = u_1u_2 \dots u_nv_1v_2 \dots v_m$. Operácia zreťazenia teda pripojí druhé slovo na koniec prvého. Znak zreťazenia (bodku) budeme väčšinou vynechávať, uvedieme ho len ak by mohlo dôjsť k omylu.

zreťazenie jazykov

$L_1 \cdot L_2 = \{uv \mid u \in L_1 \ \wedge \ v \in L_2\}$

Aj pri zreťazení jazykov budeme namiesto $L_1 \cdot L_2$ často písať L_1L_2. Uvedomte si, že podľa definície pre ľubovoľný jazyk L platí $L \cdot \emptyset = \emptyset$.

Kvôli prehľadnejšiemu zápisu definujeme aj zreťazenie slova s jazykom: $w \cdot L = \{w\} \cdot L$. (A analogicky $L \cdot w = L \cdot \{w\}$.)

mocnina slova

$w^0 = \varepsilon, \quad \forall n \geq 1; \ w^n = w \cdot w^{n-1}$

Teda napríklad $a^3bc^0a^2 = aaabaa$. Všimnite si, že naozaj $w^1 = w$. Uvedomte si, že $(ab)^2 \neq a^2b^2$.

mocnina jazyka

$L^0 = \{\varepsilon\}, \quad \forall n \geq 1; \ L^n = L \cdot L^{n-1}$

Všimnite si, že $L^0 \neq \emptyset$ a že (vďaka tomu) $L^1 = L$. Intuitívne: L^i obsahuje slová, ktoré sú zreťazením nejakých i slov z jazyka L. Zreťazením 0 slov dostávame prázdne slovo.

iterácia (uzáver, Kleeneho $*$)

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

L^* obsahuje všetky slová, ktoré dostaneme zreťazením niekoľkých slov z L. Všimnite si, že značenie Σ^*, ktoré sme už používali, je konzistentné s touto definíciou.

kladná iterácia (uzáver, Kleeneho $+$)

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

reverz (zrkadlový obraz) slova, jazyka

Nech $u = u_1u_2 \dots u_n$, potom $u^R = u_n \dots u_2u_1$. $L^R = \{u^R \mid u \in L\}$.

homomorfizmus

Nech h je zobrazenie zo Σ_1^* do Σ_2^* (t.j. h zobrazí každé slovo nad Σ_1 na nejaké slovo nad Σ_2) také, že $\forall u, v \in \Sigma_1^*; \ h(u \cdot v) = h(u) \cdot h(v)$. Potom h voláme homomorfizmus.

Ak $\forall x \in \Sigma^*; \ h(x) = \varepsilon \iff x = \varepsilon$, hovoríme, že homomorfizmus h je *nevymazávajúci*.

Obraz jazyka L ($L \subseteq \Sigma_1^*$) pri zobrazení homomorfizmom h je jazyk $h(L) = \{h(w) \mid w \in L\}$.

inverzný homomorfizmus

Nech h je homomorfizmus. Zjavne nemusí byť surjektívny ani injektívny, napriek tomu však k nemu vieme definovať zobrazenie h^{-1}, ktoré bude v istom zmysle inverzné. Toto zobrazenie budeme volať inverzný homomorfizmus.

Obraz slova v inv. homomorfizmom h^{-1} bude **jazyk** (nie slovo!) $h^{-1}(v) = \{u \mid h(u) = v\}$. Teda h^{-1} zobrazí slovo v na jazyk tých slov, ktoré h zobrazí na v.

Obraz jazyka L pri zobrazení inverzným homomorfizmom h^{-1} je jazyk $h^{-1}(L) = \bigcup_{v \in L} h^{-1}(v) = \{u \mid h(u) \in L\}$. Teda je to jazyk tých slov, ktoré h zobrazí na slovo z L.

Poznámka 1.2.1 Z definície homomorfizmu vyplýva, že pre ľubovoľný homomorfizmus $h : \Sigma_1^* \to \Sigma_2^*$ a ľubovoľné slovo $u = u_1u_2 \dots u_n \in \Sigma_1^*$ platí: $h(u) = h(u_1u_2 \dots u_n) = h(u_1) \cdot h(u_2) \cdot \dots \cdot h(u_n)$. Preto každý homomorfizmus je jednoznačne určený obrazmi jednotlivých písmen zo Σ_1. Teda homomorfizmus h je nevymazávajúci, ak $\forall x \in \Sigma; \ h(x) \neq \varepsilon$.

Príklad 1.2.1 Nech $L_1 = \{\varepsilon, a, ab\}$, $L_2 = \{a, aa\}$. Potom:
$L_1 \cup L_2 = \{\varepsilon, a, aa, ab\}$
$L_1 \cap L_2 = \{a\}$
$L_1 \cdot L_2 = \{a, aa, aaa, aba, abaa\}$

Príklad 1.2.2 Nech h je homomorfizmus taký, že $h(a) = 0$, $h(b) = 101$, $h(c) = 01$. Potom $h(\varepsilon) = \varepsilon$, $h(abac) = 0101001$, $h^{-1}(0101) = \{ab, cc\}$, $h^{-1}(1) = \emptyset$.

1.3 Automaty

Základným cieľom teórie formálnych jazykov je formalizovať pojem výpočtu a zaoberať sa tým, ktoré problémy sú z hľadiska algoritmickej zložitosti nakoľko ťažké. Preto v nasledujúcich kapitolách definujeme niekoľko typov automatov, ktoré budú mať čím ďalej, tým väčšiu výpočtovú silu. Záverom nášho snaženia bude definícia a skúmanie vlastností Turingovych strojov, ktoré sa považujú za (jednu možnú) formalizáciu klasického počítača.

Všetky spomínané automaty budú fungovať na rovnakom princípe: keď dostanú vstupné slovo, niečo počítajú a prípadne časom vyhlásia, že slovo je dobré. Jazyk rozpoznávaný (akceptovaný) takýmto automatom bude množina slov, ktoré vyhlási za dobré.

1.4 Gramatiky

Jednu možnosť ako špecifikovať jazyky nám dávajú vyššie spomenuté typy automatov. Ukážeme si ešte jeden aparát, ktorý je veľmi podobný deklaratívnym programovacím jazykom – gramatiky. Definujeme si niekoľko typov gramatík, neskôr sa budeme každým z nich podrobnejšie zaoberať a ukážeme, ako zložité jazyky je ktorý typ gramatík schopný špecifikovať (a teda ako zložité problémy je schopný riešiť).

Definícia 1.4.1 ***Frázová gramatika*** *je štvorica $G = (N, T, P, \sigma)$, kde N a T sú abecedy neterminálnych a terminálnych*[2] *symbolov ($N \cap T = \emptyset$), $\sigma \in N$ je začiatočný neterminál a $P \subseteq_{kon} (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ je konečná množina prepisovacích pravidiel.*

Poznámka 1.4.1 Fakt, že $(u, v) \in P$ môžeme tiež zapisovať $u \underset{P}{\longrightarrow} v$, $u \underset{G}{\longrightarrow} v$, prípadne len $u \to v$ (ak vieme, o akú množinu pravidiel P, resp. gramatiku G ide). Pravidlá $u \to v_1$, $u \to v_2$, $\ldots u \to v_n$ s rovnakou ľavou stranou zapisujeme skrátene $u \to v_1 \mid v_2 \mid \ldots \mid v_n$.

Definícia 1.4.2 ***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 $x = w_1 u w_2$ a $y = w_1 v w_2$.*

Definícia 1.4.3 ***Jazyk*** *generovaný gramatikou G je množina $L(G) = \{w \in T^* \mid \sigma \underset{G}{\Longrightarrow}^* w\}$.*

Poznámka 1.4.2 Ak bude jasné, o akú gramatiku ide, budeme namiesto symbolu $\underset{G}{\Longrightarrow}$ písať $\Rightarrow$.

Poznámka 1.4.3 Uvedomme si, že $\Rightarrow$ je binárna relácia. Má zmysel uvažovať jej mocniny, prípadne tranzitívny a reflexívno-tranzitívny uzáver: Napr. $u \Rightarrow^2 v$, ak $\exists w$ také, že $u \Rightarrow w$ a $w \Rightarrow v$. Značenie $u \Rightarrow^2 v$ teda znamená, že z vetnej formy u vieme na 2 kroky odvodiť vetnú formu v. Podobne napr. značenie $u \Rightarrow^* v$ znamená, že z u vieme v vygenerovať konečným počtom krokov. (Podobne ako napr. u uzáverov jazykov symbol $\Rightarrow^+$ označuje tranzitívny a $\Rightarrow^*$ reflexívno-tranzitívny uzáver relácie $\Rightarrow$.)

Definícia 1.4.4 ***Odvodenie*** *$\sigma \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$ je postupnosť krokov odvodení.* ***Dĺžka odvodenia*** *je počet takýchto krokov.*

[2] terminálny = koncový

Definícia 1.4.5 ***Vetná forma*** *je slovo z $(N \cup T)^*$, ktoré môžeme získať odvodením zo začiatočného neterminálu.*

Definícia 1.4.6 *Hovoríme, že gramatiky G_1, G_2 sú* ***ekvivalentné****, ak $L(G_1) = L(G_2)$.*

Príklad 1.4.1 Majme nasledovnú gramatiku: $G = (\{\sigma\}, \{a, b\}, \{\sigma \to a\sigma b,\ \sigma \to \varepsilon\}, \sigma)$. Ak začneme zo začiatočného 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 $L(G) = \{a^n b^n \mid n \geq 0\}$. Toto tvrdenie teraz poriadne dokážeme.

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

1° Na $m = 0$ krokov vieme odvodiť len vetnú formu σ, pre tú tvrdenie platí.

2° Nech IP platí $\forall m \leq k$, dokážeme, že platí aj pre $m = k + 1$. Nech $\sigma \Rightarrow^k w_k \Rightarrow w_{k+1}$. Podľa IP je w_k tvaru $a^i b^i$ alebo $a^i \sigma b^i$. Keďže odvodenie ešte mohlo pokračovať, nutne $w_k = a^i \sigma b^i$. Ak sme v $(k + 1)$. kroku použili pravidlo $\sigma \to \varepsilon$, tak $w_{k+1} = a^i b^i$. Ak sme použili pravidlo $\sigma \to a\sigma b$, tak $w_{k+1} = a^{i+1} \sigma b^{i+1}$. Pre obe tieto vetné formy dokazované tvrdenie platí.

$\supseteq$: Potrebujeme dokázať, že pre každé n vieme vygenerovať slovo tvaru $a^n b^n$. Toto odvodenie vieme priamo zostrojiť: n-krát použijeme pravidlo $\sigma \to a\sigma b$ a následne raz pravidlo $\sigma \to \varepsilon$. Dostávame odvodenie $\sigma \Rightarrow^n a^n \sigma b^n \Rightarrow a^n b^n$.

Skôr než 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 množina pravidiel, sú konečné), možno gramatikám jedno-jednoznačne priradiť prirodzené čísla, a teda rôznych gramatík je spočítateľne veľa.

Na druhej strane ľahko nahliadneme, že množina $\{a, b\}^*$ je nekonečná spočítateľná, a teda množina jej podmnožín (jazykov nad abecedou $\{a, b\}$) je nespočítateľná. Rôznych jazykov je teda nespočítateľne veľa.

To značí, že k drvivej väčšine jazykov neexistuje gramatika, ktorá ich generuje. Avšak také jazyky sú veľmi zložité a je veľmi ťažké ich definovať. Pre ľubovoľný jazyk, ktorý si viete v tejto chvíli predstaviť, existuje frázová gramatika, ktorá ho generuje. Až keď budeme mať vybudovaný dostatočný aparát, ukážeme niekoľko jazykov, pre ktoré frázová gramatika neexistuje.

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

Definícia 1.4.7 ***Kontextová gramatika*** *je taká frázová gramatika, v ktorej pre každé pravidlo $u \to v$ platí $|u| \leq |v|$.*

Poznámka 1.4.4 V pôvodnej Chomského definícii mali pravidlá tvar $u\xi v \to uwv$, kde $u, v \in (N \cup T)^*$, $w \in (N \cup T)^+$, $\xi \in N$. (Teda neterminál ξ sa mohol prepísať na w len vtedy, ak sa vyskytoval v správnom kontexte – odtiaľ názov kontextová gramatika.) Dá sa ukázať, že tieto dve definície sú ekvivalentné.

Definícia 1.4.8 ***Bezkontextová gramatika*** *je taká frázová gramatika, v ktorej navyše platí $P \subseteq N \times (N \cup T)^*$.*

Poznámka 1.4.5 Bezkontextová gramatika má teda pravidlá, ktorých ľavú stranu tvorí práve jeden neterminál. Názov „bezkontextová“ pochádza teda zo skutočnosti, že príslušný neterminál môžeme prepísať bez ohľadu na kontext (okolie), v ktorom sa nachádza.

Definícia 1.4.9 ***Regulárna gramatika*** *je taká frázová gramatika, v ktorej navyše platí $P \subseteq N \times T^*(N \cup \{\varepsilon\})$.*

Poznámka 1.4.6 Regulárna gramatika má navyše (oproti bezkontextovej) obmedzenú aj pravú stranu pravidiel – tá môže obsahovať najviac jeden neterminál a ten musí byť posledný. Uvedomte si, že vďaka tomu regulárna gramatika generuje slovo „zľava doprava“.

1.5 Triedy jazykov

Trieda jazykov (family of languages) je zaužívané označenie pre množinu jazykov.[3] Niekoľko „rozumných“ tried jazykov už vieme definovať. Predovšetkým je to trieda všetkých konečných jazykov. Navyše každý typ gramatík, ktorý sme si definovali, nám určuje jednu triedu jazykov. Pre tieto triedy sa zaužívali nasledujúce názvy:

frázové (rekurzívne vyčísliteľné, recursively enumerable) jazyky
$\mathscr{L}_{RE} = \{L \mid \text{ existuje frázová gram. } G \text{ taká, že } L = L(G)\}$

kontextové (context sensitive) jazyky
Kvôli našej definícií kontextovej gramatiky (pravá strana každého pravidla musí byť aspoň tak dlhá ako ľavá) nevie žiadna kontextová gramatika vygenerovať prázdne slovo. Preto za kontextové jazyky budeme považovať nielen jazyky generované kontextovými gramatikami, ale aj jazyky, ktoré z nich dostaneme pridaním prázdneho slova.
$\mathscr{L}_{ECS} = \{L,\ L \cup \{\varepsilon\} \mid \text{ existuje kontextová gram. } G \text{ taká, že } L = L(G)\}$

Ak budeme chcieť hovoriť špeciálne o jazykoch, generovaných kontextovými gramatikami, použijeme značenie $\mathscr{L}_{CS}$:
$\mathscr{L}_{CS} = \{L \mid \text{ existuje kontextová gram. } G \text{ taká, že } L = L(G)\}$

bezkontextové (context free) jazyky
$\mathscr{L}_{CF} = \{L \mid \text{ existuje bezkontextová gram. } G \text{ taká, že } L = L(G)\}$

regulárne (regular) jazyky
$\mathcal{R} = \{L \mid \text{ existuje regulárna gram. } G \text{ taká, že } L = L(G)\}$

Poznámka 1.5.1 Tieto triedy tvoria tzv. Chomského hierarchiu – totiž platí, že $\mathscr{L}_{RE} \supseteq \mathscr{L}_{ECS} \supseteq \mathscr{L}_{CF} \supseteq \mathcal{R}$.

Prvá a tretia inklúzia sú zjavné z definície príslušných gramatík, druhá vyplýva z lemy o úprave bezkontextovej gramatiky na ε-free tvar, ktorú dokážeme v časti 3.4. Neskôr tiež ukážeme, že všetky inklúzie sú ostré, t.j. ani v jednom prípade neplatí rovnosť.

1.6 Jazyk ako problém

Celá táto časť bude prudko neformálne rozprávanie o niektorých významoch teórie formálnych jazykov. Stručne vysvetlíme, že na jazyky sa môžeme dívať ako na problémy (a naopak, každému problému bude zodpovedať nejaký jazyk). Rozhodovanie príslušnosti slova do jazyka bude ekvivalentné riešeniu príslušného problému. Problém bude tým ľahší, čím „slabší nástroj“ nám bude stačiť na jeho riešenie.

Definícia 1.6.1 ***Rozhodovací problém*** *môžeme definovať ako jazyk Y (nad pevne zvolenou abecedou) – množinu tých vstupov, pre ktoré je odpoveď kladná.*

Príklad 1.6.1 Ukážeme, že algoritmické úlohy, ktorých riešením je odpoveď áno/nie, vieme formulovať ako rozhodovací problém napríklad nad abecedou $\{0, 1\}$. Stačí jednoducho zobrať ako Y binárne zápisy tých vstupov problému, pre ktoré má byť odpoveď áno.

Napr. pre úlohu „Zistite, či daný reťazec u nad abecedou $\{a, b, c\}$ obsahuje podreťazec *abbaca*.“ vieme každý reťazec u ľahko zakódovať do postupnosti núl a jednotiek. Potom Y bude jazyk kódov tých reťazcov u, ktoré naozaj obsahujú *abbaca*. V nasledujúcej kapitole nahliadneme, že tento problém je regulárny, a teda ľahko riešiteľný.

Poznámka 1.6.1 Programátor ľahko nahliadne, že pri riešení rozhodovacieho problému ide o nájdenie algoritmu, ktorý pre každý „vstupný súbor“ (slovo nad abecedou $\{0, 1\}$) rozhodne, či je dobrý alebo nie.

[3] Už samotný jazyk je množina slov, preto sa zvolil tento odlišný názov. Formálna definícia triedy jazykov požaduje, aby trieda jazykov obsahovala aspoň jeden neprázdny jazyk – kvôli zlým vlastnostiam tried $\emptyset$ a $\{\emptyset\}$.

Naším cieľom v teórií formálnych jazykov bude zaoberať sa tým, pre ktoré jazyky (=problémy) je ako ťažké rozhodovať príslušnosť doň. Napríklad trieda regulárnych jazykov bude obsahovať tie rozhodovacie problémy, ktoré sme schopní vyriešiť v lineárnom čase s konečnou pomocnou pamäťou. Jedným z cieľov, ku ktorým časom dospejeme, bude ukázať, že trieda frázových jazykov obsahuje práve všetky algoritmicky (aspoň čiastočne) riešiteľné problémy a že existujú problémy, ktoré algoritmicky riešiť nevieme, nech by sme mali k dispozícii ľubovoľne veľa času a nekonečne veľkú pamäť.

Na to ale najskôr potrebujeme vybudovať dostatočne silný aparát a formálne definovať všetky pojmy z predchádzajúceho odseku. V nasledujúcich kapitolách definujeme niekoľko typov automatov, najsilnejší z nich bude už spomenutý Turingov stroj ako formálny model počítača. Budeme skúmať silu a vlastnosti nami definovaných modelov.

Kapitola 2

Regulárne jazyky

2.1 Deterministický konečný automat

Najskôr si zavedieme najjednoduchší druh automatu pomocou štyroch nasledujúcich definícií. Tento istý postup budeme v budúcnosti používať pri definícii zložitejších typov automatov. Deterministický konečný automat bude jednoduché zariadenie, ktoré si môžeme predstaviť nasledovne: Skladá sa z čítacej hlavy a vstupnej pásky. Čítacia hlava sa môže nachádzať v jednom z konečne veľa stavov. Idúc po vstupnej páske číta vstupné slovo a podľa prečítaných symbolov mení svoj stav. Jej stav po dočítaní slova rozhoduje, či bolo slovo dobré alebo zlé.

Definícia 2.1.1 ***Deterministický konečný automat*** *A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je konečná vstupná abeceda, $q_0 \in K$ je začiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times \Sigma \to K$ je prechodová funkcia.*

Definícia 2.1.2 ***Konfigurácia*** *deterministického konečného automatu je prvok $(q, w) \in K \times \Sigma^*$, kde q je stav automatu a w je nespracovaná časť vstupného slova.*

Poznámka 2.1.1 Konfigurácia nám vlastne udáva jednoznačný popis situácie, v ktorej sa automat A v danom okamihu výpočtu nachádza. Na základe jeho konfigurácie budeme vedieť povedať, ako bude výpočet pokračovať ďalej. (Analógia s reálnym počítačom: konfigurácia automatu v istom zmysle zodpovedá obsahu celej pamäte počítača – na jeho základe vieme povedať, čo sa môže diať ďalej.)

Definícia 2.1.3 ***Krok výpočtu*** *deterministického konečného automatu A je relácia $\vdash_A$ na konfiguráciách definovaná $(q, av) \vdash_A (p, v) \iff p = \delta(q, a)$.*

Poznámka 2.1.2 Krok výpočtu teda hovorí, v akej konfigurácii sa bude A nachádzať v nasledujúcom kroku výpočtu, ak vieme, v akej sa nachádza teraz. Výpočet automatu teda môžeme formálne zapísať ako postupnosť konfigurácií, pričom každé dve po sebe nasledujúce konfigurácie sú v relácií „krok výpočtu“.

Výpočet automatu si môžeme predstaviť tak, že automat postupne číta písmená vstupného slova a v stave si prenáša konečnú informáciu.

Definícia 2.1.4 ***Jazyk*** *akceptovaný deterministickým konečným automatom A je množina $L(A) = \{w \mid \exists q_F \in F;\ (q_0, w) \vdash_A^* (q_F, \varepsilon)\}$.*

Poznámka 2.1.3 Podobne ako pri gramatikách $\Rightarrow^*$, aj tu $\vdash^*$ je reflexívno-tranzitívny uzáver relácie $\vdash$. Značenie $Q_1 \vdash_A^* Q_2$ teda znamená, že automat A prejde z konfigurácie Q_1 do konfigurácie Q_2 na ľubovoľný konečný počet krokov.

Poznámka 2.1.4 Deterministický konečný automat označujeme skrátene DKA, prípadne DFA (z anglického deterministic finite automaton).

Poznámka 2.1.5 Konečný automat (resp. jeho δ-funkciu) zvykneme zapisovať aj prechodovou tabuľkou alebo prechodovým diagramom (viď príklad 2.1.1). Prechodová tabuľka by mala byť intuitívne jasná, zastavme sa pri prechodovom diagrame. Jednotlivé stavy automatu sa kreslia kružnicou, do ktorej sa píše názov stavu. Ak sa jedná o koncový stav, kreslíme ho dvojitou kružnicou. Na začiatočný stav v diagrame ukazuje šípka. Ak v automate platí $\delta(q, a) = p$, potom v diagrame nakreslíme šípku vedúcu z vrcholu q k vrcholu p a označíme ju symbolom a. (Každý DKA teda vieme reprezentovať vhodným ohodnoteným orientovaným grafom.)

Poznámka 2.1.6 Uvedomte si, že δ-funkcia predstavuje akýsi jednoduchý program konečného automatu. Preto hlavnou časťou konštrukcie konečného automatu pre daný jazyk bude práve konštrukcia vhodnej prechodovej funkcie.

Príklad 2.1.1 $A = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_0\})$, $\delta(q_i, a) = q_{1-i}$, $\delta(q_i, b) = q_i$ (pre $i \in \{0, 1\}$).

Tento automat zisťuje, či má vstupné slovo párny počet písmen a, čiže akceptuje jazyk $L = \{w \mid w \in \{a, b\}^* \ \wedge \ \#_a(w) \text{ je párny}\}$. Pre vstupné slovo $abaaba$ bude vyzerať výpočet takto:

$$(q_0, abaaba) \vdash (q_1, baaba) \vdash (q_1, aaba) \vdash (q_0, aba) \vdash (q_1, ba) \vdash (q_1, a) \vdash (q_0, \varepsilon)$$

Dokážeme teraz formálne, že $L(A) = L$. Matematickou indukciou od i dokážeme tvrdenie: Nech $(q_0, uv) \vdash^i (q_x, v)$. Potom $x = \#_a(u) \bmod 2$.

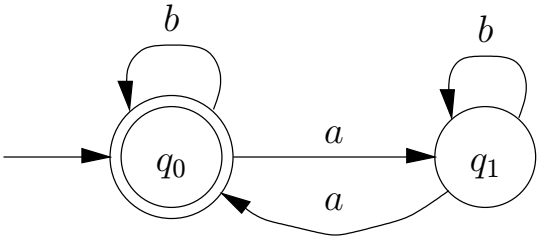
1° Pre $i = 0$ zjavne platí, lebo nutne $u = \varepsilon$.

2° Nech $(q_0, ucv) \vdash^i (q_x, cv) \vdash (q_y, v)$. Z indukčného predpokladu $x = \#_a(u) \bmod 2$. Ak $c = a$, tak z δ-funkcie vyplýva, že $y = 1 - x$. Ale aj $(\#_a(uc) \bmod 2) = 1 - (\#_a(u) \bmod 2)$ a tvrdenie platí. Analogicky pre $c = b$.

Z práve dokázaného tvrdenia vyplýva, že výpočet na slove w skončí v stave $q_{\#_a(w) \bmod 2}$, a teda A akceptuje práve slová z L, q.e.d.

$A:$	a	b
q_0	q_1	q_0
q_1	q_0	q_1

Prechodová tabuľka pre tento DKA.



Prechodový diagram pre tento DKA.

Cvičenie 2.1.2 Bolo by vhodné, aby čitateľ skôr, než bude čítať ďalej, získal predstavu o sile konečných automatov. Odporúčame skúsiť si zostrojiť DKA pre nasledujúce jazyky (nad abecedou $\{a, b\}$):

- prázdny jazyk
- jazyk $\{\varepsilon, a, aba\}$
- jazyk $\{ba^i \mid i \geq 2\}$
- jazyk $\{a^x \mid \exists k \geq 0;\ x = 6k + 2\}$
- jazyk slov obsahujúcich aspoň 4 symboly a
- jazyk slov obsahujúcich podslovo aaa
- jazyk slov obsahujúcich podslovo $ababb$
- jazyk slov obsahujúcich podslovo $ababb$ a párny počet symbolov a
- jazyk slov, ktorých tretie písmeno je a a predposledné je b
- jazyk slov, ktorých tretie písmeno je a alebo predposledné je b
- jazyk $\{a^i b^i \mid i \geq 0\}$

2.2 Nedeterministický konečný automat

Definícia nedeterministického konečného automatu pozostáva z rovnakých štyroch častí ako pri DKA, pričom sa mení len definícia automatu a kroku výpočtu. Táto zmena bude podstatná – umožníme automatu rozhodovať sa. Teda na jednom slove bude možných viacero rôznych výpočtov. Bude nás zaujímať, či existuje postupnosť „správnych rozhodnutí“, ktorá dovedie výpočet do úspešného konca.

Definícia 2.2.1 ***Nedeterministický konečný automat*** *A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je konečná vstupná abeceda, $q_0 \in K$ je začiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times (\Sigma \cup \{\varepsilon\}) \to 2^K$ je prechodová funkcia.*

Poznámka 2.2.1 Detail k značeniu: 2^X je množina všetkých podmnožín množiny X. (Niekedy sa tiež zvykne označovať $\mathcal{P}(X)$.)

Definícia 2.2.2 ***Krok výpočtu*** *nedeterministického konečného automatu A je relácia $\vdash_A$ na konfiguráciách definovaná $(q, an) \vdash_A (p, n) \iff p \in \delta(q, a)$.*

Poznámka 2.2.2 Nedeterministický konečný automat sa označuje skrátene NKA, prípadne NFA (z anglického non-deterministic finite automaton).

Poznámka 2.2.3 Hlavný rozdiel medzi deterministickým a nedeterministickým automatom je, že u nedeterministického automatu je $\delta(q, a)$ množina stavov, ktorá môže obsahovať viac stavov, prípadne byť aj prázdna. 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 výpočty na danom slove môžu byť akceptačné a niektoré nie. Slovo akceptujeme, ak aspoň jeden výpočet na ňom je akceptačný.

NKA má navyše ešte aj možnosť robiť kroky (a rozhodovať sa) bez toho, aby musel čítať písmeno zo vstupu. Takýmto krokom budeme hovoriť „kroky na ε“.

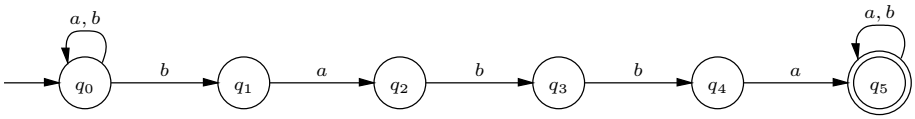
Poznámka 2.2.4 Rovnako ako DKA, aj NKA vieme znázorniť vhodným orientovaným grafom. Jediný rozdiel bude v tom, že z každého vrcholu (=stavu) môže vychádzať ľubovoľne veľa hrán, označených tým istým písmenom.

Príklad 2.2.1 Úžasnú silu, ktorú nám ponúka nedeterminizmus, si ukážeme na jednoduchom príklade. Skúste si najskôr zostrojiť DKA pre jazyk $\{w \mid w$ obsahuje podslovo $ababbababa\}$. Konštrukcia je nepríjemne komplikovaná, v princípe si musíme v stave pamätať dĺžku najdlhšieho sufixu prečítanej časti slova, ktorý je prefixom hľadaného podslova.

NKA pre jazyk $\{w \mid w$ obsahuje podslovo $u\}$ bude omnoho jednoduchší. Nech $|u| = k$, potom NKA $A = (\{q_0, \ldots, q_k\}, \Sigma, \delta, q_0, \{q_k\})$, kde:

$$\begin{aligned} \delta(q_0, x) &\ni q_0 \quad (\forall x \in \Sigma) \\ \delta(q_i, u_{i+1}) &\ni q_{i+1} \quad (\forall i \in \{0, \ldots, k-1\}) \\ \delta(q_k, x) &\ni q_k \quad (\forall x \in \Sigma) \end{aligned}$$

Ak vstupné slovo hľadané podslovo neobsahuje, akceptačný výpočet neexistuje. Ak vstupné slovo hľadané podslovo obsahuje, akceptačný výpočet existuje a nedeterminizmus ho akoby za nás nájde a vstupné slovo akceptuje.



NKA pre jazyk $\{w \mid w$ obsahuje podslovo $babba\}$.

Poznámka 2.2.5 Predchádzajúci príklad nám ponúka nasledovný pohľad na nedeterminizmus: Zatiaľ čo v prípade deterministického automatu sme museli pomocou konečnej pamäte zistiť, či vstupné slovo patrí do daného jazyka, v prípade nedeterminizmu máme k dispozícii vševediacu vílu Amálku, ktorá nás chce presvedčiť, že vstupné slovo do jazyka patrí, ukáže nám dôkaz (=poradí správne voľby, ktoré vedú k akceptačnému výpočtu, ak existuje) a my ho musíme v konečnej pamäti overiť, aby nám nepodstrčila aj zlé slová.

Na výpočet nedeterministického automatu sa teda môžeme dívať ako na akúsi hru, ktorú hrá víla Amálka na našom NKA. Dáme jej vstupné slovo a ona nám nájde (ak existuje) výpočet, ktorý toto slovo akceptuje. Možných výpočtov môže byť nekonečne veľa, to ale vševediacej víle Amálke problémy nerobí.

Poznámka 2.2.6 Iný, pri prvom stretnutí s nedeterminizmom trochu prirodzenejší pohľad je „Alibaba a nekonečno zbojníkov". Alibaba pošle svojich nekonečno zbojníkov cez vstupnú bránu (začiatočnú konfiguráciu) do jaskyne (priestoru konfigurácií) hľadať poklad. Zbojníci idú po chodbách (krokoch výpočtu) a vždy, keď narazia na križovatku (viac možností na nasledovný krok výpočtu), rozdelia sa na príslušný počet nekonečne veľkých skupín a nerušene pokračujú ďalej. Akonáhle niektorá skupina nájde poklad (akceptačnú konfiguráciu), zakričí to ostatným, poklad vynesú von a vyhrali. Ak v jaskyni poklad je, v konečnom čase ho nájdu. Ak tam poklad nie je, buď zistia, že prešli celú jaskyňu a nič nenašli, alebo budú chudáci hľadať na veky vekov.

Na nedeterministický automat sa teda môžeme dívať aj tak, že naraz skúša všetky možné výpočty a akceptuje, akonáhle akceptuje niektorý z nich.

2.3 Ekvivalencia DKA a NKA

Zjavne ku každému DKA existuje ekvivalentný NKA. Opačným smerom to už nie je až také zjavné – ako už vieme, nedeterministický automat má možnosť robiť kroky bez čítania písmena zo vstupu a vie akoby naraz skúšať viac možností. V tejto časti ukážeme, že napriek tomu ho vieme simulovať obyčajným deterministickým automatom (aj keď za cenu nárastu počtu stavov).

Cvičenie 2.3.1 Zostrojte DKA s čo najmenej stavmi pre jazyk $L = \{a^x \mid x \text{ je deliteľné 7 alebo } 13\}$. Zostrojte NKA s čo najmenej stavmi[1] pre jazyk L.

Veta 2.3.1 *K ľubovolnému nedeterministickému konečnému automatu A existuje nedeterministický konečný automat A' bez ε-prechodov (čiže taký, že $\forall q \in K_{A'};\ \delta_{A'}(q,\varepsilon) = \emptyset$), pre ktorý $L(A') = L(A)$.*

Dôkaz. Zamyslime sa najskôr, ako vyzerá akceptačný výpočet A na $w = w_1 \dots w_n$. Môžeme si ho znázorniť nasledovne:

$$\xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{w_1} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} \xrightarrow{w_2} \xrightarrow{\varepsilon} \quad \cdots \quad \xrightarrow{\varepsilon} \xrightarrow{w_n} \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon}$$

Náš automat A' bude mať na tomto slove k dispozícii len n krokov. Preto ho musíme zostrojiť tak, aby v jednom kroku simuloval niekoľko krokov A – okrem kroku na práve prečítané písmeno aj niekoľko okolitých krokov na ε. To ale nebude nič zložité.

Označme $[q]_\varepsilon = \{p \mid (q,\varepsilon) \vdash_A^* (p,\varepsilon)\}$. Teda $[q]_\varepsilon$ (tzv. epsilonový chvost stavu q) je množina tých stavov, do ktorých sa z q vieme v A dostať bez čítania písmena. Všimnite si, že aj $q \in [q]_\varepsilon$.

Nech sa A nachádza v stave q. Všimnime si kus výpočtu: „A chodí na ε, spraví krok na x a ďalej chodí na ε". Prvú časť skončí v nejakom stave $r \in [q]_\varepsilon$, druhú v nejakom $s \in \delta(r,x)$, tretiu v nejakom $t \in [s]_\varepsilon$. Ak tento kus výpočtu chceme odsimulovať v A' jedným krokom, musí mať možnosť prejsť zo stavu q na písmeno x do stavu t.

Formálne zapíšme vyššie uvedenú konštrukciu. Bude $A' = (K, \Sigma, \delta', q_0, F)$, kde

$$t \in \delta'(q,x) \quad \Longleftrightarrow \quad \exists r,s \in K;\ r \in [q]_\varepsilon \ \wedge\ s \in \delta(r,x) \ \wedge\ t \in [s]_\varepsilon$$

[1] Dá sa ukázať, že DKA ich potrebuje 91. Pre NKA nám ich stačí 21. Ako?

Zjavne každému kroku výpočtu A' zodpovedá časť nejakého výpočtu A, a teda každému akceptačnému výpočtu A' zodpovedá akceptačný výpočet A na danom slove. A naopak, akceptačný výpočet A ľahko rozdelíme na časti, v ktorých číta po jednom písmene a každej z nich vieme priradiť krok v A'. Preto naozaj $L(A) = L(A')$ a zjavne A' nepoužíva prechody na ε. Alebo nie?

Nuž, jeden detail nám zatiaľ unikol – prázdne slovo ε. Zatiaľ čo A mohol na slove ε spraviť niekoľko krokov (a prípadne prísť do akceptačného stavu), nový automat A' nemôže spraviť ani jeden krok. Preto sa môže stať, že $\varepsilon \in L(A)$, ale $\varepsilon \notin L(A')$.

Toto ale ľahko ošetríme. Jednou možnosťou je vhodná úprava množiny akceptačných stavov. (Tú si ukážeme v poznámke 2.3.1.) Teraz ukážeme inú možnú konštrukciu – zavedieme špeciálny začiatočný stav, ktorý bude akceptačný práve ak $\varepsilon \in L(A)$. Bude teda $A'' = (K \cup \{q_0''\}, \Sigma, \delta'', q_0'', F'')$, kde q_0'' je nový stav,

$$F'' = F \cup \{q_0'' \mid \varepsilon \in L(A)\} = F \cup \{q_0'' \mid F \cap [q_0]_\varepsilon \neq \emptyset\}$$

$$\forall q \in K;\ \forall x \in \Sigma;\ \delta''(q,x) = \delta'(q,x)$$

$$\forall x \in \Sigma;\ \delta''(q_0'',x) = \delta'(q_0,x)$$

Poznámka 2.3.1 Predchádzajúci dôkaz je mierne náročný na algoritmické zostrojenie A' z daného A. Môžeme si trochu uľahčiť prácu vhodnou voľbou toho, ktoré kroky na ε budeme kedy simulovať. Asi najrýchlejšie je zostrojiteľný A', v ktorom každý krok simuluje niekoľko krokov na ε, jeden krok na písmeno a dosť. Potom ale nám chýba simulácia krokov na ε po dočítaní slova. Tú ale nahradíme tým, že všetky stavy, z ktorých sa v A dalo na ε dostať do akceptačného, budú v A' akceptačné.

Formálne bude $A' = (K, \Sigma, \delta', q_0, F')$, kde:

$$\delta'(q,x) = \bigcup_{p \in [q]_\varepsilon} \delta(p,x) \qquad\qquad F' = \{q \mid F \cap [q]_\varepsilon \neq \emptyset\}$$

Veta 2.3.2 *K ľubovolnému nedeterministickému konečnému automatu A existuje deterministický konečný automat A' taký, že $L(A) = L(A')$.*

Dôkaz. BUNV[2] predpokladajme, že A neobsahuje ε-prechody. Zostrojme $A' = (K', \Sigma', \delta', q_0', F')$ takto: $K' = 2^K$, $\Sigma' = \Sigma$, $q_0' = \{q_0\}$, $F' = \{X \mid X \cap F \neq \emptyset\}$ a $\delta'(Q,a) = \bigcup_{q \in Q} \delta(q,a)$. Myšlienka tejto konštrukcie spočíva v tom, že A' si v svojom stave pamätá všetky stavy, v ktorých sa A mohol nachádzať po spracovaní príslušnej časti slova. Slovo akceptujeme, ak sa po jeho dočítaní A mohol nachádzať v akceptačnom stave.

Potrebujeme teraz formálne dokázať, že $L(A) = L(A')$. Matematickou indukciou vzhľadom na dĺžku w ľahko dokážeme nasledovné tvrdenie: Nech $(\{q_0\}, w) \vdash_{A'}^* (Q, \varepsilon)$. Potom $\forall q \in K;\ (q_0, w) \vdash_A^* (q, \varepsilon) \iff q \in Q$. (Slovom: Na slove w vie automat A skončiť práve v stavoch z Q.)

Z toho už je jasné, že ak $w \in L(A')$, tak $(\{q_0\}, w) \vdash_{A'}^* (Q, \varepsilon)$, kde $Q \in F'$. To ale znamená, že $\exists q_F \in F \cap Q$. Potom podľa vyššie uvedenej lemy $(q_0, w) \vdash_A^* (q_F, \varepsilon)$, a teda $w \in A$.

Analogicky, nech $w \in L_A$, zoberme akceptačný výpočet $(q_0, w) \vdash_A^* (q_F, \varepsilon)$. Automat A' skončí svoj výpočet v nejakom stave Q. Podľa vyššie uvedenej lemy $q_F \in Q$, preto $Q \in F'$ a teda $w \in L(A')$.

Poznámka 2.3.2 Deterministický konečný automat sa niekedy definuje tak, že je to NKA bez prechodov na ε, v ktorom $\forall q \in K,\ \forall x \in \Sigma;\ |\delta(q,x)| \leq 1$.

Dá sa ľahko ukázať, že táto definícia je ekvivalentná s našou. Líšia sa v tom, že takto definovaný DKA nemusí mať z niektorého stavu na niektoré písmeno definovaný prechod, a teda sa môže počas výpočtu zaseknúť. Ľahko ho upravíme na DKA spĺňajúci našu definíciu – stačí pridať nový „odpadový“ stav a dodefinovať všetky chýbajúce prechody tak, aby viedli doň.

Výhoda novej definície DKA je v ľahšej konštrukcii ku konkrétnemu jazyku – nemusíme sa zaoberať slovami, o ktorých sme už počas výpočtu zistili, že do nášho jazyka nepatria.

[2] bez ujmy na všeobecnosti

2.4 Normálne tvary konečných automatov a regulárnych gramatík

Ku každému jazyku existuje celá trieda gramatík, ktoré ho generujú, resp. automatov, ktoré ho akceptujú. Toto je výhodné, ak máme k danému jazyku nejakú gramatiku, resp. automat zostrojiť – máme na výber z veľa možností, stačí nám nájsť ľubovoľnú jednu z nich. (Neskôr ukážeme, že občas si môžme ešte pomôcť tým, že definujeme nový typ automatu či gramatiky, ktorý bude ekvivalentný s pôvodným a poskytne nám väčšiu voľnosť pri konštrukcii.)

Akonáhle máme o nejakom objekte (gramatike či automate) niečo dokázať, prípadne ho nejako upraviť, hodí sa nám, aby bol čo najjednoduchší. A toto dokážeme zabezpečiť pomocou rôznych viet o normálnom tvare. Uvedieme jeden príklad:

Príklad 2.4.1 Ku každému NKA $A = (K, \Sigma, \delta, q_0, F)$ existuje ekvivalentný[3] NKA, ktorý neobsahuje prechody na ε a pre ktorý platí $\forall q \in K,\ \forall x \in \Sigma;\ |\delta(q, x)| = 1$.

Spoznávate? Túto vetu sme v podstate dokázali v prechádzajúcej časti. Hovorí toľko, že ku každému NKA existuje ekvivalentný DKA. Na DKA sa teda môžeme dívať ako na normálny tvar nedet. konečného automatu. A skutočne platí to, čo sme pred chvíľou o normálnych tvaroch uviedli: Ak máme nejaký konečný automat zostrojiť, ľahšie zostrojíme NKA (porovnajte si napr. NKA a DKA pre jazyk $\{uababbv \mid u, v \in \{a, b\}^*\}$). Na druhej strane, pri dôkazoch radšej použijeme DKA, v ktorom existuje na každom slove práve jeden výpočet.

Príklad 2.4.2 Ukážeme si ešte jeden príklad normálneho tvaru (tentokrát regulárnej gramatiky) aj s poriadnym dôkazom.

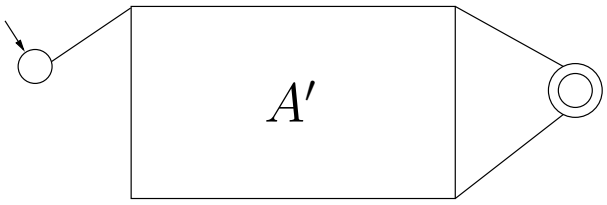
Najjednoduchšia úprava gramatiky spočíva v odstránení pravidiel tvaru $\xi \to \xi$. Tieto pravidlá určite neposunú výpočet nijako dopredu, ba ani iným smerom. Po použití takéhoto pravidla dostaneme v odvodení tú istú vetnú formu ako sme mali pred jeho použitím. Môžeme teda tvrdiť nasledovné:

Lema 2.4.1 *Nech G je regulárna gramatika a nech $G_0 = (N, T, P_0, \sigma)$ je gramatika, pre ktorú platí $P_0 = P \setminus \{\xi \to \xi \mid \xi \in N\}$. Potom $L(G_0) = L(G)$.*

Dôkaz. Dokážeme dve inklúzie rovnosti $L(G_0) = L(G)$.

- $\subseteq$: Z predpokladu $w \in L(G_0)$ vyplýva, že $\exists$ odvodenie slova w v G_0: $\sigma \underset{G_0}{\Longrightarrow} u_1 \underset{G_0}{\Longrightarrow} u_2 \underset{G_0}{\Longrightarrow} \ldots \underset{G_0}{\Longrightarrow} u_n = w$. Toto odvodenie je zároveň aj odvodením slova w v G, lebo $P_0 \subseteq P$.
- $\supseteq$: Z predpokladu $w \in L(G)$ vyplýva, že $\exists$ odvodenie slova w v G. Potrebujeme nájsť odvodenie slova w v G_0. Ukážeme si trik, ktorý sa v dôkazoch dá často využiť. Keďže existuje (aspoň jedno) odvodenie w v G, tak zjavne[4] existuje (aspoň jedno) najkratšie odvodenie w v G. V tomto odvodení sa nemohlo použiť žiadne pravidlo tvaru $\xi \to \xi$, inak by sme vedeli nájsť kratšie odvodenie. Preto je toto odvodenie aj odvodením v G_0, q.e.d.

Lema 2.4.2 *(„Prasiatkový“ normálny tvar NKA.) Ku každému NKA A existuje NKA A' taký, že $L(A) = L(A')$, A' má práve jeden akceptačný stav, počas výpočtu sa nikdy nevráti do začiatočného stavu a zastane okamžite po dosiahnutí akceptačného stavu.*



Zdôvodnenie názvu *prasiatkový normálny tvar.*

[3] T.j. akceptujúci rovnaký jazyk

[4] Lebo dĺžka odvodenia je prirodzené číslo a každá podmnožina prirodzených čísel má minimálny prvok.

Dôkaz. Nech $A = (K, \Sigma, \delta, q_0, F)$. Bez ujmy na všeobecnosti nech A neobsahuje prechody na ε (teda je v normálnom tvare z príkladu 2.4.1).

Potom zostrojíme A' nasledovne: Nech q_0', q_F' sú nové stavy. Bude $A' = (K \cup \{q_0', q_F'\}, \Sigma, \delta', q_0', \{q_F'\})$, kde δ' definujeme nasledovne:

$$\begin{aligned} \delta'(q, a) &= \delta(q, a) \quad (\forall q \in K,\ \forall a \in \Sigma) \\ \delta'(q_0', \varepsilon) &= \{q_0\} \\ \delta'(q_a, \varepsilon) &= \{q_F'\} \quad (\forall q_a \in F) \end{aligned}$$

Do automatu A sme pridali nový začiatočný stav q_0'. Keďže ten nie je nikde na pravej strane δ', zjavne sa A' počas výpočtu do začiatočného stavu nevráti. Podobne z akceptačného stavu nie sú definované žiadne prechody, preto ak ho A' dosiahne, už nemôže ďalej počítať.

Okrem toho sme pridali nový akceptačný stav a zo všetkých bývalých akceptačných stavov ε-hrany doň. Ľahko ukážeme, že takto upravený automat akceptuje rovnaký jazyk ako A. (Z akc. výpočtu A na w ľahko zostrojíme akc. výpočet A' na w. Naopak, v akc. výpočte A' na w vieme ako vyzeral prvý a posledný krok, potom ale zvyšok tohto výpočtu zodpovedá akc. výpočtu A na w)

Lema 2.4.3 *K ľubovoľnej regulárnej gramatike G existuje regulárna gramatika G' taká, že $P' \subseteq_{kon} N \times (T \cup \{\varepsilon\})(N \cup \{\varepsilon\})$ a $L(G) = L(G')$. (Čiže každú regulárnu gramatiku vieme upraviť tak, aby v každom kroku generovala najviac jeden terminál.)*

Dôkaz. Každé pravidlo, ktoré generuje viac ako jeden terminál rozbijeme na viac pravidiel, ktoré generujú po jednom terminále. Pridaním nových neterminálov zabezpečíme, aby sme nové pravidlá museli použiť v správnom poradí a dosiahnuť rovnakú vetnú formu ako v pôvodnej gramatike.

Formálna konštrukcia: Bude $G' = (N', T, P', \sigma)$. Označme k najväčší počet terminálov na pravej strane pravidla gramatiky G. Pre každé pravidlo budeme pridávať niekoľko nových neterminálov. BUNV nech $\xi_{p,i}$ (kde $p \in P$, $i \in \{1, \ldots, k-1\}$) sú nové neterminály, t.j. zatiaľ nepatria do N.[5]

Položme $N' = N \cup \{\xi_{p,i} \mid p \in P \ \wedge \ i \in \{1, \ldots, k-1\}\}$. Niektoré z nových neterminálov v skutočnosti nepoužijeme, ale je pohodlnejšie N' definovať takto ako ošetrovať rôzne okrajové prípady. Novú množinu pravidiel zostrojíme nasledovne:

Nech $p \in P$ je pravidlo. Ak $p \in (T \cup \{\varepsilon\})(N \cup \{\varepsilon\})$, tak $p \in P'$. Ak $p = (\eta \to a_1 \ldots a_n \varphi)$, tak do P' dáme pravidlá $\eta \to a_1 \xi_{p,1}$, $\xi_{p,1} \to a_2 \xi_{p,2}$, $\ldots$, $\xi_{p,n-1} \to a_n \varphi$. Analogicky nahradíme pravidlá tvaru $\eta \to a_1 \ldots a_n$.

Formálny dôkaz oboch inklúzií rovnosti $L(G) = L(G')$ je triviálny, prenechávame ho čitateľovi.

Lema 2.4.4 *K ľubovoľnej regulárnej gramatike G existuje regulárna gramatika G' taká, že $P' \subseteq_{kon} N \times (T \cup T(N \setminus \{\sigma\})) \ \cup \ \{\sigma \to \varepsilon\}$.*

Dôkaz. Uvedieme len náznak dôkazu, keďže využíva viacero konštrukcii, ktoré na tomto mieste nechceme rozoberať.

Jedna možnosť je upraviť gramatiku do normálneho tvaru z lemy 2.4.3 a postupne odstrániť pravidlá $\xi \to \eta$ (odstránenie chain rules, viď časť 3.6) a $\xi \to \varepsilon$ (odepsilonovanie, viď časť 3.4). Iná možnosť je pomocou konštrukcií z nasledujúcej časti previesť gramatiku na NKA, ten na DKA a ten späť na novú regulárnu gramatiku, ktorú následne potrebujeme len odepsilonovať.

2.5 Ekvivalencia konečných automatov a regulárnych gramatík

Už sme si všimli, že každá vetná forma odvoditeľná v regulárnej gramatike je buď terminálne slovo, alebo obsahuje práve jeden neterminál, a to na konci. Regulárna gramatika teda slovo generuje „zľava doprava“, pričom jediný spôsob, ako si môže prenášať informácie je zmenou neterminálu.

[5] A ani do $T \ldots$

Takmer okamžite vidíme paralelu s konečnými automatmi – zatiaľ čo konečný automat slovo „zľava doprava“ číta a pamätá si konečnú informáciu v stave, gramatika ho „zľava doprava“ generuje a pamätanú konečnú informáciu predstavuje neterminál na konci vetnej formy. V ďalšom texte ukážeme, že táto paralela naozaj platí a že konečné automaty akceptujú práve regulárne jazyky.

Veta 2.5.1 *K ľubovoľnému deterministickému konečnému automatu A existuje regulárna gramatika G taká, že $L(G) = L(A)$.*

Dôkaz. Gramatiku zostrojíme tak, aby generovanie slova simulovalo výpočet A. Chceme dosiahnuť, aby neterminál na konci už vygenerovanej časti slova zodpovedal stavu, v ktorom sa A bude nachádzať po prečítaní tejto časti slova. Odvodenie sa môžeme rozhodnúť ukončiť, ak aktuálny neterminál zodpovedá akceptačnému stavu.

Formálna konštrukcia: Nech $A = (K, \Sigma, \delta, q_0, F)$, potom $G = (K, \Sigma, P, q_0)$, kde $P = \{\xi \to a\eta \mid \delta(\xi, a) = \eta\} \cup \{\xi \to \varepsilon \mid \xi \in F\}$.

Matematickou indukciou od dĺžky výpočtu A ľahko dokážeme lemu, že A na slove w skončí v stave q vtedy a len vtedy, ak v G vieme odvodiť slovo wq. Formálne: $\forall w;\ (q_0, w) \vdash_A^* (q, \varepsilon) \iff q_0 \underset{G}{\Longrightarrow}^* wq$.

Z lemy už rovnosť $L(A) = L(G)$ priamo vyplýva. (Ak výpočet na w je akceptačný, končí v nejakom akc. stave q_F, potom ale v G vieme podľa lemy odvodiť wq_F a následne w. Ak vieme odvodiť w, všimnime si posledný krok. Pred ním sme museli mať vetnú formu wq_F, kde q_F je nejaký akc. stav A. Potom ale podľa lemy $(q_0, w) \vdash_A^* (q_F, \varepsilon)$, a teda A akceptuje w.)

Poznámka 2.5.1 Skôr než vyslovíme a dokážeme vetu o konverzii gramatiky na automat, uvedomme si, v čom je (jediný) problém. Gramatika vie v jednom kroku vygenerovať veľa znakov, zatiaľ čo automat ich musí čítať po jednom. Tento problém vyriešime tak, že na konverziu použijeme regulárnu gramatiku vo vhodnom normálnom tvare.

Veta 2.5.2 *K ľubovoľnej regulárnej gramatike G existuje nedeterministický konečný automat A taký, že $L(A) = L(G)$.*

Dôkaz. Vďaka leme 2.4.3 môžeme BUNV predpokladať, že G má pravidlá v uvedenom tvare (t.j. v každom kroku odvodenia vygeneruje najviac jeden terminál). Zostrojme ekvivalentný NKA A takto: $A = (K = N \cup \{q_F\}, T, \delta, \sigma, F = \{q_F\})$, pričom

$$\forall \xi \in N,\ \forall x \in T \cup \{\varepsilon\};\quad \delta(\xi, x) = \{\eta \mid (\xi \to x\eta) \in P\})\} \cup \{q_F \mid (\xi \to x) \in P\}$$

Dôkaz správnosti tejto konštrukcie je takmer izomorfný s dôkazom vety 2.5.1 o konštrukcii opačným smerom, preto ho nebudeme uvádzať.

2.6 Uzáverové vlastnosti triedy regulárnych jazykov

Definícia 2.6.1 *Hovoríme, že trieda jazykov $\mathscr{L}$ je **uzavretá** na binárnu operáciu $\circ$, ak pre všetky jazyky $L_1, L_2 \in \mathscr{L}$ platí, že $L_1 \circ L_2 \in \mathscr{L}$.*

Poznámka 2.6.1 Definícia sa ľahko rozšíri aj na n-árne operácie, $n \geq 1$ (teda vrátane unárnych).

Veta 2.6.1 *$\mathcal{R}$ je uzavretá na zjednotenie.*

Dôkaz. Myšlienka dôkazu: Z reg. gramatík pre pôvodné jazyky zostrojíme reg. gramatiku pre ich zjednotenie. Tá bude fungovať tak, že v prvom kroku odvodenia sa rozhodne, či vygeneruje slovo z L_1 alebo z L_2.

Majme $L_1, L_2 \in \mathcal{R}$. K nim existujú regulárne gramatiky G_1, G_2 také, že $L_i = L(G_i)$. Nech $G_1 = (N_1, T_1, P_1, \sigma_1)$, $G_2 = (N_2, T_2, P_1, \sigma_1)$. BUNV môžeme predpokladať, že množiny N_1, N_2 a $T_1 \cup T_2$ sú po dvoch disjunktné (neterminály vieme vhodne preznačiť).

Zostrojme regulárnu gramatiku G nasledovne: Nech σ je nový symbol, potom $G = (N_1 \cup N_2 \cup \{\sigma\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{\sigma \to \sigma_1,\ \sigma \to \sigma_2\}, \sigma)$.

Dokážeme, že $L(G) = L_1 \cup L_2$.

Nech $w \in L_1 \cup L_2$. Ak $w \in L_1$, existuje v G_1 odvodenie $\sigma_1 \underset{G_1}{\Longrightarrow}^* w$. Jemu ale zodpovedá odvodenie $\sigma \underset{G}{\Longrightarrow} \sigma_1 \underset{G}{\Longrightarrow}^* w$ v gramatike G, preto $w \in L(G)$. (Prípad $w \in L_2$ analogicky.)

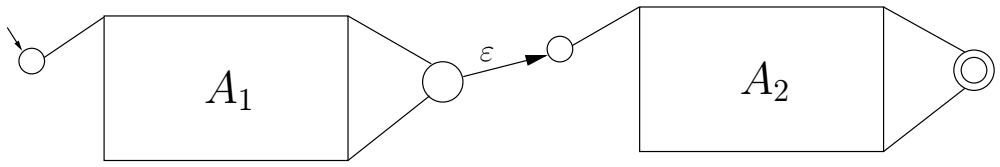
Nech teraz $w \in L(G)$. V prvom kroku odvodenia sme museli použiť jedno z pravidiel $\sigma \to \sigma_1$, $\sigma \to \sigma_2$. Opäť rozoberieme len prvý prípad. Indukciou od dĺžky odvodenia ľahko ukážeme, že všetky vetné formy odvoditeľné v G zo σ_1 sú tvaru $T_1^*(N_1 \cup \{\varepsilon\}$, a že pri každom ďalšom kroku odvodenia sme museli použiť pravidlo z P_1.[6] Potom ale odvodenie $\sigma_1 \underset{G}{\Longrightarrow}^* w$ v G je aj odvodením v G_1. To ale znamená, že $w \in L(G_1)$, a teda $w \in L_1 \cup L_2$, q.e.d.

Veta 2.6.2 *$\mathcal{R}$ je uzavretá na zreťazenie.*

Dôkaz. Tentokrát podáme dôkaz pomocou automatov. Nech teda L_1, L_2 sú regulárne jazyky. Podľa lemy 2.4.2 k nim existujú NKA v „prasiatkovom normálnom tvare“. Označme tieto automaty $A_i = (K_i, \Sigma, \delta_i, q_{0i}, \{q_{Fi}\})$. Nech navyše BUNV majú disjunktné množiny stavov, t.j. $K_1 \cap K_2 = \emptyset$.

Zostrojme potom automat $A = (K_1 \cup K_2, \Sigma, \delta, q_{01}, \{q_{F2}\})$, kde δ obsahuje obe δ_i a navyše definujeme $\delta(q_{F1}, \varepsilon) = \{q_{02}\}$.

Zjavne naozaj $L(A) = L(A_1)L(A_2)$. Pri formálnom dôkaze jednej inklúzie by sme ukázali, ako z akc. výpočtov A_1 na slove w_1 a A_2 na slove w_2 zostrojiť akc. výpočet A na w_1w_2 (stačí pridať jeden krok výpočtu). Druhá inklúzia analogicky: platí vďaka tomu, že ak A akceptuje w, tak počas výpočtu zjavne práve raz prejde z q_{F1} do q_{02}. Ale potom A_1 akceptuje už prečítanú a A_2 ešte neprečítanú časť w.



Konštrukcia NKA pre zreťazenie dvoch regulárnych jazykov.

Veta 2.6.3 *$\mathcal{R}$ je uzavretá na iteráciu.*

Dôkaz. Dôkaz vyzerá analogicky ako vo vete 2.6.2 – zoberieme automat pre L v prasiatkovom normálnom tvare a pridáme mu ε-hranu z akceptačného stavu do počiatočného.

Poznámka 2.6.2 Keďže $L^+ = L \cdot L^*$, z viet 2.6.2 a 2.6.3 vyplýva uzavretosť $\mathcal{R}$ aj na kladnú iteráciu.

Veta 2.6.4 *$\mathcal{R}$ je uzavretá na komplement.*

Dôkaz. Stačí stručne uviesť myšlienku: Ak $A = (K, \Sigma, \delta, q_0, F)$ je DKA akceptujúci jazyk L, tak $A' = (K, \Sigma, \delta, q_0, K \setminus F)$ je DKA akceptujúci jazyk $L^C = \Sigma^* \setminus L$. Totiž ak v A výpočet na w nebol akceptačný, v A' bude a naopak.

Poznámka 2.6.3 Uvedomte si, že ak by A bol nedeterministický, táto konštrukcia by nemusela fungovať.

Veta 2.6.5 *$\mathcal{R}$ je uzavretá na prienik.*

Dôkaz. Tvrdenie vyplýva napr. z de Morganovych zákonov. Totiž $L_1 \cap L_2 = (L_1^C \cup L_2^C)^C$ a už vieme, že $\mathcal{R}$ je uzavretá na zjednotenie a komplement.

Ukážeme ale ešte jeden dôkaz, tzv. konštrukciu kartézskym súčinom, kvôli tomu, že túto myšlienku ešte v budúcnosti použijeme. Konečný automat vie naraz simulovať viacero iných konečných

[6] Toto platí vďaka predpokladu, že $N_1 \cap N_2 = \emptyset$.

automatov – jednoducho tak, že si v svojom stave pamätá stavy všetkých simulovaných automatov. (Uvedomte si, že možných stavov nášho automatu je len konečne veľa.)

Formálna konštrukcia: Nech $L_1, L_2 \in \mathcal{R}$, nech $A_i = (K_i, \Sigma, \delta_i, q_{0i}, F_i)$ sú DKA také, že $L(A_i) = L_i$. Potom zostrojme automat $A = (K, \Sigma, \delta, q_0, F)$, kde $K = K_1 \times K_2$, $q_0 = [q_{01}, q_{02}]$, $F = F_1 \times F_2$ a $\delta([q_x, q_y], a) = [\delta_1(q_x, a), \delta_2(q_y, a)]$.

Indukciou ľahko dokážeme tvrdenie, že A je po prečítaní w v stave $[q_x, q_y]$ iff[7] A_1 je po prečítaní w v stave q_x a A_2 v q_y. Odtiaľ už je zjavné, že $L(A) = L(A_1) \cap L(A_2)$.

Poznámka 2.6.4 Analogickú konštrukciu sme mohli použiť aj pri dôkaze uzavretosti na zjednotenie. Tam však bolo pohodlnejšie spoľahnúť sa na nedeterminizmus, ktorý si správne tipol, do ktorého jazyka dané slovo patrí. Pri prieniku nám nedeterminizmus nepomôže – tak či tak potrebujeme overiť, či slovo patrí do oboch jazykov.

Veta 2.6.6 *$\mathcal{R}$ je uzavretá na homomorfizmus.*

Dôkaz. Myšlienka dôkazu: Pravidlá reg. gramatiky pre daný jazyk upravíme tak, že každý terminál nahradíme jeho homomorfným obrazom. Výsledná gramatika bude generovať homomorfný obraz pôvodného jazyka.

Majme $L \in \mathcal{R}$, $h : T^* \to S^*$ nech je ľub. homomorfizmus. Vieme, že existuje regulárna gramatika $G = (N, T, P, \sigma)$ taká, že $L(G) = L$. Definujme nový homomorfizmus h' nasledovne: $\forall \xi \in N;\ h'(\xi) = \xi$ a $\forall x \in T; h'(x) = h(x)$. (Homom. h' teda funguje rovnako ako h, len navyše vie zobraziť aj neterminály našej gramatiky a necháva ich na pokoji.)

Zostrojme reg. gramatiku G' nasledovne: $G' = (N, S, h'(P), \sigma)$. (Množina $h'(P)$ sú všetky pravidlá z P zobrazené homomorfizmom h', teda $h'(P) = \{\xi \to h'(w) \mid \xi \to w \in P\}$.)

Tvrdenie $L(G') = h(L)$ sa ľahko dokáže, dôkaz nebudeme uvádzať.

Veta 2.6.7 *$\mathcal{R}$ je uzavretá na inverzný homomorfizmus.*

Dôkaz. Majme $L \in \mathcal{R}$, nech $A = (K, \Sigma_1, \delta, q_0, F)$ je DKA akceptujúci L. Nech $h : \Sigma_2^* \to \Sigma_1^*$ je ľub. homomorfizmus. Ukážeme, ako zostrojiť NKA A' pre $h^{-1}(L)$.

A' má o každom slove w, ktoré dostane na vstupe, zistiť, či $h(w) \in L$. Nech $k = \max_{x \in \Sigma_2} |h(x)|$. Každé písmeno w nám teda h zobrazí na najviac k písmen. Náš automat si bude pamätať v stave slovo konečnej veľkosti $\leq k$. Vždy, keď prečíta písmeno zo vstupu, naplní si túto pamäť obrazom prečítaného písmena. Na písmenách tohto obrazu simuluje A (bez toho, aby čítal zo vstupu). Keď sa mu pamäť vyprázdni, prečíta ďalšie písmeno.

Formálne bude $A' = (K', \Sigma_2, \delta', [q_0, \varepsilon], F')$, kde:

$$\begin{aligned}
K' &= \{[q, w] \mid q \in K \ \wedge\ w \in \Sigma_1^* \ \wedge\ |w| \leq k\} \\
F' &= \{[q_F, \varepsilon] \mid q_F \in F\} \\
\delta'([q, aw], \varepsilon) &= \{[\delta(q, a), w]\} \ \ (\forall q \in K,\ \forall a \in \Sigma_1,\ \forall w \in \Sigma_1^*) \\
\delta'([q, \varepsilon], x) &= \{[q, h(x)]\} \ \ (\forall q \in K,\ \forall x \in \Sigma_2)
\end{aligned}$$

Zjavne každému výpočtu A' na w zodpovedá výpočet A na $h(w)$ a naopak, preto naozaj $L(A') = h^{-1}(L)$.

Veta 2.6.8 *$\mathcal{R}$ je uzavretá na reverz.*

Dôkaz. Myšlienka: Stačí zobrať NKA v prasiatkovom normálnom tvare, vymeniť akceptačný stav so začiatočným a obrátiť smer šípok. Výpočet v takto upravenom NKA bude zodpovedať výpočtu pôvodného NKA na reverze vstupného slova.

Formálne: Nech $L \in \mathcal{R}$, $L = L(A)$, kde $A = (K, \Sigma, \delta, q_0, \{q_F\})$ je v prasiatkovom normálnom tvare. Potom zostrojme $A' = (K, \Sigma, \delta', q_F, \{q_0\})$, kde $p \in \delta'(q, x) \iff q \in \delta(p, x)$. Potom $L(A') = L(A)^R$.

[7] iff = if and only if = vtedy a len vtedy

Totiž ľahko overíme, že v A' existuje výpočet

$$(q_F, w_1 \dots w_n) \vdash^* (q_{i_1}, w_2 \dots w_n) \vdash^* (q_{i_{n-1}}, w_n) \vdash^* (q_0, \varepsilon)$$

práve vtedy, keď v A existuje výpočet

$$(q_0, w_n \dots w_1) \vdash^* (q_{i_{n-1}}, w_{n-1} \dots w_1) \vdash^* (q_{i_1}, w_1) \vdash^* (q_F, \varepsilon)$$

Inými slovami, pre ľubovoľné $w = w_1 \dots w_n$ platí: $w \in L(A') \iff w^R \in L(A)$.

Poznámka 2.6.5 Pri dôkaze každej z uzáverových vlastností sme použili jeden z modelov (automat alebo gramatiku), ktorý sa nám zdal vhodnejší. Rozmyslite si, ako by tieto dôkazy vyzerali, ak by sme pri nich použili opačné modely.

2.7 Kleeneho veta

Lema 2.7.1 *Každý konečný jazyk je regulárny.*

Dôkaz. Nech $L = \{w_1, \dots, w_n\}$. Potom $L = L(G)$, kde $G = (\{\sigma\}, \Sigma_L, P, \sigma)$, pričom $P = \{\sigma \to w_1,\ \sigma \to w_2,\ \dots,\ \sigma \to w_n\}$.

Zhrňme si, čo už vieme povedať o triede $\mathcal{R}$. Musí obsahovať všetky konečné jazyky. V časti 2.6 sme dokázali, že musí byť uzavretá na všetky základné operácie. Teraz ukážeme, že $\mathcal{R}$ je najmenšia trieda jazykov s týmito vlastnosťami.

Veta 2.7.2 *(Kleene) $\mathcal{R}$ je najmenšia trieda jazykov, ktorá obsahuje všetky konečné jazyky a je uzavretá na zjednotenie, zreťazenie a iteráciu.*

Poznámka 2.7.1 Pri dôkaze použijeme techniku, známu v oblasti efektívnych algoritmov pod názvom *dynamické programovanie*. Presne rovnakú myšlienku ako náš dôkaz má napríklad algoritmus od Floyda a Warshalla na spočítanie dĺžok najkratších ciest v ohodnotenom grafe.

Dôkaz. Vieme, že $\mathcal{R}$ obsahuje všetky konečné jazyky a je uzavretá na vyššie uvedené operácie. Potrebujeme dokázať, že každý regulárny jazyk vieme vyrobiť z konečných jazykov pomocou konečného počtu spomínaných operácií. (Z toho bude vyplývať, že každá trieda, ktorá spĺňa podmienky zo znenia vety obsahuje všetky regulárne jazyky, a teda je nadmnožinou $\mathcal{R}$.)

Nech teda $R \in \mathcal{R}$ je ľub. regulárny jazyk, A nech je DKA taký, že $L(A) = R$. BUNV nech $K = \{q_1, \dots, q_n\}$. Budeme sa snažiť zostrojiť množinu všetkých slov, ktoré A akceptuje. Keby sme pre každé i, j vedeli zostrojiť množinu $L_{i,j}$ slov, na ktoré sa A dostane zo stavu q_i do stavu q_j, vyhrali sme – ľahko zostrojíme $L = \bigcup_{j;q_j \in F} L_{1,j}$.

Zostrojiť priamo množiny $L_{i,j}$ nie je také ľahké, preto si pomôžeme drobným trikom: Nech $R^k_{i,j}$ je množina tých slov, na ktoré A prejde z q_i do q_j a počas výpočtu nejde cez stav s číslom väčším ako k. Formálne:

$$R^k_{i,j} = \{a_1 \dots a_n \mid (q_i, a_1 \dots a_n) \vdash_A (q_{m_1}, a_2 \dots a_n) \vdash^*_A (q_{m_{n-1}}, a_n) \vdash_A (q_j, \varepsilon) \ \wedge\ \forall x; m_x \leq k\}$$

Zjavne $R^n_{i,j} = L_{i,j}$ (lebo podmienka $\forall x; m_x \leq n$ je splnená pre ľubovoľné stavy). Budeme postupne zostrojovať množiny $R^k_{i,j}$ s rastúcim k.

Pre $k = 0$ je situácia jednoduchá. Výpočet nemôže prechádzať cez žiadne stavy, preto môže mať najviac jeden krok. Teda pre $i \neq j$ je $R^0_{i,j} = \{x \mid \delta(q_i, x) = q_j\}$, v $R^0_{i,i}$ je navyše aj ε. Všetky množiny $R^0_{i,j}$ sú zjavne konečné.

Nech už vieme zostrojiť množiny $R^{k-1}_{i,j}$ (pre všetky i, j). Ukážeme, ako zostrojiť množinu $R^k_{i,j}$. Zaujímajú nás tie slová, na ktoré A prejde z q_i do q_j a ide cez stavy s číslom najviac k. Rozoberieme dve možnosti, ako tento výpočet vyzerá:

Cez stav q_k neprechádza. Potom ale prechádza cez stavy s číslom najviac $k-1$. Takýto výpočet teda mohol prebehnúť práve na slovách z $R_{i,j}^{k-1}$.

Cez stav q_k prechádza. Rozdeľme výpočet (a slovo) v miestach, kedy A bol v stave q_k. V prvej časti výpočtu sme prešli z q_i do q_k, v poslednej z q_k do q_j, medzitým sme niekoľkokrát prešli z q_k do q_k. Dôležité je uvedomiť si, že v každej časti už výpočet prechádza len cez stavy s číslom menším ako k. Pre každú časť výpočtu teda už vieme zostrojiť množinu slov, na ktoré mohol prebehnúť. Z nich teraz poskladáme množinu slov, na ktoré mohol prebehnúť celý uvažovaný výpočet.

Dostávame teda, že platí:

$$R_{i,j}^{k} \;=\; R_{i,j}^{k-1} \;\cup\; R_{i,k}^{k-1} \left(R_{k,k}^{k-1}\right)^{*} R_{k,j}^{k-1}$$

Pomocou vyššie uvedeného rekurentného vzťahu vieme ku každému DKA A zostrojiť $L(A)$ z konečných jazykov len pomocou zreťazenia, iterácie a zjednotenia. Preto každá trieda jazykov, ktorá obsahuje všetky konečné jazyky a je uzavretá na zreťazenie, iteráciu a zjednotenie, obsahuje všetky regulárne jazyky, q.e.d.

2.8 Pumpovacia lema pre regulárne jazyky

Príklad 2.8.1 Jazyk $L = \{a^n b^n \mid n \geq 0\}$ nie je regulárny. Totiž keby bol, musel by existovať DKA A, ktorý ho akceptuje. Všimnime si stavy, v ktorých je A po prečítaní slova a^i (pre $i \in \{0, \ldots, |K|\}$). Z Dirichletovho princípu musia existovať j, k $(j \neq k)$ také, že A je po prečítaní a^j a a^k v rovnakom stave. Ale keďže A akceptuje slovo $a^j b^j$, musí akceptovať aj slovo $a^k b^j$, čo je spor.

Nasledujúca veta nám dá možnosť ukázať o niektorých jazykoch, že nie sú regulárne. Budeme sa snažiť vystihnúť podstatu príkladu 2.8.1 tak, aby sa tento postup dal použiť aj na iné nieregulárne jazyky.

Veta 2.8.1 *(pumpovacia lema) Ku každému regulárnemu jazyku L existuje číslo p také, že pre každé slovo $w \in L$ také, že $|w| > p$ existujú u, v, x také, že platí:*

1. $w = uvx$
2. $|uv| \leq p$
3. $|v| \geq 1$
4. $\forall i \geq 0;\ uv^i x \in L$

Poznámka 2.8.1 Podľa vlastnosti 4 sa táto veta nazýva pumpovacou, lebo táto vlastnosť pripomína akési napumpovávanie strednej časti slova.

Dôkaz. Nech L je regulárny jazyk a A je DKA, ktorý ho akceptuje. Dokážme, že $p = |K_A|$ vyhovuje tvrdeniu vety. Nech $a_1 a_2 \ldots a_l = w \in L$, $l > p$ a nech

$$(q_0, a_1 a_2 \ldots a_l) \vdash (q_1, a_2 \ldots a_l) \vdash^* (q_p, a_{p+1} \ldots a_l) \vdash^* (q_l, \varepsilon)$$

je akceptačný výpočet A na w. Podľa Dirichletovho princípu[8] musia existovať čísla i, j $(0 \leq i < j \leq p)$ také že $q_i = q_j$. Nech $u = a_1 \ldots a_i$, $v = a_{i+1} \ldots a_j$, $x = a_{j+1} \ldots a_l$.

Poľahky nahliadneme, že slová ux, uv^2x, uv^3x, $\ldots$ náš automat akceptuje, a teda patria do L. (Totiž na slovo v prejde A zo stavu q_i do q_i. Túto časť výpočtu teda môžeme ľubovoľne veľa krát zopakovať.)

Príklad 2.8.2 Dokážeme, že jazyk $L = \{w \mid \#_a(w) = \#_b(w)\}$ nie je regulárny. Sporom. Nech je regulárny. Potom podľa pumpovacej lemy existuje p také, že každé slovo z L dlhšie ako p sa dá nejako rozdeliť a napumpovať. Aby sme dospeli ku sporu, nájdeme slovo z L, ktoré je dlhšie ako p, ale nech ho rozdelíme, ako chceme, napumpovať sa nedá.

[8] Ak rozmiestnime $n+1$ holubov do n holubníkov, existuje holubník s aspoň 2 holubmi. V našom prípade keď zoberieme postupnosť prvých $p+1$ stavov počas výpočtu A na w, existuje stav, ktorý v nej je aspoň dvakrát.

Takéto slovo je napríklad $w = a^p b^p$. Nech ho rozdelíme, ako chceme, vďaka podmienke 2 budú slová u, v obsahovať iba písmená a. Bude teda $u = a^i$, $v = a^j$, $x = a^{p-i-j}b^p$ pre vhodné $i \geq 0$, $j > 0$. Potom ale slovo $uv^2x = a^{p+j}b^p$ nepatrí do L, čo je hľadaný spor.

Poznámka 2.8.2 Jediné miesto, kde treba pri dôkaze tvrdenia „L nie je regulárny“ rozmýšľať, je voľba slova w. Musíme nájsť také slovo, ktoré sa skutočne rozdeliť a napumpovať nedá. Navyše voľbou vhodného slova si častokrát môžeme ušetriť prácu: Máme vlastne ukázať, že žiadnym prípustným spôsobom rozdelené slovo sa nedá napumpovať. Preto sa oplatí vybrať slovo, ktoré sa dá prípustne rozdeliť málo spôsobmi.

Poznámka 2.8.3 Uvedomte si, že podmienka 4 hovorí: $\forall i \geq 0;\ uv^i x \in L$. Presnejšie: aj slovo uv^0x musí patriť do L. Toto sa nám niekedy môže hodiť, ako ukážeme v nasledujúcom príklade.

Príklad 2.8.3 Dokážeme, že jazyk $L = \{a^i b^j \mid i \geq j\}$ nie je regulárny. Sporom. Nech je regulárny. Potom podľa pumpovacej lemy existuje p také, že každé slovo z L dlhšie ako p sa dá nejako rozdeliť a napumpovať. Aby sme dospeli ku sporu, opäť nájdeme slovo z L, ktoré je dlhšie ako p, ale nech ho rozdelíme, ako chceme, napumpovať sa nedá.

Takéto slovo je opäť napríklad $w = a^p b^p$. Nech ľubovoľne rozdelíme w, budú síce pre $i \geq 1$ slová $uv^i x$ patriť do L, ale slovo $uv^0x = ux$ do L patriť nebude, čo je hľadaný spor.

Poznámka 2.8.4 Nezabudnite, že pumpovacia lema je len implikácia. To, že L spĺňa podmienky z pumpovacej lemy, ešte neznamená, že je regulárny. Inými slovami, pomocou pumpovacej lemy sa dá iba ukázať, že L regulárny **nie je**.

2.9 Myhill-Nerodova veta

V tejto časti dokážeme vetu, ktorá presne charakterizuje všetky regulárne jazyky a navyše hovorí o minimálnom DKA akceptujúcom daný regulárny jazyk. Najskôr ale definujeme potrebné pojmy.

Definícia 2.9.1 *Binárna relácia $\heartsuit$ na Σ^* sa nazýva **sprava invariantná** (vzhľadom na operáciu zreťazenie), ak platí:*

$$x \heartsuit y \iff \forall v;\ xv \heartsuit yv$$

Definícia 2.9.2 *Počet tried relácie ekvivalencie nazývame **index**. Ak je tento počet konečný, hovoríme, že relácia je **konečného indexu**.*

Príklad 2.9.1 Zmysel tejto definície je nasledujúci: Namiesto „slová x, y sú v relácií $\heartsuit$“ hovorme: „slová x, y sú (vzhľadom na $\heartsuit$) nerozlíšiteľné“. Potom skutočnosť, že relácia $\heartsuit$ je sprava invariantná znamená toľko, že ak zoberieme dve nerozlíšiteľné slová x, y, tak nech za ne doplníme čokoľvek, vždy dostaneme opäť dve nerozlíšiteľné slová.

Teda napríklad ak $ab \heartsuit baa$ (a $\heartsuit$ je sprava invariantná), tak nutne aj $abca \heartsuit baaca$.

V ďalšom texte tejto časti definujeme k danému DKA A reláciu ekvivalencie – dve slová budeme považovať za nerozlíšiteľné, ak A po ich prečítaní skončí v tom istom stave. Zjavne táto relácia bude sprava invariantná – ak A skončí v rovnakom stave na x a y, tak zjavne skončí v rovnakom stave aj na slovách xv a yv (pre ľubovoľné v). Túto jej vlastnosť neskôr využijeme.

Veta 2.9.1 *(Myhill-Nerode) Majme jazyk $L \subseteq \Sigma^*$. Nasledujúce tvrdenia sú ekvivalentné:*

1. *L je regulárny jazyk*
2. *L je zjednotením niekoľkých tried ekvivalencie nejakej sprava invariantnej relácie ekvivalencie konečného indexu*
3. *Relácia R_L definovaná $uR_Lv \iff (\forall x;\ ux \in L \iff vx \in L)$ je reláciou ekvivalencie konečného indexu.*

Dôkaz. Ukážeme niekoľko implikácií, z ktorých bude vyplývať platnosť dokazovaného tvrdenia.

$1 \Rightarrow 2$: Nech L je regulárny jazyk. Zoberme DKA $A = (K, \Sigma, \delta, q_0, F)$, ktorý ho akceptuje. Hľadanú reláciu ekvivalencie definujeme tak, že dve slová sú ekvivalentné práve vtedy, ak na nich A skončí v tom istom stave. Zjavne ide o reláciu ekvivalencie. Každá trieda ekvivalencie zodpovedá jednému stavu automatu, preto naša relácia ekvivalencie je konečného indexu. Je sprava invariantná, lebo keď na slove u aj v automat A skončí v tom istom stave q, tak pre ľubovoľné ux a vx skončí v tom istom stave (v tom, do ktorého sa z q dostane na slovo x). No a L je zjednotením tých tried ekvivalencie, ktoré zodpovedajú akceptačným stavom automatu A.

Ešte raz formálne. Príkladom hľadanej relácie ekvivalencie je relácia $\heartsuit$ definovaná nasledovne:

$$u \heartsuit v \iff \exists p \in K;\ (q_0, u) \vdash_A^* (p, \varepsilon) \ \wedge\ (q_0, v) \vdash_A^* (p, \varepsilon)$$

$2 \Rightarrow 3$: Prv, než pristúpime k dôkazu je potrebné si uvedomiť, že táto implikácia hovorí len o reláciách ekvivalencie, a teda by sa mala dať dokázať bez toho, aby sme čosi vedeli o konečných automatoch. A teraz už k samotnému dôkazu.

Nech teda L je zjednotením niekoľkých tried ekvivalencie nejakej relácie ekvivalencie $\heartsuit$, ktorá spĺňa podmienky zo znenia vety. Nie je ťažké nahliadnuť, že R_L je vždy reláciou ekvivalencie (t.j. je reflexívna, symetrická a tranzitívna). Pôjde teda o to, či bude konečného indexu, teda či má konečný počet tried ekvivalencie.

Ukážeme, že každá trieda ekvivalencie R_L je zjednotením niekoľkých tried ekvivalencie relácie $\heartsuit$. Z toho už priamo vyplýva, že aj R_L má konečne veľa tried ekvivalencie (najviac toľko ako $\heartsuit$).

Nech u, v sú ľubovoľné slová také, že $u \heartsuit v$. Rozmyslite si, že stačí ukázať, že $u R_L v$. (Potom totiž ľubovoľná trieda ekvivalencie R_L obsahuje spolu s ľubovoľným slovom w aj všetky slová, ktoré sú s ním v relácii $\heartsuit$, teda celú príslušnú triedu ekvivalencie relácie $\heartsuit$.)

Keďže $\heartsuit$ je sprava invariantná, pre všetky x platí $ux \heartsuit vx$. Jazyk L je zjednotením niekoľkých tried ekvivalencie relácie $\heartsuit$, teda pre ľubovoľné x buď obe slová ux, vx do L patria, alebo doň nepatrí ani jedno z nich. To ale znamená, že $u R_L v$, q.e.d.

$3 \Rightarrow 1$: Na základe existencie R_L treba zostrojiť konečný automat A pre jazyk L. Zamyslime sa najskôr nad reláciou R_L. Nie je ťažké nahliadnuť, že R_L je vždy reláciou ekvivalencie (t.j. je reflexívna, symetrická a tranzitívna). Čo nám ale táto relácia hovorí?

Všimnime si dve slová u, v, ktoré sú v tejto relácií. Zoberme definíciu R_L a dosaďme $x = \varepsilon$. Dostávame, že $u \in L \iff v \in L$. Teda v L buď sú obe slová, alebo ani jedno. Potom ale L je zjednotením niekoľkých tried ekvivalencie R_L. (Majme ľubovoľnú triedu ekvivalencie. Ak niektoré slovo z nej patrí do L, musia tam podľa vyššie uvedeného tvrdenia patriť všetky.)

Teraz si všimnime dve slová u, v, ktoré v tejto relácií nie sú. To ale znamená, že existuje x také, že BUNV $ux \in L$ a $vx \notin L$. Majme ľubovoľný DKA A akceptujúci jazyk L. Potom po prečítaní slov u a v musí skončiť v rôznych stavoch. Totiž ak by skončil v rovnakom stave, potom by v rovnakom stave skončil aj pre slová ux a vx – ale jedno z nich má akceptovať a druhé nie.

Relácia R_L nám teda hovorí: Každý DKA pre jazyk L musí mať aspoň toľko stavov, koľko má R_L tried ekvivalencie. Triviálne z toho vyplýva, že ak R_L nie je konečného indexu, L nie je regulárny.[9] Ukážeme, že ak R_L je konečného indexu, tak existuje DKA pre L, ktorý má rovnako stavov, ako tried ekvivalencie.

Na základe znalosti R_L teda zostrojíme *minimálny* DKA pre L nasledovne:[10] Automat si bude v stave pamätať, do ktorej triedy ekvivalencie R_L doteraz prečítané slovo patrí. Ak po prečítaní celého slova sme v triede ekvivalencie, ktorá je podmnožinou L, akceptujeme.

[9] Ak má R_L nekonečne veľa tried, musel by mať aj ľubovoľný DKA pre L nekonečne veľa stavov – každý DKA však má stavov len konečne veľa.

[10] Aby sme boli presní: Zostrojíme ho na základe znalosti relácie R_L a toho, zjednotením ktorých jej tried ekvivalencie je L.

Formálne: Nech $[u] = \{v \mid uR_Lv\}$ je trieda ekvivalencie obsahujúca u.
Bude $A = (K, \Sigma, \delta, [\varepsilon], F)$, kde:
$K = \{[u] \mid u \in \Sigma^*\}$ je konečná množina tried ekvivalencie R_L
$F = \{[u] \mid u \in L\}$ je konečná množina tých tried ekvivalencie, ktoré sú podmnožinou L
$\delta([u], c) = [uc]$

Musíme ukázať o našej δ-funkcii, že je to skutočne funkcia, teda že jednému prvku je priradený práve jeden prvok. Ukážme, že našu δ-funkciu sme zadefinovali korektne, teda že funkčná hodnota nezávisí od voľby reprezentanta u triedy ekvivalencie $[u]$. Nech teda $[u] = [v]$, teda uR_Lv. Potom $\forall x \in \Sigma^*;\ ux \in L \iff vx \in L$. Preto aj $\forall c \in \Sigma,\ \forall x \in \Sigma^*;\ ucx \in L \iff vcx \in L$. Preto aj slová uc a vc sú v relácií R_L, a teda $[uc] = [vc]$, q.e.d.

Tiež zjavne platí, že $([\varepsilon], w) \vdash^* ([w], \varepsilon)$ a $[w] \in F \iff w \in L$, preto naozaj $L(A) = L$.

Príklad 2.9.2 Jazyk $L = \{w \mid w \in \{a, b\}^* \ \wedge\ \#_a(w) = \#_b(w)\}$ nie je regulárny. Tvrdenie dokážeme Myhill-Nerodovou vetou. Ukážeme, že R_L nie je konečného indexu. Ku slovám $u = a^i$ a $v = a^j$ (pre $i \neq j$) totiž existuje slovo $x = b^i$, ktoré ich *odlišuje* – je $ux \in L$, ale $vx \notin L$, a teda u a v nie sú v relácií R_L. Našli sme teda nekonečne veľa slov, z ktorých žiadne dve nie sú ekvivalentné, a teda R_L naozaj nie je konečného indexu.

(Inými slovami, našli sme nekonečnú množinu slov takých, že každý DKA pre L nesmie na žiadnych dvoch z nich skončiť v tom istom stave.)

Príklad 2.9.3 Trieda R je uzavretá na komplement.

Nech L je regulárny jazyk. Potom je zjednotením niekoľkých tried ekvivalencie vhodnej relácie ekvivalencie. Potom ale L^C je zjednotením zvyšných jej tried, a teda je regulárny.

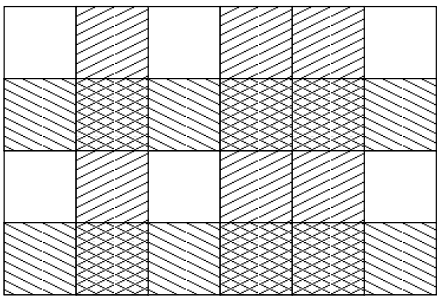
Príklad 2.9.4 Trieda R je uzavretá na zjednotenie a prienik.

Nech L_1, L_2 sú regulárne jazyky, nech R_1, R_2 sú zodpovedajúce relácie ekvivalencie z druhého bodu M-N vety. Definujme novú reláciu ekvivalencie $\heartsuit$, pričom $u \heartsuit v$ iff uR_1v aj uR_2v. Zjavne nová relácia ekvivalencie je tiež sprava invariantná a konečného indexu. Totiž ak R_i má t_i tried ekvivalencie, $\heartsuit$ ich má zjavne najviac $t_1 \times t_2$ – každá trieda ekvivalencie je prienikom jednej triedy ekvivalencie R_1 a jednej triedy R_2 (a niektoré tieto prieniky môžu byť prázdne).

A zjavne ako $L_1 \cup L_2$, tak aj $L_1 \cap L_2$ je zjednotením vhodných tried ekvivalencie našej novej relácie.

(Myšlienka za touto konštrukciou je nasledovná: výsledná relácia zodpovedá DKA, ktorý simuluje naraz oba pôvodné DKA. Na dvoch slovách skončíme v tom istom stave iff skončili v tom istom stave oba pôvodné DKA. Vhodne dodefinujeme množinu akceptačných stavov.)

Myšlienku predchádzajúceho príkladu vieme graficky znázorniť:



Obdĺžnik predstavuje Σ^*, riadky sú triedy ekvivalencie relácie R_1, stĺpce sú triedy ekvivalencie R_2. Vyšrafované riadky tvoria L_1, vyšrafované stĺpce tvoria L_2. Políčka sú triedy ekvivalencie novej relácie $\heartsuit$, jazyky $L_1 \cup L_2$ a $L_1 \cap L_2$ dostaneme ako zjednotenie vhodných políčok.

2.10 Regulárne výrazy

Ukázali sme si zatiaľ dve možnosti, ako zadať ľubovoľný regulárny jazyk – konečné automaty a regulárne gramatiky.[11] Teraz si ukážeme tretiu možnosť – regulárne výrazy. Ich výhodou bude jednoduchší zápis v porovnaní s automatmi aj gramatikami.

Definícia 2.10.1 ***Regulárny výraz*** *môžeme rekurzívne definovať nasledovne:*

- $\overline{\emptyset}$ *je regulárny výraz, predstavujúci jazyk* $\emptyset$.
- *Nech* $x \in \Sigma$, *potom* $\overline{x}$ *je regulárny výraz, predstavujúci jazyk* $\{x\}$.
- *Nech* $\overline{R_1}$, $\overline{R_2}$ *sú regulárne výrazy, predstavujúce jazyky* L_1, L_2, *potom* $\overline{(R_1 + R_2)}$ *je regulárny výraz, predstavujúci jazyk* $L_1 \cup L_2$.
- *Nech* $\overline{R_1}$, $\overline{R_2}$ *sú regulárne výrazy, predstavujúce jazyky* L_1, L_2, *potom* $\overline{R_1 R_2}$ *je regulárny výraz, predstavujúci jazyk* $L_1 \cdot L_2$.
- *Nech* $\overline{R}$ *je regulárny výraz, predstavujúci jazyk* L, *potom* $\overline{(R_1)*}$ *je regulárny výraz, predstavujúci jazyk* L_1^*.
- *Nič iné nie je regulárny výraz a žiaden regulárny výraz nepredstavuje žiaden iný jazyk.*

Príklad 2.10.1 Reg. výraz $\overline{((a+b)) * baa((a+b))*}$ predstavuje jazyk $L = \{ubaav \mid u, v \in \{a, b\}^*\}$.

Veta 2.10.1 *Každý regulárny výraz predstavuje regulárny jazyk.*

Dôkaz. Zjavné – jazyky $\emptyset$ a $\{x\}$ (pre $x \in \Sigma$) sú regulárne, regulárne jazyky sú uzavreté na zjednotenie, zreťazenie a iteráciu.

Veta 2.10.2 *Ku každému regulárnemu jazyku existuje regulárny výraz, ktorý ho predstavuje.*

Dôkaz. Ku každému regulárnemu jazyku existuje DKA, ktorý ho akceptuje. Ľahko upravíme postup z dôkazu Kleeneho vety (veta 2.7.2) tak, aby nám postupne zostrojil regulárne výrazy pre všetky jazyky $R_{i,j}^k$.

Poznámka 2.10.1 Sila regulárnych výrazov spočíva v tom, že nimi vieme jednoducho popísať niektoré regulárne jazyky. Mnohé programy (obzvlášť na Unix-like platformách) preto používajú upravenú verziu regulárnych výrazov pri vyhľadávaní či nahrádzaní vzoriek. Za všetky spomeňme *grep*, *sed*, *vim* a *perl*. Popíšeme teraz, ako vyzerajú tieto upravené regulárne výrazy (regexpy). Naša definícia bude jemne zjednodušená, jej cieľom nie je uvádzať všetky technické detaily ale predviesť približný vzhľad regexpov.

Ak slovo w bude patriť do jazyka predstavovaného regexpom R, budeme hovoriť, že R **matchuje** slovo w.

Regexp je reťazec, zložený z niekoľkých **vetiev**, oddelených znakom `|`. **Regexp** matchuje tie slová, ktoré matchuje aspoň jedna **vetva**. (Znak `|` teda plní funkciu logického or, zjednotenia jazykov, resp. + z našej formálnej definície.)

Vetva je reťazec, zložený z niekoľkých **častí**. **Vetva** matchuje práve všetky slová, ktoré sú zreťazením slov matchujúcich jednotlivé časti.

Časť je **atóm**, ktorý môže byť nasledovaný jedným znakom `*`, `+` alebo `?`. Nech **atóm** matchuje slová z L. Potom: **Atóm** nasledovaný znakom `*` matchuje slová z L^*. **Atóm** nasledovaný znakom `+` matchuje slová z L^+. **Atóm** nasledovaný znakom `?` matchuje slová z $L \cup \{\varepsilon\}$.

Atóm je buď `(R)` (kde `R` je opäť **regexp**), alebo znak `x` (ktorý matchuje reťazec x), alebo jeden zo špeciálnych znakov `.`, `^`, `$`. Znak `.` matchuje jeden ľubovoľný znak. Znak `^` matchuje začiatok slova, znak `$` matchuje koniec slova.

Príklad 2.10.2 Regexp `^aab` matchuje všetky slová, začínajúce *aab*.

[11] V oboch prípadoch nám stačí na popísanie ľubovoľného regulárneho jazyka konečná informácia. Myhill-Nerodova veta síce hovorí, že každý regulárny jazyk vieme zadať ako zjednotenie niekoľkých tried ekvivalencie vhodnej relácie ekvivalencie – nehovorí ale o tom, ako túto reláciu konečne popísať!

Príklad 2.10.3 Regexp `baba|aabaa` matchuje slová, ktoré obsahujú podslovo *baba* alebo *aabaa*.

Príklad 2.10.4 Regexp `^((a*b)(a*b))*a*$` matchuje slová nad abecedou $\{a, b\}$, ktoré obsahujú párny počet b.

Príklad 2.10.5 Regexp `baba.*aabaa|aabaa.*baba` matchuje slová, ktoré obsahujú nepretínajúce sa výskyty podslov *baba* a *aabaa*.

2.11 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. Ukážeme, že niektoré z nich sú ekvivalentné s klasickým konečným automatom. To nám dá väčšiu voľnosť pri dokazovaní regulárnosti niektorých jazykov.

2.11.1 Automaty s pružnou hlavou

Regulárna gramatika mohla v jednom kroku vygenerovať viac terminálov. Definujeme teraz automat, ktorý bude schopný v jednom kroku prečítať viac symbolov zo vstupu. Líšiť sa bude len v definícii samotného automatu a kroku výpočtu. Jediná zmena pri definícii kroku výpočtu spočíva v tom, že čítané slovo môže byť z Σ^* (nie len z $\Sigma \cup \{\varepsilon\}$).

Definícia 2.11.1 ***Nedeterministický konečný automat s pružnou hlavou*** *je 5-ica $A = (K, \Sigma, \delta, q_0, F)$, kde všetko je rovnaké ako u NKA, len $\delta : K \times \Sigma^* \to 2^K$ je* ***konečná*** *prechodová funkcia. (Teda množina tých dvojíc (q, w), pre ktoré $\delta(q, w) \neq \emptyset$ je konečná.)*

Lema 2.11.1 *NKA s pružnou hlavou akceptujú práve regulárne jazyky.*

Dôkaz. Zjavne ku každému klasickému NKA existuje ekvivalentný NKA s pružnou hlavou. Naopak, keď máme NKA s pružnou hlavou, ľahko zostrojíme s ním ekvivalentnú regulárnu gramatiku.

Poznámka 2.11.1 Iná možnosť je zostrojiť priamo konečný automat, ktorý by vždy prečítal niekoľko písmen, tie si pamätal v stave (podobne ako automat pre $h^{-1}(L)$, viď vetu 2.6.7) a následne spravil na ε príslušný krok.

2.11.2 Viachlavé automaty

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

Definícia 2.11.2 ***Viachlavý NKA*** *je 6-ica $A = (K, \Sigma, \delta, q_0, F, k)$, kde K, Σ, q_0, F sú ako pri NKA, $k > 0$ je počet hláv a $\delta : K \times (\Sigma \cup \{\varepsilon\})^k \to 2^K$ je prechodová funkcia.*

Definícia 2.11.3 ***Konfiguráciou*** *viachlavého NKA nazveme $(k+1)$-ticu $(q, w_1, \ldots, w_k)$, kde $q \in K$ je aktuálny stav a $w_i \in \Sigma^*$ je časť slova, ktorú ešte neprečítala i-ta hlava.*

Definícia 2.11.4 ***Krokom výpočtu*** *viachlavého NKA A nazveme binárnu reláciu $\vdash_A$, definovanú na množine konfigurácií nasledovne:*

$$(q, a_1 w_1, \ldots, a_k w_k) \vdash_A (p, w_1, \ldots, w_k) \iff p \in \delta(q, a_1, \ldots, a_k)$$

$$(\text{kde } p, q \in K,\ a_i \in (\Sigma \cup \{\varepsilon\}),\ w_i \in \Sigma^*)$$

Definícia 2.11.5 ***Jazyk*** *akceptovaný viachlavým NKA A je množina*

$$L(A) = \{w \mid \exists q_F \in F;\ (q_0, \underbrace{w, \ldots, w}_{k}) \vdash_A^* (q_F, \underbrace{\varepsilon, \ldots, \varepsilon}_{k})\}$$

Poznámka 2.11.2 Iný spôsob definície používaný v literatúre je taký, že na konci vstupného slova pridáme špeciálny znak označujúci koniec slova. Takto modifikovaný automat teda vie spoznať, ktoré jeho hlavy už dočítali vstup. Rozmyslite si, či je takto definovaný k-hlavý automat ekvivalentný s automatom z našej definície.

Poznámka 2.11.3 Analogicky vieme definovať aj deterministické viachlavé konečné automaty. Bude pri nich záležať na tom, či vieme zistiť, že je hlava na konci vstupného slova?

Poznámka 2.11.4 Podobne sa dajú definovať aj dvojsmerné viachlavé automaty, a prípadne vieme definovať viachlavé automaty, ktoré vedia spoznať, že niektoré dve hlavy sa nachádzajú na tom istom mieste vo vstupnom slove. Týmito modelmi sa už zaoberať nebudeme, vystačíme si s jednoduchými jednosmernými viachlavými automatmi, ktoré sme definovali vyššie.

Poďme sa teraz zaoberať otázkou, aká je výpočtová sila viachlavých konečných automatov vo vzťahu k známym triedam jazykov. Predovšetkým si uvedomme, že ich sila nie je menšia, ako sila obyčajných konečných automatov (s jednou hlavou). Naopak, už dvojhlavý automat dokáže akceptovať napr. jazyky $L_1 = \{a^n b^n \mid n > 0\}$ a $L_2 = \{w\#w \mid w \in \{a,b\}^*\}$, ktoré nie sú regulárne, ba dokonca aj jazyk $L_3 = \{a^n b^n c^n \mid n > 0\}$, ktorý nie je ani bezkontextový.

Ľahko ale nahliadneme, že jazyk $L_4 = \{w\#w^R \mid w \in \Sigma^*\}$ nedokážeme akceptovať k-hlavým NKA pre žiadne k.

Na podobnej myšlienke sú založené nasledujúce jazyky:

$$L_m = \{w_1\#w_2\#\ldots\#w_{2m} \mid \forall i;\ (w_i \in \{a,b\}^* \ \wedge\ w_i = w_{2m+1-i}\}$$

Dá sa dokázať, že pre jazyk L_m existuje k-hlavý deterministický konečný automat práve vtedy, ak platí $m \leq k(k-1)/2$.

Našu krátku exkurziu medzi viachlavé konečné automaty teda uzavrieme nasledujúcim zhrnutím: trieda jazykov, rozpoznávaná viachlavými automatmi, je:
– ostrou nadmnožinou regulárnych jazykov
– neporovnateľná s triedou bezkontextových jazykov
– ostrou podmnožinou (rozšírených) kontextových jazykov

Navyše platí, že pre každé k je trieda jazykov, pre ktoré existuje k-hlavý konečný automat, je vlastnou podmnožinou triedy jazykov, pre ktoré existuje $(k+1)$-hlavý konečný automat.

2.11.3 Dvojsmerné automaty

Na rozdiel od viachlavých automatov, dvojsmerné automaty prekvapivo nebudú znamenať nárast výpočtovej sily automatov. Dôkaz tohto tvrdenia nebude patriť k najtriviálnejším. Na druhej strane jeho význam je značný – v prvom rade pri dokazovaní, že niektoré jazyky sú regulárne.

Príklad 2.11.1 Nech $L \in \mathcal{R}$. Rozmyslite si, ako (resp. či vôbec) vieme zostrojiť NKA akceptujúci jazyk $\sqrt{L} = \{w \mid ww \in L\}$. Zostrojiť dvojsmerný automat pre tento jazyk je triviálne.

Základnou myšlienkou dvojsmerných automatov je, že na základe informácie o stave, v ktorom sa nachádza a prečítaného symbolu prejde do iného stavu a čítacia hlava sa môže posunúť doprava, doľava alebo zostať na mieste. Teda môžeme sa v prípade potreby vrátiť späť k písmenám, ktoré sme už čítali. Aby automat vedel povedať, že sa už nachádza na začiatku, resp. na konci slova, budeme na začiatku a konci slova mať zarážky $\not\in$ a \$. Tieto zarážky nebude smieť prekročiť.

Definícia 2.11.6 ***Dvojsmerným (nedeterministickým konečným) automatom*** *(2NKA) nazývame 5-icu $A = (K, \Sigma, \delta, q_0, F)$, kde K, Σ, q_0, F sú rovnaké ako pri DKA, $\not\in, \$ \notin \Sigma$, $\delta : K \times (\Sigma \cup \{\not\in, \$\}) \to 2^{K\times\{-1,0,1\}}$ je prechodová funkcia, pričom platí:*

$$\forall q \in K;\ \delta(q, \not\in) \subseteq K \times \{0, 1\}$$

$$\forall q \in K;\ \delta(q, \$) \subseteq K \times \{-1, 0\}$$

(Teda ak je automat na niektorom konci slova, nesmie ho prekročiť.)

Definícia 2.11.7 ***Konfiguráciou*** *2NKA A nazývame trojicu $(q, ¢w\$, i)$, kde $q \in K$ je aktuálny stav, $w \in \Sigma^*$ je vstupné slovo a $i \in \{0, \ldots, |w| + 1\}$ je pozícia hlavy.*

Poznámka 2.11.5 Uvedomte si, že konfiguráciu musíme definovať ináč ako u klasického konečného automatu – totiž pri klasickom konečnom automate konfigurácia neobsahuje už prečítanú časť slova. (Niekedy hovoríme, že konečný automat číta vstup *deštruktívne*.)

Definícia 2.11.8 *Nech w_i je i-ty znak slova w, pričom definujeme $w_0 = ¢$ a $w_{|w|+1} = \$$. Potom* ***krokom výpočtu*** *2NKA A nazývame binárnu reláciu $\vdash_A$ na množine konfigurácií definovanú nasledovne:*

$$(q, ¢w\$, i) \vdash_A (p, ¢w\$, i + j) \iff (p, j) \in \delta(q, w_i)$$

Definícia 2.11.9 ***Jazyk*** *akceptovaný 2NKA A je množina*
$L(A) = \{w \mid \exists q_F \in F; (q_0, ¢w\$, 1) \vdash_A^* (q_F, ¢w\$, |w| + 1)\}$

Poznámka 2.11.6 Analogicky vieme definovať aj deterministické dvojsmerné automaty (2DKA).

Veta 2.11.2 *Ku každému DKA A existuje ekvivalentný 2NKA A'.*

Dôkaz. Vyzerá identicky, len δ-funkcia navyše vracia vždy 1, teda pohyb doprava.

Lema 2.11.3 *Ku každému 2NKA A existuje ekvivalentný 2NKA A', ktorý v každom kroku výpočtu pohne hlavou a do akceptačného stavu vie prejsť len na znaku $\$$.*

Dôkaz. Každý riadok δ-funkcie, ktorý hovorí 2NKA A, že má zostať na mieste vieme zavedením nového stavu rozdeliť na dva riadky, pri prvom spraví „krok tam“ a pri druhom „krok späť“. Z pôvodných akceptačných stavov pridáme krok do stavu, v ktorom A' prejde na koniec slova a až tam prejde do nového akceptačného stavu.

Príklad 2.11.2 Nech n je konkrétne prirodzené číslo. Skonštruujeme 2DKA pre jazyk $L_n = \{w\#w \mid w \in \{a, b, c\}^n\}$. Navyše tento 2DKA bude spĺňať podmienky o normálnom tvare podľa lemy 2.11.3.

Všimnite si, že keďže n je nejaká dopredu zvolená konštanta, jazyk L_n je konečný, a teda regulárny. Klasický DKA pre tento jazyk by si v momente prechodu cez mriežku potreboval v stave pamätať informáciu o celom slove w. (Premyslite si, že to znamená, že každý DKA pre tento jazyk musí mať aspoň 3^n stavov. Vidíte súvis medzi touto úvahou a Myhill-Nerodovou vetou?) Dvojsmerný automat sa dokáže k prvej časti slova vrátiť, čím sa zbavuje potreby pamätať si celé slovo w naraz. Tomu bude zodpovedať aj rádovo menší počet potrebných stavov.

Náš automat bude overovať rovnosť slov pred mriežkou a za mriežkou po jednom písmene. Tento cyklus zopakuje n-krát. V i-tom opakovaní si pozrie i-te písmeno na ľavej strane a zapamätá si ho v stave. Potom prejde hlavou do pravej časti vstupu a tam skontroluje rovnosť s i-tym znakom za mriežkou. Čo všetko si potrebuje pamätať v stave? Okrem overovaného písmena a počítadla vykonávaného cyklu ešte pozíciu, kde v slove sa nachádza čítacia hlava, aby rozpoznal, že je nad správnym písmenom. Stav $[i, j, x]$ bude znamenať, že overujeme rovnosť i-tych znakov slov pred a za mriežkou, nachádzame sa na j-tej pozícii vstupného slova a aktuálne overované písmeno je x (prípadne špeciálny znak ?, ak sme tento znak ešte neprečítali). Znak ¢ označíme ako nultý. Nech $\Sigma = \{a, b, c, \#\}$.

Formálne, náš automat bude $A_n = (K, \Sigma, \delta, [1, 0, ?], \{q_F\})$, kde:

$$\begin{aligned}
K &= \{[i, j, x] \mid i \in \{1, \ldots, n\}, j \in \{0, \ldots, 2n + 1\}, x \in \{a, b, c, ?\}\} \\
&\quad \cup \{q_{spat,i} \mid i \in \{1, \ldots, n - 1\}\} \cup \{q_F, q_{koniec}, q_{koniec2}\} \\
F &= \{q_F\} \\
\delta([i, j, y], x) &= ([i, j + 1, y], 1) \quad (j \neq i, j \neq n + 1, j \neq n + i + 1, x \in \{a, b, c\}, y \in \{a, b, c, ?\}) \\
\delta([i, n + 1, y], \#) &= ([i, n + 2, y], 1) \quad (y \in \{a, b, c\}) \\
\delta([i, i, ?], x) &= ([i, i + 1, x], 1) \quad (i \in \{1, \ldots, n\}, x \in \{a, b, c\})
\end{aligned}$$

$$
\begin{aligned}
\delta([i, n+i+1, x], x) &= (q_{spat,i}, -1) \quad (i \in \{1, \ldots, n-1\}, x \in \{a, b, c\}) \\
\delta(q_{spat,i}, x) &= (q_{spat,i}, -1) \quad (i \in \{1, \ldots, n-1\}, x \in \Sigma) \\
\delta(q_{spat,i}, ¢) &= ([i+1, 1, ?], 1) \quad (i \in \{1, \ldots, n-1\}) \\
\delta([n, 2n+1, x], x) &= (q_{koniec}, 1) \quad (x \in \{a, b, c\}) \\
\delta(q_{koniec}, \$) &= (q_{koniec2}, -1) \\
\delta(q_{koniec2}, x) &= (q_F, 1) \quad (x \in \{a, b, c\})
\end{aligned}
$$

Slovné vysvetlenie jednotlivých riadkov δ-funkcie: Prvý riadok je všeobecná situácia – ak sme na políčku, ktoré nás práve nezaujíma, ideme doprava. Druhý riadok overí, či je na pozícii $n+1$ mriežka (vždy, keď cez ňu prechádza). Tretí riadok je čítanie písmena, ktoré práve ideme kontrolovať, a jeho zapamätanie v stave; štvrtý riadok je overenie, či sa takto zapamätané písmeno rovná svojmu „páru“. Piaty a šiesty riadok opisujú návrat hlavy na začiatok pásky a prechod na kontrolu nasledujúceho písmena. Po úspešnom skontrolovaní posledného, n-tého páru príde k slovu zvyšok δ-funkcie, ktorý ešte overí, či už za skontrolovaným slovom nič ďalšie nenasleduje. (Rozmyslite si, že ak je vstupné slovo kratšie ako $2n+1$ znakov, výpočet sa už skôr zasekne.)

Čitateľ si iste všimol, že niektoré stavy nie sú dosiahnuteľné, avšak pomáhajú prehľadnosti konštrukcie. Počet stavov je $n(2n+2)4+n+2$, teda pre veľké n ich je rádovo menej ako v prípade jednosmerného deterministického automatu. (Tento počet stavov rastie polynomiálne od n, zatiaľ čo u DKA rástol exponenciálne.)

Veta 2.11.4 *Ku každému 2NKA A existuje ekvivalentný NKA A'.*

Dôkaz. BUNV nech A je v normálnom tvare z lemy 2.11.3, zjednoduší to našu konštrukciu.

Všimnime si, ako môže A využiť to, že sa môže vracať späť. Koľkokrát sa môže vrátiť na tú istú pozíciu na páske? Samozrejme, že potenciálne nekonečne veľa krát. Všimnime si však najkratší akceptačný výpočet. Počas neho sa určite nenachádza dvakrát na tej istej pozícii v tom istom stave – zodpovedajúci kus výpočtu by sme mohli vynechať. Automat A' sa teda bude snažiť nájsť taký akceptačný výpočet A, pri ktorom A nebol dvakrát v tom istom stave na tej istej pozícii.

Všimnime si ešte raz nejaký akceptačný výpočet A. Každému políčku na páske môžeme priradiť postupnosť dvojíc (s, d), kde s je stav, v ktorom sa A nachádzal, keď mal hlavu na tomto políčku a d je smer, ktorým odišiel. Túto postupnosť budeme volať *prechodová postupnosť*. Ak A nebol dvakrát v tom istom stave na tej istej pozícii, každá prechodová postupnosť má dĺžku najviac $|K|$. Takúto prechodovú postupnosť si A' vie pamätať v stave (bude mať rádovo $2^{|K|}|K|!$ stavov).

A' bude pracovať nasledovne: Na začiatku si tipne prechodovú postupnosť pre ¢. Čítajúc vstup, ku každému písmenu vstupu si nedeterministicky tipne prechodovú postupnosť, ktorá bude „pasovať“ na predchádzajúcu – t.j. takú, aby sa prechody medzi týmito postupnosťami naozaj mohli na prečítané symboly uskutočniť.[12] Uvedomte si, že keď poznáme δ, vieme pre ľubovoľné dve prechodové postupnosti (a zodpovedajúce dva symboly) algoritmicky rozhodnúť, či na seba pasujú – a teda zostrojiť δ'. Na konci výpočtu si už náš NKA len tipne prechodovú postupnosť pre \$, ktorá bude „pasovať“ na predchádzajúcu. Posledným členom tejto postupnosti by mal byť akceptačný stav.

V celej konštrukcii uvažujeme normálny tvar podľa lemy 2.11.3, v ktorom automat vždy pohne hlavou a preto je hodnota smeru posunu 1 alebo -1. Z akceptačného stavu na konci vstupu už ale nevykonávame žiadne kroky, preto v tomto špeciálnom prípade môžeme uvažovať dvojice $(q_F, 0)$ pre $q_F \in F$.

A' teda akceptuje, ak posledný člen poslednej prechodovej postupnosti pre \$ je $(q_F, 0)$, kde $q_F \in F$. (Teda F' tvoria tie prechodové postupnosti, ktoré spĺňajú túto podmienku.)

Príklad 2.11.3 Konštrukcia z príkladu 2.11.2 je v normálnom tvare podľa lemy 2.11.3. Uvažujme automat A_2 rozpoznávajúci jazyk L_2 z príkladu 2.11.2 a vstupné slovo $ab\#ab$. Ak označíme $u = ¢ab\#ab\$$, potom je akceptačný výpočet nasledovný:

[12] Pre prvý symbol to navyše znamená, že prvý člen postupnosti má byť $(q_0, \pm 1)$.

$$([1,0,?],u,0) \vdash ([1,1,?],u,1) \vdash ([1,2,a],u,2) \vdash ([1,3,a],u,3) \vdash$$
$$([1,4,a],u,4) \vdash (q_{spat,1},u,3) \vdash (q_{spat,1},u,2) \vdash (q_{spat,1},u,1) \vdash$$
$$(q_{spat,1},u,0) \vdash ([2,1,?],u,1) \vdash ([2,2,?],u,2) \vdash ([2,3,b],u,3) \vdash$$
$$([2,4,b],u,4) \vdash ([2,5,b],u,5) \vdash (q_{koniec},u,6) \vdash (q_{koniec2},u,5) \vdash (q_F,u,6)$$

Tomuto výpočtu zodpovedajú nasledovné prechodové postupnosti na jednotlivých znakoch:

¢	$([1,0,?],1),([q_{spat,1},1)$
a	$([1,1,?],1),(q_{spat,1},-1),([2,1,?],1)$
b	$([1,2,a],1),(q_{spat,1},-1),([2,2,?],1)$
#	$([1,3,a],1),(q_{spat,1},-1),([2,3,b],1)$
a	$([1,4,a],-1),([2,4,b],1)$
b	$([2,5,b],1),(q_{koniec2},1)$
\$	$(q_{koniec},-1),(q_F,0)$

Skúste si ručne overiť, že postupnosti pre # a po nej nasledujúce a na seba „pasujú“. Rozmyslite si, ako by sa takéto overovanie robilo mechanicky.

Kapitola 3

Bezkontextové jazyky

Ako názov kapitoly napovedá, budeme sa v nej zaoberať výlučne bezkontextovými jazykmi, preto keď v texte tejto kapitoly hovoríme o gramatike, myslíme tým bezkontextovú gramatiku (ak nebude explicitne uvedené ináč).

Pripomeňme si, že bezkontextové gramatiky sú frázové gramatiky, ktorých pravidlá sú podmnožinou $N \times (N \cup T)^*$.

3.1 Redukované bezkontextové gramatiky

V tejto časti definujeme, ktoré neterminály sú v bezkontextovej gramatike zbytočné a ako ich odtiaľ odstrániť.

Definícia 3.1.1 *Gramatika $G = (N, T, P, \sigma)$ je v **redukovanom tvare**, ak pre všetky neterminály ξ platí: $(\exists u, v \in (N \cup T)^*;\ \sigma \Rightarrow^* u\xi v) \wedge (\exists w \in T^*;\ \xi \Rightarrow^* w)$. Povedané slovami, každý neterminál musí byť dosiahnuteľný (vyskytovať sa v niektorej odvoditeľnej vetnej forme) a musí sa dať z neho odvodiť (aspoň jedno) terminálne slovo.*

Príklad 3.1.1 Nech je daná gramatika $G = (N = \{\sigma, A, B\}, T = \{a, b\}, P, \sigma)$, kde

$$\begin{aligned} P = \{\ \sigma &\rightarrow \sigma A \mid \sigma b \mid aa \\ A &\rightarrow aA \mid AB \\ B &\rightarrow aB \mid bb\ \} \end{aligned}$$

Všimnime si, že môžeme odstrániť neterminál A (a teda všetky pravidlá, ktoré ho obsahujú). Akonáhle sa totiž vo vetnej forme vyskytne A, už z nej určite nedokážeme odvodiť terminálne slovo. Jeho odstránením zjavne nezmeníme jazyk, ktorý G generuje. Zostane nám gramatika G' s $N' = \{\sigma, B\}$ a $P' = \{\sigma \rightarrow \sigma b,\ \sigma \rightarrow aa,\ B \rightarrow aB,\ B \rightarrow bb\}$.

Potom ale neterminál B aj so všetkými svojimi pravidlami je tiež úplne zbytočný, lebo žiadna vetná forma odvoditeľná v G' neobsahuje B. Preto môžeme odstrániť aj B (a opäť aj všetky pravidlá, ktoré ho obsahujú). Zostala nám gramatika $G'' = (\{\sigma\}, \{a, b\}, \{\sigma \rightarrow \sigma b,\ \sigma \rightarrow aa\}, \sigma)$, ktorá generuje rovnaký jazyk ako G a je v redukovanom tvare.

V ďalšom texte ukážeme, ako k danej bezkontextovej gramatike zostrojiť ekvivalentnú v redukovanom tvare.

Lema 3.1.1 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika. Hľadanú množinu dosiahnuteľných neterminálov označme H. (Teda $H = \{\xi \mid \exists u, v \in (N \cup T)^*;\ \sigma \Rightarrow^* u\xi v\}$.)*

Definujme induktívne nasledujúce množiny:

$$H_0 = \{\sigma\}, \quad H_{i+1} = H_i \cup \{\xi \mid \exists \eta \in H_i;\ \exists u, v \in (N \cup T)^*;\ (\eta \rightarrow u\xi v) \in P\}$$

Potom $H = H_{|N|-1}$.

Dôkaz. Nie je ťažké nahliadnuť, že H_k je množina neterminálov, ktoré sa môžu vyskytnúť vo vetnej forme gramatiky G po najviac k krokoch odvodenia. Zjavne $\forall k; H_k \subseteq H_{k+1}$ a $H = \bigcup_{k\geq 0} H_k$. Všimnime si, že ak pre nejaké m je $H_m = H_{m+1}$, tak aj $H_{m+1} = H_{m+2}$, $H_{m+2} = H_{m+3}$, ... Takéto m zjavne existuje, lebo neterminálov je len konečne veľa. A takisto zjavne je $H = \bigcup_{k=0}^{m} H_k = H_m$.

Zoberme najmenšie možné m. Potom $H_m \supsetneq H_{m-1} \supsetneq \ldots \supsetneq H_1 \supsetneq H_0$. To ale znamená, že $|H_m| \geq |H_{m-1}| + 1 \geq \ldots \geq |H_1| + (m-1) \geq |H_0| + m = m + 1$. Ale $H_m \subseteq N$, preto $|H_m| \leq n$, a teda $m \leq n - 1$. Potom ale $H = H_m = H_{n-1}$.

To isté ešte raz slovami: Všímajme si veľkosť množín H_i. Akonáhle v niektorom kroku nenájdeme žiaden nový dosiahnuteľný neterminál, môžeme skončiť, lebo vieme, že aktuálna množina H_i je už H. Koľko krokov môžeme najviac urobiť? Máme $|N| - 1$ neterminálov, ktoré v H_0 nie sú. V každom kroku musí do aktuálnej H_i pribudnúť aspoň jeden z nich, preto krokov bude najviac $|N| - 1$.

Poznámka 3.1.1 Uvedomte si, že táto lema nám dáva návod ako hľadanú množinu H efektívne zostrojiť.

Lema 3.1.2 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech H je množina dosiahnuteľných neterminálov v G. Zostrojme gramatiku $G' = (H, T, P' = P \cap (H \times (H \cup T)^*), \sigma)$. Potom $L(G) = L(G')$.*

Dôkaz. Inklúzia $L(G') \subseteq L(G)$ je zjavná, lebo $P' \subseteq P$, a teda každé odvodenie v G' je aj odvodením v G. Dokážeme opačnú inklúziu. Zoberme odvodenie nejakého slova w v G. V tomto odvodení sa nemohol vyskytnúť žiadny neterminál z $N \setminus H$ (lebo by bol dosiahnuteľný, a teda mal patriť do H). To ale znamená, že toto odvodenie je aj odvodením v G'.

Lema 3.1.3 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika. Nech $S = \{\xi \mid \exists w \in T^*;\ \xi \underset{G}{\Longrightarrow}^* w\}$ je množina tých neterminálov G, z ktorých sa dá odvodiť (aspoň jedno) terminálne slovo. Nech $u \underset{G}{\Longrightarrow}^* v$, kde $v \in T^*$. Potom $u \in (S \cup T)^*$.*

Dôkaz. Sporom. Všimnime si slová z $(N \cup T)^*$ také že nepatria do $(S \cup T)^*$, ale vieme z nich odvodiť terminálne slovo. Nech u je to (ľubovoľné) z nich, z ktorého vieme odvodiť terminálne slovo na najmenší možný počet krokov. Slovo u obsahuje (aspoň jeden) neterminál $\xi \notin S$. Nech $u = u_1 \xi u_2$. Neterminálu ξ sa musíme niekedy[1] zbaviť – prepísať ho použitím nejakého pravidla $\xi \to w$. Všetky neterminály z w sú (vďaka voľbe slova u) v S, preto z každého z nich vieme odvodiť terminálne slovo. Potom ale vieme odvodiť terminálne slovo aj z ξ, čo je spor s $\xi \notin S$.

Táto lema vlastne hovorí, že akonáhle sa vo vetnej forme vyskytne neterminál, ktorý nepatrí do S, už sa nám z nej nepodarí odvodiť terminálne slovo.

Lema 3.1.4 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech S je vyššie definovaná množina neterminálov G, z ktorých sa dá odvodiť terminálne slovo. Definujme induktívne nasledujúce množiny:*

$$S_0 = \{\xi \mid \exists v \in T^*;\ \xi \Rightarrow v\}, \quad S_{i+1} = S_i \cup \{\xi \mid \exists w \in (S_i \cup T)^*;\ (\xi \to w) \in P\}$$

Potom $S = S_{|N|-1}$.

Dôkaz. Tvrdenie sa ukáže rovnako ako lema 3.1.1. Keď neskôr definujeme stromy odvodenia, ľahko nahliadneme, že množina S_i predstavuje tie neterminály, pre ktoré existuje strom odvodenia s hĺbkou najviac i pre nejaké terminálne slovo.

Lema 3.1.5 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech S je vyššie definovaná množina neterminálov G, z ktorých sa dá odvodiť terminálne slovo. Zostrojme gramatiku $G' = (S, T, P' = P \cap (S \times (S \cup T)^*), \sigma)$. Potom $L(G) = L(G')$.*

[1] Dá sa dokonca ukázať, že $u \in T^*\{\xi\}T^*$, a teda ξ prepíšeme hneď v prvom kroku, ale nepotrebujeme to.

Dôkaz. Toto tvrdenie sa dokáže analogicky ako lema 3.1.2. Jedna inklúzia je zjavná, pri dôkaze druhej zoberme ľubovoľné odvodenie ľubovoľného slova w v G. Idúc od jeho konca vieme o každom neterminále ukázať, že patrí do S. (Ak sme použili pravidlo $\xi \to w$, o všetkých neterminálach v w vieme, že patria do S, preto aj $\xi \in S$.) Preto toto odvodenie je aj odvodením v G'.

Veta 3.1.6 *K ľubovoľnej bezkontextovej gramatike $G = (N, T, P, \sigma)$ existuje redukovaná gramatika G' s ňou ekvivalentná.*

Dôkaz. Z gramatiky G najskôr podľa lemy 3.1.5 odstránime neterminály, z ktorých nevieme odvodiť terminálne slovo, z výslednej gramatiky G_1 podľa lemy 3.1.2 odstránime nedosiahnuteľné neterminály. Tvrdíme, že takto získaná gramatika G_2 je redukovaná.

Každý neterminál $\xi \in N_{G_2}$ je v G_1 dosiahnuteľný v nejakom odvodení $\sigma \Rightarrow^* u\xi v$. Toto odvodenie je zjavne aj odvodením v G_2 (odstránili sme len nedosiahnuteľné neterminály, ktoré sa v ňom nevyskytujú), preto ξ je v G_2 dosiahnuteľný.

Analogicky ľubovoľný neterminál $\xi \in N_{G_2}$ patrí aj do N_{G_1}. To znamená, že v G preň existovalo odvodenie nejakého terminálneho slova. Toto odvodenie je podľa lemy 3.1.3 aj odvodením v G_1. Potom ale z rovnakého dôvodu ako vyššie[2] je to aj odvodenie v G_2. Preto z ξ vieme v G_2 odvodiť terminálne slovo, q.e.d.

Poznámka 3.1.2 Keby sme poradie krokov vymenili, výsledná gramatika ešte nemusí byť redukovaná. Vyskúšajte si to napr. na príklade 3.1.1. Ak si nevieme zapamätať správne poradie, stačí striedavo odstraňovať zlé neterminály jedného a druhého typu až kým nedostaneme redukovanú gramatiku. (V lepšom prípade to bude po druhom, v horšom po treťom kroku.)

Poznámka 3.1.3 Uvedomte si, že dôkaz vety 3.1.6 mohol vyzerať nasledovne: Striedavo z gramatiky odstraňujeme zlé neterminály jedného a druhého typu. Každé úspešné odstránenie nám zmenší počet neterminálov. Preto po konečnom počte krokov dostaneme gramatiku, ktorá neobsahuje zlé neterminály ani jedného typu. Náš dôkaz navyše hovorí, že redukovanú gramatiku našim postupom získame „takmer hneď".

3.2 Stromy odvodenia

Definícia 3.2.1 *Induktívne definujme* ***strom s vrcholmi z množiny*** *M:*
– *Ľubovoľný symbol $x \in M$ je strom.*
– *Pre ľubovoľný symbol $x \in M$, ľub. $k > 0$ a stromy $T_1, T_2, \ldots, T_k$ aj $x(T_1)(T_2)\ldots(T_k)$ je strom.*
– *Nič iné nie je strom.*

Definícia 3.2.2 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, potom každému odvodeniu $\xi \Rightarrow^* w$ (kde $\xi \in (N \cup T \cup \{\varepsilon\})$, $w \in (N \cup T)^*$) v G vieme jednoznačne priradiť strom s vrcholmi z $M = N \cup T \cup \{\varepsilon\}$ nasledovným induktívnym[3] postupom:*

Odvodeniu $\xi \Rightarrow^0 \xi$ priradíme strom ξ. Odvodeniu $\xi \Rightarrow \varepsilon$ priradíme strom $\xi(\varepsilon)$.

Nech teraz $\xi \Rightarrow u_1 \ldots u_k \Rightarrow^* v$ (kde $k > 0$) je nejaké odvodenie. Všimnime si u_i (to je terminál alebo neterminál). Keď teraz budeme prechádzať naše odvodenie a vyberať len tie kroky odvodenia, v ktorých prepisujeme časť vetnej formy, ktorá vzniká z u_i, dostaneme odvodenie $u_i \Rightarrow^* v_i$. Pritom zjavne $v = v_1 \ldots v_k$. Nech sme odvodeniu $u_i \Rightarrow^* v_i$ priradili strom T_i. Potom nášmu odvodeniu priradíme strom $\xi(T_1)\ldots(T_k)$.

Takto zostrojený strom voláme ***strom odvodenia****.*

Poznámka 3.2.1 Strom odvodenia si môžete predstaviť nasledovne: x je samostatný vrchol, $x(T_1)\ldots(T_k)$ je vrchol x, pod ktorým visia (v poradí zľava doprava) stromy $T_1, \ldots, T_k$.

Vrcholy stromu, pod ktorými nič nevisí, voláme listy. Indukciou[4] sa ľahko dokáže, že keď zoberieme ľubovoľný strom odvodenia, tak symboly v listoch (čítané zľava doprava) predstavujú odvodenú vetnú formu.

[2] Už vieme, že ξ je dosiahnuteľný, preto sú dosiahnuteľné aj všetky neterminály v tomto odvodení.
[3] Alebo ak sa vám viac páči pohľad z opačnej strany, rekurzívnym.
[4] Napríklad od dĺžky odvodenia.

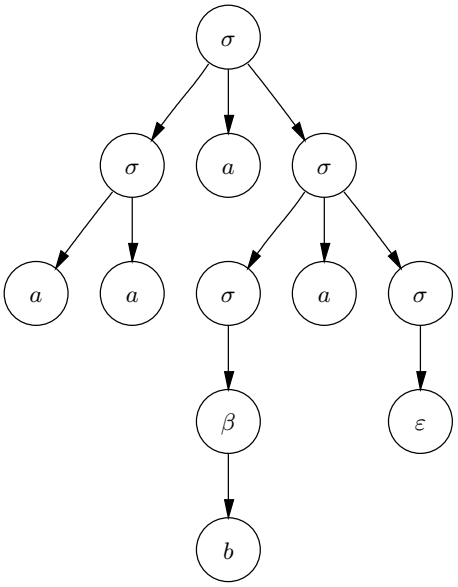
Analogicky sa dá ukázať, že strom odvodenia vieme zostrojovať spolu s odvodením: Jednoducho keď máme použiť pravidlo $\xi \to w_1 \dots w_k$, nájdeme list obsahujúci prepisované ξ a zavesíme pod neho k vrcholov, obsahujúcich $w_1, \dots, w_k$. (Resp. jeden vrchol obsahujúci ε ak $k = 0$.)

Poznámka 3.2.2 Uvedomte si, že strom odvodenia sme definovali pre **jedno konkrétne** odvodenie v jednej konkrétnej gramatike. Strom odvodenia ešte neurčuje gramatiku, ba ani odvodenie. Jednému stromu odvodenia môže zodpovedať viacero rôznych odvodení (líšiacich sa poradím použitia pravidiel). Toto je vlastne dôvod, prečo stromy odvodenia definujeme – dve odvodenia v bezkontextovej gramatike budeme považovať za rovnaké, ak majú rovnaký strom odvodenia.

Príklad 3.2.1 Nech $G = (N = \{\sigma, \beta\}, T = \{a, b\}, P, \sigma)$, kde $P = \{\sigma \to \sigma a \sigma \mid aa \mid \beta \mid \varepsilon,\ \beta \to b\}$. V tejto gramatike vieme odvodiť napr. slovo *aaaba*. Jedno možné odvodenie je nasledujúce (prepisovaný neterminál je podčiarknutý):

$$\underline{\sigma} \Rightarrow \sigma a \underline{\sigma} \Rightarrow \sigma a \underline{\sigma} a \sigma \Rightarrow \underline{\sigma} a \beta a \sigma \Rightarrow aaa \beta a \underline{\sigma} \Rightarrow aaa \underline{\beta} a \Rightarrow aaaba$$

Tomuto odvodeniu zodpovedá nasledujúci strom odvodenia:



Definícia 3.2.3 *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**.*

Príklad 3.2.2 Príkladom viacznačnej gramatiky je $G = (\{\sigma\}, \{a, b\}, \{\sigma \to \sigma a \sigma b \sigma \mid \varepsilon\}, \sigma)$. Napríklad pre slovo *abab* ľahko nájdeme dva rôzne stromy odvodenia. Príkladom jednoznačnej gramatiky je $G = (\{\sigma\}, \{a, b\}, \{\sigma \to a \sigma b \mid \varepsilon\}, \sigma)$.

Príklad 3.2.3 Aj gramatika z príkladu 3.2.1 je viacznačná – slovo *aaaba* vieme odvodiť aj nasledovne: $\underline{\sigma} \Rightarrow \underline{\sigma} a \sigma \Rightarrow \sigma a \sigma a \sigma \Rightarrow^* aaaba$. Tomuto odvodeniu zodpovedá iný strom odvodenia ako ten uvedený v príklade 3.2.1.

Definícia 3.2.4 *Hovoríme, že bezkontextový jazyk L je **jednoznačný**, ak existuje jednoznačná bezkontextová gramatika G, taká, že $L = L(G)$. Inak hovoríme, že L je **vnútorne viacznačný**.*

Príklad 3.2.4 Príkladom vnútorne viacznačného jazyka je jazyk $L = \{a^i b^j c^k \mid (i = j) \vee (j = k)\}$.

Poznámka 3.2.3 Idea dôkazu, že tento jazyk je naozaj vnútorne viacznačný: Každá z dvoch častí L si vyžaduje svoj spôsob odvodzovania. Ale slová tvaru $a^n b^n c^n$ sa potom dajú odvodiť dvoma spôsobmi, lebo nevieme kontrolovať, či nastali obe rovnosti naraz.

Definícia 3.2.5 *Hovoríme, že odvodenie $w_0 = \sigma \Rightarrow w_1 \Rightarrow^* w_n$ je* ***ľavým krajným odvodením****, ak platí, že v každom kroku odvodenia prepisujeme najľavejší neterminál vetnej formy. Formálne povedané, musí platiť:*

$$\forall i;\ \exists t \in T^*,\ \xi \in N,\ u, w \in (N \cup T)^*;\ (\xi \to w) \in P\ \wedge\ w_i = t\xi u\ \wedge\ w_{i+1} = twu$$

Poznámka 3.2.4 Uvedomte si, že každému stromu odvodenia prislúcha práve jedno ľavé krajné odvodenie. (Inými slovami, jeden strom zodpovedá veľa odvodeniam a práve jedno z nich je ľavé krajné.) Odtiaľ ľahko nahliadneme, že každé slovo z $L(G)$ má v G (aspoň jedno) ľavé krajné odvodenie. Navyše môžeme podať alternatívnu definíciu jednoznačnej gramatiky: Gramatika je jednoznačná, ak každé slovo má jednoznačné ľavé krajné odvodenie.

Príklad 3.2.5 Odvodeniu (a stromu odvodenia) z príkladu 3.2.1 zodpovedá ľavé krajné odvodenie

$$\underline{\sigma} \Rightarrow \underline{\sigma}a\sigma \Rightarrow aaa\underline{\sigma} \Rightarrow aaa\underline{\sigma}a\sigma \Rightarrow aaa\underline{\beta}a\sigma \Rightarrow aaaba\underline{\sigma} \Rightarrow aaaba$$

3.3 Chomského normálny tvar

Definícia 3.3.1 *Hovoríme, že gramatika $G = (N, T, P, \sigma)$ je v* ***Chomského normálnom tvare****, ak $P \subseteq N \times (NN \cup T \cup \{\varepsilon\})$.*

Poznámka 3.3.1 Motivácia za Chomského normálnym tvarom: Každému odvodeniu v gramatike, ktorá je v Chomského normálnom tvare, prislúcha striktne binárny strom odvodenia. (Buď použijeme pravidlo tvaru $N \to NN$ a dostaneme dva podstromy, alebo pravidlom $N \to (T \cup \{\varepsilon\}$ vetvu ukončíme.) Ako to už pri normálnych tvaroch býva, táto vlastnosť sa môže hodiť pri dôkazoch a konštrukciách.

Veta 3.3.1 *Ku každej bezkontextovej gramatike G existuje gramatika G' s ňou ekvivalentná, ktorá je v Chomského normálnom tvare.*

Dôkaz. Nech $G = (N, T, P, \sigma)$. Zostrojme najskôr gramatiku G_1 ekvivalentnú s G, ktorá bude mať na pravej strane každého pravidla buď len (najviac jeden) terminál, alebo len neterminály. Nech $\xi_x \mid x \in T$ sú nové neterminály, nech h je homomorfizmus, ktorý je na N identita a $\forall x \in T;\ h(x) = \xi_x$. Potom

$$G_1 = (N \cup \{\xi_x \mid x \in T\}, T, P_1, \sigma)$$

$$P_1 = \{\eta \to h(w) \mid (\eta \to w) \in P\}\ \cup\ \{\xi_x \to x \mid x \in T\}$$

Na pravej strane pravidiel G sme teda terminály nahradili novými neterminálmi a pridali sme pravidlá, ktorými ich prepíšeme na pôvodné terminály. Ľahko nahliadneme, že $L(G_1) = L(G)$. Teraz už len potrebujeme upraviť pravidlá, ktoré majú na pravej strane neterminály, na správnu dĺžku. Použijeme konštrukciu podobnú ako v dôkaze lemy 2.4.3 o normálnom tvare reg. gramatík.

Nech k je maximum spomedzi dĺžok pravých strán pravidiel. Nech ξ_ε a $\psi_{p,i}$ (pre $p \in P_1$, $i \in \{1, \ldots, k-1\}$) sú nové neterminály. Bude

$$\begin{aligned} G' &= (N', T, P', \sigma) \\ N' &= N_1 \cup \{\xi_\varepsilon\} \cup \{\psi_{p,i} \mid p \in P_1, i \in \{1, \ldots, k-1\}\} \end{aligned}$$

Pritom P' bude obsahovať pravidlá $\{\xi_x \to x \mid x \in T\}$ a $\xi_\varepsilon \to \varepsilon$. Každé pravidlo $(\eta \to w) \in P_1$, kde $w \in N^+$, nahradíme v P' niekoľkými pravidlami – ak $|w| = 2$, pravidlo je dobré a necháme ho, ak $|w| = 1$, nahradíme ho pravidlom $\eta \to w\xi_\varepsilon$, ak $|w| > 2$, rozdelíme ho na niekoľko pravidiel pomocou nových neterminálov $\psi_{\eta \to w,i}$.

Formálne:

$$\begin{aligned} P' &= \{\xi_\varepsilon \to \varepsilon\} \\ &\cup \{\xi_x \to x \mid x \in T\} \\ &\cup \{\eta \to \varphi\xi_\varepsilon \mid (\eta \to \varphi) \in P_1\} \\ &\cup \{\eta \to \varphi_1\varphi_2 \mid (\eta \to \varphi_1\varphi_2) \in P_1\} \\ &\cup \{\eta \to \varphi_1\psi_{\pi,1} \mid \pi = (\eta \to \varphi_1 \dots \varphi_k) \in P_1 \ \wedge \ k > 2\} \\ &\cup \{\psi_{\pi,i} \to \varphi_{i+1}\psi_{\pi,i+1} \mid \pi = (\eta \to \varphi_1 \dots \varphi_k) \in P_1 \ \wedge \ k > 2 \ \wedge \ i \in \{1, \dots, k-3\}\} \\ &\cup \{\psi_{\pi,k-2} \to \varphi_{k-1}\varphi_k \mid \pi = (\eta \to \varphi_1 \dots \varphi_k) \in P_1 \ \wedge \ k > 2\} \end{aligned}$$

Zjavne G' je v Chomského normálnom tvare. Dôkaz tvrdenia $L(G') = L(G)$ prenechávame čitateľovi.

Poznámka 3.3.2 Dá sa definovať aj tzv. prísny Chomského normálny tvar. Gramatika $G = (N, T, P, \sigma)$ je v prísnom Chomského normálnom tvare, ak $P \subseteq \{\sigma \to \varepsilon\} \cup (N \times T) \cup (N \times N'N')$ (kde $N' = N \setminus \{\sigma\}$). Teda na pravej strane pravidla sú buď dva neterminály alebo jeden terminál. (A navyše máme pravidlo $\sigma \to \varepsilon$, ktorým môžeme v prvom kroku odvodenia vygenerovať prázdne slovo.)

Previesť danú bezkontextovú gramatiku do prísneho Chomského normálneho tvaru je o dosť zložitejšie. Začneme zavedením neterminálov ξ_ε a ξ_x ako v pôvodnom dôkaze. Výslednú gramatiku musíme odepsilonovať (viď časť 3.4, vrátane poznámky 3.4.2). Potom z nej odstrániť chain rules (viď časť 3.6). Jediný problém potom tvoria pravidlá, ktoré majú na pravej strane viac ako dva neterminály, tie nahradíme rovnako ako v pôvodnom dôkaze.

3.4 Bezepsilonové gramatiky

Definícia 3.4.1 *Bezkontextová gramatika G je v **bezepsilonovom tvare** (je ε-free), ak $P \subseteq N \times (N \cup T)^+$.*

Poznámka 3.4.1 Pravidlu $\xi \to \varepsilon$ hovoríme tiež epsilonové. Gramatike v bezepsilonovom tvare skrátene hovoríme bezepsilonová gramatika.

V tejto časti ukážeme, že ku každej bezkontextovej gramatike existuje „takmer ekvivalentná“, ktorá nepotrebuje vymazávať – t.j. neobsahuje epsilonové pravidlá. Ekvivalentnú gramatiku nemusíme vedieť zostrojiť preto, že bezepsilonová gramatika nevie vygenerovať slovo ε – toto však bude jediný problém.

Veta 3.4.1 *K ľubovoľnej bezkontextovej gramatike $G = (N, T, P, \sigma)$ existuje bezepsilonová bezkontextová gramatika G' taká, že $L(G) \setminus \{\varepsilon\} = L(G')$.*

Dôkaz. Nech $H = \{\xi \mid \xi \underset{G}{\Longrightarrow}^* \varepsilon\}$ je množina vymazávajúcich neterminálov v G. (Ľahko ju zostrojíme podobne, ako napr. množinu dosiahnuteľných neterminálov v leme 3.1.1.) Upravíme pravidlá gramatiky G tak, aby mala možnosť tieto neterminály nevygenerovať – to bude mať rovnaký efekt, ako keby ho vygenerovala a následne zmazala.

Napr. ak by sme v G mali pravidlá $\sigma \to \gamma\beta\beta$ a $\beta \to \varepsilon$, môžeme pridať pravidlá $\sigma \to \gamma\beta$ a $\sigma \to \gamma$, čím sa pravidlo $\beta \to \varepsilon$ stane zbytočným – keď β nechceme, nevygenerujeme ju a nemusíme ju potom vymazávať.

Formálne: Bude $G' = (N, T, P', \sigma)$, kde

$$\begin{aligned} P' = \Big\{ \alpha \to \beta_1 \dots \beta_k \ \Big| \ & k > 0 \ \wedge \ \beta_i \in (N \cup T)^* \ \wedge \ \beta_1 \dots \beta_k \neq \varepsilon \ \wedge \\ & \exists \gamma_1, \dots, \gamma_{k-1} \in H; \ (\alpha \to \beta_1\gamma_1\beta_2\gamma_2 \dots \gamma_{k-1}\beta_k) \in P \Big\} \end{aligned}$$

Ku každému pravidlu z P, ktoré má na pravej strane vymazávajúce neterminály, pridáme všetky jeho verzie, v ktorých niektoré z nich chýbajú. (Jediný špeciálny prípad, ktorý si treba uvedomiť, je že nikdy nepridávame nové pravidlo $\xi \to \varepsilon$. Toto by nás mohlo napadnúť, ak máme pravidlo tvaru $\xi \to w$, kde $w \in H^+$. Potom ale aj $\xi \in H$, a teda pravidlo $\xi \to \varepsilon$ nepotrebujeme pridať – neterminálu ξ sa tiež vieme zbaviť „o krok skôr“.)

Formálny dôkaz, že $L(G) \setminus \{\varepsilon\} = L(G')$ by mohol vyzerať približne nasledovne: Nech $w \neq \varepsilon$, $w \in L(G)$. Zoberieme odvodenie w v G. Jemu prislúcha strom odvodenia. Označme v ňom všetky listy obsahujúce ε a aj všetky vrcholy, zodpovedajúce neterminálom, z ktorých sa odvodilo slovo ε. Teraz odstráňme všetky označené vrcholy. Dostaneme strom odvodenia w v G'.

Naopak, nech $w \in L(G')$. Zjavne $w \neq \varepsilon$. Zoberme odvodenie w v G'. Budeme postupne podľa neho zostrojovať odvodenie w v G. Vždy, keď sme v G' použili pravidlo π, v G použijeme pravidlo, z ktorého π vzniklo. Tým sa nám možno vo vetnej forme objaví navyše niekoľko neterminálov z H, tie ale vieme konečným počtom krokov vymazať a pokračujeme v odvodení.

Poznámka 3.4.2 Kvôli riešeniu problémov so slovom ε sa niekedy zvykne definovať, že gramatika je ε-free, ak má začiatočný neterminál σ a jej pravidlá sú podmnožinou $\{\sigma \to \varepsilon\} \cup N \times ((N \setminus \{\sigma\}) \cup T)^+$. (Teda gramatika buď v prvom kroku vygeneruje ε, alebo sa správa ako bezepsilonová gramatika z našej definície.) Takto definované ε-free gramatiky zjavne tvoria normálny tvar bezkontextových gramatík.

3.5 Greibachovej normálny tvar

Definícia 3.5.1 *Hovoríme, že bezkontextová gramatika $G = (N, T, P, \sigma)$ je v **Greibachovej normálnom tvare**, ak platí $P \subseteq N \times T(N \setminus \{\sigma\} \cup T)^* \cup \{\sigma \to \varepsilon\}$.*

Poznámka 3.5.1 Definícia sa dá jemne sprísniť – môžeme požadovať, aby pravidlá boli podmnožinou $N \times T(N \setminus \{\sigma\})^* \cup \{\sigma \to \varepsilon\}$. Takýto tvar pravidiel z nášho dosiahneme nahradením terminálov neterminálmi ξ_x (pre $x \in T$) rovnako ako pri konštrukcii Chomského normálneho tvaru.

Poznámka 3.5.2 Motivácia za Greibachovej normálnym tvarom: Ak máme gramatiku v Greibachovej normálnom tvare, v každom kroku odvodenia určite vygenerujeme aspoň jeden terminálny symbol. Z toho okrem iného vyplýva, že vieme zhora odhadnúť dĺžku odvodenia ľubovoľného slova $w \in L(G)$.

Definícia 3.5.2 *Ak ξ je neterminálom gramatiky $G = (N, T, P, \sigma)$, potom ξ-**pravidlom** nazveme každé pravidlo $(\xi \to w) \in P$. (Teda každé pravidlo s ľavou stranou ξ a ľubovoľnou pravou stranou.)*

Lema 3.5.1 *Nech $G = (N, T, P, \sigma)$, nech $\pi = (\xi \to \eta w) \in P$. Nech $\eta \to u_1 \mid \ldots \mid u_n$ sú všetky η-pravidlá. Potom $G' = (N, T, P', \sigma)$, kde $P' = (P \setminus \{\pi\}) \cup \{\xi \to u_i w \mid 1 \leq i \leq n\}$ je ekvivalentná s G.*

Lema 3.5.2 *Nech $G = (N, T, P, \sigma)$ je gramatika, Nech $P_\xi = \{\xi \to \xi w_i \mid 1 \leq i \leq n\}$ je množina všetkých ξ-pravidiel, ktorých pravá strana začína neterminálom ξ. Nech $\{\xi \to u_j \mid 1 \leq j \leq m\}$ sú zvyšné ξ-pravidlá. Nech η je nový neterminál. Potom $G' = (N \cup \{\eta\}, T, P', \sigma)$, kde*

$$P' = (P \setminus P_\xi) \cup \{\xi \to u_j \eta \mid 1 \leq j \leq m\} \cup \{\eta \to w_i \eta \mid 1 \leq i \leq n\} \cup \{\eta \to w_i \mid 1 \leq i \leq n\}$$

je ekvivalentná s G.

Poznámka 3.5.3 Dôkaz oboch liem je triviálny. Ich používaním vo vhodnom poradí upravíme danú bezkontextovú gramatiku do požadovaného tvaru.

Veta 3.5.3 *Ku každej bezkontextovej gramatike G existuje ekvivalentná gramatika G' v Greibachovej normálnom tvare.*

Dôkaz. BUNV nech G je v ε-free tvare (poznámka 3.4.2). Ak $\varepsilon \in L(G)$, pravidlo $\sigma \to \varepsilon$ nateraz z G odstránime, na konci ho do G' späť pridáme. Pravá strana každého pravidla v G má teda dĺžku aspoň 1.

Nech $G = (N = \{\xi_1, \ldots \xi_n\}, T, P, \sigma)$. (Teda ľubovoľným spôsobom si očíslujeme neterminály v G.) Potrebujeme z G odstrániť pravidlá, ktorých pravé strany začínajú neterminálom. To spravíme v dvoch fázach. Najskôr sa zbavíme pravidiel tvaru $\xi_i \to \xi_j w$, kde $i \geq j$, následne odstránime aj zvyšné zlé pravidlá.

Predpokladajme, že pravidlá, ktoré majú na ľavej strane jeden spomedzi neterminálov $\xi_1, \ldots, \xi_{k-1}$ už sú v požadovanom tvare (všetky pravé strany ξ_i-pravidiel začínajú neterminálom alebo terminálom s vyšším číslom). Upravme teraz ξ_k-pravidlá do tohto tvaru.

Pomocou lemy 3.5.1 sa postupne zbavíme všetkých pravidiel tvaru $\xi_k \to \xi_1 w$, $\xi_k \to \xi_2 w$, $\ldots$, $\xi_k \to \xi_{k-1} w$. Rozmyslite si, že keď odstraňujeme pravidlo $\xi_k \to \xi_i w$, všetky ξ_i-pravidlá už majú na začiatku pravej strany terminál s vyšším číslom alebo neterminál. Preto keď postupom z lemy 3.5.1 odstránime pravidlo $\xi_k \to \xi_i w$, nevzniknú žiadne nové pravidlá typov, ktoré sme už odstránili.

Ešte potrebujeme odstrániť pravidlá tvaru $\xi_k \to \xi_k w$. Nech ξ_{k-n} je nový neterminál, ktorý pridáme pri použití lemy 3.5.2 na ich odstránenie. (Uvedomte si, že všetky ξ_{k-n}-pravidlá, ktoré použitím lemy vzniknú, majú na začiatku pravej strany buď terminál, alebo pôvodný neterminál. Tieto pravidlá teda už sú požadovaného tvaru.)

Teraz už jediné zlé pravidlá v našej gramatike sú tvaru $\xi_i \to \xi_j$, kde $i < j$. Na ich odstránenie opäť použijeme lemu 3.5.1. Lemu postupne použijeme na všetky ξ_{n-1}-pravidlá, ξ_{n-2} pravidlá, atď. (Všetky ξ_n-pravidlá sú v poriadku. Keď nahrádzame pravidlo $\xi_i \to \xi_j w$, všetky ξ_j-pravidlá už majú na začiatku pravej strany terminál, lebo $i < j$. Po dosadení teda vzniknú opäť len dobré pravidlá.)

3.6 Reťazcové pravidlá (Chain rules)

Definícia 3.6.1 ***Reťazcovým pravidlom*** *(chain rule) nazývame každé pravidlo z $N \times N$.*

Poznámka 3.6.1 Je pomerne ľahké zbaviť sa chain rules – stačí zaviesť nový neterminál ξ_ε, tým „doplniť“ pravú stranu každého chain rule a pridať pravidlo $\xi_\varepsilon \to \varepsilon$. Na takomto riešení sa nám nepáči okrem iného to, že pridá do gramatiky epsilonové pravidlo. Pre viacero konštrukcii by bolo dobré vedieť sa zbaviť chain rules tak, aby sme gramatiku „čo najmenej porušili“. Budeme sa preto zbavovať chain rules tak, že budeme iba vhodne „dosádzať“ jedno pravidlo do druhého.

Poznámka 3.6.2 Vysvetlíme, čo sme mysleli slovom „dosádzať“ a ukážeme, prečo treba dosádzať vhodne. Nech sa chceme zbaviť pravidla $\xi \to \psi$. Jeden možný spôsob je zobrať všetky pravidlá $\psi \to w_i$ a „dosadiť“ ich do pravej strany pravidla $\xi \to \psi$. Dostaneme teda nové pravidlá $\xi \to w_i$ a ľahko nahliadneme, že pravidlo $\xi \to \psi$ je už zbytočné.

Nestačí však týmto spôsobom v ľubovoľnom poradí po jednom odstraňovať chain rules, dosadením nám totiž môžu vzniknúť nové a dokonca sa môžeme zacykliť. Vyskúšajte si to na gramatike s pravidlami $\sigma \to \xi$, $\xi \to \psi$, $\psi \to \xi$. (Odstránením $\sigma \to \xi$ nám pribudne $\sigma \to \psi$, odstránením toho nám opäť pribudne $\sigma \to \xi$.)

Veta 3.6.1 *K ľubovoľnej bezkontextovej gramatike G existuje ekvivalentná bezkontextová gramatika G', ktorá neobsahuje chain rules a navyše pravá strana každého pravidla v G' je pravou stranou niektorého pravidla v G.*

Dôkaz. Ukážeme konštrukciu založenú na teórií grafov. Zostrojme orientovaný graf, ktorého vrcholy sú neterminály našej gramatiky. Každé pravidlo $\xi \to \psi$ znázorníme orientovanou hranou z ξ do ψ.

Teraz nech $M_\xi = \{\psi \mid \text{z } \xi \text{ je dosiahnuteľný } \psi\}$. Z grafu ľahko zostrojíme M_ξ pre každý $\xi \in N$ prehľadávaním (napr. do šírky) v smere hrán. Zostrojíme $G' = (N, T, P', \sigma)$, kde

$$P' = \{\xi \to w \mid \xi \in N \ \wedge \ w \notin N \ \wedge \ \exists \psi \in M_\xi; \ (\psi \to w) \in P\}$$

Namiesto každého pravidla $\psi \to w$ máme teda v G' pravidlá $\xi \to w$ pre všetky ξ, z ktorých sa (používaním chain rules) dalo odvodiť ψ. Chain rules sú teda už zbytočné – všetko, čo sme vedeli z ľubovoľného neterminálu ξ odvodiť tak, že sme ho niekoľkokrát prepísali pomocou vhodného chain rule a potom spravili ďalší krok odvodenia vieme teraz odvodiť z ξ priamo na jeden krok. Formálny dôkaz správnosti konštrukcie prenechávame čitateľovi.

Poznámka 3.6.3 Predchádzajúcu konštrukciu vieme spraviť aj ináč, navyše v prípade, že gramatika obsahuje cyklické chain rules sa nám podarí zredukovať počet jej neterminálov.

Množinu neterminálov M budeme volať *množina ekvivalentných neterminálov*, ak

$$\forall \xi, \psi \in M,\ \forall u, v \in (N \cup T)^*;\ \sigma \Rightarrow^* u\xi v \iff \sigma \Rightarrow^* u\psi v$$

Zjavne jazyk generovaný G sa nezmení, ak všetky neterminály nejakej množiny ekvivalentných neterminálov nahradíme jedným novým neterminálom.

V našom grafe nájdeme silno súvislé komponenty. Zjavne každý silno súvislý komponent S je množina ekvivalentných neterminálov – totiž

$$\forall \xi, \psi \in S;\ \xi \Rightarrow^* \psi \ \wedge\ \psi \Rightarrow^* \xi$$

Neterminály z každého silno súvislého komponentu teda nahradíme jedným novým neterminálom a následne z gramatiky odstránime všetky pravidlá tvaru $\xi \to \xi$.

Takto dostaneme gramatiku G'. Jej zodpovedajúci graf chain rules je už acyklický. Na odstránenie zvyšných chain rules stačí spracúvať vrcholy „odspodu“ v topologickom usporiadaní. Aby sme sa zbavili pravidla $\xi \to \psi$, stačí „dosadiť“ pravidlá z už spracovaného neterminálu ψ. Teda pridáme pravidlá $\xi \to w$ (kde $(\psi \to w) \in P'$) a vyhodíme pravidlo $\xi \to \psi$.

3.7 Zásobníkové automaty

Najväčším obmedzením konečného automatu bola jeho konečná pamäť. Pokúsme sa ho „vylepšiť“ tak, že mu dáme k dispozícii jeden nekonečný zásobník.[5]

Definícia 3.7.1 ***Nedeterministický zásobníkový automat*** *je 7-ica $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde K je konečná množina stavov, Σ je konečná abeceda vstupných symbolov, Γ je konečná 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.*

Poznámka 3.7.1 2^X_{kon} je množina všetkých **konečných** podmnožín množiny X. V našom prípade to znamená, že v ľubovoľnej situácií má automat len konečne veľa možností, medzi ktorými sa musí rozhodnúť.

Poznámka 3.7.2 Všimnime si, že podľa našej definície δ-funkcie bude zásobníkový automat musieť v každom kroku čítať písmeno zo zásobníka. To ale znamená, že ak sa mu niekedy počas výpočtu zásobník vyprázdni, automat sa zasekne. Začiatočný zásobníkový symbol Z_0 je teda v zásobníku (aj) preto, aby sa automat nezasekol hneď na začiatku výpočtu.

Definícia 3.7.2 ***Konfiguráciou*** *nedeterministického zásobníkového automatu nazývame trojicu (q, w, s), kde $q \in K$ je aktuálny stav, $w \in \Sigma^*$ neprečítaná časť vstupného slova a $s \in \Gamma^*$ obsah zásobníka.*

Definícia 3.7.3 ***Krokom výpočtu*** *nedeterministického zásobníkového automatu A nazývame reláciu $\vdash_A$ na množine konfigurácií definovanú takto:*

$$(q, au, sZ) \vdash_A (p, u, st) \iff (p, t) \in \delta(q, a, Z)$$

kde $p, q \in K$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, $s, t \in \Gamma^*$

[5] Dátová štruktúra s operáciami push (vlož prvok) a pop (vyber najneskôr vložený prvok).

Poznámka 3.7.3 Automat teda funguje tak, že v jednom kroku výpočtu vyberie jedno písmeno zo zásobníka, prečíta najviac jedno písmeno zo vstupu, podľa príslušnej časti δ-funkcie sa rozhodne, ako zmení stav a aké symboly vloží na vrch zásobníka.

Poznámka 3.7.4 V literatúre vládne nejednotnosť ohľadom toho, či vrch zásobníka písať vľavo alebo vpravo. My používame notáciu, v ktorej je vrch zásobníka 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 vie dostať do akceptačného stavu. Druhá možnosť je taká, že za akceptované slovo považujeme také, po prečítaní ktorého sa vyprázdni zásobník.[6] Neskôr dokážeme, že rozhodnutie, ktorú z týchto definícií použiť neovplyvní silu nedeterministického zásobníkového automatu.

Definícia 3.7.4 ***Jazyk*** *akceptovaný zásobníkovým automatom A* ***akceptačným stavom*** *je množina $L(A) = \{w \mid \exists q_F \in F,\ s \in \Gamma^*;\ (q_0, w, Z_0) \vdash_A^* (q_F, \varepsilon, s)\}$.*

Definícia 3.7.5 ***Jazyk*** *akceptovaný zásobníkovým automatom A* ***prázdnou pamäťou*** *je množina $N(A) = \{w \mid \exists q \in K;\ (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \varepsilon)\}$.*

Poznámka 3.7.5 Niekedy sa používa konvencia, že ak $F \neq \emptyset$, tak $L(A)$ je jazyk akceptovaný stavom, ak $F = \emptyset$, tak $L(A)$ je jazyk akceptovaný prázdnou pamäťou.

Z definície zásobníkových automatov priamo vyplýva, že ich sila nie je menšia ako sila konečných automatov. Zatiaľ však ešte nevieme ani to, či je väčšia. Nasledujúci príklad ukazuje, že áno.

Príklad 3.7.1 Zostrojíme zásobníkový automat pre jazyk $L = \{a^n b^n \mid n \geq 0\}$, o ktorom vieme, že nie je regulárny.

Náš automat bude akceptovať L prázdnou pamäťou. Bude

$$A = (K = \{q_a, q_b\}, \Sigma = \{a, b\}, \Gamma = \{Z, A\}, \delta, q_a, Z, F = \emptyset)$$

$$\begin{aligned}
\delta(q_a, a, X) &\ni (q_a, XA) \quad (\forall X \in \Gamma) \\
\delta(q_a, b, A) &\ni (q_b, \varepsilon) \\
\delta(q_b, b, A) &\ni (q_b, \varepsilon) \\
\delta(q, \varepsilon, Z) &\ni (q, \varepsilon) \quad (\forall q \in K)
\end{aligned}$$

Myšlienka: V stave q_a číta písmená a a na zásobník dáva symboly A. Keď príde prvé b, zmení stav na q_b. Vždy, keď chce prečítať zo vstupu b, musí zo zásobníka vyhodiť jedno A. Keď má na vrchu zásobníku Z, môže ho vymazať. Ak vymaže Z skôr ako na konci slova, zasekne sa. Aby ho ale mohol vymazať na konci slova (a teda akceptovať), zjavne musel prečítať najskôr niekoľko a, potom toľko isto b.

Poznámka 3.7.6 Zásobníkové automaty budeme označovať skratkou PDA (z anglického *push-down automaton*). Nezabúdajte, že nami definovaný PDA je nedeterministický. (Keď neskôr definujeme aj deterministický zásobníkový automat, budeme ho označovať DPDA. Ak by mohlo dôjsť k omylu, budeme o PDA hovoriť ako o „nedeterministickom PDA“.)

[6] Keby sme chceli byť puntičkári, je tu ešte tretia možnosť – automat musí súčasne vyprázdniť zásobník a prejsť do akceptačného stavu.

3.8 Ekvivalencia medzi akceptáciou prázdnou pamäťou a akceptačným stavom

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

Veta 3.8.1 *K ľubovoľnému zásobníkovému automatu $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ existuje zásobníkový automat A' taký, že $N(A') = L(A)$.*

Dôkaz. Myšlienka konštrukcie: Simulujeme A. Ak A akceptoval, prejdeme do nového stavu, v ktorom vyprázdnime zásobník.

Nech q_z je nový stav, Z'_0 nový zásobníkový symbol. Potom zostrojíme automat A' nasledovne: Bude $A' = (K' = K \cup \{q_z\}, \Sigma, \Gamma, \delta', q_0, Z'_0, F' = \emptyset)$, kde:
$\delta'(q_0, \varepsilon, Z'_0) = \{(q_0, Z'_0 Z_0)\}$
$\forall q \in K,\ \forall x \in \Sigma \cup \{\varepsilon\},\ \forall Z \in \Gamma;\ \delta'(q, x, Z)$ obsahuje $\delta(q, x, Z)$
$\forall q \in F,\ \forall Z \in \Gamma;\ \delta'(q, \varepsilon, Z) \ni (q_z, Z)$
$\forall Z \in \Gamma;\ \delta'(q_z, \varepsilon, Z) = \{(q_z, \varepsilon)\}$

Ak A akceptoval w stavom, znamená to, že existoval výpočet $(q_0, w, Z_0) \vdash_A^* (q_F, \varepsilon, x)$, kde $q_F \in F$ a $x \in \Gamma^*$. Potom ale v A' existuje akceptačný výpočet

$$(q_0, w, Z'_0) \vdash_{A'} (q_0, w, Z'_0 Z_0) \vdash_{A'}^* (q_F, \varepsilon, Z'_0 x) \vdash_{A'} (q_z, \varepsilon, Z'_0 x) \vdash_{A'}^* (q_z, \varepsilon, \varepsilon)$$

Naopak, ak A' akceptoval w prázdnou pamäťou, musel skončiť v q_z (lebo v inom stave nevie zo zásobníka vyhodiť Z'_0) a dočítať vstupné slovo. V okamihu, keď prešiel do q_z, mal prečítané vstupné slovo a bol v stave, ktorý je v A akceptačný. Ľahko nahliadneme, že výpočtu do tohto okamihu zodpovedá akceptačný výpočet A na w.

Veta 3.8.2 *K ľubovoľnému zásobníkovému automatu A existuje zásobníkový automat A' taký, že $N(A) = L(A')$.*

Dôkaz. Analogicky ako v predchádzajúcej vete. Uvedieme len myšlienku konštrukcie. Opäť pridáme na spodok zásobníka nový zásobníkový symbol. Keď teraz A' vidí na vrchu zásobníka tento symbol, simulovaný automat A má prázdnu pamäť. Automat A' teda prejde do nového akceptačného stavu. Ak už dočítal vstupné slovo, akceptuje, inak sa zasekne.

3.9 Ekvivalencia bezkontextových gramatík a zásobníkových automatov

V tejto časti vyriešime otázku sily zásobníkových automatov – ukážeme, že sú rovnako silné ako bezkontextové gramatiky.

Veta 3.9.1 *K ľubovoľnej bezkontextovej gramatike $G = (N, T, P, \sigma)$ existuje zásobníkový automat A, ktorý akceptuje $L(G)$ prázdnou pamäťou.*

Dôkaz. Myšlienka: Automat bude obsahovať iba jeden stav, množina zásobníkových symbolov bude $N \cup T$. Budeme nedeterministicky hádať a simulovať ľavé krajné odvodenie vstupného slova w v G, pričom na zásobníku „zhora nadol“ budeme mať nespracovanú časť vetnej formy. (Na začiatku je tam teda neterminál σ.) Automat bude pracovať nasledovne: Ak na vrchu zásobníka uvidí neterminál ξ, vyberie ho a nahradí ho reverzom pravej strany uhádnutého ξ-pravidla. Ak na vrchu zásobníka je terminál, prekáža nám, lebo nemôžeme pokračovať v odvádzaní. Vieme sa ho ale zbaviť. Totiž tento terminál by už zostal na začiatku vetnej formy uloženej v zásobníku až do konca odvodenia. To ale znamená, že sa musí rovnať prvému doteraz neprečítanému písmenu vstupného slova. Ak áno, automat ho vyhodí a pokračuje v odvádzaní ďalej. Ak nie, automat sa zasekne.

Formálne: Bude $A = (K = \{q\}, \Sigma = T, \Gamma = N \cup T, \delta, q, \sigma, F = \emptyset)$, pričom:

$$\begin{aligned} \delta(q, \varepsilon, \xi) &= \{(q, w^R) \mid (\xi \to w) \in P\} \\ \delta(q, x, x) &= \{(q, \varepsilon)\} \quad (\forall x \in T) \end{aligned}$$

Indukciou by sa ľahko dokázalo, že $(q, uv, \sigma) \vdash_A^* (q, v, w)$ práve vtedy, keď $\sigma \underset{G}{\Longrightarrow}^* uw^R$. Z toho už priamo vyplýva rovnosť $N(A) = L(G)$.

Veta 3.9.2 *Nech A je zásobníkový automat. Potom existuje bezkontextová gramatika G taká, že $L(G) = N(A)$.*

Dôkaz. Nech $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ je zásobníkový automat, ktorý akceptuje jazyk L prázdnou pamäťou. Ukážeme, ako zostrojiť bezkontextovú gramatiku generujúcu L.

Potrebujeme vygenerovať množinu všetkých slov, na ktoré mohol v A prebehnúť akceptačný výpočet. Na to ale potrebujeme presnejšie vedieť, ako taký výpočet vyzerá. Pozrime sa na konfiguráciu $\mathcal{C}$ niekde uprostred akceptačného výpočtu. Sme v nejakom stave q_x, na zásobníku máme slovo $Z_1 \ldots Z_k$.[7] V každom kroku výpočtu sa buď veľkosť zásobníka zväčší, alebo zostane rovnaká, alebo sa **práve o jedna** zmenší. Automat ale akceptuje prázdnou pamäťou, teda skôr či neskôr musí všetkých k znakov zo zásobníka „vyžrať“.[8] Ale vyžierať ich môže iba po jednom. Preto sa skôr či neskôr vyskytne konfigurácia $\mathcal{D}$, kde je **prvýkrát** na zásobníku len $k-1$ znakov (a sme v nejakom stave q_y).

Všimnime si kus výpočtu, ktorým automat prešiel z konfigurácie $\mathcal{C}$ do konfigurácie $\mathcal{D}$. Tento istý kus výpočtu môže prebehnúť kedykoľvek, keď sme v stave q_x a na vrchu zásobníka je Z_k. (Totiž k znakom pod Z_k sme sa ešte nedostali.) Výsledok tohto kusu výpočtu bude ten, že Z_k „vyžerieme“ zo zásobníka a skončíme v stave q_y.

A na tomto poznatku založíme konštrukciu našej bezkontextovej gramatiky. Jej neterminály budú trojice $[q_x, Z, q_y]$, kde $q_x, q_y \in K$, $Z \in \Gamma$. Pritom budeme chcieť, aby sa z neterminálu $[q_x, Z, q_y]$ dali odvodiť práve tie slová w, na ktoré mohol prebehnúť vyššie uvedený kus výpočtu – na začiatku sme v stave q_x, na vrchu zásobníku je Z, na konci sme v stave q_y a zásobník je prvýkrát o jedno nižší.

Stav $[q_x, Z, q_y]$ budeme teda čítať „sme v stave q_x, vyžerieme Z a skončíme v q_y“.

Ale čo s pravidlami, odkiaľ sa vezmú? Predsa vzniknú podľa δ-fcie automatu A. Poďme zostrojiť pravidlá, na ktorých ľavej strane bude neterminál $[q_x, Z, q_y]$. Ako mohol vyzerať hľadaný kus výpočtu? V prvom kroku sme museli použiť niektorú z možností, ktoré nám ponúka $\delta(q_x, Z, s)$ pre $s \in \Sigma \cup \{\varepsilon\}$.

Nech napríklad $(q_z, ABC) \in \delta(q_x, Z, a)$. Potom hľadaný kus výpočtu mohol vyzerať nasledovne:

- Prečítali sme a zo vstupu, Z zo zásobníka. Zmenili sme stav na q_z, na zásobník sme postupne vložili A, B, C. Tieto symboly teraz potrebujeme v našom kuse výpočtu zo zásobníka vyžrať.
- Preto nasleduje kus výpočtu, kde začíname v q_z, vyžerieme C a skončíme v nejakom q_1.
- Z q_1 vyžerieme B a skončíme v nejakom q_2.
- No a z q_2 vyžerieme A a skončíme v q_y, v ktorom máme skončiť. (Neterminál, pre ktorý hľadáme pravidlá, je $[q_x, Z, q_y]$, t.j. náš kus výpočtu končí v q_y.)

Preto pre tento „riadok“ δ-fcie dostávame v zostrojovanej gramatike sadu pravidiel

$$[q_x, Z, q_y] \;\to\; a[q_z, C, q_1][q_1, B, q_2][q_2, A, q_y] \qquad (\forall q_1, q_2 \in K)$$

(Chceme vygenerovať všetky slová, na ktoré sme z q_x mohli vyžrať Z a skončiť v q_y. V prvom kroku odvodenia sa vlastne rozhodneme, ako vyzeral prvý krok výpočtu a cez ktoré stavy sme išli pri vyžieraní toho, čo nám ten prvý krok nahádzal na zásobník.)

Špeciálne ak napr. $(q_z, \varepsilon) \in \delta(q_x, Z, a)$, vieme Z vyžrať priamo na jeden krok, pri ktorom čítame a a skončíme v q_z. Preto nám v gramatike pribudne pravidlo $[q_x, Z, q_z] \to a$.

[7] Vrch zásobníka je napravo, teda na vrchu je Z_k.

[8] Slovo *vyžrať* je natoľko názorné, že v ďalšom texte ho budeme používať bez ostychu a úvodzoviek.

Jazyk L sú vlastne tie slová, pre ktoré vieme v stave q_0 vyžrať zo zásobníka symbol Z_0 a skončiť v ľubovoľnom stave. Preto si pridáme do našej gramatiky počiatočný neterminál σ a pravidlá $\sigma \to [q_0, Z_0, q]$, kde $q \in K$. V prvom kroku odvodenia sa teda rozhodneme, v akom stave skončíme a následne vygenerujeme slovo, na ktoré môže zodpovedajúci kus výpočtu prebehnúť.

Formálna konštrukcia: Nech $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$. Potom zostrojíme $G = (N, T, P, \sigma)$, kde $N = (K \times \Gamma \times K) \cup \{\sigma\}$, $T = \Sigma$ a

$$\begin{aligned} P \;=\; & \Big\{ [q,Z,p] \to x[r,Z_k,q_1][q_1,Z_{k-1},q_2]\dots[q_{k-1},Z_1,p] \\ & \quad \Big| \;\; x \in (\Sigma \cup \{\varepsilon\}) \;\wedge\; k \geq 2 \;\wedge\; (r, Z_1 \dots Z_k) \in \delta(q,x,Z) \;\wedge\; p, q_i \in K \Big\} \\ \cup \; & \Big\{ [q,Z,p] \to x[r,Z_1,p] \;\Big|\; x \in (\Sigma \cup \{\varepsilon\}) \;\wedge\; (r,Z_1) \in \delta(q,x,Z) \;\wedge\; p \in K \Big\} \\ \cup \; & \Big\{ [q,Z,r] \to x \;\Big|\; x \in (\Sigma \cup \{\varepsilon\}) \;\wedge\; (r,\varepsilon) \in \delta(q,x,Z) \Big\} \end{aligned}$$

3.10 Uzáverové vlastnosti bezkontextových jazykov

V tomto odseku sa budeme venovať otázkam uzavretosti triedy bezkontextových jazykov na základné operácie. Uvedomme si, že pri dôkazoch môžeme využívať ako bezkontextové gramatiky, tak aj zásobníkové automaty. Zopakujme si, že triedu všetkých bezkontextových jazykov označujeme $\mathscr{L}_{CF}$.

Veta 3.10.1 *$\mathscr{L}_{CF}$ je uzavretá na zjednotenie.*

Dôkaz. Analogicky ako vo vete 2.6.1 pre regulárne jazyky. Keď máme bezkontextové gramatiky pre jazyky L_1, L_2 (s disjunktnými množinami neterminálov a so začiatočnými neterminálmi σ_1, σ_2), bezkontextovú gramatiku pre ich zjednotenie zostrojíme tak, že pridáme nový začiatočný neterminál σ, v ktorom sa rozhodneme, či vygenerujeme slovo z prvého alebo druhého jazyka. (Teda pribudnú nám pravidlá $(\sigma \to \sigma_1 \mid \sigma_2)$.)

Veta 3.10.2 *$\mathscr{L}_{CF}$ je uzavretá na zreťazenie.*

Dôkaz. Myšlienka je rovnaká ako pri zjednotení, len namiesto pravidla $(\sigma \to \sigma_1 \mid \sigma_2)$ pridáme pravidlo $(\sigma \to \sigma_1\sigma_2)$.

Veta 3.10.3 *$\mathscr{L}_{CF}$ je uzavretá na iteráciu.*

Dôkaz. Zostávame pri „dôkazoch na jedno kopyto“. Nech bezkontextová gramatika pre pôvodný jazyk je $G = (N, T, P, \sigma)$. Pridáme nový začiatočný neterminál ξ a pravidlá $\xi \to \sigma\xi \mid \varepsilon$.

Lema 3.10.4 *Jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$ nie je bezkontextový.*

Poznámka 3.10.1 Dopustíme sa prehrešku voči matematickej kultúre a toto tvrdenie dokážeme až v časti 3.11, keď budeme mať na to dostatočný aparát.

Veta 3.10.5 *$\mathscr{L}_{CF}$ nie je uzavretá na prienik.*

Dôkaz. Jazyky $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$ a $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$ sú zjavne bezkontextové, ich prienik však bezkontextový nie je.

Veta 3.10.6 *$\mathscr{L}_{CF}$ je uzavretá na prienik s regulárnym jazykom.*

Dôkaz. Jeden pohľad na to, kde bol problém pri uzavretosti $\mathscr{L}_{CF}$ na prienik je nasledovný: Zásobníkový automat nedokáže (pomocou jedného zásobníka) simulovať dva zásobníkové automaty. Keď robíme prienik bezkontextového a regulárneho jazyka, simulované automaty už majú dokopy len jeden zásobník. Preto simulácii už nič nestojí v ceste. Konštrukcia bude podobná ako pri prieniku dvoch regulárnych jazykov, iba navyše musíme simulovať aj prácu so zásobníkom.

Formálne: Nech $A_1 = (K_1, \Sigma, \Gamma_1, \delta_1, q_{01}, F_1$ je PDA, $A_2 = (K_2, \Sigma, \delta_2, q_{02}, F_2)$ je DKA. Zostrojíme PDA A taký, že $L(A) = L(A_1) \cap L(A_2)$. Bude $A = (K, \Sigma, \Gamma_1, \delta, [q_{01}, q_{02}], F)$, kde:

$$\begin{aligned} K &= K_1 \times K_2 \\ F &= F_1 \times F_2 \\ \delta([q_1, q_2], x, Z) &= \{([p_1, p_2], w) \mid p_2 = \delta_2(q_2, x) \ \wedge \ (p_1, w) \in \delta_1(q_1, x, Z)\} \quad (\forall x \in \Sigma) \\ \delta([q_1, q_2], \varepsilon, Z) &= \{([p_1, q_2], w) \mid (p_1, w) \in \delta_1(q_1, \varepsilon, Z)\} \end{aligned}$$

Dôkaz, že naozaj $L(A) = L(A_1) \cap L(A_2)$ je triviálny – k akceptačnému výpočtu A ľahko zostrojíme zodpovedajúce akceptačné výpočty v oboch A_i a naopak.

Veta 3.10.7 *$\mathscr{L}_{CF}$ nie je uzavretá na komplement.*

Dôkaz. Sporom. Nech je uzavretá na komplement. Vieme, že je uzavretá aj na zjednotenie. Potom ale vďaka de Morganovym zákonom ($L_1 \cap L_2 = (L_1^C \cup L_2^C)^C$) je uzavretá aj na prienik, čo je spor.

Príklad 3.10.1 Uvedieme aj jeden (možno prekvapivý) príklad. Z lemy 3.10.4 vieme, že jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$ nie je bezkontextový. Jeho komplement však bezkontextový je. (Rozmyslite si, prečo. Hint: Nedeterministicky si tipneme, prečo vstupné slovo nepatrí do L.) Našli sme teda bezkontextový jazyk (L^C), ktorého komplement nie je bezkontextový.

Veta 3.10.8 *$\mathscr{L}_{CF}$ je uzavretá na homomorfizmus.*

Dôkaz. Rovnako ako u regulárnych jazykov – keď máme gramatiku G a homomorfizmus h, zostrojíme G' tak, že každý výskyt terminálu x v pravidlách nahradíme slovom $h(x)$. Zjavne bude $L(G') = h(L(G))$.

Veta 3.10.9 *$\mathscr{L}_{CF}$ je uzavretá na inverzný homomorfizmus.*

Dôkaz. Opäť zafunguje rovnaký postup ako u regulárnych jazykov. Nech L je bezkontextový jazyk, A PDA, ktorý ho akceptuje, h homomorfizmus. Potom PDA A' pre $h^{-1}(L)$ bude vyzerať nasledovne:

A' má o každom slove w, ktoré dostane na vstupe, zistiť, či $h(w) \in L$. Nech $k = \max_{x \in \Sigma_2} |h(x)|$. Každé písmeno w nám teda h zobrazí na najviac k písmen. Náš automat si bude pamätať v stave slovo konečnej veľkosti $\leq k$. Vždy, keď prečíta písmeno zo vstupu, naplní si túto pamäť obrazom prečítaného písmena. Na písmenách tohto obrazu simuluje A (bez toho, aby čítal zo vstupu). Keď sa mu pamäť vyprázdni, prečíta ďalšie písmeno zo vstupu (zásobník pri tomto kroku nezmení).

Veta 3.10.10 *$\mathscr{L}_{CF}$ je uzavretá na reverz.*

Dôkaz. Keď chceme zostrojiť gramatiku pre reverz bezkontextového jazyka L, stačí zobrať gramatiku pre L a reverznúť v nej všetky pravé strany pravidiel.

Poznámka 3.10.2 Uvedomte si, že pri regulárnej gramatike sme si toto nemohli dovoliť, výsledná gramatika by nebola regulárna.

3.11 Pumpovacia lema pre bezkontextové jazyky

Poznámka 3.11.1 Táto veta sa v literatúre tiež označuje ako $p - q$-lema, alebo tiež lema o vkladaní. Podobne ako pri pumpovacej leme pre regulárne jazyky pôjde o nástroj, pomocou ktorého budeme o niektorých jazykoch vedieť ukázať, že nie sú bezkontextové.

Veta 3.11.1 *(pumpovacia lema) K ľubovoľnému $L \in \mathscr{L}_{CF}$ existujú čísla p, q také, že pre každé $w \in L$ také, že $|w| > p$ existujú u, v, x, y, z také, že*

1. *$w = uvxyz$*

2. $|vxy| \leq q$
3. $|vy| \geq 1$
4. $\forall i \geq 0;\ uv^i xy^i z \in L$

Poznámka 3.11.2 Všimnite si, že ak leme vyhovujú hodnoty p, q, tak vyhovujú aj hodnoty r, r, kde $r = \max(p, q)$. Mohli sme teda aj túto lemu vysloviť s jednou premennou (podobne ako u regulárnych jazykov). Z historických dôvodov sa ale zvykne uvádzať v tejto podobe.

Dôkaz. Nech G je bezkontextová gramatika generujúca L. BUNV predpokladajme, že G je ε-free a neobsahuje chain rules. Nech dĺžka najdlhšej pravej strany pravidla v G je k. Potom každý strom odvodenia, v ktorom najhlbší neterminál je v hĺbke najviac h, je stromom odvodenia nejakého slova dĺžky najviac k^{h+1}. Inými slovami, keď zoberieme $p = k^{|N|}$, tak (každý) strom odvodenia každého slova z $L(G)$ dlhšieho ako p má nejaký neterminál v hĺbke aspoň $|N|$.

Majme teraz pevné $w \in L(G)$, $|w| > p$ a pevný strom jeho odvodenia. Zoberme v ňom ľubovoľnú najdlhšiu cestu z koreňa do listu. Poďme po nej odspodu. Vďaka tomu, že obsahuje aspoň $|N|+1$ neterminálov a Dirichletovmu princípu sa nám skôr či neskôr niektorý neterminál zopakuje. Nech ξ je prvý takýto neterminál.

Vynechajme zo stromu odvodenia podstrom A s koreňom vo vyššom z nájdených dvoch výskytov ξ. Dostávame odvodenie $\sigma \Rightarrow^* u\xi z$. Podobne keď zo stromu A vynecháme podstrom B s koreňom v nižšom výskyte x, dostávame odvodenie $\xi \Rightarrow^* v\xi y$. Podstrom B zodpovedá odvodeniu $\xi \Rightarrow^* x$.

Teraz ale vieme zostrojiť odvodenia nových slov z $L(G)$. Časť odvodenia $\xi \Rightarrow^* v\xi y$ totiž môžeme použiť ľubovoľne veľa krát, čím vygenerujeme niekoľko kópií slov v a y. Formálne teda:

$$\sigma \Rightarrow^* u\xi z \underbrace{\Rightarrow^* uv\xi yz \Rightarrow^* \ldots \Rightarrow^*}_{i \text{ krát}} uv^i\xi y^i z \Rightarrow^* uv^i xy^i z$$

Vďaka tomu, že G je ε-free a neobsahuje chain rules, slová v a y nemôžu byť obe prázdne. Najhlbší neterminál v strome A je v hĺbke najviac $|N|$, preto dĺžka slova vxy je najviac $q = k^{|N|+1}$.

Príklad 3.11.1 Ukážeme, že jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$ nie je bezkontextový.

Sporom. Nech teda existujú p, q z pumpovacej lemy. Zoberme slovo $w = a^{p+q+7}b^{p+q+7}c^{p+q+7}$. Toto slovo je určite dlhšie ako p. Ale v ľubovoľnom prípustnom rozdelení w na u, v, x, y, z musí byť $|vxy| \leq q$. To ale znamená, že vxy nemôže naraz obsahovať znaky a, b aj c. Potom ale v slove uv^2xy^2z nemôžu byť rovnaké počty a, b a c, a teda $uv^2xy^2z \notin L$, čo je spor.

Príklad 3.11.2 Ukážeme, že jazyk $L = \{w\#w \mid w \in \{a, b\}^*\}$ nie je bezkontextový.

Sporom. Nech teda existujú p, q z pumpovacej lemy. Čitateľ ľahko overí, že stačí zobrať slovo $w = a^{p+q+4}b^{p+q+4}\#a^{p+q+4}b^{p+q+4}$. To spĺňa podmienku z pumpovacej lemy, ale nedá sa nijako rozdeliť a napumpovať, čo je hľadaný spor.

Príklad 3.11.3 Ukážeme, že jazyk $L = \{w \mid w \in \{a, b, c\}^* \ \wedge\ \#_a(w) = \#_b(w) = \#_c(w)\}$ nie je bezkontextový.

Opäť sporom. Nech L je bezkontextový. Trieda bezkontextových jazykov je uzavretá na prienik s regulárnym jazykom, preto musí byť bezkontextový aj jazyk $L' = L \cap \{a^i b^j c^k \mid i, j, k \geq 0\}$. Ale zjavne jazyk $L' = \{a^n b^n c^n \mid n \geq 0\}$ bezkontextový nie je, čo je spor.

Poznámka 3.11.3 Rovnako ako pri regulárnych jazykoch je táto pumpovacia lema len nutnou podmienkou bezkontextovosti jazyka, **nie** postačujúcou. Existujú nebezkontextové jazyky, ktoré podmienky z lemy spĺňajú.

Jedným z nedostatkov je skutočnosť, že niektoré nebezkontextové jazyky majú časť, ktorá je bezkontextová, a teda sa dá pumpovať. Tento problém sčasti rieši silnejšia Ogdenova lema, ktorá nám umožní približne špecifikovať, kde v slove sa má pumpovaná časť nachádzať.

Príklad 3.11.4 Zostrojíme jazyk, ktorý nie je bezkontextový, ale spĺňa podmienky z pumpovacej lemy. Vieme, že jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$ nie je bezkontextový. Zoberme teda jazyk $L' =$

$\{a^n b^n c^n d^m \mid n \geq 0, m \geq 1\}$, ktorý zjavne tiež nie je bezkontextový. Teraz v pumpovacej leme stačí pre ľubovoľné slovo w zobrať $u = a^n b^n c^n$, $v = d$, $x = y = \varepsilon$ a z rovné zvyšku slova. Pridaním d-čok dostaneme opäť slovo z nášho jazyka. Každé slovo teda vieme napumpovať. Alebo nie?

Kde je problém? My sme zatiaľ ukázali, že každé slovo vieme napumpovať na i-tu, kde $i > 0$. Problém je v tom, že aj slovo $uv^0xy^0z = uxz$ musí patriť do nášho jazyka. A tu s nepríjemným prekvapením zistíme, že slovo $a^n b^n c^n d$ „na nultú“ napumpovať nevieme. Tento drobný nedostatok musíme nejako zaplátať.

Zoberme trochu zložitejší jazyk $L'' = L' \cup \{w \mid w \in \{a, b, c\}^*\}$. Ľubovoľné slovo z L' vieme napumpovať postupom popísaným vyššie, pričom slová tvaru $a^n b^n c^n d$ napumpované „na nultú“ teraz padnú do novej časti jazyka. A ľubovoľné slovo z novej časti jazyka zjavne zostane v tejto časti jazyka, nech ho rozdelíme a napumpujeme akokoľvek (určite dostaneme opäť nejaké slovo obsahujúce len písmená a, b, c). Pre tento jazyk teda platí pumpovacia lema.

Na druhej strane, tento jazyk nie je bezkontextový. Vieme, že trieda bezkontextových jazykov je uzavretá na prienik s reg. jazykom. Keby teda bol L'' bezkontextový, bol by bezkontextový aj jazyk $L'' \cap \{a^i b^j c^k d \mid i, j, k \geq 0\} = \{a^n b^n c^n d \mid n \geq 0\}$. Tento jazyk však bezkontextový zjavne nie je. Preto ani L'' nie je bezkontextový, q.e.d.

Veta 3.11.2 *(Ogdenova lema) Pre ľubovoľný bezkontextový jazyk L existuje číslo n také, že pre každé $w \in L$, $|w| > n$ a pre každé označenie n alebo viac symbolov v slove w existujú u, v, x, y, z také, že*

1. $w = uvxyz$
2. *vxy obsahuje najviac n označených symbolov*
3. *vy obsahuje aspoň 1 označený symbol*
4. $\forall i \geq 0;\ uv^i xy^i z \in L$

Dôkaz. Myšlienka je podobná ako pri dôkaze pumpovacej lemy. Uvedieme preto len odlišné miesta dôkazu. 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 každý vetviaci bod má práve dvoch synov, existuje cesta z koreňa do niektorého listu, na ktorej leží aspoň $\log_2 n$ vetviacich bodov. Ak by sme zvolili $2^{|N|+1}$, potom na tejto ceste budú existovať dva rovnaké neterminály vo vetviacich bodoch. Ďalej postupujeme rovnako ako v dôkaze pumpovacej lemy.

Príklad 3.11.5 Zoberme jazyk, o ktorom sme už ukázali, že pumpovaciu lemu spĺňa:

$$L = \{a^n b^n c^n d^m \mid n, m \geq 0\} \cup \{w \mid w \in \{a, b, c\}^*\}$$

Pomocou Ogdenovej lemy ukážeme, že tento jazyk nie je bezkontextový.

Sporom. Nech existuje n. Zoberme slovo $a^n b^n c^n d^n$ a označme v ňom všetky písmená a. Ukážeme, že toto slovo nevieme prípustne rozdeliť a napumpovať na druhú, čo bude hľadaný spor. Totiž: Zjavne ak bude niektorá z pumpovaných častí obsahovať dve rôzne písmená, po napumpovaní na druhú „sa nám písmená pomiešajú“ a zjavne dostaneme slovo, ktoré nepatrí do L. Každé z dvoch pumpovaných podslov teda musí byť tvorené len rovnakými písmenami. Pumpovaná časť musí obsahovať aspoň jeden označený symbol a, a teda jedno z dvoch pumpovaných slov je tvorené samými písmenami a. Preto vo w napumpovanom na druhú bude symbolov a viac ako n. Nemáme ale ako zabezpečiť, aby aj symbolov b a c bolo rovnako, a teda ani teraz napumpované slovo nebude patriť do L.

Všimnite si, že na rozdiel od pumpovacej lemy (kde vznikol problém, lebo sa dali pumpovať symboly d) sme vhodným označením symbolov „donútili lemu“ pumpovať tú časť slov, vďaka ktorej jazyk nie je bezkontextový.

Poznámka 3.11.4 Všimnite si, že pumpovacia lema je špeciálnym prípadom Ogdenovej lemy. Dostaneme ju, ak vo vybranom slove označíme všetky symboly.

3.12 Zistenie príslušnosti slova do bezkontextového jazyka

Bezkontextovými gramatikami často špecifikujeme syntax programovacích jazykov. Pri kompilácií programov napísaných v týchto jazykoch je jednou z najdôležitejších otázok efektívne *parsovanie* – pre dané slovo (=program) zistiť, či patrí do jazyka všetkých korektných programov a ak áno, tak nájsť jeho jedno možné odvodenie.

Uvedieme algoritmus Cockeho, Youngera a Kasamiho (známy ako „algoritmus CYK“), ktorým vieme[9] zistiť, či dané slovo w patrí do jazyka generovaného danou gramatikou G. Tento algoritmus používa myšlienku *dynamického programovania*.

Ak $w = \varepsilon$, iba zistíme, či je σ v množine vymazávajúcich neterminálov a skončili sme. Nech teda $w = w_1 \dots w_n$, kde $n > 0$. Gramatiku G prevedieme do prísneho Chomského normálneho tvaru. Budeme postupne vo vhodnom poradí konštruovať množiny neterminálov $N_{i,j}$ (kde $1 \leq i \leq j \leq n$) také, že

$$\xi \in N_{i,j} \iff \xi \Rightarrow^* w_i \dots w_j$$

Pritom pri konštrukcii $N_{i,j}$ budeme využívať skôr vytvorené množiny.

Množiny $N_{i,j}$ budeme zostrojovať usporiadané podľa hodnoty $(j - i)$, t.j. neskôr zostrojené množiny budú zodpovedať dlhším podslovám slova w.

Vďaka tvaru gramatiky G vieme ľahko zostrojiť každú množinu $N_{i,i}$ – je to množina tých ξ, pre ktoré v G existuje pravidlo $\xi \to w_i$.

Ako teraz zostrojiť množinu $N_{i,j}$, ak už poznáme množiny zodpovedajúce všetkým kratším podslovám? Chceme nájsť všetky neterminály ξ, z ktorých sa dá odvodiť slovo $w_i \dots w_j$. V prvom kroku tohto odvodenia určite použijeme nejaké pravidlo $\xi \to \beta\gamma$, čím dostaneme dva neterminály. Následne z prvého odvodíme $w_i \dots w_k$ (pre vhodné k) a z druhého $w_{k+1} \dots w_j$. Preto:

$$N_{i,j} = \left\{ \xi \;\middle|\; \exists k \in \{i, \dots, j-1\},\ \beta \in N_{i,k},\ \gamma \in N_{k+1,j};\ (\xi \to \beta\gamma) \in P \right\}$$

Slovo w patrí do $L(G)$ iff množina $N_{1,n}$ obsahuje začiatočný neterminál σ. Algoritmus sa ľahko dá upraviť tak, aby v prípade $w \in L(G)$ našiel aj jeden strom odvodenia slova w.

[9] v polynomiálnom čase od veľkosti vstupu

Kapitola 4

Zložitejšie modely

V tejto kapitole si ukážeme dva zložitejšie modely a rozoberieme ich silu. Prvým bude a-prekladač ako zovšeobecnenie konečného automatu, druhým deterministický zásobníkový automat.

4.1 A-prekladače

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 druhú pásku.

Definícia 4.1.1 ***A-prekladač*** *(konečne stavový prekladač s akceptačnými stavmi) je 6-tica* $M = (K, \Sigma_1, \Sigma_2, H, q_0, F)$*, kde* K *je konečná množina stavov,* Σ_1 *je konečná vstupná abeceda,* Σ_2 *je konečná výstupná abeceda,* q_0 *je začiatočný stav automatu,* F *je množina akceptačných stavov a* $H \subseteq_{kon} K \times \Sigma_1^* \times \Sigma_2^* \times K$ *je prechodová funkcia. (Štvorica* $(q, u, v, p) \in H$ *bude znamenať: „ak sme v stave* q *a prečítame slovo* u*, môžeme zapísať slovo* v *a zmeniť stav na* p*“.)*

Poznámka 4.1.1 A-prekladač zadefinujeme ďalej rovnakým spôsobom, ako sme definovali konečné automaty. Ukážeme potom iný spôsob definície, pomocou homomorfizmov, inverzných homomorfizmov a prienikov s regulárnymi jazykmi.

Definícia 4.1.2 ***Konfiguráciou*** *a-prekladača nazývame usporiadanú trojicu* (q, u, v)*, kde* $q \in K$ *je aktuálny stav,* $u \in \Sigma_1^*$ *neprečítaná časť vstupného slova a* $v \in \Sigma_2^*$ *už zapísaná časť výstupného slova.*

Definícia 4.1.3 ***Krokom výpočtu*** *a-prekladača nazývame reláciu* $\vdash$ *definovanú na konfiguráciách takto:*

$$(q, xu, v) \vdash (p, u, vy) \quad \Longleftrightarrow \quad (q, x, y, p) \in H$$

Poznámka 4.1.2 Môžeme si všimnúť, že čítanie vstupu je podobné ako pri automate s pružnou hlavou. A-prekladač dokáže naraz prečítať celý úsek slova.

Definícia 4.1.4 ***Obrazom jazyka*** L *a-prekladačom* M *nazývame množinu*
$M(L) = \{w \mid \exists u \in L;\ q_F \in F;\ (q_0, u, \varepsilon) \vdash_M^* (q_F, \varepsilon, w)\}$

Teraz prichádza ten sľubovaný okamih, keď si ukážeme inú definíciu výpočtu a obrazu v zobrazení a-prekladačom pomocou už spomínaných homomorfizmov, ktoré nám uľahčia dôkaz niektorých tvrdení o a-prekladačoch. Čitateľ môže pouvažovať, prečo sú dané definície ekvivalentné.

Poznámka 4.1.3 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 ekvivalencie DKA a NKA, kedy stav jedného automatu predstavoval množinu stavov iného.

Definícia 4.1.5 *Homomorfizmy pr_i, $i \in \{1,2,3,4\}$ z množiny H^* také, že $pr_i((x_1,x_2,x_3,x_4)) = x_i$ nazývame **projekcie**. Jednotlivé projekcie majú svoje obrazy postupne v množinách K^*, Σ_1^*, Σ_2^* a K^*.*

Definícia 4.1.6 ***Výpočtom*** *a-prekladača M nazývame postupnosť $h_1, h_2, \ldots, h_k$ štvoríc z H takú, že platí:*

1. $pr_1(h_1) = q_0$
2. $\forall i;\ pr_4(h_i) = pr_1(h_{i+1})$

*Slovne: h_i je postupnosť „riadkov" prechodovej funkcie A, ktoré mohli byť postupne použité počas akceptačného výpočtu. Ak navyše $pr_4(h_k) \in F$, hovoríme, že výpočet je **akceptačný**.*

Definícia 4.1.7 ***Jazyk akceptačných výpočtov*** *a-prekladača M je jazyk* $\Pi_M = \{w \mid w \in H^* \ \wedge\ w$ je akc. výpočet$\}$.

Lema 4.1.1 *Pre ľubovoľný a-prekladač M je jazyk Π_M jeho akceptačných výpočtov regulárny.*

Definícia 4.1.8 ***Zobrazením*** *jazyka L a-prekladačom M nazývame množinu* $M(L) = pr_3(pr_2^{-1}(L) \cap \Pi_M)$.

Poznámka 4.1.4 Vysvetlíme túto definíciu: Nech $h \in H$. Na h sa môžeme dívať ako na krok výpočtu. Potom ale $pr_2(h)$ je časť slova, ktorú počas tohto kroku prečítame a $pr_3(h)$ je časť slova, ktorú zapíšeme. Keď teraz zoberieme nejaký výpočet $v \in H^*$, tak analogicky počas neho prečítame slovo $pr_2(v)$ a zapíšeme slovo $pr_3(v)$.

Teda $pr_2^{-1}(L)$ sú všetky postupnosti „riadkov"[1] z H, počas ktorých sa dokopy prečítalo slovo z L. Z nich prienikom s Π_M vyberieme akceptačné výpočty a-prekladača M. (Teda ten prienik zabezpečí, aby na seba nadväzovali stavy v po sebe idúcich krokoch, prvý stav bol začiatočný a posledný akceptačný.) Zaujíma nás tretia projekcia výsledku – slová, ktoré M počas týchto výpočtov zapísal.

4.1.1 Zobrazenia a-prekladačom

Veta 4.1.2 *Triedy $\mathcal{R}$ a $\mathscr{L}_{CF}$ sú uzavreté na zobrazenie a-prekladačom.*

Dôkaz. Tvrdenie vyplýva zo skutočnosti, že (podľa alternatívnej definície) vieme každé zobrazenie a-prekladačom poskladať z vhodného inverzného homomorfizmu, prieniku s regulárnym jazykom a homomorfizmu. Obe triedy sú na tieto operácie uzavreté.

Veta 4.1.3 *Ku každému homomorfizmu $h : \Sigma_1^* \to \Sigma_2^*$ existuje a-prekladač M taký, že $\forall L \subseteq \Sigma_1^*;\ M(L) = h(L)$.*

Dôkaz. Bude $M = (K = \{q\}, \Sigma_1, \Sigma_2, H, q, F = \{q\})$, kde $H = \{(q, x, h(x), q) \mid x \in \Sigma_1\}$.

Veta 4.1.4 *Ku každému homomorfizmu $h : \Sigma_1^* \to \Sigma_2^*$ existuje a-prekladač M taký, že $\forall L \subseteq \Sigma_2^*;\ M(L) = h^{-1}(L)$.*

Dôkaz. Bude $M = (K = \{q\}, \Sigma_2, \Sigma_1, H, q, F = \{q\})$, kde $H = \{(q, h(x), x, q) \mid x \in \Sigma_1\}$.

Veta 4.1.5 *Ku každému regulárnemu jazyku R existuje a-prekladač M taký, že $\forall L \subseteq \Sigma_R^*;\ M(L) = L \cap R$.*

Dôkaz. Nech $A = (K, \Sigma_R, \delta, q_0, F)$ je DKA taký, že $L(A) = R$. Bude $M = (K, \Sigma_R, \Sigma_R, H, q_0, F)$, kde $H = \{(q, x, x, \delta(q,x)) \mid q \in K,\ x \in \Sigma_R\}$. Automat M kopíruje vstupné slovo na výstup a akceptuje len vtedy, ak akceptoval A.

[1] Pozor, nemusí ešte ísť o výpočet!

4.1.2 Normálny tvar a-prekladača

Veta 4.1.6 *Ku každému a-prekladaču M existuje ekvivalentný, ktorého pravidlá sú podmnožinou $K \times (\Sigma_1 \cup \{\varepsilon\}) \times (\Sigma_2 \cup \{\varepsilon\}) \times K$.*

Dôkaz. Uvedieme len myšlienku: Pre každé $h \in H_M$ zavedieme nové stavy, pomocou ktorých najskôr po jednom písmene načítame $pr_2(h)$, potom po jednom písmene zapíšeme $pr_3(h)$.

Poznámka 4.1.5 S využitím tohto normálneho tvaru a-prekladača sa dá ľahšie dokázať, že zobrazenia a-prekladačmi sú uzavreté na skladanie. Toto tvrdenie ale presahuje rámec nášho textu, preto ho tu nebudeme uvádzať.

4.1.3 Použitie a-prekladačov

Keďže triedy $\mathcal{R}$ aj $\mathscr{L}_{CF}$ sú uzavreté na zobrazenie a-prekladačom, môžeme a-prekladač použiť pri dôkaze, že daný jazyk L nie je regulárny, resp. bezkontextový – nájdeme vhodný a-prekladač M a ukážeme o (jednoduchšom) jazyku $M(L)$, že nie je regulárny, resp. bezkontextový.

Príklad 4.1.1 Ukážeme, že jazyk $L = \{a^i b^i c^i d^j \mid i, j > 0\} \cup \{a, b, c\}^*$ nie je bezkontextový. Sporom. Keby bol, potom je bezkontextový aj jazyk $M(L)$, kde:

$$\begin{aligned} M &= (K, \{a,b,c,d\}, \{a,b,c\}, H, q_a, \{q_d\}) \\ K &= \{q_a, q_b, q_c, q_d\} \\ H &= \{\ (q_a, a, a, q_a), \\ &\quad\ (q_a, \varepsilon, \varepsilon, q_b), \\ &\quad\ (q_b, b, b, q_b), \\ &\quad\ (q_b, \varepsilon, \varepsilon, q_c), \\ &\quad\ (q_c, c, c, q_c), \\ &\quad\ (q_c, d, \varepsilon, q_d)\ \} \end{aligned}$$

Ale ľahko nahliadneme, že $M(L) = \{a^i b^i c^i \mid i > 0\}$, čo nie je bezkontextový jazyk.

4.2 Deterministické zásobníkové automaty

Kým v prípade konečných automatov bola sila nedeterministických a deterministických automatov rovnaká, v tomto prípade sa bude sila nedeterministických a deterministických automatov líšiť. Taktiež ukážeme, že na rozdiel od nedeterministických PDA nie je jedno, či definujeme akceptáciu stavom alebo prázdnou pamäťou.

Významom tohto modelu je jeho ľahká implementácia na počítači. Podrobnejšie sa využitím deterministických zásobníkových automatov budeme zaoberať v kapitole 8, kde uvádzame prehľad základných pojmov z oblasti syntaktickej analýzy.

Definícia 4.2.1 ***Deterministickým zásobníkovým automatom*** *(DPDA) nazveme 7-icu $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde význam všetkých symbolov je rovnaký ako u nedeterministických PDA, $\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to (K \times \Gamma^*) \cup \{\emptyset\}$ je prechodová funkcia, spĺňajúca:*
$\forall q \in K,\ \forall Z \in \Gamma;\ \delta(q, \varepsilon, Z) \neq \emptyset \Rightarrow (\forall x \in \Sigma;\ \delta(q, x, Z) = \emptyset)$

Poznámka 4.2.1 Všimnite si, že DPDA sme definovali tak, aby mal možnosť robiť kroky na ε (a teda napr. prečítať spolu s jedným písmenom zo vstupu viac písmen zo zásobníka). Navyše δ-funkcia môže byť pre niektoré dvojice $(stav, symbol\ z\ \Gamma)$ „nedefinovaná“ (vracať $\emptyset$).

Aby ale takto definovaný automat bol deterministický, krok DPDA na ε povolíme len vtedy, ak v danej situácii nemá možnosť spraviť krok na žiadne písmeno. Automat si teda v žiadnom okamihu nemôže vybrať – podľa stavu a symbolu na vrchu zásobníka buď vstup číta alebo nie.

Definícia 4.2.2 ***Konfiguráciou*** *deterministického zásobníkového automatu A rozumieme trojicu (q, w, s), kde $q \in K$ je aktuálny stav, $w \in \Sigma^*$ nedočítaná časť vstupu a $s \in \Gamma^*$ slovo na zásobníku.*

Definícia 4.2.3 ***Krokom výpočtu*** *deterministického zásobníkového automatu A nazveme reláciu $\vdash_A$ definovanú nasledovne:*

$$(q, aw, sZ) \vdash_A (p, w, s\gamma) \iff \delta(q, a, z) = (p, \gamma)$$

Definícia 4.2.4 ***Jazykom*** *rozpoznávaným deterministickým zásobníkovým automatom A nazývame množinu*

$$L(A) = \{w \mid \exists q_F \in F, \gamma \in \Gamma^*;\ (q_0, w, Z_0) \vdash_A^* (q_F, \varepsilon, \gamma)\}$$

Príklad 4.2.1 Zostrojíme DPDA akceptujúci jazyk $L = \{w \mid w \in \{a, b\}^* \ \wedge \ \#_a(w) = \#_b(w)\}$.

Jediný rozdiel oproti nedeterministickému PDA je ten, že v okamihu, keď je zásobník „skoro prázdny“ sa nemôžeme nedeterministicky rozhodnúť, že prejdeme do akceptačného stavu a skončiť. V akceptačnom stave musíme byť vždy, keď sme dovtedy prečítali rovnako písmen a a b. Vyriešime to tak, že keď nie sme v akceptačnom stave a zo zásobníka čítame „zarážku“, pomocou kroku na ε prejdeme do akceptačného stavu. Pri čítaní ďalšieho symbolu zo vstupu zmeníme stav späť na neakceptačný.

Formálna konštrukcia (konvencia: neuvedené riadky δ vracajú $\emptyset$):

$$\begin{aligned}
A &= (K = \{q, p\}, \Sigma = \{a, b\}, \Gamma = \{Z, a, b\}, \delta, q, Z, \{p\}) \\
\delta(q, \varepsilon, Z) &= (p, Z) \\
\delta(p, x, Z) &= (q, Zx) \ \ (\forall x \in \{a, b\}) \\
\delta(q, x, x) &= (q, xx) \ \ (\forall x \in \{a, b\}) \\
\delta(q, x, y) &= (q, \varepsilon) \ \ (\forall x, y \in \{a, b\},\ x \neq y)
\end{aligned}$$

Definícia 4.2.5 ***Jazykom*** *rozpoznávaným deterministickým zásobníkovým automatom A* ***prázdnou pamäťou*** *nazývame množinu*

$$N(A) = \{w \mid \exists q \in K;\ (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \varepsilon)\}$$

Poznámka 4.2.2 Triedu jazykov, ktoré dokážu akceptovať deterministické zásobníkové automaty, označujeme $\mathscr{L}_{DPDA}$. Triedu jazykov, ktoré dokážu akceptovať deterministické zásobníkové automaty prázdnou pamäťou, si označíme $\mathscr{L}_{eDPDA}$. V ďalšom texte ukážeme, že $\mathscr{L}_{eDPDA}$ má pomerne nepríjemné vlastnosti a je **vlastnou** podmnožinou $\mathscr{L}_{DPDA}$.

4.2.1 Normálny tvar pre DPDA

Lema 4.2.1 *Ku každému deterministickému zásobníkovému automatu A existuje DPDA A' taký, že $L(A) = L(A')$ a A' vždy dočíta vstupné slovo a zastane.*

Dôkaz. A' bude samozrejme simulovať A. Ak nastane jedna z dvoch možných „problémových situácií“, jednoducho dočíta vstup a neakceptuje ho. Jednoduchší problém je ten, že A sa zasekne s prázdnym zásobníkom. Toto vieme ľahko zistiť tak, že A' si na začiatku na spodok zásobníka vloží novú „zarážku“.

Zložitejší problém bude zistiť, že A do nekonečna robí kroky na ε. A si bude počítať (okrem iného) za sebou idúce kroky A na ε, aby dokázal zistiť, kedy sa A zacyklil. Ukážeme, že všetko potrebné si dokáže pamätať v stave.

Ak A robí do nekonečna kroky na ε, sú dve možnosti: Buď zásobník neobmedzene rastie, alebo sa jeho obsah cyklicky opakuje. V druhom prípade zjavne existuje výška, ktorú zásobník nikdy nedosiahne.

Označme $t = |\Gamma|$, $s = |K|$, r dĺžku najdlhšieho slova, ktoré vie A v jednom kroku vložiť na zásobník. Tvrdíme, že ak počas robenia krokov na ε výška zásobníka narastie o viac ako $V = rst + 47$, tak už zásobník porastie do nekonečna.

Zoberme postupnosť $v_0, \ldots, v_n$ výšok zásobníka od začiatku robenia krokov na ε do okamihu, kým nenarástol o viac ako V. Budeme si všímať tie okamihy i, pre ktoré platí, že zásobník po i-tom kroku bol vždy vyšší ako v_i. Postupnosť výšok zásobníka spĺňa podmienky z lemy 4.2.2 (uvedenej za týmto dôkazom). Tá nám hovorí, že takýchto okamihov je viac ako st.

Automat A teda počas tejto postupnosti krokov na ε prešiel viac ako st konfiguráciami takými, že od doby tejto konfigurácie sa nikdy nestalo, aby výška zásobníka klesla na (alebo dokonca pod) hodnotu, ktorú mala v tejto konfigurácii.

Potom ale z Dirichletovho princípu vyplýva, že medzi týmito viac ako st konfiguráciami existovali také konfigurácie K_1, K_2, ktoré sa zhodovali vo vrchnom symbole zásobníka a stave automatu. Z predchádzajúceho vieme, že v neskoršej z nich (BUNV nech je to K_2) je zásobník vyšší ako v skoršej. Ale keďže A je deterministický, ľahko nahliadneme, ako bude výpočet pokračovať – bude sa do nekonečna opakovať postupnosť krokov, vedúca z K_1 do K_2 a zásobník naozaj porastie do nekonečna. (Totiž vzhľadom na výber konfigurácií K_1, K_2 vieme, že počas tohto kusu výpočtu zásobník bol vždy vyšší ako v K_1, a teda tento kus výpočtu opäť prebehne vždy, keď sa zopakuje stav a vrchný zásobníkový symbol z K_1.)

Vždy, keď simulujeme úsek výpočtu, počas ktorého A robí kroky na ε, si budeme v stave pamätať hodnotu $v - v_{min}$, kde v je súčasná výška zásobníka a v_{min} najmenšia výška zásobníka počas tejto časti výpočtu.[2] Ak by táto hodnota mala prekročiť V, prejdeme do nového stavu, v ktorom len dočítame vstup.

Ako ale zistiť, že nastal prípad, že sa A zacyklil, no zásobník počas krokov na ε nikdy nenarastie o viac ako V od pôvodnej výšky? Zásobník sa určite nebude zmenšovať do nekonečna. Všimnime si okamih, kedy sa naposledy zmenila hodnota v_{min}. Ak A od tohto okamihu spraví viac ako $U = (|\Gamma| + 1)^{V+2}|K|$ krokov, určite sa zopakuje niektorá konfigurácia (lebo všetky možné konfigurácie sa líšia len v stave a obsahu zásobníka vo výške $\geq v_{min}$). Budeme si teda navyše v stave počítať, koľko za sebou idúcich krokov na ε sme odsimulovali. Vždy, keď sa zmení hodnota v_{min} (t.j. keď by prvé „počítadlo“ malo klesnúť pod nulu), toto druhé „počítadlo“ vynulujeme a začneme rátať odznova. Keď by toto „počítadlo“ malo prekročiť hodnotu U, dočítame slovo a skončíme.

Množina stavov nášho automatu bude teda $K' = K \times \{0, \ldots, V\} \times \{0, \ldots, U\}$. Takto upraveným automatom dokážeme simulovať A a navyše vyššie popísaným spôsobom detekovať, či sa A nezacyklil. Ľahko nahliadneme, že $L(A) = L(A')$ a že A' vždy dočíta vstupné slovo.

Lema 4.2.2 *Nech $k, r > 0$. Nech $a_0, a_1, \ldots, a_n$ je postupnosť prirodzených čísel taká, že platí:*

- $\forall i;\ a_{i+1} \in \{a_i - 1, a_i, \ldots, a_i + r\}$
- $a_n > a_0 + r(k-1)$

Potom množina $M = \{i \mid \forall j > i;\ a_j > a_i\}$ má aspoň $k+1$ členov.

Dôkaz. Najväčší prvok množiny M je $m_1 = n$. Skúmajme členy postupnosti a_i od konca. Najbližší menší prvok množiny M bude to i, kedy prvýkrát $a_i < a_n$, atď. Môžeme teda postupne zostrojiť ďalšie prvky množiny M nasledovne:

$$m_{i+1} = \max\{j \mid j < m_i \ \wedge\ a_j < a_{m_i}\}$$

Zjavne pre niektoré i už bude množina na pravej strane prázdna a žiadny nový prvok M nedostaneme. Otázka znie, koľko tých prvkov bude. Označme $|M| = x$.

Zjavne pre každé $i > 1$ platí, že $a_{m_i+1} \geq a_{m_{i-1}}$. (Keď zoberieme prvok „napravo“ od m_i, je hodnota postupnosti v ňom aspoň taká, ako v m_{i-1} – inak by sme dotyčný prvok vybrali do M namiesto m_i.) A navyše vieme, že $a_{m_i} \geq a_{m_i+1} - r$. Zložením týchto dvoch nerovností dostávame, že $a_{m_i} \geq a_{m_{i-1}} - r$.

Všimnime si teraz, že $a_{m_x} \leq a_0 < a_n - r(k-1) = a_{m_1} - r(k-1)$. Keďže $a_{m_i} \geq a_{m_1} - r(i-1)$, musí byť $x - 1 \geq k$, q.e.d.

[2] Uvedomte si, že aj hodnota v_{min} sa môže počas tejto časti výpočtu meniť. Toto „počítadlo“ vtedy jednoducho zostane na nule.

Lema 4.2.3 *Ku každému deterministickému zásobníkovému automatu A existuje DPDA A' taký, že $L(A) = L(A')$ a A' vždy dočíta vstupné slovo a zastane. Pritom ak A' vstupné slovo akceptuje, tak zastane v akceptačnom stave (a ak nie, tak nie).*

Dôkaz. Na základe lemy 4.2.1 môžeme predpokladať, že A vždy dočíta vstupné slovo. Musíme ešte ošetriť nasledujúci problém: Kvôli tomu, že deterministický zásobníkový automat môže robiť prechody na ε, môže sa stať, že A po prečítaní vstupného slova prejde do akceptačného stavu (akceptuje ho), ale potom urobí niekoľko ďalších krokov na ε, po ktorých bude v neakceptačnom stave. Keby sme len vymenili akceptačné a neakceptačné stavy, takéto slovo by akceptoval aj A'.

Preto musí nastúpiť akýsi fígeľ. Upravme automat A nasledovne: Nová množina stavov bude $K \times \{0,1\}$. V druhej zložke si budeme ukladať informáciu o tom, či sme od posledného prečítania vstupného symbolu boli v nejakom akceptačnom stave alebo nie. Ak teda A nejaké slovo w akceptuje, po tejto úprave jeho výpočet na slove w skončí v nejakom stave tvaru $(q,1)$, inak skončí v stave tvaru $(q,0)$.

To, čo by sme chceli dosiahnuť, je simulovať A kým nezistíme, že zastal a potom podľa druhej zložky stavu akceptovať alebo nie. Ešte stále je tam jeden háčik, a to pri slovách „kým nezistíme, že zastal“. Ako to chceme zistiť?

Automat zastane, ak je na konci slova a pre jeho súčasný stav a symbol na vrchu zásobníka nemá definovaný prechod na ε. Zdalo by sa, že stačí v takýchto situáciách dodefinovať prechod na ε do nového akceptačného stavu – ale **pozor**, nie je tomu tak! Pre túto dvojicu (stav,symbol) totiž A môže vedieť robiť krok na písmeno zo vstupu. Takto dodefinovaný automat by už nespĺňal našu definíciu DPDA.

Riešenie bude nasledovné: Zavedieme nové stavy $(q,2)$ a $(q,3)$. Stav $(q,i+2)$ znamená: som v stave (q,i) a idem už čítať písmeno zo vstupu (teda teraz už nebudem robiť kroky na ε). Upravíme δ-funkciu A tak, že vždy namiesto kroku na písmeno zo vstupu spraví najskôr krok na ε (a bez zmeny zásobníka) do príslušného nového stavu a až z neho urobí pôvodný krok na písmeno zo vstupu.

Formálna konštrukcia: Nech $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je DPDA, ktorý vždy dočíta vstupné slovo a zastane. Zostrojíme DPDA $A' = (K', \Sigma, \Gamma, \delta', (q_0,0), Z_0, F')$ nasledovne:

$$
\begin{aligned}
K' &= \{(q,i) \mid q \in K \ \wedge \ i \in \{0,1,2,3\}\} \\
F' &= \{(q,3) \mid q \in K\} \\
\delta'((q,i),\varepsilon,Z) &= ((p,i),\gamma) \quad \Leftarrow \quad (p,\gamma) = \delta(q,\varepsilon,Z) \ \wedge \ p \notin F \\
\delta'((q,i),\varepsilon,Z) &= ((p,1),\gamma) \quad \Leftarrow \quad (p,\gamma) = \delta(q,\varepsilon,Z) \ \wedge \ p \in F \\
\delta'((q,i),\varepsilon,Z) &= ((q,i+2),Z) \quad \Leftarrow \quad (p,\gamma) = \delta(q,x,Z) \\
\delta'((q,i+2),x,Z) &= ((p,0),\gamma) \quad \Leftarrow \quad (p,\gamma) = \delta(q,x,Z) \ \wedge \ p \notin F \\
\delta'((q,i+2),x,Z) &= ((p,1),\gamma) \quad \Leftarrow \quad (p,\gamma) = \delta(q,x,Z) \ \wedge \ p \in F
\end{aligned}
$$

Všade kde nie je explicitne nič uvedené je $p,q \in K$, $x \in \Sigma$, $Z \in \Gamma$, $\gamma \in \Gamma^*$ a $i \in \{0,1\}$.

Zjavne každý akceptačný výpočet A zodpovedá výpočtu A', ktorý končí v nejakom stave $(q,3)$. A naopak, každý akceptačný výpočet A' skončí v jednom z týchto stavov a vieme podľa neho zostrojiť akceptačný výpočet A na príslušnom vstupnom slove. Preto naozaj $L(A') = L(A)$, q.e.d.

Všimnime si ešte, že každý neakceptačný výpočet A zodpovedá výpočtu A', ktorý skončí v nejakom neakceptačnom stave $(q,2)$, neskôr sa nám to hodí pri dokazovaní, že $\mathscr{L}_{DPDA}$ je uzavretá na komplement.

4.2.2 Akceptovanie stavom, pamäťou a jednoznačné gramatiky

Veta 4.2.4 *Ku každému DPDA A existuje DPDA A' taký, že $N(A) = L(A')$. (T.j. každý deterministický zásobníkový automat rozpoznávajúci prázdnou pamäťou možno simulovať deterministickým zásobníkovým automatom rozpoznávajúcim akceptujúcim stavom.)*

Dôkaz. Analogicky ako pri nedeterministických PDA – na začiatku výpočtu si A' na spodok zásobníka pridá nový symbol, potom simuluje A. Keď niekedy prečíta zo zásobníka svoj nový symbol, vie, že A má prázdny zásobník, zmení stav na akceptačný a skončí.

Veta 4.2.5 *Nech A je DPDA, nech $w \in N(A)$. Potom $\forall u \neq \varepsilon;\ wu \notin N(A)$.*

Dôkaz. Stačí si uvedomiť, že ak A akceptuje w prázdnou pamäťou, tak po dočítaní w má prázdny zásobník a nemá ako pokračovať vo výpočte ďalej. Preto nemôže akceptovať žiadne iné slovo s prefixom w. (U nedeterministických PDA sme tento problém nemali, tam mohlo existovať viacero výpočtov na w, pri niektorých ešte zásobník nebol prázdny a mohli sme pokračovať ďalej.)

Dôsledok 4.2.6 $\mathscr{L}_{eDPDA} \subsetneq \mathscr{L}_{DPDA}$.

Dôkaz. Z vety 4.2.4 vyplýva, že $\mathscr{L}_{eDPDA} \subseteq \mathscr{L}_{DPDA}$. Z vety 4.2.5 vyplýva, že napr. jazyk $L = \{w \mid w \in \{a,b\}^* \wedge \#_a(w) = \#_b(w)\}$ nepatrí do $\mathscr{L}_{eDPDA}$ (obsahuje napr. slová ε, ab, $abbbaa$, atď.) Príklad 4.2.1 ale ukazuje, že tento jazyk patrí do $\mathscr{L}_{DPDA}$.

Poznámka 4.2.3 Ukážeme teraz, že jediný problém pri akceptovaní prázdnou pamäťou je skutočnosť, že nevieme detekovať koniec slova. Ak každému slovu z L (kde $L \in \mathscr{L}_{DPDA}$) pridáme na koniec nový znak \$, budeme už takýto upravený jazyk vedieť akceptovať aj prázdnou pamäťou. (Uvedomte si, že pridaním \$ na koniec každého slova sme dosiahli, že žiadne slovo už nie je prefixom žiadneho iného.)

Veta 4.2.7 *Nech A je DPDA, nech \$ je nový vstupný symbol. Potom existuje DPDA A' taký, že $N(A') = L(A)\$$.*

Dôkaz. BUNV nech A je v normálnom tvare z lemy 4.2.3 – teda vždy dočíta vstup a ak akceptuje, skončí v akceptačnom stave. Zostrojíme A' nasledovne: Na začiatku si vloží na spodok zásobníka vlastný nový symbol (to preto, aby vedel zo vstupu dočítať aj znak \$, ak A zároveň s akceptovaním vyprázdni zásobník). Ďalej simulujeme A. Ak A akceptoval a dočítal vstupné slovo, v novom stave vyprázdnime zásobník a akceptujeme. To dosiahneme nasledovne:

Pre všetky dvojice (q_F, Z), kde $q_F \in F$, $Z \in \Gamma$, $\delta(q, \varepsilon, Z) = \emptyset$ (teda pre všetky situácie, kedy A mohol zastať a akceptovať) definujeme $\delta(q, \$, Z) = (q_z, Z)$, kde q_z je (jeden) nový stav. Táto definícia nám zjavne neporuší deterministickosť zostrojovaného PDA a spôsobí, že v okamihu, kedy A na konci slova zastal a akceptoval, A' na znak \$ prejde do nového stavu q_z. V tomto stave už iba pomocou krokov na ε vyprázdni zásobník. Rovnosť $N(A') = L(A)\$$ je zrejmá z konštrukcie.

Veta 4.2.8 *Ku každému DPDA A existuje jednoznačná bezkontextová gramatika G taká, že $L(A) = L(G)$.*

Dôkaz. Použijeme rovnakú konštrukciu ako pre nedeterministické PDA. Rozmyslite si, že gramatika, ktorú takto dostaneme, bude jednoznačná.

Veta 4.2.9 *Existujú bezkontextové jazyky, ktoré nepatria do $\mathscr{L}_{DPDA}$, ale existuje pre ne jednoznačná bezkontextová gramatika.*

Dôkaz. V leme 4.2.13 dokážeme, že jazyk $L = \{a^i b^i \mid i \geq 0\} \cup \{a^i b^{2i} \mid i \geq 0\}$ nepatrí do $\mathscr{L}_{DPDA}$. Ľahko nahliadneme, že gramatika $G = (\{\sigma, \beta, \gamma\}, \{a, b\}, \{\sigma \to \beta \mid \gamma,\ \beta \to a\beta b \mid \varepsilon,\ \gamma \to a\gamma bb \mid \varepsilon\}, \sigma)$ je jednoznačná a že $L(G) = L$.

4.2.3 Uzáverové vlastnosti triedy $\mathscr{L}_{DPDA}$

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

Veta 4.2.10 $\mathscr{L}_{DPDA}$ *nie je uzavretá na prienik.*

Dôkaz. Jazyky $L_1 = \{a^i b^i c^j \mid i, j > 0\}$ a $L_2 = \{a^i b^j c^j \mid i, j > 0\}$ sú zjavne v $\mathscr{L}_{DPDA}$, ich prienik nie je ani len v $\mathscr{L}_{CF}$.

Veta 4.2.11 *$\mathscr{L}_{DPDA}$ je uzavretá na prienik s regulárnym jazykom.*

Dôkaz. Analogicky ako pre $\mathscr{L}_{CF}$ – z DPDA a DKA zostrojíme kartézskym súčinom DPDA pre prienik ich jazykov.

Veta 4.2.12 *$\mathscr{L}_{DPDA}$ je uzavretá na komplement.*

Dôkaz. Uvedomme si najskôr, že z viacerých dôvodov nemôžeme priamo použiť konštrukciu, ktorú sme použili pre DKA – vymeniť množiny akceptačných a neakceptačných stavov. Ako obvykle, pomôže nám normálny tvar. Nech teda A je ľubovoľný DPDA, podľa lemy 4.2.1 môžeme predpokladať, že A vždy dočíta vstupné slovo a zastane. Teraz použijeme rovnakú konštrukciu ako v leme 4.2.3, ktorou dosiahneme, že ak A akceptoval, A' zastane v nejakom stave $(q, 3)$, ak nie, tak A zastane v nejakom stave $(q, 2)$. Zjavne bude stačiť drobná zmena – množina akceptačných stavov automatu A' bude $F' = \{(q, 2) \mid q \in K\}$.

Poznámka 4.2.4 Z vety 4.2.12 vyplýva, že $\mathscr{L}_{DPDA} \subsetneq \mathscr{L}_{CF}$. Uvedieme ešte príklad jazyka, ktorý patrí do ich rozdielu.

Lema 4.2.13 *Nech $L = \{a^i b^i \mid i \geq 0\} \cup \{a^i b^{2i} \mid i \geq 0\}$. Potom $L \in \mathscr{L}_{CF} \setminus \mathscr{L}_{DPDA}$.*

Dôkaz. Sporom. Zjavne $L \in \mathscr{L}_{CF}$. Nech teda existuje DPDA A taký, že $L(A) = L$. Zostrojme teraz (nedeterministický) PDA M nasledovne:

M bude obsahovať A. Navyše pridáme jeho kópiu A', v ktorej všetky výskyty písmena b nahradíme písmenom c. Pridáme prechody na ε (bez zmeny zásobníka) z každého akceptačného stavu v A do jemu zodpovedajúceho stavu v A'.

Zamyslime sa nad tým, ako vyzerá $L(M)$. Zjavne $L(M) \supseteq L$. Navyše môže akceptovať niektoré slová, na ktorých mohol prejsť do A' a dopočítať v ňom. Všimnime si, že v akceptačnom stave je A len po prečítaní $a^n b^n$ a $a^n b^{2n}$.

V druhom prípade ak aj prejdeme do A', akonáhle prečítame písmeno zo vstupu, už sa do akceptačného stavu nedostaneme. (Sme v situácií, v ktorej by A' bol po prečítaní $a^n c^{2n}$, žiadne dlhšie slovo s týmto prefixom A' neakceptuje.)

Zaujímavejší je prvý prípad. Dostaneme sa do situácie, v ktorej by A' bol po prečítaní slova $a^n c^n$. To ale znamená, že po prečítaní c^n budeme opäť v akceptačnom stave!

Z uvedeného ľahko nahliadneme, že $L(M) = L \cup \{a^i b^i c^i \mid i \geq 0\}$. Tento jazyk ale zjavne nie je bezkontextový, preto sa takýto PDA M nemôže dať zostrojiť. Jediným problémom pri konštrukcii M bola existencia DPDA A. Tento predpoklad bol teda nesprávny – A neexistuje, a teda $L \notin \mathscr{L}_{DPDA}$.

Veta 4.2.14 *$\mathscr{L}_{DPDA}$ nie je uzavretá na zjednotenie.*

Dôkaz. Sporom. Je uzavretá na komplement. Keby bola uzavretá na zjednotenie, musela by (podľa de Morganovych zákonov) byť uzavretá aj na prienik.

Iná možnosť je všimnúť si, že jazyk z lemy 4.2.13 je zjednotením dvoch deterministických bezkontextových jazykov.

Veta 4.2.15 *$\mathscr{L}_{DPDA}$ je uzavretá na „označené zjednotenie“ (tagged union) – ak $L_1, L_2 \in \mathscr{L}_{DPDA}$, tak aj $(aL_1 \cup bL_2) \in \mathscr{L}_{DPDA}$.*

Dôkaz. DPDA pre označené zjednotenie podľa prvého znaku vstupu vie, ktorý z pôvodných DPDA má simulovať.

Poznámka 4.2.5 Podobne je $\mathscr{L}_{DPDA}$ zjavne uzavretá aj na „polooznačené zjednotenie“ – ak $L_1, L_2 \in \mathscr{L}_{DPDA}$ a a je nový symbol, tak aj $(aL_1 \cup L_2) \in \mathscr{L}_{DPDA}$.

Veta 4.2.16 *$\mathscr{L}_{DPDA}$ nie je uzavretá na homomorfizmus (ani len na nevymazávajúci).*

Dôkaz. Nech $L_1, L_2 \in \mathscr{L}_{DPDA}$ sú jazyky, ktorých zjednotenie nepatrí do $\mathscr{L}_{DPDA}$. Nech a, b sú nové symboly. Vieme, že $aL_1 \cup bL_2 \in \mathscr{L}_{DPDA}$. Zoberme homomorfizmus h, ktorý vymazáva symboly a, b. Je $h(aL_1 \cup bL_2) = (L_1 \cup L_2) \notin \mathscr{L}_{DPDA}$.

Zoberme teraz homomorfizmus g, ktorý zobrazí b na a. Je $g(aL_1 \cup bL_2) = a(L_1 \cup L_2) = L$. Zjavne ak by sme vedeli zostrojiť DPDA pre L, vedeli by sme zostrojiť DPDA pre $(L_1 \cup L_2)$, ktorý by simuloval DPDA pre L, len namiesto prvého čítania zo vstupu spravil krok na ε. Preto ani $L \notin \mathscr{L}_{DPDA}$.

Poznámka 4.2.6 Čitateľ snáď pochopil základnú myšlienku tohto dôkazu: Homomorfizmus nám môže zrušiť značky, ktoré robia jazyk deterministickým. Túto myšlienku ešte viackrát použijeme.

Veta 4.2.17 *$\mathscr{L}_{DPDA}$ je uzavretá na inverzný homomorfizmus.*

Dôkaz. Funguje konštrukcia na rovnakom princípe ako pri ostatných automatových modeloch.

Predpokladajme, že máme DPDA A rozpoznávajúci stavom jazyk L. Pre konkrétny homomorfizmus h musí DPDA rozpoznávajúci jazyk $h^{-1}(L)$ overiť, či jeho vstupné slovo w patrí do $h^{-1}(L)$. Inými slovami, potrebujeme overiť, či $h(w) \in L$. Toto vieme spraviť, keďže $h(w)$ vieme z w zostrojiť deterministicky. Nový DPDA teda vždy prečíta zo vstupu jedno písmeno x a následne odsimuluje niekoľko krokov DPDA A na $h(x)$. (Ešte nespracovanú časť $h(x)$ si môžeme pamätať v stave.)

Veta 4.2.18 *$\mathscr{L}_{DPDA}$ nie je uzavretá na zreťazenie (ani len s regulárnym jazykom).*

Dôkaz. Opäť nech $L_1, L_2 \in \mathscr{L}_{DPDA}$ sú jazyky, ktorých zjednotenie nepatrí do $\mathscr{L}_{DPDA}$. Nech a je nový symbol. Jazyk $L = aL_1 \cup L_2$ je v $\mathscr{L}_{DPDA}$, ale jazyk $\{a\}^*L = \{a\}^*(L_1 \cup L_2)$ zjavne do $\mathscr{L}_{DPDA}$ nepatrí.

Veta 4.2.19 *$\mathscr{L}_{DPDA}$ nie je uzavretá na reverz.*

Dôkaz. Myšlienka bude opäť rovnaká: Reverzom stratíme „značky“, ktoré robia jazyk deterministickým (resp. presunieme ich na konce slov, kde nám budú nanič). Nech teda $L = \{cb^ia^i \mid i \geq 0\} \cup \{db^{2i}a^i \mid i \geq 0\}$. Zjavne $L \in \mathscr{L}_{DPDA}$. Ale takisto zjavne $L^R = \{a^ib^ic \mid i \geq 0\} \cup \{a^ib^{2i}d \mid i \geq 0\}$ do $\mathscr{L}_{DPDA}$ nepatrí. (Ľahko upravíme dôkaz lemy 4.2.13.)

Kapitola 5

Kontextové jazyky

Na úvod tejto kapitoly by sme radi poznamenali, že ako prechádzame k čím ďalej, tým zložitejším modelom, je čím ďalej, tým náročnejšie robiť formálne konštrukcie a dôkazy. Navyše dĺžka týchto konštrukcií a dôkazov s ich zložitosťou rastie. Preto formálna korektnosť bude musieť niekedy ustúpiť zrozumiteľnosti. Často uvedieme len slovný popis danej konštrukcie. Budeme sa snažiť, aby bol tento popis natoľko podrobný, aby čitateľ podľa neho dokázal spraviť príslušnú konštrukciu formálne.

5.1 Rozšírené kontextové gramatiky

Ako sme si už uviedli v úvodnej kapitole pri definícií triedy $\mathscr{L}_{ECS}$ (kontextových jazykov), definícia kontextových gramatík má jeden drobný problém: Keďže pravá strana každého pravidla musí byť aspoň taká dlhá ako ľavá, nevie žiadna kontextová gramatika vygenerovať prázdne slovo.

Môžeme si však pomôcť podobne, ako pri definícii ε-free bezkontextových gramatík – dáme kontextovej gramatike navyše možnosť v prvom kroku odvodenia vygenerovať prázdne slovo: Pridáme navyše možnosť prepísať začiatočný neterminál na ε. Aby ale gramatika toto pravidlo nemohla použiť neskôr počas odvodenia, požadujeme, aby sa začiatočný neterminál nevyskytoval na pravej strane žiadneho pravidla. (A teda sa po prvom kroku odvodenia nebude vyskytovať vo vetnej forme.) Formálne bude:

Definícia 5.1.1 *Frázová gramatika $G = (N, T, P, \sigma)$ je **rozšírená kontextová gramatika**, ak jej pravidlá navyše spĺňajú podmienku $P \subseteq \left((N \cup T)^+ \times \left((N \setminus \{\sigma\}) \cup T\right)^+\right) \cup \{\sigma \to \varepsilon\}$ a pre každé $\pi = (u \to v) \in P$, $\pi \neq (\sigma \to \varepsilon)$ platí $|u| \leq |v|$.*

Zjavne keď zoberieme ľubovoľnú rozšírenú kontextovú gramatiku $G = (N, T, P, \sigma)$ a vynecháme z nej pravidlo $\sigma \to \varepsilon$ (ak ho obsahuje), dostaneme kontextovú gramatiku, ktorá generuje jazyk $L(G) \setminus \{\varepsilon\}$.

A naopak, keď zoberieme ľubovoľnú kontextovú gramatiku $G = (N, T, P, \sigma)$, pridáme jej nový začiatočný neterminál ξ a pravidlá $\xi \to \sigma \mid \varepsilon$, dostaneme rozšírenú kontextovú gramatiku pre jazyk $L(G) \cup \{\varepsilon\}$.

Preto trieda jazykov, pre ktoré existuje rozšírená kontextová gramatika, je práve trieda kontextových jazykov – $\mathscr{L}_{ECS}$.

Poznámka 5.1.1 Všimnite si drobný rozdiel v terminológií: pojmy *kontextový jazyk* a *jazyk generovaný kontextovou gramatikou* nie sú totožné. Totožné sú pojmy *kontextový jazyk* a *jazyk generovaný **rozšírenou** kontextovou gramatikou*. Dôvodov pre takúto definíciu je viacero. Jedným z najvážnejších sú zlé uzáverové vlastnosti triedy $\mathscr{L}_{CS}$, druhým je rokmi zažitá konvencia, tretím dôvodom sú lineárne ohraničené automaty, ktoré definujeme v nasledujúcej časti.

5.2 Lineárne ohraničené automaty

Opäť 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 5.2.1 ***Nedeterministickým lineárne ohraničeným automatom*** *(LBA, linear-bounded automaton) nazývame 6-ticu* $(K, \Sigma, \Gamma, \delta, q_0, F)$*, kde* K *je konečná množina stavov,* Σ *je konečná abeceda vstupných symbolov,* $\Gamma \supseteq \Sigma$ *je konečná abeceda pracovných symbolov (pričom* $\not\in, \$ \notin \Gamma$*),* $q_0 \in K$ *je začiatočný stav,* $F \subseteq K$ *je množina akceptačných stavov a* $\delta : K \times (\Gamma \cup \{\not\in, \$\}) \to 2^{K \times (\Gamma \cup \{\not\in, \$\}) \times \{-1,0,1\}}$ *je prechodová funkcia, pričom platí:*

$$\forall q \in K;\ \delta(q, \not\in) \subseteq K \times \{\not\in\} \times \{0, 1\}$$

$$\forall q \in K;\ \delta(q, \$) \subseteq K \times \{\$\} \times \{-1, 0\}$$

(Teda ak je automat na niektorom konci slova, nesmie ho prekročiť ani prepísať.)

Definícia 5.2.2 ***Konfigurácia*** *LBA je trojica* $(q, \not\in w\$, i)$*, kde* $q \in K$ *je aktuálny stav,* $w \in \Gamma^*$ *aktuálny obsah pásky a* $i \in \{0, \ldots, |w| + 1\}$ *pozícia hlavy.*

Definícia 5.2.3 *Nech* w_i *je* i*-ty znak slova* w*, pričom definujeme* $w_0 = \not\in$ *a* $w_{|w|+1} = \$$*. Potom* ***krokom výpočtu*** *LBA* A *nazývame binárnu reláciu* $\vdash_A$ *na množine konfigurácií definovanú nasledovne:*

$$(q, w_0 w_1 \ldots w_{|w|} w_{|w|+1}, i) \vdash_A (p, w_0 \ldots w_{i-1} x w_{i+1} \ldots w_{|w|+1}, i + j) \iff (p, x, j) \in \delta(q, w_i)$$

Definícia 5.2.4 ***Jazyk*** *akceptovaný LBA* A *je množina*

$$L(A) = \{w \mid \exists q_F \in F,\ x \in \Gamma^*,\ n \in \mathbb{N};\ (q_0, \not\in w\$, 1) \vdash_A^* (q_F, \not\in x\$, n)\}$$

Definícia 5.2.5 ***Deterministický LBA*** *je taký LBA, kde* $\forall q \in K,\ x \in \Gamma;\ |\delta(q, x)| \leq 1$*. Otázka, či sú deterministické a nedeterministické LBA ekvivalentné je dodnes otvoreným problémom.*

Príklad 5.2.1 Zostrojíme LBA A, ktorý bude akceptovať jazyk

$$L = \{w \mid w \in \{a, b, c\}^* \ \wedge\ \#_a(w) = \#_b(w) = \#_c(w)\}$$

Definujme $next(a) = b$, $next(b) = c$, $next(c) = a$. Bude $A = (K, \Sigma, \Gamma, \delta, q_a, F)$, kde:

$$\begin{aligned}
K &= \{q_a, q_b, q_c, q_{over}, q_{akc}\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{a, b, c, \overline{a}, \overline{b}, \overline{c}\} \\
F &= \{q_{akc}\} \\
\delta(q_x, y) &\ni (q_x, y, d) \qquad (\forall x \in \Sigma,\ \forall y \in \Gamma,\ \forall d \in \{-1, 0, 1\}) \\
\delta(q_x, x) &\ni (q_{next(x)}, \overline{x}, 0) \qquad (\forall x \in \Sigma) \\
\delta(q_a, \$) &\ni (q_{over}, \$, -1) \\
\delta(q_{over}, \overline{x}) &\ni (q_{over}, \overline{x}, -1) \qquad (\forall x \in \Sigma) \\
\delta(q_{over}, \not\in) &\ni (q_{akc}, \not\in, 0)
\end{aligned}$$

Všimnite si elegantné využitie nedeterminizmu pri našej konštrukcii. V stave q_x sa automat nedeterministicky hýbe po páske a hľadá x. Keď nejaké nájde, označí si ho (aby to isté x nezarátal viackrát) a ide hľadať ďalšie písmeno. Keď je v stave q_a (teda doteraz našiel rovnako písmen a, b a c), môže sa nedeterministicky rozhodnúť, že už má celé slovo označené a chce ho akceptovať. Musíme ešte overiť, či je tomu naozaj tak. Automat príde hlavou na koniec slova, zmení stav na

q_{over} a v tom prejde celú pásku „sprava doľava“. Ak sú naozaj všetky písmená označené, podarí sa mu prísť na začiatok slova a akceptovať.

Rozmyslite si, že na podobnej myšlienke (aj keď konštrukcia by bola komplikovanejšia) by sa dal zostrojiť aj deterministický LBA pre tento jazyk. Napríklad nasledovne: Automat by „odškrtol“ vždy prvé neodškrtnuté hľadané písmeno v slove. Keď pri hľadaní a narazí na koniec slova, skontroluje, či mu nezvýšili žiadne b ani c.

Čitateľ, ktorý sa dostal až na toto miesto, si určite vie predstaviť, ako by vyzeral formálny dôkaz, že $L = L(A)$. Inklúzia $L \subseteq L(A)$ je zjavná z vyššie popísanej myšlienky konštrukcie, ľahko nahliadneme, že na každom slove z L akceptačný výpočet existuje. Pri dôkaze opačnej inklúzie by sme mohli napríklad matematickou indukciou dokázať tvrdenie, že keď je A v stave q_a a na páske má slovo x, tak $\#_{\overline{a}}(x) = \#_{\overline{b}}(x) = \#_{\overline{c}}(x)$ a že hodnoty výrazov $A = \#_a(x) + \#_{\overline{a}}(x)$, $B = \#_b(x) + \#_{\overline{b}}(x)$ a $C = \#_c(x) + \#_{\overline{c}}(x)$ sa počas výpočtu nemenia. Odtiaľ už vyplýva, že v každom akceptovanom slove muselo byť rovnako a, b aj c.

Poznámka 5.2.1 Konfiguráciu LBA je možné definovať viacerými odlišnými spôsobmi. Uvedieme ešte dve možnosti:

(Konfigurácia s bičíkom) Konfigurácia LBA A je dvojica (q, w), kde $q \in K$ je aktuálny stav, $w \in \¢\Gamma^* \restriction \Gamma^*\$ \cup \restriction \¢\Gamma^*\$$ je slovo na páske, pričom znak $\restriction$ je pred znakom, na ktorom je práve hlava.

(Konfigurácia ako slovo) Konfigurácia LBA A je slovo $w \in \¢\Gamma^* K \Gamma^*\$ \cup K\¢\Gamma^*\$$, pričom znak z K predstavuje aktuálny stav a ostatné znaky slovo zapísané na páske. Znak z K je uvedený pred znakom, na ktorom je v danej konfigurácii hlava automatu. (Navyše požadujeme, aby $K \cap \Gamma = \emptyset$.)

Čitateľ ľahko dodefinuje krok výpočtu LBA s takto definovanou konfiguráciou a tiež ľahko nahliadne, že obe definície sú ekvivalentné s našou. Pri konkrétnych dôkazoch môže byť výhodnejšie použiť niektorú z týchto definícií, preto ich tu uvádzame.

5.3 Ekvivalentnosť kontextových gramatík a lineárne ohraničených automatov

Veta 5.3.1 *K ľubovoľnej kontextovej gramatike G existuje LBA A taký, že $L(A) = L(G)$.*

Dôkaz. Nech $G = (N, T, P, \sigma)$. Potom LBA $A = (K, \Sigma = T, \Gamma = N \cup T \cup \{B, \$'\}, \delta, q_0, F)$ bude pracovať tak, že bude simulovať odvodenie v gramatike G odzadu. Automat bude opakovať nasledujúci cyklus:

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

$$\delta(q_0, x) \quad \ni \quad (q_{\pi,1}, x, 0) \qquad (\forall x \in \Gamma,\ \forall \pi \in P)$$

2. Nedeterministicky si nájde správny výskyt pravej strany pravidla na páske.

$$\delta(q_{\pi,1}, x) \quad \ni \quad (q_{\pi,1}, x, d) \qquad (\forall x \in N \cup T,\ \forall \pi \in P,\ \forall d \in \{-1, 0, 1\})$$

3. Skontroluje, či naozaj našiel výskyt pravej strany pravidla. Ak áno, tak ho prepíše ľavou stranou daného pravidla. (Ak nie, tak sa zasekne a nepokračuje ďalej vo výpočte.) Uvedieme sadu pravidiel pre konkrétne $\pi = (u_1 \dots u_n \to v_1 \dots v_m)$. Ak $n < m$, tak nech $u_{n+1} = \dots = u_m = B$.

$$\begin{aligned}
\delta(q_{\pi,i}, v_i) &\ni (q_{\pi,i+1}, v_i, 1) && (\forall i \le m) \\
\delta(q_{\pi,m+1}, x) &\ni (pis_{u_1 \dots u_m}, x, -1) && (\forall x \in \Gamma \cup \{\$\}) \\
\delta(pis_{wa}, x) &\ni (pis_w, a, -1) && (\forall a \in \Gamma,\ \forall w \in \Gamma^*,\ |w| < m)
\end{aligned}$$

4. Uprace pásku. Tým sa myslí to, že ak bola ľavá strana vybraného pravidla kratšia ako pravá strana, vznikli nám na páske pri prepise „prázdne“ políčka B, ktoré presunie za posledné písmeno, a následne ich prepíše znakmi $\$'$ (falošný dolár – znak konca slova). Najľavejší $\$'$ od tejto chvíle považuje za koniec vstupu.

$$\begin{aligned}
\delta(pis_\varepsilon, x) &\ni (zisti, x, 1) \qquad (\forall x \in \Gamma \cup \{¢\}) \\
\delta(zisti, x) &\ni (zisti, x, 1) \qquad (\forall x \in N \cup T) \\
\delta(zisti, D) &\ni (q_0, D, -1) \qquad (\forall D \in \{\$, \$'\}) \qquad \text{– netreba posúvať} \\
\delta(zisti, B) &\ni (prines, B, 1) \\
\delta(prines, B) &\ni (prines, B, 1) \\
\delta(prines, x) &\ni (nesiem_x, B, -1) \qquad (\forall x \in N \cup T) \\
\delta(nesiem_x, B) &\ni (nesiem_x, B, -1) \qquad (\forall x \in N \cup T) \\
\delta(nesiem_x, y) &\ni (poloz_x, y, 1) \qquad (\forall x \in N \cup T,\ \forall y \in N \cup T \cup \{¢\}) \\
\delta(poloz_x, B) &\ni (prines, x, 1) \qquad (\forall x \in N \cup T) \\
\delta(prines, D) &\ni (uprac, D, -1) \qquad (\forall D \in \{\$, \$'\}) \qquad \text{– už sme všetko posunuli} \\
\delta(uprac, B) &\ni (uprac, \$', -1) \\
\delta(uprac, x) &\ni (q_0, x, 1) \qquad (\forall x \in N \cup T \cup \{¢\})
\end{aligned}$$

5. Zakaždým, keď si automat volí ďalšie pravidlo, ktoré ide „vrátiť späť“, sa namiesto toho môže nedeterministicky rozhodnúť, že už skončil. Vtedy skontroluje, či sa na páske nachádza už iba σ. Ak áno, tak zmení stav na akceptačný.

$$\begin{aligned}
\delta(q_0, x) &\ni (q_{over1}, x, 0) \qquad (\forall x \in \Gamma \cup \{¢, \$\}) \\
\delta(q_{over1}, x) &\ni (q_{over1}, x, -1) \qquad (\forall x \in \Gamma \cup \{\$\}) \\
\delta(q_{over1}, ¢) &\ni (q_{over2}, ¢, 1) \\
\delta(q_{over2}, \sigma) &\ni (q_{over3}, \sigma, 1) \\
\delta(q_{over3}, D) &\ni (q_{akc}, D, 0) \qquad (\forall D \in \{\$, \$'\})
\end{aligned}$$

Veta 5.3.2 *K ľubovoľnému LBA A existuje kontextová gramatika G taká, že $L(G) = L(A) \setminus \{\varepsilon\}$.*

Dôkaz. Nech LBA $A = (K, \Sigma, \Gamma, \delta, q_0, F)$. Pri dôkaze tejto vety sa nám hodí tretí variant konfigurácií (konfigurácia ako slovo z $¢\Gamma^* K \Gamma^* \$ \ \cup \ K¢\Gamma^*\$$), keďže krok výpočtu A predstavuje len lokálnu zmenu v takto zapísanej konfigurácií. Túto zmenu budeme simulovať v našej gramatike. Navyše BUNV predpokladáme, že δ-funkcia A je taká, že do akceptačného stavu môže prejsť len vtedy, ak je hlava na znaku $\$$.

Naša gramatika G si najskôr tipne slovo w, ktoré ide vygenerovať. Vetnú formu upraví na začiatočnú konfiguráciu A a simuluje ho. Ak A akceptuje, G slovo w naozaj vygeneruje, inak sa odvodenie zasekne.

Samotná simulácia nie je problém, napr. ak $(p, b, -1) \in \delta(q, a)$, tak konfigurácia sa pri spravení tohto kroku má zmeniť z $\ldots cqa \ldots$ na $\ldots pcb \ldots$, budeme mať teda sadu pravidiel $cqa \to pcb$ $(\forall c \in \Gamma)$.

Prvý problém: Uvedomme si, že simuláciou automatu na vstupnej konfigurácii sa pokazila samotná vstupná konfigurácia. Automatu to nevadí, pretože jemu stačí akceptovať, ale gramatika musí vygenerovať slovo, z ktorého simuláciu začala. Preto si musíme na začiatku nejako vstupnú konfiguráciu zapamätať, aby sme po úspešnom skončení simulácie dokázali „vypísať“ slovo z nášho jazyka.

Ukážeme si trik, ktorý ešte v budúcnosti veľakrát použijeme, tzv. *poschodové neterminály*. Keď bude mať gramatika pracovnú abecedu $M \cup (M \times M)$, jeden jej neterminál môže predstavovať

jeden alebo dva symboly z množiny M. Na tieto symboly sa môžeme dívať tak, že sú napísané nad sebou. Ak teda napríklad $w \in (M \times M)^*$, môžeme sa na w dívať ako na dve slová nad abecedou M napísané nad sebou.

(Analógiou tejto myšlienky pri automatoch bude *viacstopá páska*. Pri voľbe vhodnej pracovnej abecedy vie mať automat na jednom políčku zapísaných aj viac znakov naraz.)

Naša gramatika G teda bude mať takéto „poschodové“ neterminály, pričom „horné poschodie“ obsahuje počas celého výpočtu vygenerované slovo w, na „spodnom“ budeme simulovať A.

G teda začne tým, že vygeneruje vetnú formu tvaru $\not\vdash q_0 \binom{a_1}{a_1} \binom{a_2}{a_2} \dots \binom{a_n}{a_n} \$$. Zodpovedajúcu sadu pravidiel čitateľ ľahko zostrojí.

Na spodnej stope simuluje výpočet automatu A. Pravidlá na simuláciu budú kvôli dvojstopej páske vyzerať trochu inak, napr. ak $(p, b, -1) \in \delta(q, a)$, tak budeme mať teda sadu pravidiel $\binom{x}{c} q \binom{y}{a} \to p \binom{x}{c} \binom{y}{b}$ $(\forall x, y, c \in \Gamma)$.

Ak je simulovaný automat v akceptačnom stave (a teda podľa predpokladu je hlava na konci slova), môžeme vygenerovať terminálne slovo w pomocou pravidiel $q_F\$ \to Q$ (pre $q_F \in F$), $\binom{x}{a} Q \to Qx$, $\not\vdash Q \to \varepsilon$.

Ostáva posledný drobný problém: dve z posledných pravidiel nie sú kontextové. Konfigurácia je totiž o 3 znaky dlhšia ako výsledné slovo a týchto troch znakov sa potrebujeme zbaviť. To ale kontextovými pravidlami nevieme.

S týmto problémom nám opäť pomôžu viacposchodové vetné formy. Budeme mať dokonca päť poschodí: spodné dve poschodia budú doterajšie dve poschodia. Keďže potrebujeme „schovať“ tri znaky, ktoré nám kazia kontextovosť, bude prvé poschodie pre stav, druhé pre $\not\vdash$ a tretie pre \$. Presnejšie: na prvom poschodí bude zapísaný stav nad miestom, kde sa nachádza hlava, na druhom poschodí nad prvým znakom bude $\not\vdash$ a na treťom poschodí nad posledným znakom bude \$. (Ešte nejako – napr. iným znakom ako $\not\vdash$ a \$ – odlíšime, či sa hlava nachádza na cente/dolári, alebo na prvom/poslednom písmene slova.)

Formálny zápis tejto konštrukcie sa síce dá podľa popisu priamočiaro zostrojiť, ale je dosť rozsiahly a nepovažujeme za potrebné uvádzať ho. Takisto veríme, že v prípade núdze by čitateľ bol schopný vygenerovať formálny dôkaz, že $L(A) \setminus \{\varepsilon\} = L(G)$.

Dôsledok 5.3.3 *Triviálnym dôsledkom vety 5.3.2 je, že ku každému LBA existuje ekvivalentná rozšírená kontextová gramatika.*

5.4 Normálne tvary kontextových gramatík

Lema 5.4.1 *Ku každej kontextovej gramatike G existuje ekvivalentná kontextová gramatika G', ktorej pravidlá sú navyše aj podmnožinou $(N^+ \times N^+) \cup (N \times T)$.*

Dôkaz. Zavedieme nové neterminály ξ_x (kde $x \in T$). Všetky výskyty terminálu x v pravidlách (na ľavej aj pravej strane!) nahradíme neterminálom ξ_x. Pridáme pravidlá $\xi_x \to x$.

Poznámka 5.4.1 Výhoda tohto normálneho tvaru je tá, že akonáhle gramatika vygeneruje terminálny symbol, ten už zostane do konca odvodenia na svojom mieste.

Lema 5.4.2 *Ku každej kontextovej gramatike G existuje ekvivalentná kontextová gramatika G', ktorej pravidlá sú tvaru $u\xi v \to uwv$ (kde $\xi \in N$, $u, v, w \in (N \cup T)^*$, $w \neq \varepsilon$).*

Dôkaz. Opäť uvedieme len myšlienku. Každé pravidlo pôvodnej gramatiky nahradíme sadou pravidiel požadovaného tvaru, ktoré ho budú simulovať. Použitím nových neterminálov (pre ktoré bude existovať len jedno použiteľné pravidlo) zabezpečíme, aby sa novými pravidlami nedalo odvodiť nič iné.

Nech teda v G máme pravidlo $\pi = (u_1 \dots u_n \to v_1 \dots v_m)$ (kde $m \geq n \geq 2$). Potom v G' budeme mať nasledujúcu sadu pravidiel:

$$u_1 u_2 \dots u_n \quad \to \quad \pi_1 u_2 \dots u_n \tag{5.1}$$

$$\pi_i u_{i+1} \rightarrow \pi_i \pi_{i+1} \tag{5.2}$$

$$\pi_1 \dots \pi_{n-1} \pi_n \rightarrow \pi_1 \dots \pi_{n-1} \overline{\pi_n}\, v_{n+1} \dots v_m \tag{5.3}$$

$$\pi_i \overline{\pi_{i+1}} \rightarrow \boxed{\pi_i}\, \overline{\pi_{i+1}} \tag{5.4}$$

$$\boxed{\pi_i}\, \overline{\pi_{i+1}} \rightarrow \boxed{\pi_i}\, v_{i+1} \tag{5.5}$$

$$\boxed{\pi_i} \rightarrow \overline{\pi_i} \tag{5.6}$$

$$\overline{\pi_1} \rightarrow v_1 \tag{5.7}$$

Samozrejme predpokladáme, že všetko okrem pôvodných u_i a v_i sú nové neterminály, ktoré sa dovtedy v G' nevyskytovali. Všade $i \in \{1, \dots, n-1\}$.

Čitateľ si ľahko overí, že akonáhle sa použije pravidlo 5.1, máme vo vetnej forme jeden z týchto nových neterminálov. (Našli sme si výskyt ľavej strany simulovaného pravidla a označili sme si ho.) Jediná možnosť ako sa ho zbaviť je nasledovná: Používaním pravidla 5.2 si postupne označíme celú ľavú stranu pravidla. Pravidlom 5.3 následne pripíšeme na koniec prepisovanej časti znaky $v_{n+1} \dots v_m$ (ak bola pravá strana π dlhšia ako ľavá) a vo vetnej forme dostaneme nový typ neterminálu. Ten nám zabezpečí, že sa korektne prepíše aj zvyšok pravidla. Jediná možnosť, ako sa ho zbaviť, je striedavo používať pravidlá 5.4, 5.5 a 5.6, čím „prebuble“ na začiatok prepisovanej časti a tá sa korektne prepíše. Keď už je označený len prvý znak prepisovanej časti, môžeme ho pravidlom 5.7 zmeniť na v_1 a zbaviť sa ho. Tým sme ale úspešne odsimulovali použitie pravidla π.

Odporúčame rozmyslieť si nasledujúcu skutočnosť: Počas používania pravidla 5.2 sa mohli v G' robiť kroky, ktoré menili ešte neoznačenú časť pravidla. Ak sa nám ale skôr či neskôr podarilo celú ľavú stranu pravidla označiť, neprekáža nám to. (Keby sme všetky použitia pravidiel 5.1 a 5.2 spravili až v tomto okamihu, dostali by sme rovnaký výsledok.)

Poznámka 5.4.2 Ako sme už uviedli pri definícií kontextových gramatík, táto lema dokazuje ekvivalenciu pôvodnej Chomského definície kontextových gramatík s našou.

5.5 Uzáverové vlastnosti kontextových jazykov

Veta 5.5.1 *Trieda $\mathscr{L}_{ECS}$ je uzavretá na zjednotenie.*

Dôkaz. Analogicky ako u jednoduchších typov gramatík – nedeterministicky sa rozhodneme, do ktorého zo zjednocovaných jazykov generované slovo patrí a vygenerujeme ho.

(Dôkaz pomocou LBA: nedeterministicky sa rozhodneme, do ktorého zo zjednocovaných jazykov vstupné slovo patrí a príslušné LBA na ňom odsimulujeme.)

Veta 5.5.2 *Trieda $\mathscr{L}_{ECS}$ je uzavretá na zreťazenie.*

Dôkaz. Analogicky ako u bezkontextových gramatík. Navyše ale potrebujeme zabezpečiť, aby sa nám jednotlivé časti vetnej formy „nepomiešali“. Na to ale stačí, aby pôvodné gramatiky mali disjunktné množiny neterminálov a boli v normálnom tvare z lemy 5.4.1.

(Dôkaz pomocou LBA by bol komplikovanejší: nedeterministicky sa rozhodneme, kde končí prvá časť slova, odsimulujeme LBA pre prvý jazyk na prvej časti slova a ak akceptuje, tak odsimulujeme LBA pre druhý jazyk na zvyšku slova. Rozmyslite si, ako by ste riešili simuláciu na okrajoch vybranej časti slova.)

Príklad 5.5.1 Majme gramatiky:

$$G_1 = (\{\sigma_1\}, \{a, b\}, \{\sigma_1 \rightarrow a\}, \sigma_1), \qquad G_2 = (\{\sigma_2\}, \{a, b\}, \{a\sigma_2 \rightarrow bb\}, \sigma_2)$$

Rozmyslite si, že keby sme len slepo použili konštrukciu, ktorá fungovala pre bezkontextové gramatiky, nedostaneme správny výsledok.

Veta 5.5.3 *Trieda $\mathscr{L}_{ECS}$ je uzavretá na iteráciu.*

Dôkaz. Opäť použijeme osvedčený prístup s gramatikami. Podobne ako pri zreťazení potrebujeme striedavo používať dve kópie pôvodnej gramatiky, ktoré budú mať disjunktné neterminály a budú v normálnom tvare z lemy 5.4.1.

Veta 5.5.4 *Trieda $\mathscr{L}_{ECS}$ je uzavretá na prienik.*

Dôkaz. Najskôr sa trochu zamyslime. Nemôžeme simulovať oba automaty súčasne na tom istom slove, totiž môžu ho rôznym spôsobom prepisovať. Preto opäť použijeme trik s dvojstopou páskou (pracovná abeceda bude obsahovať kartézsky súčin množín pôvodných páskových symbolov).

Nech máme dané dva LBA A_1, A_2. Zostrojíme automat A, ktorý bude akceptovať $L(A_1) \cap L(A_2)$. Automat A bude pracovať v nasledovných etapách:

1. Vyrobí na páske dve stopy, skopíruje vstupné slovo na hornú aj na spodnú stopu.
2. Simuluje automat A_1 na hornej stope a ak ten akceptuje, prejde do nasledujúcej etapy.
3. Simuluje automat A_2 na spodnej stope. A akceptuje, ak automat A_2 akceptuje.

Formálna konštrukcia by vyzerala nasledovne: Nech $A_i = (K_i, \Sigma_i, \Gamma_i, \delta_i, q_{0i}, F_i)$ sú zadané LBA. Zostrojíme $A = (K, \Sigma = \Sigma_1 \cup \Sigma_2, \Gamma, \delta, q_R, F)$, kde:

$$\begin{aligned} K &= \{q_R, q_L, q_I\} \quad \cup \quad \{\ [i,q] \mid i \in \{1,2\}, q \in K_i\} \\ \Gamma &= \Sigma \ \cup \ (\Gamma_1 \times \Gamma_2) \\ F &= \{\ [2,f] \mid f \in F_2\} \end{aligned}$$

Stavy q_R a q_L slúžia na zdvojenie pásky na začiatku výpočtu, q_I na presun hlavy po prvej simulácii. Inak si v stave pamätáme poradové číslo a stav práve simulovaného LBA. Zdvojenie pásky:

$$\begin{aligned} \delta(q_R, x) &\ni (q_R, [x,x], 1) \\ \delta(q_R, \$) &\ni (q_L, \$, -1) \\ \delta(q_L, x) &\ni (q_L, x, -1) \qquad (\forall x \in \Gamma) \\ \delta(q_L, \text{¢}) &\ni (\ [1, q_{01}], \text{¢}, 1) \end{aligned}$$

Simulácia prvého automatu na hornej stope pásky (formálne teda meníme prvú z dvoch zložiek pracovného symbolu):

$$\begin{aligned} \delta(\ [1,q],\ [x,y]\ ) &\ni (\ [1,p],\ [z,y],\ d) \qquad (\forall\, [x,y] \in \Gamma,\ \forall p,q \in K_1,\ (p,z,d) \in \delta_1(q,x)\ ) \\ \delta(\ [1,q],\ E) &\ni (\ [1,p],\ E,\ d) \qquad (\forall E \in \{\text{¢}, \$\},\ \forall p,q \in K_1,\ (p,E,d) \in \delta_1(q,E)\ ) \\ \delta(\ [1,f],\ [x,y]\ ) &\ni (q_I,\ [x,y],\ 0) \qquad (\forall\, [x,y] \in \Gamma,\ \forall f \in F_1) \\ \delta(\ [1,f],\ E) &\ni (q_I,\ E, 0) \qquad (\forall E \in \{\text{¢}, \$\},\ \forall f \in F_1) \end{aligned}$$

Ak prvý automat akceptoval, opäť presunieme hlavu na začiatok pásky a spustíme simuláciu druhého automatu:

$$\begin{aligned} \delta(q_I, x) &\ni (q_I, x, -1) \qquad (\forall x \in \Gamma) \\ \delta(q_I, \text{¢}) &\ni (\ [2, q_{02}], \text{¢}, 1) \\ \delta(\ [2,q],\ [x,y]\ ) &\ni (\ [2,p],\ [x,z],\ d) \qquad (\forall\, [x,y] \in \Gamma,\ \forall p,q \in K_2,\ (p,z,d) \in \delta_2(q,y)\ ) \\ \delta(\ [2,q],\ E) &\ni (\ [2,p],\ E,\ d) \qquad (\forall E \in \{\text{¢}, \$\},\ \forall p,q \in K_2,\ (p,E,d) \in \delta_2(q,E)\ ) \end{aligned}$$

Dôkaz správnosti tejto konštrukcie by bol zdĺhavý ale očividný, preto ho neuvádzame.

Veta 5.5.5 *$\mathscr{L}_{ECS}$ 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 $h(w) \in L$.

Pri dokazovaní uzavretosti $\mathscr{L}_{CF}$ na inverzný homomorfizmus sme používali buffer, ktorý sme dokázali držať v stave. 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“. 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. Na začiatku ho prejde zľava doprava a na políčko, kde bolo písmeno x, zapíše $h(x)$. (Uvedomte si, že na to mu stačí konečná pracovná abeceda.) Potom sa posunie na začiatok slova $h(w)$ a začne na ňom simulovať prácu automatu A. Tu treba prepísať δ-funkciu automatu A tak, aby vedel chodiť po viacstopej páske nahor a dole a vedel riešiť aj problém „štrbavého hrebeňa“, t.j. ošetriť problém prázdneho políčka na vstupnej páske. (Napr. v stave si vieme pamätať, na ktorom zo symbolov na aktuálnom políčku je hlava simulovaného automatu.)

Veta 5.5.6 *$\mathscr{L}_{ECS}$ je uzavretá na nevymazávajúci homomorfizmus.*

Dôkaz. Opäť pomocou automatov. Nech A je LBA pre daný jazyk, nech h je nevymazávajúci homomorfizmus. Automat pre $h(L(A))$ bude fungovať nasledovne: Nedeterministicky si tipneme, z akého slova u vzniklo vstupné slovo w. Keďže homomorfizmus je nevymazávajúci, $|u| \leq |w|$, teda u sa nám zmestí na jednu stopu pásky. Overíme, že naozaj $h(u) = w$ a odsimulujeme pôvodný automat na slove u.

Veta 5.5.7 *Ku každému frázovému jazyku L existuje kontextový jazyk L' a homomorfizmus h také, že $L = h(L')$*

Dôkaz. Majme frázovú gramatiku $G = (N, T, P, \sigma)$ takú, že $L(G) = L$. Zostrojme kontextovú gramatiku G' tak, že doplníme každé nekontextové pravidlo G tak, aby generovalo navyše niekoľko nových neterminálov ξ. Pridajme pravidlá tvaru $\xi x \to x\xi$ (ktorými sa neterminály ξ môžu presunúť na koniec vetnej formy, aby nám neprekážali pri odvodení) a pravidlo $\xi \to e$, kde e je nový terminál.

Zoberme homomorfizmus h taký, že $\forall x \in T;\ h(x) = x$ a $h(e) = \varepsilon$. Tvrdíme, že $h(L(G')) = L(G)$. Myšlienka je teda taká, že kontextová gramatika G' na miestach, kde G niečo maže, vygeneruje niekoľko terminálov e, ktoré potom z výsledného slova odstránime.

Formálne: Ak $w \in L(G)$, tak existuje jeho odvodenie v G. Potom ľahko v G' odvodíme slovo we^x (pre vhodné x) – po odsimulovaní kroku G vždy všetky nové ξ presunieme na koniec vetnej formy, na konci odvodenia ich prepíšeme na e. Potom ale $w \in h(L(G'))$.

Naopak ak $w \in h(L(G'))$, vieme v G' vygenerovať slovo w', z ktorého vymazaním všetkých e dostaneme w. Zoberme odvodenie w' v G', vynechajme z neho kroky presúvajúce a prepisujúce ξ a nahraďme zvyšné kroky im zodpovedajúcimi krokmi v G. Dostaneme odvodenie slova w v G, teda $w \in L(G)$.

Dôsledok 5.5.8 *$\mathscr{L}_{ECS}$ nie je uzavretá na homomorfizmus.*

Poznámka 5.5.1 Keby sme chceli byť detailisti, na to, aby sme dokázali tento dôsledok, potrebujeme ešte ukázať, že $\mathscr{L}_{RE} \supsetneq \mathscr{L}_{ECS}$. Zatiaľ ešte nemáme dostatočný aparát na to, aby sme ukázali, že niektorý jazyk nie je kontextový, preto sa k tomuto tvrdeniu neskôr vrátime.

Poznámka 5.5.2 Trieda $\mathscr{L}_{CS}$ je uzavretá na zjednotenie, prienik, zreťazenie, kladnú iteráciu, inverzný homomorfizmus, nevymazávajúci homomorfizmus a reverz. (Dôkazy sú takmer identické vyššie uvedeným dôkazom pre $\mathscr{L}_{ECS}$.) Z pochopiteľných dôvodov $\mathscr{L}_{CS}$ nie je uzavretá na iteráciu (ktorá do jazyka vždy pridáva slovo ε), na homomorfizmus ani na komplement.

5.6 Szelepczényiho veta

Matematici zaoberajúci sa teóriou formálnych jazykov po desaťročia hľadali odpoveď na otázku, či je trieda kontextových jazykov uzavretá na komplement. Väčšina z týchto bádateľov sa totiž mylne domnievala, že odpoveď na túto otázku je záporná. Preto hľadali dôkazy toho, že trieda $\mathscr{L}_{ECS}$ nie je uzavretá na komplement. Ukázalo sa však, že opak je pravdou. Za tento veľký výsledok vďačíme slovenskému matematikovi Róbertovi Szelepczényimu (ktorý v tom čase študoval na MFF UK Bratislava).

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 čítal 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 5.6.1 *(Immerman, Szelepczényi) Trieda $\mathscr{L}_{ECS}$ je uzavretá na komplement.*

Dôkaz. Nech $L \in \mathscr{L}_{ECS}$ je akceptovaný 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. Vďaka nedeterminizmu A si nemôžeme dovoliť zmeniť množinu akceptačných stavov na $K \setminus F$. Potrebujeme pre dané vstupné slovo overiť, či **žiaden** výpočet A na ňom nie je akceptujúci.

Základnou myšlienkou je, že A' bude generovať konfigurácie, do ktorých sa A mohol dostať, a zisťovať o nich, či sú dosiahnuteľné. Ak narazí na dosiahnuteľnú akceptačnú konfiguráciu, zasekne/zacyklí sa a slovo neakceptuje. Ak úspešne preverí všetky dosiahnuteľné konfigurácie a žiadna z nich nie je akceptačná, slovo na vstupe akceptuje.

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 $|\Gamma|^{|w|} \times |K| \times (|w| + 2)$. (Máme $|\Gamma|$ možností pre obsah každého políčka pásky, $|K|$ možností pre stav, $|w| + 2$ možností pre pozíciu hlavy.)

Ďalej bude náš dôkaz pokračovať svojím koncom. Predpokladajme, že poznáme počet dosiahnuteľných konfigurácií Q. Neskôr ukážeme, že toto číslo vieme naozaj vypočítať. Automat A' bude postupne na páske generovať všetky možné konfigurácie. Pre každú z nich nedeterministicky uhádne, či je dosiahnuteľná automatom A. V prípade, že háda áno, skontroluje, či naozaj. V prípade, že táto konfigurácia je naozaj dosiahnuteľná, v konečnom čase to overí[1] a pokračuje ďalej, inak neakceptuje. Automat A' zjavne nemôže žiadnu nedosiahnuteľnú konfiguráciu prehlásiť za dosiahnuteľnú – nepodarilo by sa mu to overiť. Oklamať nás už môže len jedným smerom – prehlásiť niektorú dosiahnuteľnú konfiguráciu za nedosiahnuteľnú. My ale vieme (presnejšie A' má niekde na páske zapísané) číslo Q. Ak si A' bude musieť počítať počet konfigurácií, ktoré prehlási za dosiahnuteľné, donútime ho tak „hovoriť pravdu“. (Ak A' prejde všetky konfigurácie a zistí, že za dosiahnuteľné ich vyhlásil menej ako Q, zasekne sa.)

Na tomto mieste si treba uvedomiť, že číslo Q si automat A' dokáže pamätať na jednej stope pásky. Totiž pre dostatočne veľkú[2] konštantu c je $|\Gamma|^{|w|} \times |K| \times (|w| + 2) \leq c^{|w|}$. Takže číslo Q zapísané v sústave so základom c sa vojde na jednu stopu pásky.[3]

Jediné, čo ostáva ukázať, je, ako A' spočíta číslo Q. Postupne bude generovať čísla Q_0, Q_1, Q_2, ..., ktoré budú vyjadrovať počet dosiahnuteľných konfigurácií automatu A na danom 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 $Q_i = Q_{i+1}$ a takisto platí, že ak pre nejaké i je $Q_i = Q_{i+1}$, tak $Q_i = Q$. Stačí teda generovať čísla Q_i, kým sa neprestanú zväčšovať.

Číslo Q_0 poznáme, je rovné jednej. Ukážme, že na základe poznania čísla Q_i si A' dokáže vypočítať Q_{i+1}. (Naozaj to musí zvládnuť len na základe poznania čísla Q_i a nie zapamätania si

[1] Na jednej stope pásky nedeterministicky simuluje správny výpočet A zo zač. konfigurácie do práve overovanej.
[2] Napríklad $c = |\Gamma| + 3|K|$.
[3] Technický detail: uvedené tvrdenie platí pre $|w| \geq 1$. Prípad $w = \varepsilon$ ale ľahko vyriešime samostatne.

všetkých Q_i konfigurácii, na to totiž nemá na páske dosť miesta.) Neskôr ukážeme, že keď poznáme Q_i, vieme na jednej stope pásky vygenerovať všetkých Q_i konfigurácií dosiahnuteľných na najviac i krokov. Teraz ale predpokladajme, že už to vieme spraviť a ukážme, ako potom spočítať hodnotu Q_{i+1}.

Nech M_i je množina konfigurácií, ktoré A vie dosiahnuť na najviac i krokov. Budeme postupovať nasledovne: A' si nastaví hodnotu Q_{i+1} na Q_i. Zakaždým, keď nájdeme novú konfiguráciu z $M_{i+1} \setminus M_i$, zväčšíme Q_{i+1} o jedno.

Budeme postupne generovať všetky konfigurácie z M_i. Nech máme práve vygenerovanú m-tú z nich, označme ju K_m. Skúsime odsimulovať všetkými možnosťami nasledujúci krok automatu A z nej. Takto dostaneme konečne veľa konfigurácií z M_{i+1}. O každej z nich potrebujeme zistiť, či ju máme započítať. Keď chceme konfiguráciu započítať, musíme overiť, že nie je v M_i (opäť vygenerujeme všetky konfigurácie z M_i a porovnáme ju s nimi) a že je nová, ešte nezapočítaná v Q_{i+1} (ešte raz vygenerujeme všetky konfigurácie z M_i, overíme, že zo žiadnej vygenerovanej pred K_m sa nedá dostať do overovanej konfigurácie na jeden krok).

Ak vieme (na základe toho, že poznáme Q_i) vygenerovať v konečnom čase všetky konfigurácie z M_i, vyššie popísaným postupom v konečnom čase spočítame hodnotu Q_{i+1}. Opakovaním tohto postupu kým $Q_i \neq Q_{i+1}$ spočítame hľadanú hodnotu Q. Zostáva ukázať posledný detail – ako na základe znalosti Q_i vygenerovať všetky konfigurácie v M_i.

Použijeme rovnakú fintu ako na začiatku dôkazu: Postupne prejdeme všetky možné konfigurácie. O každej si nedeterministicky tipneme, či patrí do M_i. Ak sme si tipli, že áno, overíme, či naozaj a zvýšime počítadlo vygenerovaných konfigurácií o 1. Ak na konci máme v počítadle hodnotu menšiu ako Q_i, zasekneme sa.

Na záver poznamenajme, 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 si počítame počet vygenerovaných možností (a tým donútime nedeterminizmus vygenerovať všetky) dostala názov **inductive counting**.

Kapitola 6

Rekurzívne vyčísliteľné a rekurzívne jazyky

6.1 Turingove stroje

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 (TS), ba dokonca všetko, čo je algoritmické (teda naprogramovateľné). V tejto časti sa budeme zamýšľať aj nad vlastnosťami TS, ktoré budú väčšinou analogické ako pre 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 a jednou hlavou pohybujúcou sa doprava, doľava, alebo stojacou na mieste. V kroku výpočtu hlava načíta písmeno na páske, podľa neho a stavu sa rozhodne, aký znak na jeho miesto zapíše, ako zmení stav a ktorým smerom pohne hlavu.

Na začiatku predpokladáme, že sa hlava nachádza na prvom písmenku vstupného slova, pričom na miestach, kde sa vstupné slovo nenachádza, je na páske zapísaný špeciálny znak **B** (blank, prázdne).

Konfigurácie môžeme zapisovať podobne ako u LBA s tým, že zapisujeme iba neprázdnu časť pásky (aby konfigurácia bola konečná). Aby neprázdna časť pásky bola súvislá, požadujeme, aby TS nesmel zapisovať **B**. (Uvedomte si, že silu TS to neobmedzí, môže používať iný symbol, „falošný blank“, ku ktorému sa bude správať rovnako ako ku blanku.) Na výber pri definícií konfigurácie máme rovnaké možnosti ako pri LBA.

Akceptovanie Turingovym strojom zabezpečíme akceptačným stavom – TS akceptuje akonáhle dosiahne akceptačný stav, nemusí ani len dočítať vstupné slovo.

Definícia 6.1.1 ***Nedeterministický Turingov stroj*** *je 6-ica* $(K, \Sigma, \Gamma, \delta, q_0, F)$*, kde* K *je konečná množina stavov,* Σ *je konečná abeceda vstupných symbolov,* Γ *je konečná abeceda pracovných symbolov (*$\Sigma \subseteq \Gamma$*,* $\mathbf{B} \notin \Gamma$*),* $q_0 \in K$ *je začiatočný stav,* $F \subseteq K$ *je množina akceptačných stavov a* $\delta : K \times (\Gamma \cup \{\mathbf{B}\}) \to 2^{K \times \Gamma \times \{-1,0,1\}}$ *je prechodová funkcia.*

Definícia 6.1.2 ***Konfigurácia*** *TS je prvok z* $K\mathbf{B}\Gamma^* \cup \Gamma^* K \Gamma^+ \cup \Gamma^* K \mathbf{B}$.

Poznámka 6.1.1 Čitateľ isto podľa príkladu pri LBA dokáže definovať „konfiguráciu s bičíkom“ aj „konfiguráciu s pozíciou hlavy od začiatku (neprázdnej časti) pásky“.

Definícia 6.1.3 ***Krok výpočtu*** *TS je relácia* $\vdash$ *na konfiguráciách definovaná nasledovne:*

$$\begin{aligned} &\forall u, v \in \Gamma^+,\ \forall x, y, z \in \Gamma,\ \forall q \in K : \\ &\qquad uqxv \vdash upzv \iff (p, z, 0) \in \delta(q, x) \\ &\qquad uqxv \vdash uzpv \iff (p, z, 1) \in \delta(q, x) \end{aligned}$$

$$uyqxv \vdash upyzv \quad \iff \quad (p, z, -1) \in \delta(q, x)$$

Veríme, že čitateľ si sám doplní definíciu o prípady, kedy je hlava pri okrajoch zapísanej časti pásky.

Definícia 6.1.4 ***Jazyk*** *akceptovaný TS A je množina*
$L(A) = \{w \mid \exists q_F \in F,\ u, v \in (\Gamma \cup \{\mathbf{B}\})^*;\ (q_0 w) \vdash_A^* (u q_F v)\}$.

Poznámka 6.1.2 Táto definícia je trochu „lenivá“ v tom, že pripúšťa ako u a v ľubovoľné postupnosti znakov pracovnej abecedy a blankov. My ale vieme, že zapísaná časť pásky je vždy súvislá, preto by sa dalo aj presnejšie popísať, ako u a v vyzerajú. Zbytočne by sme tým ale definíciu zneprehľadnili.

Definícia 6.1.5 ***Deterministický TS*** *je taký TS, kde* $\forall q \in K,\ \forall x \in \Gamma \cup \{\mathbf{B}\};\ |\delta(q, x)| \leq 1$.

Poznámka 6.1.3 Deterministický TS budeme skrátene označovať DTS. Ak budeme chcieť zdôrazniť, že konkrétny TS je nedeterministický, budeme ho označovať NTS.

Poznámka 6.1.4 Iná možnosť je formálne definovať DTS podobne, ako sme definovali jednoduchšie typy automatov, teda ako TS, ktorého prechodová funkcia je typu $\delta : K \times (\Gamma \cup \{\mathbf{B}\}) \to (K \times \Gamma \times \{-1, 0, 1\})$. Výhodou tejto definície je trochu ľahší zápis konkrétneho DTS. Je ľahké nahliadnuť, že obe definície sú ekvivalentné.

Cvičenie 6.1.1 Bolo by vhodné, aby čitateľ skôr, než bude čítať ďalej, získal predstavu o sile Turingovych strojov. Odporúčame skúsiť si zostrojiť nasledujúce (deterministické alebo nedeterministické) Turingove stroje:

- TS akceptujúci prázdny jazyk
- TS akceptujúci daný konečný jazyk
- TS simulujúci daný konečný automat
- TS simulujúci daný zásobníkový automat (zásobník si môže napr. písať na pásku za vstup)
- TS, ktorý po spustení na slove $w \in \{a, b\}^*$ na jeho koniec pripíše znak # a zaň skopíruje w
- TS pre jazyk $\{a^x \# a^y \mid x \text{ delí } y\}$
- TS pre jazyk $\{a^x \mid x \text{ je prvočíslo}\}$
- **nedeterministický** TS pre jazyk $\{a^x \mid x \text{ nie je prvočíslo}\}$ – jednoduchší ako v predchádzajúcom príklade!
- TS, ktorý po spustení na slove $w \in \{a, b, c\}^*$ toto prepíše na $a^\alpha b^\beta c^\gamma$, kde $\alpha = \#_a(w)$, atď.
- TS, ktorý keď spustíme na prázdnej páske a necháme pracovať do nekonečna, napíše na ňu slovo $abaabaaabaaaab\ldots$
- TS, ktorý rozdelí vstupné slovo na polovice a na jeho prvej polovici odsimuluje iný TS
- TS, ktorý rozdelí vstupné slovo na polovice, odsimuluje iný TS na jeho prvej polovici a ak akceptoval, tak ho odsimuluje aj na druhej polovici pôvodného vstupu

Poznámka 6.1.5 Uvedomte si, že posledné z cvičení sú príkladom toho, že Turingov stroj vie používať iné TS ako „procedúry“. Ak máme TS, ktorý niečo robí a chceme ho použiť pri konštrukcii nového TS, stačí vhodne premenovať stavy pôvodného TS, pridať jeho δ-funkciu do zostrojovaného TS a upraviť zostrojovanú δ-funkciu tak, aby vedela „zavolať pridanú procedúru“ a po jej skončení pokračovať ďalej.

6.2 Ekvivalencia det. a nedet. Turingovych strojov

Zjavne každý DTS je zároveň aj NTS. Ukážeme, že na deterministickom Turingovom stroji vieme simulovať nedeterministický.

Veta 6.2.1 *Ku každému NTS A existuje DTS A' taký, že $L(A) = L(A')$.*

Dôkaz. A' bude na vstupnom slove w prehľadávať do šírky priestor konfigurácií, v ktorých sa A na w mohol nachádzať. Teda najskôr upraví vstupné slovo na začiatočnú konfiguráciu, potom funguje v nekonečnom cykle: nájde na páske prvú nespracovanú konfiguráciu, označí ju ako spracovanú, odsimuluje všetky možné kroky A z tejto konfigurácie a nové konfigurácie pripíše na koniec zapísanej časti pásky.

Takto A' zjavne skôr či neskôr[1] nájde každú konfiguráciu, do ktorej sa A na w mohol dostať. (A' najskôr vygeneruje tie konfigurácie, kam sa A mohol dostať na jeden krok, potom všetky dosiahnuteľné na dva kroky, atď.) A' akceptuje, akonáhle nájde nejakú akcept. konfiguráciu A.

6.3 Varianty Turingovych strojov

Ukážeme si niekoľko iných možností, ako definovať Turingov stroj. Väčšina z nich bude rozpoznávať jazyky a budú ekvivalentné s našou definíciou. Okrem nich si ukážeme niekoľko „exotickejších" variantov, ktoré budú robiť niečo trochu iné. Definície budeme robiť neformálne – ale veríme, že dostatočne podrobne na to, aby nebol problém spraviť podľa nich príslušnú formálnu definíciu.

NTS a DTS s jednosmerne nekonečnou páskou

Rozdiel je v tom, že príslušný TS definujeme tak, že páska je nekonečná len smerom doprava, na ľavom okraji je endmarker ¢, ktorý TS (podobne ako LBA) nesmie prekročiť ani prepísať.

Simulovať takýto TS na našom je triviálne. Simulácia opačným smerom:

Jedna možnosť je mať špeciálnu množinu stavov („procedúru"), ktorá vie posunúť celé slovo na páske o jedno doprava. Keď chceme zapísať na ¢, najskôr si ňou slovo na páske posunieme doprava, potom príslušný znak zapíšeme na prvé políčko pásky.

Druhá možnosť je vyrobiť si na začiatku výpočtu dvojstopú pásku, ktorá akoby vznikne preložením pôvodnej oboma smermi nekonečnej pásky – na hornej stope bude časť pásky idúca doprava, na dolnej zvyšok pásky idúci doľava. V stave si pamätáme, či sme v simulovanom TS na hornej alebo dolnej stope.

Pri NTS je ešte iná možnosť: Na začiatku výpočtu posúvame vstupné slovo doprava, kým sa nedeterministicky nerozhodneme, že už máme naľavo od neho dosť miesta. Potom odsimulujeme výpočet.

Viacpáskový TS s hlavou na každej páske

Takýto TS funguje tak, že v jednom kroku sa podľa svojho stavu a znakov, ktoré čítajú jednotlivé hlavy rozhodne, čo ktorá hlava zapíše a kam sa pohne.

Pomocou jednej hlavy by sme takýto automat mohli simulovať pomocou viacerých stôp na jednej páske. 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.

Iná možnosť by bola mať na každej stope zaznačené, kde sa práve ktorá hlava nachádza. Automat pri simulácii jedného kroku prebehne zapísanú časť pásky, v stave si „pozbiera", čo ktorá hlava číta, potom podľa δ-funkcie simulovaného automatu zmení stav, znova prebehne pásku, zapíše čo má a príslušne poposúva značky, kde je ktorá hlava. Treba si uvedomiť, že to, čo číta konečný počet hláv, si vieme zapamätať v stave.

[1] Uvedomte si, že ako popis každej konfigurácie, tak aj počet možných krokov z nej je konečný, preto každú konfiguráciu spracuje v konečnom čase.

TS s viac hlavami na páske

Pri definícii potrebujeme vyriešiť situáciu, čo sa stane, ak bude chcieť viac hláv naraz písať na to isté políčko. Sú viaceré možnosti, napr. hlavy očíslujeme, úspešne zapíše tá s najväčším číslom.

Pri simulácií budeme mať na páske viac stôp, pre každú hlavu jednu, kde si značíme pozíciu príslušnej hlavy. Znaky načítané jednotlivými hlavami si môžeme zapamätať v stave, a potom postupne porobiť to, čo pôvodný automat s viacerými hlavami. (Ak ich bude simulovať v poradí, v akom sú očíslované, automaticky vyrieši aj problém s viac hlavami na jednom políčku – jednoducho tam zostane to, čo tam zapísala posledná simulovaná hlava.)

TS s dvojrozmernou páskou

Tento TS nemá pásku, ale rovinu rozdelenú na políčka tvoriace štvorcovú sieť. Vstup w je zapísaný na políčkach so súradnicami $[0,0],\ldots,[|w|-1,0]$, všade inde sú blanky. Hlava je na začiatku na políčku $[0,0]$. Prechodová funkcia nám vráti napr. pohyb v smere osi x a v smere osi y.

Pri jeho simulácií ukážeme všeobecný postup, ktorý sa dal použiť na simuláciu ľubovoľného z doteraz uvedených modelov: Na páske budeme mať zoznam neprázdnych políčok – trojice $[x,y,c]$, kde x,y sú súradnice políčka a c je znak na tom políčku. Toto všetko samozrejme vhodne zakódované, napr. čísla x, y budú zapísané v dvojkovej sústave. Teda napr. trojicu $[6,5,c]$ by sme mohli mať na páske ako #0110&0101&c# (kde # a & sú špeciálne oddeľovacie znaky, prvá 0 je znamienkový bit). Na konci pásky budeme mať ešte dvojicu súradníc, na ktorých sa nachádza hlava.

Na simuláciu použijeme TS s dvomi hlavami na našej páske. Jedna bude na mieste, kde sú zapísané súradnice simulovanej hlavy, druhá behá po páske. Vždy, keď máme odsimulovať krok pôvodného automatu, druhá hlava prebehne po páske a nájde popis príslušného políčka. (Dve hlavy máme jedine na to, aby sa im ľahko porovnávali súradnice – jedna číta súradnice hlavy, druhá zároveň s tým súradnice políčka.) Ak ho nenájde, tak sme čítali blank. Tak či tak si vieme v stave zapamätať čítaný znak, rozhodnúť sa, aký krok spravil pôvodný automat a odsimulovať ho – či už prepísať znak na danom políčku, alebo (ak sme prepísali blank) na začiatok pásky pridať nové políčko a na koniec zmeniť príslušne súradnice hlavy.

Uznávame, že tento popis je na toto miesto až príliš voľný a neformálny, no keď čitateľ získa trochu praxe s Turingovymi strojmi, zistí, že každý z popísaných krokov vedia TS ľahko realizovať.

TS počítajúci funkciu

Tento model nebude priamo ekvivalentný s klasickým TS. Súvis medzi ním a klasickým TS presahuje rámec tohto textu.

Definícia 6.3.1 *Hovoríme, že DTS* ***počíta funkciu*** $f: N \to N$*, ak po spustení na vstupe a^x po konečnom počte krokov prejde do akceptačného stavu a na páske je (okrem prípadných falošných blankov) slovo $a^{f(x)}$.*

Poznámka 6.3.1 Túto definíciu môžeme jemne upraviť: Hovoríme, že DTS **počíta funkciu** $f: \Sigma_1^* \to \Sigma_2^*$, ak po spustení na vstupe $w \in \Sigma_1^*$ po konečnom počte krokov prejde do akceptačného stavu a na páske je (okrem prípadných falošných blankov) slovo $f(w) \in \Sigma_2^*$.

Generujúci TS

Ukážeme jeden mierne odlišný model,[2] ktorým zdôvodníme, prečo budeme odteraz frázové jazyky volať *rekurzívne vyčísliteľné*.

Definícia 6.3.2 ***Generujúci TS*** *je DTS A s dvomi páskami, ktorý bude fungovať nasledovne: Neskončí výpočet dosiahnutím akceptačného stavu, t.j. pracuje prípadne aj do nekonečna. Druhá*

[2] Tentokrát opäť ekvivalentný s klasickým TS...

páska je „výstupná“ – vždy, keď je A v akceptačnom stave, slovo, ktoré je na nej,[3] *sa akoby vypíše na výstup. Jazyk $L(A)$ generovaný týmto TS je množina slov, ktoré v konečnom čase vypíše na výstup.*

Poznámka 6.3.2 Tento stroj teda vymenúva v nejakom poradí všetky slová nejakého jazyka. V angličtine sa pre túto činnosť používa pojem „enumerate“ (vymenovať, vyčísliť), pri preklade do slovenčiny sa (možno trochu nešťastne) zvolil pojem vyčísliť. Prívlastok „rekurzívne“ pochádza od ešte iného modelu, ako sú Turingove stroje a frázové gramatiky – rekurzívnych a čiastočne rekurzívnych funkcií.[4]

Poznámka 6.3.3 Zjavne jazyk generovaný generujúcim TS vieme akceptovať klasickým Turingovym strojom – keď dostaneme vstup w, stačí simulovať generujúci TS, kým nevypíše w na výstup. (Ak ho nevypíše nikdy, jednoducho ho budeme simulovať do nekonečna, teda w neakceptujeme.)

Veta 6.3.1 *Ku každému DTS A existuje generujúci TS A' taký, že $L(A) = L(A')$.*

Dôkaz. Potrebujeme odsimulovať A na všetkých slovách zo Σ^* a vypísať tie, ktoré akceptuje. Zjavne to nemôžeme robiť postupne – čo ak sa na niektorom z nich zacyklí? Budeme ho musieť simulovať naraz na všetkých. Aby sa „na každé slovo dostalo“, spravíme to nasledovne:

A' bude do nekonečna opakovať nasledujúci cyklus: Najskôr pridá ďalšie vstupné slovo medzi už spracúvané, potom odsimuluje po jednom kroku A na všetkých už spracúvaných slovách. Každé slovo, ktoré A akceptuje, vypíšeme na výstup.

Zjavne A' vygeneruje len slová z $L(A)$, ukážme, že ich vygeneruje všetky. Nech $w \in L(A)$. Spracúvaných slov je v každom okamihu konečne veľa, preto každý prechod cyklom skončí v konečnom čase. To ale znamená, že v konečnom čase (rádovo po $|\Sigma|^{|w|}$ prechodoch cyklom) sa medzi spracúvané slová zaradí aj w. Akceptačný výpočet A na w je konečne dlhý, po príslušnom počte prechodov cyklom w vypíšeme.

6.4 Ekvivalencia Turingovych strojov a frázových gramatík

Veta 6.4.1 *Ku každej frázovej gramatike G existuje NTS A taký, že $L(G) = L(A)$.*

Dôkaz. Analogicky ako pri LBA, simuláciou odvodenia (buď od začiatku alebo od konca). Pri LBA sme uviedli konštrukciu, kedy od konca hádame odvodenie a príslušne upravujeme vetnú formu až kým nedostaneme iba začiatočný neterminál. Teraz uvedieme druhú možnú konštrukciu.

Vstupné slovo si odložíme na jednu stopu pásky. Na druhej stope nedeterministicky hádame a simulujeme jeho odvodenie v G. Keďže máme k dispozícii nekonečne dlhú pásku, už nás netrápi, že vetná forma počas výpočtu môže byť dlhšia ako vstupné slovo. Ak sa nám vstupné slovo podarí odvodiť, akceptujeme.

Veta 6.4.2 *Ku každému NTS A existuje frázová gramatika G taká, že $L(A) = L(G)$.*

Dôkaz. Opäť analogicky ako pri LBA: Gramatika bude mať dvojposchodové neterminály. Najskôr na hornom poschodí vygeneruje ľubovoľné slovo. To skopíruje na dolné poschodie, upraví ho na začiatočnú konfiguráciu A (najlepšie v nami definovanom tvare, kde je pozícia hlavy vyznačená symbolom stavu) a prepisovaním vetnej formy simuluje výpočet A. Ak sa vo vetnej forme vyskytne akceptačný stav, G prepíše horné poschodie neterminálov na terminály a ostatné nadbytočné neterminály zmaže.

[3] Prípadne môžeme definovať výstupnú abecedu $\Sigma' \subseteq \Gamma$, ako výstupné slovo berieme len slovo tvorené symbolmi zo Σ'^* – kvôli tomu, aby sme vedeli vygenerovať aj kratšie slovo ako sme vygenerovali v predchádzajúcom kroku. Inou možnosťou je povoliť generujúcemu TS zapisovať blanky na výstupnú pásku.

[4] Dá sa ukázať, že rekurzívne funkcie sú ekvivalentné s funkciami počítanými TS z predchádzajúcej definície.

6.5 Uzáverové vlastnosti rekurzívne vyčísliteľných jazykov.

Pripomenieme, že triedu rekurzívne vyčísliteľných (t.j. frázových) jazykov označujeme $\mathscr{L}_{RE}$. Pri drvivej väčšine vlastností bude situácia rovnaká ako pre kontextové jazyky a budú sa dať použiť takmer identické dôkazy.

Veta 6.5.1 *$\mathscr{L}_{RE}$ je uzavretá na zjednotenie, prienik, zreťazenie, iteráciu a reverz.*

Dôkaz. Rovnako ako pre kontextové jazyky.

Veta 6.5.2 *$\mathscr{L}_{RE}$ je uzavretá na homomorfizmus a inverzný homomorfizmus.*

Dôkaz. Keďže na nekonečnej páske máme miesta dosť, nemusíme sa trápiť s tým, že nám na ňu obraz nevôjde. Pre jednoduchšiu konštrukciu môžeme použiť dvojpáskový TS. Jednoducho vstupné slovo zobrazíme, pričom obraz slova (resp. pri homomorfizme nedeterministicky tipnutý správny vzor slova) píšeme na druhú pásku. Na tomto slove potom simulujeme TS pre pôvodný jazyk.

Na záver jedno možno prekvapivé tvrdenie, v ktorom sa kontextové a frázové jazyky líšia:

Veta 6.5.3 *$\mathscr{L}_{RE}$ nie je uzavretá na komplement.*

Dôkaz. Keď v nasledujúcich častiach vybudujeme dostatočný aparát, ukážeme jazyk, ktorý je rekurzívne vyčísliteľný (lema 6.7.1), ale jeho komplement rekurzívne vyčísliteľný nie je (veta 6.7.2).

Poznámka 6.5.1 Uvedomte si, že z tejto vety vyplýva, že $\mathscr{L}_{ECS} \neq \mathscr{L}_{RE}$ a keďže zjavne $\mathscr{L}_{ECS} \subseteq \mathscr{L}_{RE}$, tak $\mathscr{L}_{ECS} \subsetneq \mathscr{L}_{RE}$. Preto naozaj kontextové jazyky nie sú uzavreté na homomorfizmus (viď dôsledok 5.5.8).

6.6 Kódovanie jazykov a TS

TS nad dvojpísmenkovou abecedou

Ukážeme, že ku každému TS existuje „v podstate ekvivalentný“ TS s pracovnou abecedou $\{0,1\}$. Úvodzovky sú tam preto, že ekvivalentný TS nemusíme vedieť zostrojiť – totiž vstupná abeceda pôvodného TS môže obsahovať aj iné symboly ako 0 a 1.

Lema 6.6.1 *Ku každému jazyku L nad abecedou Σ takou, že $|\Sigma| = n$ existuje homomorfizmus h taký, že $h(L)$ je jazyk nad abecedou $\{0,1\}$ a $h^{-1}(h(L)) = L$.*

Dôkaz. Nech $\Sigma_L = \{a_0, \ldots, a_{n-1}\}$. Stačí položiť napr. $h(a_i) = 0^i 1$.

Poznámka 6.6.1 Význam tejto lemy je nasledovný: Každý jazyk nad ľubovoľne veľkou abecedou vieme homomorfizmom „zakódovať“ do vhodného jazyka a príslušným inverzným homomorfizmom ho „dekódovať“ späť. Keďže $\mathscr{L}_{RE}$ je uzavretá na tieto operácie, buď sú rekurzívne vyčísliteľné oba jazyky, alebo ani jeden z nich. Stačí sa nám teda zaujímať o TS so vstupnou abecedou $\{0,1\}$.

Veta 6.6.2 *Ku každému TS A so vstupnou abecedou $\{0,1\}$ a pracovnou abecedou Γ existuje ekvivalentný TS A' s pracovnou abecedou $\{0,1\}$.*

Dôkaz. Nech $\Gamma = \{a_0 = 0, a_1 = 1, a_2, \ldots, a_{n-1}\}$. Náš TS A' si bude na páske symbol a_i reprezentovať postupnosťou symbolov tvaru[5] $0^{i+1}1^{n-i}$.

Na začiatku si „preloží“ vstupné slovo do svojho kódovania symbolov, t.j. každú nulu prepíše na slovo 01^n a každú jednotku na $0^2 1^{n-1}$. Odteraz simuluje TS A, pričom prečítanie symbolu z pásky (a rovnako aj zápis) mu trvá rádovo n krokov, počas ktorých prejde príslušný kúsok pásky a v stave si „poskladá“ prečítané písmeno.

[5] Mohli sme zvoliť ľubovoľný iný jednoznačne dekódovateľný tvar, napr. $0^i 1$ ako pri dôkaze predchádzajúcej lemy. Pri konštrukcii TS je však náš tvar výhodnejší – všetky symboly reprezentujeme slovami rovnakej dĺžky, ktoré začínajú 0 a končia 1, takže sa TS ľahko hľadajú.

Dôsledok 6.6.3 *Každý rekurzívne vyčísliteľný jazyk teda vieme „zakódovať“ do jazyka nad abecedou* $\{0,1\}$ *a ten akceptovať Turingovym strojom, ktorý nepoužíva žiadne iné symboly.*

Kódovanie TS

V tejto časti si povieme o zakódovaní daného Turingovho stroja do postupností z $\{0,1\}^*$. Je zjavne nemožné zakódovať konkrétne vstupné symboly, ktoré používa, ale lema 6.6.1 hovorí, že to ani nie je nutné. Navyše vieme, že výpočtová sila DTS a NTS je rovnaká. Preto sa (v tejto aj ďalších častiach) obmedzíme na deterministické Turingove stroje so vstupnou abecedou $\{0,1\}$. Keď budeme mať daný kód TS (kód TS A označujme $<A>$), budeme môcť okrem iného neskôr zostrojiť tzv. univerzálny TS, ktorý dostane na vstup kód nejakého TS A a vstupné slovo w a bude simulovať stroj A na slove w.

Poznámka 6.6.2 Pre programátora by nemal byť problém predstaviť si zakódovanie konkrétneho TS do postupnosti núl a jednotiek – jednoducho nejaký jeho popis uložíme do súboru a ako kód zoberieme výsledný súbor po bitoch. Ukážeme ale jednu možnú „matematickejšiu“ a hlavne exaktnejšiu definíciu kódu TS.

Zostrojíme najskôr kód nad abecedou $\{0,1,\#\}$. Kód bude vyzerať napr. nasledovne:

$$\#\#(\text{veľkosť K})\#\#(\text{veľkosť }\Gamma)\#\#(\text{čísla akc. stavov})\#\#(\delta - \text{funkcia})\#\#$$

Pritom predpokladáme, že stavy sú očíslované od 0 do $|K|-1$, páskové symboly od 0 do $|\Gamma|-1$ (0,1 sú vstupné symboly, symbol $|\Gamma|$ je blank). Všetky čísla (veľkosti K, Γ a čísla akceptačných stavov) sú zapísané po bitoch, teda v dvojkovej sústave, a prípadne od seba oddeľované znakom $\#$.

Riadok δ-funkcie $(q_y, c_t, d) \in \delta(q_x, c_s)$ zapíšeme $x\#s\#y\#t\#d$. (Opäť, čísla x, y, s, t sú uvedené po bitoch, d je 0, 1 alebo 11 (namiesto -1).) Jednotlivé riadky δ-funkcie sú oddelené $\#\#$.

Takto vieme ľubovoľný TS zakódovať (nie nutne jednoznačne) do slova nad abecedou $\{0,1,\#\}$. Keď na takýto kód TS aplikujeme homomorfizmus taký, že $h(0) = 1$, $h(1) = 01$, $h(\#) = 001$, dostaneme jednoznačne dekódovateľný kód TS nad abecedou $\{0,1\}$.

Zjavne nie každé slovo nad abecedou $\{0,1\}$ je kódom nejakého TS. Aby sme sa zbavili tejto nemilej vlastnosti, definujeme, že ak sa slovo nad abecedou $\{0,1\}$ nedá vyššie uvedeným postupom dekódovať na Turingov stroj, tak je to kód TS akceptujúceho prázdny jazyk.

Turingov stroj s kódom w budeme značiť A_w.

6.7 Diagonálny a univerzálny jazyk

Pripomenieme, že sa zaoberáme len deterministickými Turingovymi strojmi so vstupnou abecedou $\{0,1\}$.

Každé prirodzené číslo vieme jednoznačne zapísať v dvojkovej sústave, tým dostaneme slovo nad abecedou $\{0,1\}$. Preto vždy, keď v tejto časti hovoríme o čísle na vstupe TS, myslíme tým toto zodpovedajúce slovo. A naopak, každé slovo nad touto abecedou môžeme chápať ako číslo. (Vďaka nulám na začiatku viacerým slovám zodpovedá to isté číslo, to nám ale nebude prekážať.)

Definícia 6.7.1 ***Diagonálny jazyk*** *je jazyk* $L_D = \{w \mid w \in \{0,1\}^* \ \wedge \ w \in L(A_w)\}$.

Poznámka 6.7.1 Diagonálny jazyk je teda (voľne povedané) jazyk kódov tých Turingovych strojov, ktoré svoj kód akceptujú.

Lema 6.7.1 *Diagonálny jazyk je rekurzívne vyčísliteľný.*

Dôkaz. Zostrojíme preň viacpáskový TS, ktorý bude fungovať nasledovne: Vstupné slovo skopíruje na druhú pásku. Na slovo na prvej páske sa bude dívať ako na kód stroja A_w, ktorý má simulovať, na slovo na druhej páske ako na vstup. Na tretiu pásku si na začiatku napíše číslo

začiatočného stavu stroja A_w. (Uvedomte si, že stav simulovaného A_w si nemôže pamätať vo svojom stave, vopred totiž nevieme povedať, koľko stavov bude A_w mať.) Slovo na druhej páske, predstavujúce vstup, „preloží“ do pracovnej abecedy A_w.

Teraz bude simulovať A_w. V jednom kroku simulácie z druhej pásky zistí čítaný symbol, na prvej páske prejde kód δ-funkcie, nájde riadok zodpovedajúci stavu a prečítanému symbolu a odsimuluje ho (skopíruje nový stav na tretiu pásku, upraví slovo na druhej páske a príslušne na nej posunie hlavu).

Veta 6.7.2 *Komplement diagonálneho jazyka nie je rekurzívne vyčísliteľný.*

Poznámka 6.7.2 Technika, ktorú sme použili pri zostrojení tohto jazyka, pochádza z Cantorovho dôkazu, že reálnych čísel je nespočítateľne veľa a volá sa *diagonalizácia*. Cantor na základe predpokladu, že reálnych čísel je spočítateľne veľa zostrojil reálne číslo rôzne od všetkých reálnych čísel. Ako ukážeme, my sme zostrojili jazyk, ktorý sa líši od všetkých jazykov akceptovaných TS.

Dôkaz. Sporom. Nech L_D^C je rekurzívne vyčísliteľný. Potom ale existuje TS A so vstupnou abecedou $\{0,1\}$, ktorý ho akceptuje. Teda je $L(A) = L_D^C$. Tento TS má nejaký kód $w = <A>$. Pozrime sa, ako sa správa A na svojom kóde.

Nech A slovo $w = <A>$ akceptuje, teda $w \in L(A)$. Zároveň podľa definície L_D slovo w patrí aj do L_D. To ale znamená, že $w \notin L_D^C = L(A)$, čo je spor.

Nech teraz A slovo $w = <A>$ neakceptuje, teda $w \notin L(A)$. Potom podľa definície L_D slovo w nepatrí do L_D. To ale znamená, že $w \in L_D^C = L(A)$, čo je opäť spor.

Turingov stroj A teda nemôže svoj kód ani akceptovať, ani neakceptovať. To ale nie je možné, preto takýto TS neexistuje a jazyk L_D^C naozaj nie je rekurzívne vyčísliteľný.

Poznámka 6.7.3 Myšlienka dôkazu ešte raz inými slovami. Zobrali sme všetky Turingove stroje. Kódy týchto strojov vieme zoradiť do postupnosti a očíslovať prirodzenými číslami. Zostrojili sme jazyk L_D^C. Ten sa od jazyka akceptovaného i-tym Turingovym strojom v tejto postupnosti (teda stroja A_i) líši v tom, či obsahuje i-te slovo. Preto sa tento jazyk líši od všetkých jazykov akceptovaných TS, a teda nie je rekurzívne vyčísliteľný.

	$<A_1>$	$<A_2>$	$<A_3>$	...	$<A_k>$	...
A_1	0					
A_2		1				
A_3			0			
...						
A_k	1	0	1	...	?	
...						

V tabuľke je 1, ak stroj slovo akceptuje, v opačnom prípade 0. Ak by stroj akceptujúci L_D^C existoval, potom by sa nachádzal v postupnosti ako A_k pre nejaký index k. Spor dostaneme, ak sa zamyslíme, čí patrí slovo $<A_k>$ do $L(A_k) = L_D^C$.

Definícia 6.7.2 ***Univerzálny jazyk*** *je jazyk* $L_U = \{<A>\#w \ ; \ w \in L(A)\}$.

Poznámka 6.7.4 Univerzálny jazyk je teda (voľne povedané) jazyk dvojíc (Turingov stroj A, slovo w) takých, že A akceptuje w.

Praktický príklad univerzálneho TS sú rôzne emulátory, pomocou ktorých vieme na počítači emulovať iný počítač. Podobným príkladom je interpreter, ktorý vie interpretovať ľubovoľný program v danom jazyku. Uvedomte si, že presne to isté bude robiť TS pre univerzálny jazyk – dostane popis programu v nami špecifikovanom jazyku, vstup a má daný program na danom vstupe simulovať.

Lema 6.7.3 *Univerzálny jazyk je rekurzívne vyčísliteľný.*

Dôkaz. Analogicky ako pre diagonálny jazyk, len tentokrát môžu byť kód stroja a vstupné slovo navzájom rôzne.

Lema 6.7.4 *Komplement univerzálneho jazyka nie je rekurzívne vyčísliteľný.*

Dôkaz. Sporom. Nech $L_U^C \in \mathscr{L}_{RE}$. Ukážeme, že aj $L_D^C \in \mathscr{L}_{RE}$, čo bude spor. Nech A je TS taký, že $L(A) = L_U^C$. Zostrojme TS A' nasledovne: Keď dostane na vstupe slovo w, prepíše ho na $w\#w$ a simuluje na ňom TS A. Zjavne $L(A') = L_D^C$.

Poznámka 6.7.5 Predchádzajúci dôkaz bol príkladom postupu, ktorý budeme v budúcnosti často používať – redukcie. Bližšie sa s týmto pojmom zoznámime v nasledujúcich častiach.

6.8 Rekurzívne jazyky, riešiteľnosť, rozhodnuteľnosť

Definícia 6.8.1 ***Algoritmus*** *je deterministický TS, ktorý na každom vstupe zastaví.*

Poznámka 6.8.1 Uvedomte si, že táto definícia je dosť abstraktná. Ako neskôr ukážeme, z kódu TS sa nedá zistiť, či tento TS na každom vstupe zastaví. (Učene povedané, ide o sémantické, nie o syntaktické obmedzenie.)

Definícia 6.8.2 *Symbolom $\mathscr{L}_{rec}$ označujeme triedu jazykov, pre ktoré existuje deterministický TS, ktorý zastaví na každom vstupe a akceptuje daný jazyk. Túto triedu voláme trieda* ***rekurzívnych jazykov****.*

Lema 6.8.1 *$\mathscr{L}_{rec}$ je podmnožinou $\mathscr{L}_{RE}$.*

Veta 6.8.2 *$\mathscr{L}_{rec}$ je uzavretá na komplement.*

Dôkaz. Zjavné. DTS pre L^C simuluje DTS pre L. Ten určite zastane, simulujúci TS už len zneguje jeho výstup.

Dôsledok 6.8.3 *Z lemy 6.8.1 a vety 6.8.2 vyplýva, že $\mathscr{L}_{rec} \subsetneq \mathscr{L}_{RE}$.*

Poznámka 6.8.2 Načrtneme tu teraz jednu predstavu o triedach $\mathscr{L}_{rec}$ a $\mathscr{L}_{RE}$. DTS, ktorý vždy zastaví si môžeme predstaviť ako čiernu krabičku, do ktorej vopcháme slovo a ona nám v konečnom čase odpovie *áno* alebo *nie* podľa toho, či slovo patrí do zodpovedajúceho jazyka. Všeobecný TS je krabička, u ktorej ak je odpoveď *áno*, tak ju určite dostaneme v konečnom čase, ale odpovede *nie* sa dočkať nemusíme.

Uvedomme si, že existencia áno-hovoriacej a nie-hovoriacej krabičky môžu byť principiálne dve rôzne veci. Koniec koncov, $\mathscr{L}_{RE}$ nie je uzavretá na komplement, takže pre niektoré jazyky môže existovať len jedna z týchto krabičiek.

Veta 6.8.4 *(Post) Nech $L \in \mathscr{L}_{RE}$ a $L^C \in \mathscr{L}_{RE}$. Potom $L \in \mathscr{L}_{rec}$.*

Dôkaz. Myšlienka: Majme dve čierne krabičky – jednu, ktorá určite v konečnom čase povie *áno*, ak $w \in L$, druhú, ktorá v konečnom čase povie *áno*, ak $w \in L^C$. Keď dostaneme vstupné slovo w, vopcháme ho do oboch krabičiek a naraz ich pustíme. V konečnom čase jedna z nich určite skončí, vieme odpoveď a skončíme.

Ešte raz formálnejšie. Nech A_1, A_2 sú DTS pre L, L^C. Zostrojíme dvojpáskový DTS A, ktorý na každom vstupe zastane a bude akceptovať jazyk L. Bude fungovať nasledovne: Skopíruje vstupné slovo w na druhú pásku. Striedavo simuluje jeden krok A_1 na prvej páske a jeden krok A_2 na druhej. Ak A_1 akceptuje, A akceptuje. Ak A_2 akceptuje, A sa zasekne a neakceptuje. Keďže buď $w \in L$ alebo $w \in L^C$, skôr či neskôr jeden zo simulovaných TS musí akceptovať, preto A vždy zastane.

Zopakujeme definíciu, ktorú sme uviedli v prvej kapitole pri voľnej diskusii o súvise medzi jazykmi a problémami. Veríme, že čitateľ s vedomosťami z predchádzajúcich kapitol už má dostatočný nadhľad, aby vedel ľubovoľný problém, kde je odpoveď áno/nie, zakódovať ako vhodný jazyk.

Definícia 6.8.3 ***Rozhodovací problém*** *môžeme definovať ako jazyk Y (nad pevne zvolenou abecedou) predstavujúci množinu tých vstupov, pre ktoré je odpoveď kladná.*

Poznámka 6.8.3 Uvedomte si, že (ak sa zaoberáme Turingovymi strojmi) v podstate nezáleží na konkrétnom kódovaní inštancii problému do slov jazyka. Totiž ľahko nahliadneme, že keď zakódujeme ten istý problém do jazyka dvoma rôznymi rozumnými[6] postupmi, bude existovať TS, ktorý bude schopný prekladať jeden kód na druhý. Ak teda existuje TS riešiaci problém zakódovaný jedným spôsobom, tak existuje TS riešiaci dotyčný problém zakódovaný ľubovoľným rozumným spôsobom – preloží kód zo vstupného na taký, pre ktorý už problém vieme riešiť a vyrieši ho.

Z tohto dôvodu budeme v nasledujúcom texte výrazy „jazyk“ a „rozhodovací problém“ (a tiež „akceptovať jazyk“ a „riešiť rozhodovací problém“) používať ako synonymá.

Čitateľ láskavo prepáči prudkú neformálnosť tohto zdôvodnenia, formalizmus by tu ale zasahoval ďaleko nad rámec tohto textu.

Definícia 6.8.4 *Hovoríme, že problém je* ***turingovsky riešiteľný****, ak existuje DTS, ktorý zastaví na každom vstupe a rieši daný problém.*

Poznámka 6.8.4 Uvedomte si, že táto definícia zahŕňa aj problémy, ktoré nie sú rozhodovacie (teda kde výstupom je niečo iné ako áno/nie). Okrem iného hovorí, že rozhodovací problém je turingovsky riešiteľný vtedy, ak jeho zakódovaním do jazyka dostaneme rekurzívny jazyk.

Téza 6.8.5 *(Church-Turing) Turingovská riešiteľnosť je ekvivalentná algoritmickej riešiteľnosti.*

Poznámka 6.8.5 Táto téza uvádza tvrdenie, ktoré už intuitívne dlhšiu dobu používame pri rôznych myšlienkach dôkazov a pod. Tézu principiálne nemôžeme dokázať – totiž nevieme presne, čo pojem algoritmická riešiteľnosť znamená. Jednou z motivácií k rozvoju teórie formálnych jazykov bolo práve skúsiť čo najlepšie vystihnúť, ktoré problémy sú riešiteľné a ktoré nie. Našlo sa veľa diametrálne odlišných prístupov (Turingove stroje, frázové gramatiky, čiastočne rekurzívne funkcie, Minského registrové stroje, lambda-kalkul, ...), no ukázalo sa, že všetky vedú k tak isto silnému modelu. Preto sa tento model prijal ako „definícia“ algoritmickej riešiteľnosti.

Poznámka 6.8.6 V ďalšom texte budeme namiesto „turingovská“ či „algoritmická riešiteľnosť“ hovoriť jednoducho riešiteľnosť.

Definícia 6.8.5 *Hovoríme, že problém je* ***rozhodnuteľný****, ak je to riešiteľný rozhodovací problém.*

Definícia 6.8.6 *Ak rozhodovací problém nie je rozhodnuteľný, hovoríme, že je* ***nerozhodnuteľný****.*

Definícia 6.8.7 *Hovoríme, že problém je* ***čiastočne rozhodnuteľný****, ak je to rozhodovací problém a jeho zakódovaním do jazyka dostaneme rekurzívne vyčísliteľný jazyk.*

Poznámka 6.8.7 Na tomto mieste je dôležité uvedomiť si, že rekurzívne jazyky zodpovedajú rozhodnuteľným a rekurzívne vyčísliteľné jazyky čiastočne rozhodnuteľným problémom. Tiež si všimnite, že problém môže byť čiastočne rozhodnuteľný, ale nerozhodnuteľný.

Príklad 6.8.1 Problém príslušnosti slova do regulárneho jazyka (daného konečným automatom) je rozhodnuteľný.

Dôkaz. Ľahko zostrojíme jazyk zodpovedajúci tomuto problému, napr.: $L = \{(\text{kód konečného automatu})\#(\text{kód slova})\}$ – oba kódy sú napr. nad abecedou $\{0, 1\}$. Túto časť dôkazu budeme v budúcnosti vynechávať – predpokladáme, že čitateľ si už dokáže predstaviť, ako zakódovať inštanciu ľubovoľného rozumného problému do konečnej abecedy.

Takisto ľahko zostrojíme algoritmus (DTS, ktorý na každom vstupe zastaví) pre tento jazyk – jednoducho odsimuluje automat, ktorého kód dostane na vstupe na slove, ktoré tiež dostane na vstupe a zistí, či dané slovo akceptuje alebo nie.

[6] T.j. takými, že kódovanie aj dekódovanie je algoritmický postup.

Príklad 6.8.2 Problém príslušnosti slova do rekurzívne vyčísliteľného jazyka je čiastočne rozhodnuteľný, ale nie je rozhodnuteľný.

Dôkaz. Jazyk zodpovedajúci tomuto problému je univerzálny jazyk L_U. Vieme, že $L_U \in \mathscr{L}_{RE}$, $L_U^C \notin \mathscr{L}_{RE}$. Potom napríklad z vety 6.8.2 vyplýva, že $L_U \notin \mathscr{L}_{rec}$.

6.9 Základné nerozhodnuteľné problémy

V predchádzajúcej časti sme dokázali, že problém príslušnosti slova do rekurzívne vyčísliteľného jazyka (univerzálny jazyk) a problém, či daný TS akceptuje svoj kód (diagonálny jazyk) nie sú rozhodnuteľné. Pri dôkaze, že $L_U \notin \mathscr{L}_{rec}$ sme použili myšlienku redukcie, ktorú teraz formálne zadefinujeme.

6.9.1 Turingovská a many-to-one redukcia

Začneme tým, že si formálne definujeme pojem redukcie. Tento postup budeme používať pri ukazovaní o problémoch, že nie sú riešiteľné. Samotný pojem „redukcia“ vznikol z nasledovnej predstavy: Ukážeme, že nový, neznámy problém vieme zredukovať (t.j. oslabiť) tak, že dostaneme známy nerozhodnuteľný problém.

Definícia 6.9.1 ***TS s orákulom*** *S (kde S je nejaký jazyk) je špeciálny dvojpáskový TS, ktorý má k dispozícii „čiernu krabičku“, ktorej sa vie ako jeden krok výpočtu opýtať a ona mu odpovie áno/nie podľa toho, či slovo na druhej páske patrí do S. (Štandardný TS je teda TS s orákulom $\emptyset$, t.j. bez čiernej krabičky.)*

Veta 6.9.1 *(Turingovská redukcia) Nech $L_1 \notin \mathscr{L}_{rec}$. Nech L_2 je ľubovoľný jazyk. Ak existuje DTS s orákulom L_2, ktorý vždy zastane a akceptuje L_1, tak $L_2 \notin \mathscr{L}_{rec}$.*

Dôkaz. Myšlienka vety preložená do ľudskej reči: Nech problém 1 nevieme rozhodovať. Nech ale vieme rozhodovať problém 1 pomocou čiernej krabičky rozhodujúcej problém 2. Potom nevieme rozhodovať ani problém 2. Myšlienka dôkazu: Keby sme vedeli rozhodovať problém 2, vieme zostrojiť dotyčnú čiernu krabičku a s jej pomocou rozhodovať problém 1. To ale nevieme.

To isté trochu formálnejšie. Sporom. Nech A je dotyčný DTS s orákulom, nech $L_2 \in \mathscr{L}_{rec}$. Potom pre L_2 existuje algoritmus B. Keď teraz upravíme DTS A tak, aby namiesto pýtania sa orákula odsimuloval B na svojej druhej páske, dostaneme DTS, ktorý vždy zastaví a akceptuje L_1. To je spor s $L_1 \notin \mathscr{L}_{rec}$. Preto $L_2 \notin \mathscr{L}_{rec}$, q.e.d.

Poznámka 6.9.1 Analogicky sa dá definovať Turingovská redukcia aj pre čiastočnú rozhodnuteľnosť: Ak $L_1 \notin \mathscr{L}_{RE}$, ale vieme preň zostrojiť TS pomocou TS pre L_2, tak aj $L_2 \notin \mathscr{L}_{RE}$.

Poznámka 6.9.2 Keď máme k dispozícii čiernu krabičku rozhodujúcu problém 2, môžeme ju pri rozhodovaní problému 1 použiť ľubovoľne (konečne) veľa krát. Pri problémoch, ktoré sú si podobné, však často bude stačiť jedno volanie tejto čiernej krabičky – jej odpoveď bude zároveň našou odpoveďou. V takomto prípade si vystačíme s jednoduchšou redukciou.

Veta 6.9.2 *(Many-to-one redukcia) Nech $L_1 \notin \mathscr{L}_{rec}$. Nech L_2 je ľubovoľný jazyk. Nech jazyk L_i je nad abecedou Σ_i. Ďalej nech $f : \Sigma_1^* \to \Sigma_2^*$ je funkcia zachovávajúca príslušnosť do jazyka – t.j. $\forall w;\ w \in L_1 \iff f(w) \in L_2$. Ak existuje DTS počítajúci funkciu f, tak $L_2 \notin \mathscr{L}_{rec}$.*

Dôkaz. Opäť najskôr preložíme dokazovanú vetu do ľudskej reči: Nech problém 1 nevieme rozhodovať. Ak vieme prekladať vstupy problému 1 na vstupy problému 2 tak, že prekladom sa nezmení odpoveď, nemôžeme vedieť rozhodovať ani problém 2 – potom by sme totiž vedeli rozhodovať problém 1 tak, že preložíme vstup a rozhodneme problém 2.

To isté trochu formálnejšie. Sporom. Nech M je DTS rátajúci funkciu f, nech A je DTS rozhodujúci L_2. Zostrojíme DTS A' rozhodujúci L_1 nasledovne: Na vstupnom slove w simuluje M. Keď ten akceptuje, na páske máme slovo $f(w)$. Na tom A' odsimuluje A.

6.9.2 Problém zastavenia

Predstavme si úlohu naprogramovať softvér, ktorý pre daný ľubovoľný program a dáta rozhodne, či program spustený na týchto dátach niekedy skončí alebo nie. Ukážeme smutnú skutočnosť: taký softvér nie sme schopní napísať.

Formalizáciou tohto problému je jazyk $L_{HALT} = \{<A>\#w \mid A \text{ na } w \text{ zastaví}\}$.

Veta 6.9.3 *$L_{HALT} \notin \mathscr{L}_{rec}$ (T.j. problém zastavenia nie je rozhodnuteľný.)*

Dôkaz. Zredukujeme L_{HALT} na L_U, t.j. ukážeme, že ak by sme vedeli rozhodovať L_{HALT}, vedeli by sme rozhodovať aj L_U. Majme teda čiernu krabičku rozhodujúcu L_{HALT}. Zostrojme DTS M, ktorý vždy zastaví a akceptuje L_U.

Náš DTS M dostane na vstupe nejaký kód TS A a slovo w, má rozhodnúť, či A akceptuje w. Zistí to nasledovne: Upraví kód A, presnejšie jeho prechodovú funkciu tak, aby sa namiesto zaseknutia zacyklil. (T.j. na koniec pripíše nové pravidlá pre tie dvojice (stav,písmeno), pre ktoré nebola prechodová funkcia definovaná.) Takto dostane kód nového TS A'. Pritom A' zjavne akceptuje w práve vtedy, ak na ňom zastane. Preto sa M opýta čiernej krabičky na vstup $<A'>\#w$. Dostane odpoveď, či A' na w zastane. To je ale zároveň odpoveď na otázku, či A akceptuje w – skončili sme, podľa odpovede M akceptuje alebo nie.

Poznámka 6.9.3 Predchádzajúci dôkaz používal Turingovskú redukciu. Šikovný čitateľ si už uvedomil, že stačilo použiť many-to-one redukciu. Zjavne vieme na TS počítať funkciu, ktorá pre slovo $<A>\#w$ vráti slovo $<A'>\#w$, kde A' je A upravený tak, aby sa zacyklil, ak neakceptuje. Zjavne $<A>\#w \in \mathscr{L}_U \iff <A'>\#w \in \mathscr{L}_{HALT}$. Preto $L_{HALT} \notin \mathscr{L}_{rec}$, q.e.d.

Poznámka 6.9.4 Problém zastavenia zjavne je čiastočne rozhodnuteľný. Z Postovej vety vyplýva, že jeho komplement nie je ani čiastočne rozhodnuteľný.

Poznámka 6.9.5 Ukážeme si ešte jeden dôkaz nerozhodnuteľnosti problému zastavenia, tentokrát bez pomoci redukcie. Náš dôkaz nebude úplne formálny. Nebudeme hovoriť o konkrétnom výpočtovom modeli (ako sú napr. Turingove stroje či frázové gramatiky), ale všeobecne o algoritmoch. Naším cieľom bude dokázať nasledovné tvrdenie:

Nech $F(P, x)$ je funkcia, ktorej vstupom je dvojica: program P a jeho vstup x. Pritom

$$F(P,x) = \begin{cases} 1 \leftarrow \text{ak } P \text{ na } x \text{ zastane} \\ 0 \leftarrow \text{inak} \end{cases}$$

Potom neexistuje žiaden algoritmus, počítajúci túto funkciu.

Pritom zanedbáme kopu detailov, ktoré by museli byť súčasťou formálneho dôkazu. Nebude nás veľmi trápiť, v akom jazyku je program P napísaný (je nám jedno, či je to Pascal alebo δ-funkcia Turingovho stroja). Takisto sa nebudeme zaoberať rôznymi abecedami, všetko predsa vieme zakódovať do postupnosti 0 a 1. Navyše sa dohodneme, že ak P nie je korektný program, tak $\forall x; F(P, x) = 0$.

Dôkaz bude sporom. Predpokladajme, že existuje algoritmus **Rozhodni**, rátajúci funkciu F. (Teda pre všetky P a x je **Rozhodni**$(P, x) = F(P, x)$.) Zostrojme potom algoritmus **Popleť**, ktorý bude využívať **Rozhodni** ako procedúru a bude fungovať nasledovne: Ak **Rozhodni**$(P, x) = 1$, **Popleť**(P, x) sa zacyklí, inak skončí. Teda **Popleť** na vstupe (P, x) zastane práve vtedy, keď sa P na vstupe x zacyklí.

Základný trik dôkazu je podstrčiť programu na vstup jeho vlastný zdrojový kód. Vieme, že **Rozhodni**(P, P) nám povie, či P na svojom kóde zastane alebo nie. Čo potom robí algoritmus **Naopak**(P) = **Popleť**(P, P)? Ak P na svojom kóde zastane, **Popleť**(P, P) (čiže **Naopak**(P)) sa zacyklí. Ak sa P na svojom kóde zacyklí, **Popleť**(P, P) zastane.

Preto sa algoritmus **Naopak** na vstupe P správa (naozaj) naopak ako algoritmus P na vstupe P. Ale tu sa už blížime k hľadanému sporu. **Naopak** sa totiž nemôže správať naopak sám od seba!

Zistime teda, čo sa stane, keď spustíme **Naopak** na jeho vlastnom zdrojovom kóde. Ak **Naopak**(**Naopak**) zastane, znamená to, že **Naopak** sa na vstupe **Naopak** zacyklil, čo je spor. A

ak **Naopak**(**Naopak**) nezastane, znamená to, že **Naopak** na vstupe **Naopak** zastal, čo je opäť spor.

Všetky kroky nášho dôkazu boli korektné a dospeli sme k sporu. Preto nutne pôvodný predpoklad nebol správny, čiže algoritmus **Rozhodni** neexistuje.

6.9.3 Postov korešpondenčný problém

Daných je konečne veľa typov domín. Domín každého typu máme k dispozícii neobmedzene veľa. Každé domino sa skladá z horného a dolného políčka, v každom políčku je napísané nejaké slovo. Keď poskladáme niekoľko domín do radu vedľa seba, na hornom aj dolnom poschodí dostaneme nejaké slovo. Úlohou je pre danú sadu domín zistiť, či sa dajú domináposkladať do radu tak, aby sa tieto dve slová rovnali.

Formálna verzia tejto úlohy: Nech $X = (x_1, \ldots, x_n)$ a $Y = (y_1, \ldots, y_n)$, kde $x_i, y_i \in \Sigma^+$. (Slovo x_i je napísané na hornom poschodí domín i-teho typu, slovo y_i je na ich dolnom poschodí.) Úlohou je zistiť, či existuje $k \geq 1$ a postupnosť indexov $i_1, \ldots, i_k \in \{1, \ldots, n\}$ také, že $x_{i_1} \ldots x_{i_k} = y_{i_1} \cdots y_{i_k}$.

Poznámka 6.9.6 Tejto úlohe hovoríme Postov korešpondenčný problém alebo skrátene PKP. Asi neuškodí pár slov, odkiaľ sa tento trochu zvláštny názov vzal: „Postov“ je podľa jeho autora, ktorý sa volal Post, „korešpondenčný“ vzniklo z anglického correspond (zodpovedať) – slovo poskladané na hornom poschodí musí zodpovedať slovu na spodnom poschodí.

Poznámka 6.9.7 Dvojicu (X, Y) budeme volať *inštancia PKP*. Aby nedošlo k omylu, i-te domino (obsahujúce slová x_i hore a y_i dole) budeme značiť $[x_i, y_i]$.

Ukážeme, že tento problém nie je rozhodnuteľný. Zdôrazňujeme, že riešením tohto problému nie je hľadanie indexov i_j, ale len odpoveď, či taká postupnosť indexov existuje. Najskôr tento problém jemne modifikujeme a výsledný problém potom zredukujeme na problém zastavenia TS.

Modifikovaný Postov korešpondenčný problém (MPKP) je rovnaký ako PKP, ale pýtame sa, či existuje riešenie, v ktorom $i_1 = 1$ (teda začíname prvým dominom).

Poznámka 6.9.8 Každé riešenie modifikovaného PKP je aj riešením PKP. Ak v PKP existuje riešenie, v modifikovanom PKP riešenie existovať nemusí. Napríklad pre sadu domín

aa	aaa	b
bb	aa	ab

Veta 6.9.4 *Ak je rozhodnuteľný MPKP, tak je rozhodnuteľný aj PKP.*

Dôkaz. Nech je MPKP rozhodnuteľný. Ukážeme, že potom je rozhodnuteľný aj PKP. Nech A je TS rozhodujúci MPKP. Potom TS A' rozhodujúci PKP zostrojíme nasledovne: Dostane vstup (X, Y). Postupne bude n-krát spúšťať A, pričom pri i-tom spustení mu dá na vstup (X_i, Y_i), kde $X_i = (x_i, \ldots, x_n, x_1, \ldots, x_{i-1})$, Y_i analogicky. Teda A' sa postupne opýta A, či existuje riešenie začínajúce prvým, druhým, ..., n-tým dominom. Ak niekedy dostane odpoveď áno, dá aj A' odpoveď áno a skončí. Ak vždy dostane odpoveď nie, skončí s odpoveďou nie.

Veta 6.9.5 *Ak je rozhodnuteľný PKP, tak je rozhodnuteľný aj MPKP.*

Dôkaz. Nech je PKP rozhodnuteľný. Ukážeme, že potom je rozhodnuteľný aj MPKP. Nech A je TS rozhodujúci PKP. Potom TS A' rozhodujúci MPKP zostrojíme nasledovne: Dostane vstup (X, Y). Tento vstup teraz upravíme tak, aby sme sa zbavili riešení, ktoré začínajú iným ako prvým dominom. Všetky riešenia, ktoré zostanú, budú zodpovedať práve riešeniam pôvodného MPKP. Na takto upravený vstup sa opýtame TS A a tak zistíme, či má daný MPKP riešenie.

Zoberme si prvé domino $[x_1, y_1]$ a urobme z neho $[\#\overline{x_1}\#, \#\overline{y_1}]$, kde znak # dáme medzi každé dve písmená vnútri slov x_1 a y_1. Toto bude naše „štartovacie domino“. Teraz upravíme všetky domináy (vrátane prvého) tak, aby sa dali za seba napájať: Teda domino $[x_i, y_i]$ prerobíme na

$[\overline{x_i}\#, \#\overline{y_i}]$, kde znak # dáme opäť medzi každé dve písmená vnútri slov x_i a y_i. Na záver ešte vyrobíme domino $[\#, \#\#]$, ktorým môžeme ukončiť prikladanie domín.

Formálne: Majme sadu domín (X, Y), $|X| = n$. Vytvoríme sadu (X', Y') s $n + 2$ dominami nasledovne: Nech p, z sú homomorfizmy také, že $\forall x \in \Sigma$; $p(x) = \#x$ a $z(x) = x\#$. (Teda p dáva # pred a z za každé písmeno zobrazeného slova.) Bude:
$x'_1 = \#z(x_1)$, $y'_1 = p(y_1)$
$x'_{i+1} = z(x_i)$, $y'_{i+1} = p(y_i)$ (pre $i \in \{1, \ldots, n\}$)
$x'_{n+2} = \#$, $y'_{n+2} = \#\#$

Ľahko nahliadneme, že každé riešenie MPKP pre (X, Y) zodpovedá riešeniu PKP pre (X', Y'). A naopak, každé riešenie PKP pre (X', Y') musí zjavne začínať prvým dominom[7] a teda zodpovedá nejakému riešeniu MPKP pre (X, Y). Preto naozaj stačí, keď rozhodneme, či má PKP riešenie pre (X', Y').

Príklad 6.9.1 Riešenia MPKP pre sadu domín

bbaa	aaa	b
bb	aa	ab

zodpovedajú riešeniam PKP pre sadu domín

#b#b#a#a#	b#b#a#a#	a#a#a#	b#	#
#b#b	#b#b	#a#a	#a#b	##

Dôsledok 6.9.6 *PKP je rozhodnuteľný práve vtedy, ak je rozhodnuteľný MPKP.*

Veta 6.9.7 *MPKP je nerozhodnuteľný.*

Dôkaz. Sporom – redukciou na univerzálny jazyk. Predpokladajme, že je MPKP rozhodnuteľný, t.j. existuje algoritmus (TS A), ktorý pre každú inštanciu 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 M a slovo w. Na základe tohto vstupu A' vyrobí domináa, ktoré budú simulovať výpočet TS M na slove w. Riešenia MPKP s týmito dominami budú zodpovedať práve akceptačným výpočtom M na w. Ak by sme teda vedeli rozhodovať MPKP, tak by bol $L_U \in \mathscr{L}_{rec}$, čo bude hľadaný spor.

Ostáva ukázať, ako zostrojiť hľadanú sadu domín. Myšlienka: Horné aj dolné slovo bude predstavovať postupnosť konfigurácií TS A na w. Počas simulovania výpočtu bude horné slovo obsahovať o jednu konfiguráciu viac ako dolné. Tvar prikladaných domín zabezpečí, že ďalšia konfigurácia na hornom poschodí naozaj vznikne z predchádzajúcej krokom výpočtu.

K_0	K_1	$\ldots$	K_{n-1}	K_n
K_0	K_1	$\ldots$	K_{n-1}	

V takejto situácií môžeme prikladať len domináa, ktorých dolné slovo „pasuje“ na n-tú konfiguráciu. Ich horné slová vytvoria na konci horného poschodia $(n + 1)$. konfiguráciu.

Naším prvým dominom (ktorým podľa definície MPKP musí riešenie začínať) bude $[\#q_0w\#, \#]$ – na hornom poschodí máme začiatočnú konfiguráciu TS M, dolné je zatiaľ prázdne. Budeme používať znaky # na označenie, kde končí jedna konfigurácia a začína nasledujúca.

Celú konfiguráciu nemôžeme skopírovať na jeden krok (možných konfigurácií je nekonečne veľa), preto budeme mať sadu kopírovacích domín $[x, x]$ pre $x \in \Gamma$ a domino $[\#, \#]$, ktorým začneme kopírovanie ďalšej konfigurácie.

Chýba nám najdôležitejšia časť domín – tá, ktorá bude simulovať krok výpočtu M. Pre každý riadok δ-funkcie M budeme mať domino (resp. sadu domín), ktoré ho simuluje:
Ak $\delta(q, a) = (p, b, 1)$, tak budeme mať domino $[bp, qa]$.

[7] Jediný iný kandidát je posledné domino, všetky ostatné majú rôzny prvý znak horného a dolného slova. Zjavne ale začať posledným dominom k riešeniu nevedie.

Ak $\delta(q,a) = (p,b,0)$, tak budeme mať domino $[pb, qa]$.
Ak $\delta(q,a) = (p,b,-1)$, tak budeme mať sadu domín $[pcb, cqa]$ pre $c \in \Gamma$.

Ak napr. v n-tej konfigurácií je M v stave q na písmene a, tak môžeme pri odvádzaní $(n+1)$. konfigurácie priložiť domino $[bp, qa]$, čím odsimulujeme použitie $\delta(q,a) = (p,b,1)$.

Treba ešte doriešiť situácie, kedy je hlava TS na blanku na začiatku alebo konci pásky. Vtedy sa nám hodí zarážka #. Dostaneme nasledujúce domináa:
Ak $\delta(q,a) = (p,b,-1)$, tak budeme mať navyše domino $[\#p\mathbf{B}b, \#qa]$.
Ak $\delta(q,\mathbf{B}) = (p,b,1)$, tak budeme mať navyše domino $[bp\#, q\#]$.
Ak $\delta(q,\mathbf{B}) = (p,b,0)$, tak budeme mať navyše domino $[pb\#, q\#]$.
Ak $\delta(q,\mathbf{B}) = (p,b,-1)$, tak budeme mať navyše sadu domín $[pcb\#, cq\#]$.

Prikladaním takýchto domín teda zjavne vieme simulovať výpočet TS M (a nič iné). Zostáva zabezpečiť, aby sme vedeli odvodenie ukončiť, keď sa simulovaný M dostane do akceptačného stavu. Zavedieme nový symbol Q a sadu domín $[Q, q_F]$ pre $q_F \in F$. Akonáhle sa v hornom slove vyskytne Q, simulácia výpočtu končí a ideme zabezpečiť, aby dolné slovo „dobehlo“ horné.

Na to nám stačí pridať domináa $[Q, Qa]$ a $[Q, aQ]$ pre $a \in \Gamma \cup \{\mathbf{B}\}$. V každej ďalšej kópií konfigurácie na hornom poschodí bude o jeden symbol menej ako v predchádzajúcej. Časom sa dostaneme do situácie, kedy je horné slovo od dolného dlhšie len o $Q\#$. Posledné naše domino bude $[\#, Q\#\#]$, ktorým môžeme v takejto situácií prikladanie domín ukončiť.

Zjavne riešenia MPKP s týmito dominami zodpovedajú práve akceptačným výpočtom M na w. Ak by sme teda vedeli rozhodovať MPKP, tak by bol $L_U \in \mathscr{L}_{rec}$, čo je spor.

Dôsledok 6.9.8 *PKP je nerozhodnuteľný.*

6.10 Rozhodnuteľnosť otázok o jazykoch známych tried

V tejto časti sa budeme zaoberať rozhodnuteľnosťou niekoľkých základných otázok o jazykoch tried Chomského hierarchie. Jazyk bude na vstupe zadaný gramatikou príslušného typu, ktorá ho generuje. (Alebo ekvivalentne môže byť zadaný automatom, ktorý ho akceptuje. Tieto dva spôsoby sú zjavne ekvivalentné, keďže gramatiky a automaty zodpovedajúcich si typov vieme medzi sebou algoritmicky prevádzať.) Bude nás zaujímať, či zadaná gramatika generuje prázdnu množinu, konečný resp. nekonečný jazyk, Σ^*, či je prienik dvoch bezkontextových jazykov prázdny a podobne. Ukážeme, ktoré z týchto problémov sú rozhodnuteľné a ktoré nie.

Na úvod uvedieme tabuľku, v ktorej o každej z klasických tried jazykov a každom nami uvažovanom probléme uvedieme, či je pre jazyky z tejto triedy rozhodnuteľný (R), nerozhodnuteľný (N), prípadne či je triviálny[8] (T).

	Problém	$\mathcal{R}$		$\mathscr{L}_{CF}$		$\mathscr{L}_{ECS}$		$\mathscr{L}_{RE}$
1.	prázdnosť $L(G)$	R	$\Leftarrow$	R		N	$\Rightarrow$	N
2.	konečnosť $L(G)$	R	$\Leftarrow$	R		N	$\Rightarrow$	N
3.	$L(G) = \Sigma^*$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
4.	$L(G_1) = L(G_2)$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
5.	$L(G_1) \subseteq L(G_2)$	R		N	$\Rightarrow$	N	$\Rightarrow$	N
6.	$L(G_1) \cap L(G_2) = \emptyset$	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
9.	$w \in L(G)$	R	$\Leftarrow$	R	$\Leftarrow$	R		N

Celú tabuľku T si môžeme predstaviť ako dvojrozmerné pole a indexovať ho $T[p, \mathscr{L}]$, kde p je číslo problému, teda riadok a $\mathscr{L}$ je trieda jazykov, teda stĺpec. Doplňme ešte tabuľku o informácie, že $T[3, \mathscr{L}_{CF}] \Rightarrow T[4, \mathscr{L}_{CF}]$ a $T[4, \mathscr{L}_{CF}] \Rightarrow T[5, \mathscr{L}_{CF}]$. Dokážme teraz tvrdenia uvedené v tejto tabuľke:

Poznámka 6.10.1 Všetky tvrdenia znázornené v tabuľke $\Rightarrow$ by mali byť zrejmé – ak problém nie je rozhodnuteľný pre menšiu triedu jazykov, nebude rozhodnuteľný ani pre väčšiu (ktorá tú menšiu

[8] T.j. vždy je tá istá odpoveď.

obsahuje). A naopak, ak problém je rozhodnuteľný pre danú triedu jazykov, bude rozhodnuteľný aj pre menšie triedy (de facto tým istým algoritmom). Uvedomte si, že v 8. riadku ide zakaždým o iný problém.

Veta 6.10.1 *$T[1, \mathscr{L}_{CF}]$: Problém prázdnosti bezkontextového jazyka je rozhodnuteľný.*

Dôkaz. Zostrojíme množinu neterminálov, z ktorých sa v danej gramatike dá odvodiť terminálne slovo (viď lemu 3.1.4). Zjavne jazyk generovaný gramatikou je prázdny iff začiatočný neterminál v tejto množine nie je.

Lema 6.10.2 *$T[1, \mathscr{L}_{RE}]$: Problém prázdnosti rekurzívne vyčísliteľného jazyka je nerozhodnuteľný.*

Dôkaz. Dopustíme sa menšej nekorektnosti – túto lemu dokážeme až v nasledujúcej časti ako vetu 6.11.4.

Veta 6.10.3 *$T[1, \mathscr{L}_{ECS}]$: Problém prázdnosti kontextového jazyka je nerozhodnuteľný.*

Dôkaz. Podľa vety 5.5.7 ku každému jazyku $L \in \mathscr{L}_{RE}$ existuje homomorfizmus h a jazyk $L' \in \mathscr{L}_{ECS}$ také, že $L = h(L')$. Navyše dôkaz spomenutej vety nám dáva algoritmický postup, ako z frázovej gramatiky pre L zostrojiť daný homomorfizmus a rozšírenú kontextovú gramatiku pre L'.

Zostáva si uvedomiť, že L je prázdny práve vtedy, keď je prázdny L'. Teda ak by sme vedeli rozhodovať prázdnosť pre jazyky z $\mathscr{L}_{ECS}$, vedeli by sme ju rozhodovať pre všetky jazyky z $\mathscr{L}_{RE}$ – to ale nevieme.

Veta 6.10.4 *$T[2, \mathscr{L}_{CF}]$: Problém konečnosti bezkontextového jazyka je rozhodnuteľný.*

Dôkaz. Z pumpovacej lemy (veta 3.11.1) vieme, že každé slovo dlhšie ako nejaké p (ktoré vieme z G vypočítať) sa dá rozdeliť a napumpovať. Zjavne akonáhle $L(G)$ obsahuje slovo w dlhšie ako p, je $L(G)$ nekonečný – napumpovaním w na mocniny ≥ 2 dostaneme nekonečne veľa navzájom rôznych slov z $L(G)$. Naopak, ak $L(G)$ obsahuje len slová dĺžky $\leq p$, je zjavne konečný, lebo takých slov nad T je len konečne veľa.

Stačí teda vypočítať p a zistiť, či $L(G)$ obsahuje slovo dlhšie ako p. Zostrojíme gramatiku G' pre jazyk $L(G) \cap T^{p+1}T^*$. (Jazyk $T^{p+1}T^*$ je regulárny, G' ľahko zostrojíme.[9]) Teraz už len pre G' rozhodneme, či generuje neprázdny jazyk.

Veta 6.10.5 *$T[2, \mathscr{L}_{ECS}]$: Problém konečnosti kontextového jazyka je nerozhodnuteľný.*

Dôkaz. Ukážeme, že keby sme o každom jazyku z $\mathscr{L}_{ECS}$ vedeli rozhodnúť, či je konečný, vedeli by sme rozhodnúť, či je prázdny. Nech L je jazyk, o ktorom chceme rozhodnúť, či je prázdny. Jazyk $L' = L\{a\}^*$ je zjavne z $\mathscr{L}_{ECS}$. Ak L je prázdny, L' je prázdny, inak je L' nekonečný.

Veta 6.10.6 *$T[3, \mathscr{L}_{CF}]$: Pre danú bezkontextovú gramatiku je nerozhodnuteľné, či generuje Σ^*.*

Dôkaz. Redukciou na PKP. Majme na vstupe sadu domín $(X = (x_1, \ldots, x_n), Y = (y_1, \ldots, y_n))$ – t.j. inštanciu PKP. Zostrojíme dve bezkontextové gramatiky G_1, G_2 tak, aby ich slová zodpovedali možným priloženiam domín. $L(G_1)$ bude popisovať horné poschodie, $L(G_2)$ dolné. Cieľom je dosiahnuť, aby každé slovo z prieniku zodpovedalo jednému korektnému priloženiu domín a naopak.

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

$$L(X) = \{i_k \ldots i_2 i_1 \# x_{i_1} x_{i_2} \ldots x_{i_k} \mid k \geq 1 \ \wedge \ i_j \in \{1, \ldots, n\}\}$$

[9] Napr.: Z G zostrojíme ekvivalentný zásobníkový automat, zostrojíme konečný automat pre $T^{p+1}T^*$, kartézskym súčinom zostrojíme automat pre prienik ich jazykov a ten prevedieme späť na bezkontextovú gramatiku.

Každé slovo z $L(X)$ vlastne popisuje jedno slovo poskladané zo slov množiny X aj s indexmi slov, z ktorých vzniklo. Všimnite si, že indexy sú uvedené v opačnom poradí. Vďaka tomu $L(X) \in \mathscr{L}_{DPDA}$, a teda $L(X)$ je bezkontextový. Analogicky definujeme $L(Y)$. No a zjavne v $L(X) \cap L(Y)$ sú práve tie slová, kde tá istá postupnosť indexov dá to isté slovo, teda práve riešenia príslušnej inštancie PKP.

Keďže jazyky $L(X)$, $L(Y)$ patria do $\mathscr{L}_{DPDA}$, aj jazyky $L(X)^C$, $L(Y)^C$ sú v $\mathscr{L}_{DPDA}$, a teda sú bezkontextové. Potom je ale aj jazyk $(L(X) \cap L(Y))^C = L(X)^C \cup L(Y)^C$ bezkontextový. Keby sme o ňom vedeli rozhodnúť, či je rovný Σ^*, vedeli by sme rozhodnúť, či je $L(X) \cap L(Y)$ prázdny, a teda či má inštancia PKP (X, Y) riešenie.

Veta 6.10.7 *$T[3, \mathcal{R}]$: Pre danú regulárnu gramatiku je rozhodnuteľné, či generuje Σ^*.*

Dôkaz. Trieda $\mathcal{R}$ je uzavretá na komplement. Zostrojíme gramatiku pre komplement a rozhodneme, či generuje prázdny jazyk. Analogicky bude vyzerať dôkaz tvrdení $T[5, \mathcal{R}]$ a $T[6, \mathcal{R}]$.

Poznámka 6.10.2 Vzťah $T[4, \cdot]$ a $T[5, \cdot]$: Ak je pre gramatiky daného typu rozhodnuteľné, či $L(G_1) \subseteq L(G_2)$, tak je rozhodnuteľná aj rovnosť generovaných jazykov – rozhodneme, či $L(G_1) \subseteq L(G_2)$ a či $L(G_2) \subseteq L(G_1)$. Inými slovami, ak nie je rozhodnuteľná rovnosť generovaných jazykov, nie je rozhodnuteľné ani či jeden je podmnožinou druhého.

Veta 6.10.8 *$T[5, \mathcal{R}]$: Pre dané regulárne gramatiky G_1, G_2 je rozhodnuteľné, či $L(G_1) \subseteq L(G_2)$.*

Dôkaz. Rozhodneme, či je prázdny jazyk $L(G_1) \setminus L(G_2) = L(G_1) \cap L(G_2)^C$.

Veta 6.10.9 *$T[4, \mathscr{L}_{CF}]$: Pre dané bezkontextové gramatiky G_1, G_2 je nerozhodnuteľné, či $L(G_1) = L(G_2)$.*

Dôkaz. Keby bol tento problém rozhodnuteľný, vedeli by sme rozhodnúť o ľubovoľnej bezkontextovej gramatike G, či generuje Σ^*. (Rozhodli by sme, či G generuje rovnaký jazyk ako triviálna bezkontextová gramatika pre Σ^*.) Ako sme ale ukázali v tvrdení $T[3, \mathscr{L}_{CF}]$, to rozhodovať nevieme.

Veta 6.10.10 *$T[6, \mathcal{R}]$: Problém prázdnosti prieniku dvoch regulárnych jazykov je rozhodnuteľný.*

Dôkaz. Trieda $\mathcal{R}$ je uzavretá na prienik. Zostrojíme gramatiku pre prienik daných jazykov a rozhodneme, či generuje prázdny jazyk.

Veta 6.10.11 *$T[6, \mathscr{L}_{CF}]$: Problém prázdnosti prieniku dvoch bezkontextových jazykov je nerozhodnuteľný.*

Poznámka 6.10.3 Úlohou je zistiť pre dané $L_1, L_2 \in \mathscr{L}_{CF}$, či $L_1 \cap L_2 = \emptyset$. Tento problém sa tiež niekedy podáva ako problém frustrovaného programátora takto: Chudák programátor zistil, že každý kompilátor jeho jazyka[10] je iný a nie všetko mu ten či onen zožerie. Normy nie sú až tak jednotné a tak chce vedieť, či vôbec existuje program, ktorý skompilujú dva rôzne kompilátory.

Dôkaz. Redukciou na PKP. Majme na vstupe inštanciu PKP $(X = (x_1, \ldots, x_n), Y = (y_1, \ldots, y_n))$. Zostrojíme bezkontextové gramatiky pre jazyky $L(X)$, $L(Y)$, ktoré sme definovali vyššie. Prienik týchto dvoch jazykov je prázdny práve vtedy, ak naša inštancia PKP nemá riešenie. Teda ak by sme vedeli pre ľubovoľné dve bezkontextové gramatiky rozhodnúť, či je prienik nimi generovaných jazykov prázdny, vedeli by sme rozhodnúť PKP.

Veta 6.10.12 *$T[7, \mathscr{L}_{CF}]$: Pre danú bezkontextovú gramatiku je nerozhodnuteľné, či generuje regulárny jazyk.*

[10] Predpokladajme, že dotyčný programovací jazyk je bezkontextový.

Dôkaz. Jazyk $L(X) \cap L(Y)$ je buď prázdny, alebo nekonečný. Rozmyslite si, že ak je nekonečný, tak nie je regulárny – neplatí preň pumpovacia lema. Teda $L(X) \cap L(Y)$ je regulárny iff inštancia PKP (X, Y) nemá riešenie. Jazyk $(L(X) \cap L(Y))^C$ je bezkontextový. Ak inštancia PKP (X, Y) nemá riešenie, je to Σ^*, a teda je regulárny. Ak riešenie má, tento jazyk regulárny nie je. (Je to komplement neregulárneho jazyka.) Keby sme teda vedeli rozhodnúť, či daný bezkontextový jazyk je regulárny, vedeli by sme rozhodovať PKP.

Veta 6.10.13 *$T[8, \mathscr{L}_{CF}]$: Pre dané bezkontextové gramatiky G_1, G_2 je nerozhodnuteľné, či jazyk $L(G_1) \cap L(G_2)$ je bezkontextový.*

Dôkaz. Opäť redukciou na PKP. Definujme tentokrát jazyky $L(X, Y) = L(X)\#L(Y)^R$, $L_{sym} = \{w \mid w = w^R\}$. Ľahko nahliadneme, že oba tieto jazyky sú bezkontextové. Zjavne opäť slová z jazyka $L = L(X, Y) \cap L_{sym}$ zodpovedajú práve riešeniam inštancie PKP (X, Y). Ak táto inštancia nemá riešenie, $L = \emptyset$, a teda je bezkontextový. V opačnom prípade ľahko nahliadneme, že L nespĺňa pumpovaciu lemu pre bezkontextové jazyky, a teda nie je bezkontextový.

Veta 6.10.14 *$T[9, \mathscr{L}_{ECS}]$: Problém príslušnosti slova do kontextového jazyka je rozhodnuteľný.*

Dôkaz. Budeme hľadať najkratšie odvodenie daného slova v rozšírenej kontextovej gramatike, ktorá generuje daný jazyk. V najkratšom odvodení sa žiadna vetná forma neopakuje. Navyše vďaka tomu, že vstupná gramatika je kontextová, žiadna vetná forma počas odvodenia nebude dlhšia ako výsledné slovo. Takýchto odvodení je ale zjavne konečne veľa a stačí ich všetky vyskúšať.

Poznámka 6.10.4 Uvedomte si, že nie každá bezkontextová gramatika je aj rozšírená kontextová. Každú bezkontextovú gramatiku však vieme odepsilonovať a výsledná ekvivalentná gramatika už je rozšírená kontextová. Preto naozaj $T[9, \mathscr{L}_{ECS}] \Rightarrow T[9, \mathscr{L}_{CF}]$. Napriek tomu ešte explicitne uvedieme dva odlišné postupy, ako rozhodovať príslušnosť slova do bezkontextového jazyka.

Veta 6.10.15 *$T[9, \mathscr{L}_{CF}]$: Problém príslušnosti slova do bezkontextového jazyka je rozhodnuteľný.*

Dôkaz. Vstupom je gramatika G a slovo w. Upravíme G do Greibachovej normálneho tvaru (veta 3.5.3). Vieme, že odvodenie w v upravenej gramatike má najviac $|w|$ krokov (resp. jeden krok, ak $w = \varepsilon$). Takýchto odvodení je ale len konečne veľa, tak ich vyskúšame a zistíme, či $w \in L(G)$.

Iný možný postup je použiť dynamické programovanie – algoritmus pánov Cockeho, Youngera a Kasamiho (viď časť 3.12). Podotknime, že tento algoritmus je na rozdiel od predchádzajúceho *efektívny*,[11] ale z hľadiska rozhodnuteľnosti sú ekvivalentné – tu nás totiž časová zložitosť nezaujíma, stačí nám ľubovoľne pomalý algoritmus riešiaci daný problém.

Poznámka 6.10.5 Tvrdenie $T[9, \mathscr{L}_{RE}]$ sme dokázali v príklade 6.8.2.

Poznámka 6.10.6 Navyše ešte uvedieme dva problémy. Nech G_1 je regulárna gramatika, G_2 je bezkontextová. Potom problém $L(G_1) \subseteq L(G_2)$ je nerozhodnuteľný (vedeli by sme rozhodovať $L(G_2) = \Sigma^*$), ale problém $L(G_2) \subseteq L(G_1)$ je rozhodnuteľný (rozhodneme, či je jazyk $L(G_2) \cap L(G_1)^C$ prázdny).

6.11 Riceove vety

Definícia 6.11.1 *Nech A je TS. Potom **jazyk platných výpočtov** TS A je jazyk*

$$LPV_A = \left\{ w_1 \# w_2^R \# w_3 \# \dots w_n^{R^{(n+1) \bmod 2}} \;\middle|\; n \geq 1, w_i \text{ sú konfigurácie } A,\ w_1 \text{ je začiatočná},\ w_n \text{ je akceptačná},\ \forall i; w_i \vdash w_{i+1} \right\}$$

[11] Jeho časová zložitosť je polynomiálna od veľkosti vstupu. Časová zložitosť predchádzajúceho algoritmu je exponenciálna.

Definícia 6.11.2 ***Jazyk neplatných výpočtov*** *TS A je $LNV_A = LPV_A^C$.*

Poznámka 6.11.1 Jazyk LPV_A nemusí byť bezkontextový.

Lema 6.11.1 *Jazyk LNV_A je vždy bezkontextový.*

Dôkaz. Zásobníkový automat si nedeterministicky tipne, kde je chyba a overí, že naozaj. Chyba môže byť buď v tom, že prvá konfigurácia nie je začiatočná, posledná nie je akceptačná (oboje vieme overiť triviálne), alebo že dve po sebe uvedené konfigurácie na seba nenadväzujú (to vieme overiť vďaka tomu, že každá druhá konfigurácia je reverznutá).

Lema 6.11.2 *K ľubovoľnému TS A existujú bezkontextové jazyky L_1, L_2 také, že $LPV_A = L_1 \cap L_2$.*

Dôkaz. Prvý jazyk zabezpečí, že $\forall i;\ w_{2i} \vdash w_{2i+1}$, druhý jazyk zvyšok.

Lema 6.11.3 $L(A) = \emptyset \iff LPV_A = \emptyset$

Definícia 6.11.3 *Každú podmnožinu S triedy $\mathscr{L}_{RE}$ budeme volať* ***vlastnosť*** *rekurzívne vyčísliteľných jazykov. O jazykoch z S hovoríme, že vlastnosť S majú, o ostatných jazykoch hovoríme, že túto vlastnosť nemajú.*

Poznámka 6.11.2 Analogicky vieme definovať vlastnosť ľubovoľnej triedy jazykov.

Definícia 6.11.4 *Hovoríme, že vlastnosť S triedy $\mathscr{L}_{RE}$ je* ***rozhodnuteľná****, ak je jazyk $L_S = \{<A> \mid L(A) \in S\}$ rekurzívny. (Teda ak o každom kóde TS vieme rozhodnúť, či ním akceptovaný jazyk vlastnosť S má alebo nemá.)*

Veta 6.11.4 *Prázdnosť rekurzívne vyčísliteľného jazyka je nerozhodnuteľná.*

Dôkaz. Redukciou na univerzálny jazyk. Nech vieme rozhodovať prázdnosť pre rekurzívne vyčísliteľné jazyky. Zostrojme DTS pre L_U, ktorý na každom vstupe zastane.

Náš stroj dostane na vstupe $<A>$ a w. Z nich zostrojí kód takého TS A_w, ktorý pracuje nasledovne: odignoruje vstup, ktorý dostane, prepíše ho slovom w (ktoré má „zadrôtované“ v prechodovej funkcii) a simuluje na ňom A. Zjavne kód takéhoto A_w vieme z $<A>$ a w algoritmicky zostrojiť.

Načo nám ale bude? Všimnime si, že $L(A_w)$ je buď $\emptyset$ (ak $w \notin L(A)$), alebo Σ^* (ak $w \in A$). Ak by sme vedeli rozhodovať prázdnosť pre TS, vedeli by sme rozhodnúť, či je $L(A_w)$ prázdny, a teda či A akceptuje w.

Poznámka 6.11.3 Prázdnosť pre $\mathscr{L}_{RE}$ nie je ani čiastočne rozhodnuteľná. Ale komplement, t.j. neprázdnosť čiastočne rozhodnuteľná je – uhádneme slovo z jazyka a odsimulujeme na ňom príslušný TS.

Položme si otázku, ktoré vlastností rekurzívne vyčísliteľných jazykov vlastne sú rozhodnuteľné. Ukážeme, že práve predvedeným spôsobom vieme dokázať nerozhodnuteľnosť takmer všetkých vlastností rekurzívne vyčísliteľných jazykov.

Definícia 6.11.5 *Vlastnosť S jazykov z triedy $\mathscr{L}$ sa nazýva* ***netriviálna****, ak existujú $L_1, L_2 \in \mathscr{L}$ také, že $L_1 \in S$ a $L_2 \notin S$. V opačnom prípade hovoríme, že S je* ***triviálna****.*

Veta 6.11.5 *(Rice) Každá netriviálna vlastnosť rekurzívne vyčísliteľných jazykov je nerozhodnuteľná.*

Dôkaz. Redukciou na univerzálny jazyk. Nech S je netriviálna vlastnosť $\mathscr{L}_{RE}$, ktorú vieme rozhodovať. BUNV nech $\emptyset \notin S$ (inak zoberieme komplement S, tiež netriviálnu vlastnosť, ktorú tiež vieme rozhodnúť). Nech $L \in S$ je ľubovoľný jazyk. Taký jazyk určite existuje a existuje preň TS A_L. Na základe týchto predpokladov zostrojme DTS pre L_U, ktorý na každom vstupe zastane.

Náš stroj dostane na vstupe $<A>$ a w. Z nich zostrojí kód takého TS A_w, ktorý pracuje nasledovne: Vstupné slovo x, ktoré dostane, si odloží na jednu pásku. Na druhú si napíše slovo w (ktoré má „zadrôtované“ v prechodovej funkcii) a simuluje na ňom A. Ak A slovo w akceptuje, vráti sa k pôvodnému vstupu a simuluje na ňom TS A_L. (Uvedomte si, že keď poznáme $<A>$, $<A_L>$ a w, vieme algoritmicky zostrojiť náš A_w.)

Rozoberme dva prípady: Ak $w \in L(A)$, tak A_w akceptuje x práve vtedy, keď ho akceptuje A_L. Preto $L(A_w) = L$. Ak $w \notin L(A)$, tak A_w sa k simulácií A_L nikdy nedostane, preto $A_w = \emptyset$.

My vieme rozhodovať vlastnosť S. Rozhodneme preto, či nami zostrojený A_w má vlastnosť S. Ak dostaneme odpoveď áno, tak $L(A_w) = L$, a teda $w \in L(A)$. Ak dostaneme odpoveď nie, tak $L(A_w) = \emptyset$ a $w \notin L(A)$. Tým sme teda rozhodli, či $w \in L(A)$. To ale nevieme, preto nemôžeme vedieť rozhodovať ani S.

Dôsledok 6.11.6 *Prázdnosť, konečnosť, regulárnosť, bezkontextovosť, atď. rekurzívne vyčísliteľných jazykov je nerozhodnuteľná. Každá z týchto vlastností je totiž netriviálna (máme konečné aj nekonečné jazyky, prázdne aj neprázdne, . . .).*

Ukážeme teraz, že ani s čiastočnou rozhodnuteľnosťou to nebude príliš ružové.

Poznámka 6.11.4 Uvedomte si, že každý konečný jazyk vieme zakódovať do slova nad $\{0, 1\}$ podobne, ako sme kódovali TS – zapíšeme najskôr veľkosť abecedy, potom postupne v lexikografickom poradí slová. Na každú *množinu* konečných jazykov sa teda môžeme dívať ako na *jazyk* ich kódov.

Veta 6.11.7 *(Rice) Vlastnosť S rekurzívne vyčísliteľných jazykov je čiastočne rozhodnuteľná práve vtedy, keď sú splnené nasledovné tri vlastnosti:*

1. *Ak $L \in S$, tak aj každý $L' \in \mathscr{L}_{RE}$, ktorý je nadmnožinou L, má vlastnosť S.*
2. *Ak $L \in S$, tak existuje konečná podmnožina L, ktorá má vlastnosť S.*
3. *Množina konečných jazykov v S je rekurzívne vyčísliteľná.*

Dôkaz.

- Ak neplatí 1, tak $L_S \notin \mathscr{L}_{RE}$. Sporom. Nech $L_S \in \mathscr{L}_{RE}$, nech A_S je TS, ktorý ho akceptuje. Ďalej nech $L_1 \subseteq L_2$ sú rekurzívne vyčísliteľné jazyky, $L_1 \in S$, $L_2 \notin S$. Nech A_1, A_2 sú TS pre L_1, L_2. Zostrojíme TS pre L_U^C, čo bude hľadaný spor.

  Náš stroj na vstupe dostane $<A>$ a w. Z nich zostrojí kód TS A_w, ktorý bude fungovať nasledovne: Ak $w \in L(A)$, tak $L(A_w)$ bude L_2, inak L_1. Spustíme A_S na $<A_w>$. Ten akceptuje iff $L(A_w)$ má vlastnosť S, teda $L(A_w) = L_1$, teda $w \notin L(A)$. Teda náš stroj akceptuje práve komplement univerzálneho jazyka, čo je hľadaný spor.

  Ukážeme ešte, ako bude vyzerať (t.j. ako zostrojiť kód) A_w s požadovanou vlastnosťou. A_w dostane vstup x. Naraz bude simulovať: A_1 na slove x, A_2 na slove x a A na slove w. Slová z L_1 máme vždy akceptovať, preto ak A_1 akceptuje, aj A_w akceptuje. Navyše ak časom zistíme, že A akceptoval w, máme akceptovať aj tie slová, ktoré sú v $L_2 \setminus L_1$. Preto ak A akceptoval w, akceptujeme aj tie x, ktoré akceptoval A_2.

- Ak neplatí 2, tak $L_S \notin \mathscr{L}_{RE}$. Sporom. Nech $L_S \in \mathscr{L}_{RE}$, nech A_S je TS, ktorý ho akceptuje. Ďalej nech $L \in S$, ale žiadna konečná podmnožina L nie je v S. Nech A_L je TS pre L. Zostrojíme TS pre L_U^C, čo bude hľadaný spor.

  Náš stroj na vstupe dostane $<A>$ a w. Z nich zostrojí kód TS A_w, ktorý bude fungovať nasledovne: Ak $w \in L(A)$, tak $L(A_w)$ bude nejaká konečná podmnožina L, inak $L(A_w)$ bude L. Spustíme A_S na $<A_w>$. Ten akceptuje iff $L(A_w)$ má vlastnosť S, teda $L(A_w) = L$, teda $w \notin L(A)$. Teda náš stroj akceptuje práve komplement univerzálneho jazyka, čo je hľadaný spor.

  Ukážeme ešte, ako bude vyzerať (t.j. ako zostrojiť kód) A_w s požadovanou vlastnosťou. A_w dostane vstup x. Naraz bude simulovať A_L na slove x a A na slove w. Ak A_L akceptuje, aj

A_w akceptuje. Ak A akceptuje, A_w sa zasekne. Zjavne ak $w \notin L(A)$, tak $L(A_w) = L$. Ak $w \in L(A)$, nech akceptačný výpočet A na w má k krokov. Potom A_w akceptuje tie slová z L, ktoré A_L akceptuje na najviac k krokov. Takýchto slov je ale len konečne veľa.

- Ak neplatí 3, tak $L_S \notin \mathscr{L}_{RE}$. Nepriamo. Nech $L_S \in \mathscr{L}_{RE}$, nech A_S je TS, ktorý ho akceptuje. Ukážeme, že potom platí 3. Zostrojíme generujúci TS, ktorý bude generovať kódy všetkých konečných jazykov, ktoré majú vlastnosť S.

  Ľahko zostrojíme TS, ktorý bude generovať kódy všetkých konečných jazykov. Ku každému z nich ľahko zostrojíme kód TS, ktorý ho akceptuje. Na každom z týchto kódov TS potrebujeme odsimulovať A_S a vypísať tie kódy konečných jazykov, ktorých TS A_S akceptuje. Keďže ale A_S sa môže aj zacykliť, musíme ho simulovať „naraz na všetkých TS“, nie postupne. Toto sa dá dosiahnuť napr. nasledovne: Striedavo vygenerujeme jeden nový kód konečného jazyka (a zodpovedajúci TS) a odsimulujeme po jednom kroku A_S na každom z už vygenerovaných TS. Vždy, keď A_S niektorý TS akceptuje, kód príslušného konečného jazyka vypíšeme.

- Ukážeme, že ak platí 1, 2, aj 3, tak vieme zostrojiť TS A_S taký, že $L(A_S) = L_S$. A_S na vstupe dostane $<A>$ a má zistiť, či $L(A)$ má vlastnosť S. Ak $L(A)$ má túto vlastnosť, tak ju (podľa 2) má aj nejaká jeho konečná podmnožina. Inými slovami, ak žiadna z nich vlastnosť S nemá, tak ani $L(A)$ ju nemá. Budeme skúmať všetky konečné podmnožiny $L(A)$. Ak žiadna z nich vlastnosť S nemá, $<A>$ neakceptujeme.[12] Na druhej strane, akonáhle nájdeme nejakú konečnú podmnožinu $L(A)$ s vlastnosťou S, podľa 1 aj $L(A)$ má vlastnosť S – teda A_S akceptuje.

  Podľa 3 vieme generovať všetky konečné jazyky s vlastnosťou S. Na všetkých slovách každého z týchto jazykov potrebujeme simulovať A. Použijeme podobný postup ako v predchádzajúcom bode – striedavo vygenerujeme jeden nový konečný jazyk a odsimulujeme jeden krok A na každom zo slov už vygenerovaných jazykov.[13] Akonáhle A akceptuje všetky slová niektorého konečného jazyka, akceptujeme.

Dôsledok 6.11.8 *Nasledujúce vlastnosti rekurzívne vyčísliteľných jazykov nie sú ani čiastočne rozhodnuteľné:*

- *prázdnosť (porušený bod 1, nadmnožiny prázdneho jazyka nie sú prázdne)*
- *rovnosť Σ^* (porušený bod 2)*
- $L(A) \in \mathscr{L}_{rec}$ *(porušený bod 1)*
- $|L(A)| = 1$ *(porušený bod 1)*
- $L(A) \setminus L_U \neq \emptyset$ *(porušený bod 3)*

Pristavíme sa pri poslednom tvrdení. Keby sme vedeli vymenovať všetky konečné jazyky s danou vlastnosťou, zjavne vieme vymenovať všetky jednoslovné jazyky s touto vlastnosťou. Jednoslovné jazyky s danou vlastnosťou sú práve jazyky obsahujúce slovo z L_U^C. Upravme stroj, ktorý generuje jednoslovné jazyky s danou vlastnosťou tak, aby namiesto jednoslovných jazykov generoval dotyčné slová. Potom tento stroj generuje L_U^C, čo je spor.

Dôsledok 6.11.9 *Nasledujúce vlastnosti rekurzívne vyčísliteľných jazykov sú čiastočne rozhodnuteľné:*

- *neprázdnosť*
- $|L(A)| \geq 10$
- $11011 \in L(A)$

[12] Uvedomte si, že budeme počítať do nekonečna, ak $L(A)$ je nekonečný.
[13] Ako jazykov, tak slov v nich je konečne veľa, preto táto fáza v konečnom čase skončí.

Kapitola 7

Úvod do teórie zložitosti

7.1 Model TS a definície zložitosti

Čo sa týka Turingovych strojov a vôbec algoritmov ako takých, skúmajú sa hlavne dve hľadiská zložitosti Turingovych strojov, resp. algoritmov. 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. Namiesto priestorovej niekedy hovoríme o pamäťovej zložitosti, tieto dva pojmy sú ekvivalentné.

V celej tejto kapitole budeme ako model TS používať **TS so vstupnou a niekoľkými pracovnými páskami**. Vstupnú pásku smie len čítať, na ostatné pásky môže aj zapisovať. Vstupná páska je obmedzená na dĺžku vstupného slova (podobne ako u LBA), pracovné pásky sú doprava nekonečné.

Definícia 7.1.1 *Hovoríme, že deterministický Turingov stroj A je $S(n)$* ***priestorovo ohraničený****, ak každý výpočet A na slove w dĺžky n použije na každej z pracovných pások maximálne $S(n)$ políčok.*

Poznámka 7.1.1 Model so vstupnou a pracovnými páskami používame preto, aby mali zmysel aj ohraničenia, kde $S(n) < n$. Na akceptovanie jednoduchých jazykov nám bude stačiť aj menší ako lineárny pracovný priestor. Keby sme ale do použitého priestoru rátali aj vstupné slovo, nevedeli by sme takéto jazyky odlíšiť od jazykov, kde potrebujeme lineárny pracovný priestor.

Čo sa týka nedeterministických Turingovych strojov, sú možné tri prístupy definície priestorovej ohraničenosti.

Definícia 7.1.2 *Hovoríme, že nedeterministický Turingov stroj A je $S(n)$* ***priestorovo ohraničený****, ak:*

- *(silná def.) Každý výpočet na vstupnom slove w dĺžky n použije maximálne $S(n)$ políčok na každej z pracovných pások.*
- *(stredná def.) Každý výpočet na každom vstupnom slove $w \in L(A)$ dĺžky n použije maximálne $S(n)$ políčok na každej z pracovných pások.*
- *(slabá def.) Pre každé vstupné slovo $w \in L(A)$ dĺžky n existuje výpočet, ktorý použije maximálne $S(n)$ políčok na každej z pracovných pások.*

Poznámka 7.1.2 Pre funkcie vypočítateľné na TS je jedno, ktorú z definícií priestorovej ohraničenosti pre NTS používame. Pred samotným výpočtom si totiž môžeme na každej páske označiť $S(n)$ políčok a potom prerušiť každý výpočet, ktorý by potreboval viac ako $S(n)$ políčok na niektorej páske.

Definícia 7.1.3 *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 7.1.4 *Hovoríme, že nedeterministický Turingov stroj je $T(n)$* ***časovo ohraničený****, ak:*

- *(silná def.) Každý výpočet na každom slove w dĺžky n urobí najviac $T(n)$ krokov.*
- *(stredná def.) Každý výpočet na každom slove $w \in L(A)$ dĺžky n urobí najviac $T(n)$ krokov.*
- *(slabá def.) Pre každé slovo $w \in L(A)$ dĺžky n existuje výpočet, ktorý urobí najviac $T(n)$ krokov.*

Poznámka 7.1.3 V praxi sa zvykne najčastejšie používať slabá definícia, urobíme tak aj my. Dá sa použiť podobný prístup ako pri priestorovej zložitosti – tieto definície sú ekvivalentné pre funkcie, ktoré sú „časovo zostrojiteľné“ (t.j. existuje DTS, ktorý na vstupe dĺžky n urobí $T(n)$ krokov a akceptuje). Potom totiž si náš NTS vie „merať čas“ (simuluje kroky tohto DTS zároveň s výpočtom) a zaseknúť sa, ak by mal prekročiť povolený limit.

Zamyslite sa nad tým, ako „časovo zostrojiť“ niektoré elementárne funkcie. Presvedčte sa, že napr. polynomické funkcie, ktoré sú na N kladné, sú „časovo zostrojiteľné“.

7.2 Vety o kompresii, zrýchlení a redukcii počtu pások

Čitateľ by si už mal uvedomovať, že medzi silou TS, ktorý je priestorovo obmedzený funkciou n^7 a TS, ktorý je obmedzený funkciou $2n^7$ nebude prílišný (vlastne žiadny) rozdiel. Trochu prekvapivejšie už bude tvrdenie, že každý TS vieme zrýchliť tak, že jeho čas výpočtu bude k-krát menší (kde k je vopred zvolená konštanta). Dôsledkom týchto viet bude, že môžeme pri obmedzovaní na čas a pamäť ignorovať konštanty a dôležitý bude len asymptotický rast funkcie, ktorá bude udávať obmedzenie.

Veta 7.2.1 *(O kompresii pásky) K ľubovoľnému deterministickému Turingovmu stroju A, ktorý je $S(n)$ priestorovo ohraničený a konštante $k > 1$ existuje deterministický Turingov stroj A', ktorý je $\lceil S(n)/k \rceil$ priestorovo ohraničený a platí $L(A) = L(A')$.*

Dôkaz. Stačí použiť novú pracovnú abecedu $\Gamma' = \Gamma^k$. Každé políčko pásky A' bude predstavovať k políčok TS A.

Podobná veta platí aj o zrýchlení. Budeme chcieť simulovať naraz k krokov pôvodného TS. Na to najskôr použijeme vhodnú kompresiu pásky, aby sme v dotyčnom jednom kroku zvládli prečítať všetky potrebné symboly. Musíme si však uvedomiť, že vo všeobecnosti potrebujeme prečítať celý vstup, na čo potrebujeme n krokov.[1] Preto budeme musieť predpokladať, že toto je zanedbateľne malý čas oproti času výpočtu pôvodného TS.

Ak chceme simulovať v jednom kroku 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. Nech používame ľubovoľne veľkú kompresiu pásky, môže sa stať, že hlava simulovaného TS sa bude nachádzať pri okraji úseku pásky, ktorý je zapísaný na príslušnom políčku. V takomto prípade sa budeme musieť pozrieť aj na susedné políčko.

Veta 7.2.2 *(O lineárnom zrýchlení) Nech A je ľubovoľný deterministický Turingov stroj, ktorý je $T(n)$ časovo ohraničený a nech*

$$\lim_{n\to\infty} \frac{T(n)}{n} = \infty$$

Nech $k > 1$ je ľubovoľná konštanta. Potom existuje deterministický Turingov stroj A' taký, že $L(A) = L(A')$, A' je $T'(n)$ časovo ohraničený a od nejakého n_0 je $T(n') \leq T(n)/k$.

[1] Počas týchto n krokov si vstup môžeme zároveň prepísať na komprimovanú pracovnú pásku, takže ďalšie čítanie vstupu už bude rýchle.

Dôkaz. Uvedomme si najskôr, že krátke slová nám robia pri urýchlení problém. Napr. TS, ktorý akceptuje slovo *aaaaa* na 7 krokov (ale neakceptuje *aaaab*) by ho po 10-násobnom urýchlení mal akceptovať na 1 krok, čo ale nie je možné. Zostrojíme preto TS, ktorý bude k-krát rýchlejší len na dostatočne dlhých slovách. Uvedomte si, že nie je problém tento TS na záver upraviť tak, že všetky slová z $L(A)$ kratšie ako n_0 do neho „zadrôtujeme“, takže pre $n < n_0$ bude $T'(n) = n$. Keďže vo všeobecnosti TS musí vstup dočítať, nič lepšie sa dosiahnuť nedá. Ukážeme teraz, ako zostrojiť hľadaný A'.

Nech platia uvedené predpoklady. Turingov stroj A' bude pracovať nasledovne: Najprv reorganizuje pásku do blokov po m políčkach. (Číslo m je vhodná konštanta. Neskôr ukážeme, aký je vzťah medzi k a m.) To znamená, že každých m znakov vstupu prepíše na jeden znak na jednej pracovnej páske. Pracovné pásky budú mať abecedu Γ_A^m.

Ďalej A' simuluje prácu pôvodného automatu A s tým, že sa rozšíri množina stavov na $K \times \{1, 2, \ldots, m\}$, aby sme si pamätali číslo skutočného políčka, na ktorom by bol TS A v rámci aktuálneho bloku. TS A' spraví 4 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 a vráti sa späť. Určite si teraz v stave pamätá aspoň m políčok pôvodnej pásky naľavo aj napravo od pozície hlavy, preto vie odsimulovať nasledujúcich m krokov TS A. 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 TS A vieme simulovať ôsmimi krokmi TS A'. Ak má A' byť od A k-krát rýchlejší, stačí zvoliť $m = 8k + h$, kde h je konštanta „na pokrytie režijných nákladov na začiatku“.

Aby bol dôkaz kompletný, formálne dokážeme, že $m = 8k + 47$ naozaj stačí. Čas behu TS A' na vstupe veľkosti n bude

$$T'(n) = n + \left\lceil \frac{n}{m} \right\rceil + \left\lceil \frac{8T(n)}{m} \right\rceil$$

(Skomprimujeme vstup, vrátime sa hlavou na jeho začiatok a simulujeme A.) Je

$$T'(n) < n + \frac{n+m}{m} + \frac{8T(n)+m}{m} = \frac{8T(n)+2m+(m+1)n}{m} < \frac{8T(n)+(m+1)(n+2)}{m}$$

Treba ukázať, že od nejakého n_0 je $T'(n) \leq T(n)/k$. Na to stačí, aby

$$\frac{8T(n)+(m+1)(n+2)}{m} < \frac{T(n)}{k}$$

odkiaľ po prepísaní dostávame, že má platiť

$$\frac{k(8k+48)}{47}(n+2) < T(n)$$

Keďže však $\lim_{n\to\infty}(T(n)/n) = \infty$, také n_0 zjavne existuje.

V nasledujúcom texte ukážeme, že i počet použitých pások možno redukovať. Bohužiaľ, v niektorých prípadoch tým čosi stratíme.

Veta 7.2.3 *(O redukcii počtu pások z k na jednu pre priestor) K ľubovoľnému deterministickému TS A, ktorý má k pracovných pások a je $S(n)$ priestorovo ohraničený existuje ekvivalentný TS A', ktorý je tiež $S(n)$ priestorovo ohraničený a má len jednu pracovnú pásku.*

Dôkaz. Stačí použiť viacstopú pracovnú pásku. (Jedna možnosť je mať na každej stope zaznačenú aj pozíciu hlavy, druhá možnosť je posúvať jednotlivé stopy tak, aby boli všetky hlavy stále nad tým istým políčkom.) Uvedomte si, že simulácia jedného kroku pôvodného TS nám môže trvať až $S(n)$ krokov, keďže musíme prejsť celú pásku a nájsť všetky simulované hlavy. (Resp. $kS(n)$ krokov, lebo musíme poposúvať jednotlivé pásky.)

Dôsledok 7.2.4 *(O redukcii počtu pások z k na jednu pre čas) K ľubovoľnému deterministickému TS A, ktorý má vstupnú a k pracovných pások a je $T(n)$ časovo ohraničený existuje ekvivalentný klasický TS A' (s jedinou páskou, na ktorú môže aj zapisovať), ktorý je $T^2(n)$ časovo ohraničený.*

Dôkaz. Keďže A je $T(n)$ časovo ohraničený, je aj $T(n)$ priestorovo ohraničený. Zostrojíme ekvivalentný TS s jednou páskou podľa dôkazu predchádzajúcej vety. Ľahko nahliadneme, že je $T^2(n)$ časovo ohraničený.

Naša redukcia počtu pások na jednu síce spôsobí značné spomalenie, redukcia na dve pásky však na tom bude oveľa lepšie.

Veta 7.2.5 *(O redukcii počtu pások z k na dve pre čas) K ľubovoľnému deterministickému k-páskovemu Turingovmu stroju A pracujúcemu 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. Použijeme metódu posunu stopy pásky namiesto posunu hlavy. No nebudeme posúvať celú stopu, ale väčšinou len jej malý kúsok. Podstata použitej myšlienky spočíva v tom, že si pásku rozdelíme na myslené bloky o veľkostiach $1, 2, 4, 8, \ldots, 2^k$ políčok. Fungovanie automatu A' vysvetlíme na príklade.

Príklad sem časom pribudne. Možno :-)

Poznámka 7.2.1 Mohlo by nás zaujímať, či sa podobný výsledok nedá dosiahnuť aj pri redukcii z k pások na jednu. (Čo ak nami ukázaný postup nebol optimálny?) Ukážeme ale príklad jazyka, ktorý vieme na dvojpáskovom TS akceptovať v lineárnom čase, zatiaľ čo na klasickom jednopáskovom TS potrebujeme kvadratický čas. Takýmto jazykom je $L = \{wcw^R \mid w \in \{a, b\}^*\}$.

Dôkaz, že klasický TS potrebuje na jeho akceptovanie kvadratický čas, presahuje rámec tohto textu. Jeho myšlienka je v zostrojení prechodových postupností (podobne ako sme urobili v dôkaze vety 2.11.4 o ekvivalencii NKA a 2NKA) a dokázaní, že pre vhodné rôzne slová musia byť prechodové postupnosti navzájom rôzne. Z toho vyplýva, že niektoré z vybraných slov majú veľký súčet dĺžok prechodových postupností, lebo krátkych prechodových postupností je málo. Ale časová zložitosť výpočtu TS na slove je práve súčet dĺžok prechodových postupností pre všetky políčka pásky.

Dôsledok 7.2.6 *Vo všeobecnosti nevieme spraviť redukciu z k pások na jednu lepšie ako za cenu kvadratického spomalenia.*

7.3 Triedy zložitosti

Na základe definícií časovo a priestorovo ohraničeného TS rozčleníme jazyky do tried zložitosti – podľa najmenšieho času, resp. pamäte, ktoré stačia na jeho rozpoznanie Turingovym strojom.

Definícia 7.3.1 *Každá funkcia nám určuje triedu jazykov, ku ktorým existuje ňou ohraničený TS. Musíme rozlíšiť, či tento TS je deterministický a či ide o časové alebo pamäťové ohraničenie. Definujeme triedy*

$$\begin{aligned}
DSPACE(S(n)) &= \{L \mid \exists \text{ DTS } A, \text{ kt. je } S(n) \text{ priestorovo ohraničený a } L = L(A)\} \\
NSPACE(S(n)) &= \{L \mid \exists \text{ NTS } A, \text{ kt. je } S(n) \text{ priestorovo ohraničený a } L = L(A)\} \\
DTIME(T(n)) &= \{L \mid \exists \text{ DTS } A, \text{ kt. je } T(n) \text{ časovo ohraničený a } L = L(A)\} \\
NTIME(T(n)) &= \{L \mid \exists \text{ NTS } A, \text{ kt. je } T(n) \text{ časovo ohraničený a } L = L(A)\}
\end{aligned}$$

Poznámka 7.3.1 Pripomíname, že naďalej hovoríme o modeli TS so vstupnou páskou len na čítanie a k pracovnými páskami.

Poznámka 7.3.2 Definujú sa tiež triedy $(N/D)SPACETIME(S(n), T(n))$, kde obmedzíme TS ako priestor, tak aj čas.

Poznámka 7.3.3 Ako ohraničenie môžeme použiť aj asymptotickú notáciu, teda namiesto $f(n)$ uvedieme $O(f(n))$. Z viet 7.2.1 o kompresii pásky a 7.2.2 o lineárnom zrýchlení vyplýva, že pre

rozumné funkcie $f(n)$ je jedno, či zoberieme ako ohraničenie $f(n)$ alebo $cf(n)$, kde c je kladná konštanta. V takomto prípade je teda jedno, či použijeme ohraničenie $f(n)$ alebo $O(f(n))$.

Teda napr. $DSPACE(O(n))$ je trieda jazykov, pre ktoré existuje $O(n)$ priestorovo ohraničený DTS. Z vety o kompresii pásky vieme, že $DSPACE(O(n)) = DSPACE(n)$. Formálne môžeme definovať $DSPACE(O(f(n))) = \bigcup_{g(n)=O(f(n))} DSPACE(g(n))$, pre ostatné triedy analogicky.

Definícia 7.3.2 *Špeciálne nás budú neskôr zaujímať triedy, kde je čas alebo priestor obmedzený nejakým polynómom, prípadne inou jednoduchou funkciou, pričom nás bude zaujímať len typ tejto funkcie, nie konkrétne koeficienty v nej. Najznámejšie takéto triedy jazykov sú:*

$$\begin{aligned}
P &= \bigcup_{f \text{ je polynóm}} DTIME(f(n)) = \bigcup_{k>0} DTIME(O(n^k)) = DTIME(n^{O(1)}) \\
NP &= \bigcup_{f \text{ je polynóm}} NTIME(f(n)) = \bigcup_{k>0} NTIME(O(n^k)) = NTIME(n^{O(1)}) \\
PSPACE &= \bigcup_{k>0} DSPACE(n^k) = DSPACE(n^{O(1)}) \\
NPSPACE &= \bigcup_{k>0} NSPACE(n^k) = NSPACE(n^{O(1)}) \\
L &= DSPACE(\log n) \\
NL &= NSPACE(\log n)
\end{aligned}$$

Teraz ukážeme, že niektoré z takto definovaných tried jazykov už poznáme pod inými menami.

Veta 7.3.1 $\mathscr{L}_{ECS} = NSPACE(n) = NSPACE(O(n))$

Dôkaz. Prvá rovnosť vyplýva z definície LBA, druhá z vety 7.2.1 o kompresii pásky. Táto veta poukazuje na skutočnosť, že LBA je skutočne Turingov stroj s lineárne obmedzeným priestorom.

Veta 7.3.2 $\mathcal{R} = NSPACE(1) = NSPACE(O(1)) = DSPACE(1) = DSPACE(O(1))$

Dôkaz. Podľa lemy 7.5.1 a definícií uvedených tried zjavne $NSPACE(O(1)) \supseteq DSPACE(O(1))$, $DSPACE(O(1)) \supseteq DSPACE(1)$ a $NSPACE(O(1)) \supseteq NSPACE(1)$. Platí $DSPACE(1) \supseteq \mathcal{R}$, lebo pre každý regulárny jazyk existuje DTS (simulujúci príslušný DKA), ktorý ho akceptuje dokonca bez používania pomocnej pamäte. A platí aj $\mathcal{R} \supseteq NSPACE(O(1))$. Keď totiž máme NTS, ktorý na každej z k pracovných pások použije najviac c políčok, vieme si týchto kc políčok pamätať v stave dvojsmerného nedeterministického konečného automatu, ktorým príslušný TS ľahko odsimulujeme. Z ukázaných inklúzií už vyplýva platnosť všetkých rovností.

Poznámka 7.3.4 A zjavne platí aj $\mathcal{R} = DSPACETIME(1, n)$.

Poznámka 7.3.5 Z triviálne platnej inklúzie $\mathscr{L}_{CF} \subseteq \mathscr{L}_{ECS}$ a rovnosti $\mathscr{L}_{ECS} = NSPACE(n)$ dostávame $\mathscr{L}_{CF} \subseteq NSPACE(n)$. Rozmyslite si, ako by ste túto inklúziu dokázali priamo.

7.4 Uzáverové vlastnosti tried zložitosti

U väčšiny tried zložitosti sa dajú použiť osvedčené postupy, ktorými sme dokazovali a vyvracali uzáverové vlastnosti pre kontextové a frázové jazyky. (Niekedy budeme musieť navyše využiť vety 7.2.1 o kompresii pásky a 7.2.2 o lineárnom zrýchlení.) Uvedieme preto len niekoľko jednoduchých príkladov.

Veta 7.4.1 *Pre ľubovoľnú funkciu f sú triedy $DSPACE(f(n))$ a $NSPACE(f(n))$ uzavreté na zjednotenie a prienik.*

Dôkaz. Stačí postupne odsimulovať oba príslušne ohraničené automaty.

Poznámka 7.4.1 Pri uzavretosti $NSPACE(f(n))$ na zjednotenie musíme robiť simulácie paralelne, pri ostatných stačí sériovo. Rozmyslite si, prečo. Uzavretosť $NSPACE(f(n))$ na zjednotenie sa však dá dokázať aj ľahšie, s využitím nedeterminizmu – ako?

Veta 7.4.2 *Pre ľubovoľnú funkciu f, pre ktorú $\lim_{n\to\infty} f(n)/n = \infty$, sú triedy $DTIME(f(n))$ a $NTIME(f(n))$ uzavreté na zjednotenie a prienik.*

Dôkaz. Opäť spravíme presne rovnakú konštrukciu ako v predchádzajúcej vete. Vieme, že pre každé slovo dĺžky n, ktoré výsledný TS akceptuje, existuje výpočet, ktorý má najviac $2f(n)+O(n)$ krokov. Potom ale podľa vety 7.2.2 o lineárnom zrýchlení existuje aj $f(n)$ časovo ohraničený TS pre ten istý jazyk.

Veta 7.4.3 *Pre ľubovoľnú funkciu f sú triedy $DSPACE(f(n))$ a $NSPACE(f(n))$ uzavreté na inverzný homomorfizmus.*

Dôkaz. Vstupné slovo zobrazíme daným homomorfizmom a odsimulujeme na ňom TS pre pôvodný jazyk. Takto zostrojený TS použije najviac $c.f(n)$ políčok pásky pre vhodné c. Potom ale podľa vety 7.2.1 o kompresii pásky existuje ekvivalentný $f(n)$ priestorovo ohraničený TS pre daný jazyk.

Veta 7.4.4 *Pre ľubovoľnú funkciu f, pre ktorú $f(n) \geq n$,* **nie** *sú triedy $DSPACE(f(n))$ a $NSPACE(f(n))$ uzavreté na homomorfizmus.*

Dôkaz. Analogicky ako vo vete 5.5.7, kde sa toto tvrdenie dokazuje pre $\mathscr{L}_{ECS} = NSPACE(n)$. Problémy robí vymazávajúci homomorfizmus.

7.5 Vzťahy medzi deterministickým a nedeterministickým časom a priestorom

Lema 7.5.1 *$NTIME(f(n)) \subseteq NSPACE(f(n))$ a $DTIME(f(n)) \subseteq DSPACE(f(n))$.*

Dôkaz. Zjavne každý $f(n)$ časovo ohraničený TS stihne na každej páske zapísať najviac prvých $f(n)$ políčok, a teda je aj $f(n)$ priestorovo ohraničený.

Lema 7.5.2 *$DTIME(f(n)) \subseteq NTIME(f(n))$ a $DSPACE(f(n)) \subseteq NSPACE(f(n))$.*

Dôkaz. Každý DTS je zároveň aj (rovnako obmedzený) NTS.

Lema 7.5.3 *Nech $\forall n;\ f(n) \leq g(n)$. Potom $DTIME(f(n)) \subseteq DTIME(g(n))$ (a rovnako aj pre ostatné typy tried).*

Poznámka 7.5.1 Uvedomte si, že uvedené lemy platia, aj keď namiesto $f(n)$ píšeme $O(f(n))$.

Veta 7.5.4 *Nech $\forall n;\ f(n) > \log n$, pričom f je „slušná“ (napr. „časovo zostrojiteľná“) funkcia. Potom $\forall L \in DSPACE(f(n))\ \exists c \in N;\ L \in DTIME(c^{f(n)})$.*

Dôkaz. Nech $L \in DSPACE(f(n))$, nech A je príslušne priestorovo obmedzený DTS s jednou pracovnou páskou, ktorý ho akceptuje. Potom na slove dĺžky n má A iba $|\Gamma|^{f(n)}|K|(n+2)(f(n)+1)$ možných konfigurácií. Túto hodnotu môžeme zhora odhadnúť hodnotou $d^{f(n)}$ pre dostatočne veľké d. Simulujúci stroj si označí na jednej páske priestor $d^f(n)$ (to vie spraviť v čase úmernom $d^{f(n)}$). Bude simulovať A a zároveň na tejto páske rátať jeho kroky. Keď zistí, že A spravil viac ako $d^{f(n)}$ krokov, zasekne sa. Zjavne jeho časová zložitosť je najviac $kd^{f(n)}$ (pre vhodné k). Potom ale je tento simulujúci TS $c^{f(n)}$ časovo ohraničený napr. pre $c = kd$.

Veta 7.5.5 *Nech $\forall n;\ f(n) > n$, pričom f je „slušná“ funkcia.*
Potom $\forall L \in NTIME(f(n))\ \exists c \in N;\ L \in DTIME(c^{f(n)})$.

Dôkaz. Simulujúci DTS bude prehľadávať do šírky všetky dosiahnuteľné konfigurácie pôvodného NTS. V každej konfigurácií je len konečne veľa možností pre ďalší krok, preto ich počet rastie exponenciálne s dĺžkou výpočtu. Keď prezrieme konfigurácie dosiahnuteľné na $f(n)$ krokov a nenájdeme žiadnu akceptačnú, zasekneme sa.

Veta 7.5.6 *(Savitch) Nech $\forall n;\ f(n) > \log n$, pričom f je „slušná“ funkcia.*
Potom $NSPACE(f(n)) \subseteq DSPACE(f^2(n))$.

Dôkaz. Nech A je $f(n)$ pamäťovo ohraničený nedeterministický TS. Zostrojíme ekvivalentný deterministický TS, ktorému stačí priestor $f^2(n)$.

Možných konfigurácií A pri výpočte na w je $Q = |K|(f(|w|) + 2)(|\Gamma_A| + 1)^{f(|w|)}$, teda exponenciálne veľa od $f(|w|)$. Najviac tak dlhý môže byť aj najkratší akceptujúci výpočet A na w. Preto riešenie používajúce prehľadávanie do šírky, hĺbky či backtracking potrebuje až pamäť až exponenciálnu od $f(|w|)$ (pre každú navštívenú konfiguráciu si musí niečo pamätať). Bude treba použiť inú myšlienku.

Vieme si spočítať číslo Q, to nám zhora odhaduje dĺžku najkratšieho akceptačného výpočtu. Na jednej páske teda máme Q (vo vhodnej sústave), na ďalšej postupne generujeme všetky možné akceptačné konfigurácie. Tých je konečne veľa. Teraz sme v situácií, že máme počiatočnú a koncovú konfiguráciu a chceme overiť, či sa z prvej dá dostať do druhej na $\leq Q$ krokov.

To bude robiť rekurzívna „procedúra“ Over(K_1, K_2, $krokov$). Ak $krokov < 3$, vyskúšame všetky možnosti. Inak: Ak hľadaný výpočet existuje, existuje nejaká konfigurácia K_3 v jeho „strede“. Do nej sa dá z K_1 dostať na najviac $\lfloor krokov/2 \rfloor$ krokov, z nej do K_2 na najviac $\lceil krokov/2 \rceil$ krokov. On the other hand, ak takúto konfiguráciu K_3 nájdeme, hľadaný výpočet existuje. Naša procedúra teda postupne generuje všetky konfigurácie K_3, pre každú najskôr zavolá Over(K_1, K_3, $\lfloor krokov/2 \rfloor$) a potom Over(K_3, K_2, $\lceil krokov/2 \rceil$).

Pri každom vnorení sa počet krokov hľadaného výpočtu zmenší približne na polovicu, vnorení je teda $O(\log Q) = O(f(|w|))$. Pri každom si potrebujeme pamätať práve skúšanú strednú konfiguráciu a počty krokov do a z nej. Priestor potrebný na jedno vnorenie je teda $O(f(|w|))$. Výsledný DTS teda patrí do $DSPACE(O(f^2(n)))$ a podľa vety o kompresii pásky aj do $DSPACE(f^2(n))$.

Dôsledok 7.5.7 *Zjavne $PSPACE = NPSPACE$, lebo ku každému $p(n)$ priestorovo ohraničenému NTS vieme podľa Savitchovej vety zostrojiť ekvivalentný $p^2(n)$ priestorovo ohraničený DTS. Ak $p(n)$ je polynóm, aj $p^2(n)$ je polynóm.*

Veta 7.5.8 $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$

Dôkaz. Prvá inklúzia je triviálna.
Druhá: NTS, ktorý používa len $\log N$ políčok pracovnej pásky na vstupe dĺžky N, vie mať rôznych konfigurácií[2] len polynomiálne veľa od N. Preto vieme v polynomiálnom čase vyrobiť graf, ktorého vrcholy zodpovedajú konfiguráciám a (orientované) hrany zodpovedajú krokom výpočtu pôvodného NTS. Následne prehľadaním tohto grafu zistíme, či je niektorá akceptačná konfigurácia dosiahnuteľná.
Tretia je opäť triviálna.
Štvrtá: Nech A je $p(n)$ časovo ohraničený NTS. Budeme prehľadávať do hĺbky strom jeho možných konfigurácií, pričom prehľadávať končíme v hĺbke $p(n)$. Na páske si pamätáme aktuálnu cestu v strome, na to nám bohate stačí pamäť $p^2(n)$ (budeme si pamätať postupnosť najviac $p(n)$ konfigurácií).

Poznámka 7.5.2 Napriek diametrálne odlišným definíciám týchto tried nevieme skoro nič o tom, či sa rovnajú. Je známe, že $NL \subsetneq PSPACE$, a teda aspoň jedna z rovností $NL = P$, $P = NP$ a $NP = PSPACE$ neplatí – nevieme však, ktorá to je. Prevládajúci názor je, že neplatí ani jedna z nich. Bližšie si ich vzťah načrtneme v nasledujúcej časti.

[2]Horný odhad sa spraví rovnako ako horný odhad počtu rôznych konfigurácií pre LBA.

7.6 Redukcie, ťažké a úplné problémy

Túto časť začneme príkladom, na ktorom ukážeme koncept, ktorý neskôr vo všeobecnejšej podobe formálne definujeme.

Príklad 7.6.1 Uvažujme nasledovný problém, známy pod názvom SAT: Daná je boolovská formula s n premennými, rozhodnite, či existuje ohodnotenie premenných, pri ktorom je splnená.[3]

Zjavne $SAT \in NP$, stačí nedeterministicky uhádnuť jedno správne ohodnotenie premenných a overiť, že pre uhádnuté ohodnotenie je naozaj daná formula splnená.

V rokoch 1971 až 1973 Cook a Levin publikovali nasledujúci výsledok: Majme ľubovoľný problém, ktorý patrí do NP. Potom k tomuto problému existuje nedeterministický TS, ktorý ho v polynomiálnom čase rozhoduje. K tomuto stroju a ľubovoľnému danému slovu vieme v polynomiálnom čase (od súčtu veľkosti popisu stroja a dĺžky slova) zostrojiť boolovskú formulu, ktorá je splniteľná práve vtedy, ak daný stroj dané slovo akceptuje.

Čo nám tento výsledok hovorí?

Ak by problém SAT bol v P, tak by sme každý problém z NP vedeli riešiť v deterministickom polynomiálnom čase – zoberieme nedeterministický stroj pre daný problém, k tomu v polynomiálnom čase zostrojíme ekvivalentnú boolovskú formulu, a následne v polynomiálnom čase rozhodneme, či je splniteľná.[4]

Ak by teda SAT patril do P, tak každý problém z NP patrí do P, a teda $P = NP$. Inými slovami, v NP neexistuje žiaden problém „ťažší“ ako SAT.

Ukážeme teraz všeobecnejšiu definíciu, ktorá zachytáva túto myšlienku.

Definícia 7.6.1 *Nech T a R sú triedy zložitosti a X problém. Hovoríme, že problém X je T-ťažký (pri R-obmedzenej redukcii), ak pre každý problém $Y \in T$ existuje redukcia z Y na X, ktorá je v R.*

Hovoríme, že problém X je T-úplný (pri R-obmedzenej redukcii), ak je T-ťažký a patrí do T.

Ak je z kontextu jasné, ako obmedzenú redukciu používame, môžeme túto časť vynechať a hovoriť jednoducho o T-ťažkých a T-úplných problémoch.

Príklad 7.6.2 Vyššie spomínaný jazyk SAT je NP-úplný (pri redukcii z P, teda deterministickej a polynomiálne časovo obmedzenej).

Poznámka 7.6.1 Sama o sebe nám definícia T-ťažkých a T-úplných problémov veľa nehovorí. Na to, aby bola užitočná, potrebujeme vhodne zvoliť triedu R, teda to, aké redukcie povoľujeme. Redukcie by vo všeobecnosti mali byť v porovnaní s triedou T ľahké. Pri voľbe, aké redukcie povoliť a aké nie, chceme dosiahnuť, aby platilo: „Ak vieme efektívne riešiť nejaký T-úplný problém X, tak potom vieme efektívne riešiť každý problém z T tak, že ho zredukujeme na X a následne vyriešime X.“

Poznámka 7.6.2 Majme triedu T a podtriedu S, o ktorej nás zaujíma, či $S = T$. Zvoľme teda obmedzenie na redukciu tak, aby platilo, že každý algoritmus tvaru „zredukuj problém Y na X, a následne vyrieš X algoritmom z S“ patril do S.

Ak sa nám toto podarí, dosiahneme, že T-úplné problémy budú najvhodnejšími kandidátmi na zistenie, či $S = T$: Ak rovnosť neplatí, všetky T-úplné problémy sú nutne protipríkladmi. Na druhej strane, akonáhle by sme o nejakom T-úplnom probléme ukázali, že patrí do S, vyplynulo by z toho automaticky, že $S = T$.

V každej z tried NL, P, NP a $PSPACE$ poznáme problémy, ktoré sú pre ňu pri vhodnej redukcii úplné, a teda v určitom zmysle najťažšie v nej. Ukážeme si príklady takýchto problémov:

[3] Názov SAT pochádza z anglického „satisfiability“, t. j. splniteľnosť.

[4] Tu je dobré si uvedomiť, pri slovách „v polynomiálnom čase rozhodneme“ hovoríme o čase polynomiálnom od veľkosti dotyčnej boolovskej formuly. Tá však vznikla v polynomiálnom čase od veľkosti vstupu, a preto aj jej veľkosť vieme ohraničiť nejakým polynómom. Keď označíme veľkosť vstupu x, časovú zložitosť tvorby formuly $p(x)$ a časovú zložitosť algoritmu rozhodujúceho SAT $q(x)$, tak časová zložitosť celého tohto algoritmu bude $O(q(p(x)))$. Ale aj $q(p(x))$ je polynóm, a teda časová zložitosť celého algoritmu je polynomiálna.

- $PATH$: Daný je orientovaný graf G a dva jeho vrcholy s a t, rozhodnite, či v G existuje cesta z s do t.
  $PATH$ patrí do NL, a ak by patril do L, tak $L = NL$.

- $HALTN$: Daný je deterministický Turingov stroj A, vstup w a unárne zapísané číslo n, rozhodnite, či A na w po nanajvýš n krokoch zastane.
  $HALTN$ patrí do P, a ak by patril do NL, tak $NL = P$.

- SAT: Daná je boolovská formula s n premennými, rozhodnite, či existuje ohodnotenie premenných, pri ktorom je splnená.
  SAT patrí do NP, a ak by patril do P, tak $P = NP$.

- $TQBF$: Daná je úplne kvantifikovaná boolovská formula s n premennými, rozhodnite, či je pravdivá.
  $TQBF$ patrí do $PSPACE$, a ak by patril do NP, tak $NP = PSPACE$.
  Navyše vieme, že $TQBF$ nepatrí do NL.

Kapitola 8

Syntaktická analýza

Syntaktická analýza je veľmi široká problematika, ktorá nachádza uplatnenie pri tvorbe kompilátorov alebo prekladačov prirodzených jazykov. Dve základné témy, ktorými sa budeme v tejto kapitole zaoberať, sú *parsovanie* a *preklad*. Cieľom parsovania bude k danej gramatike a slovu čo najefektívnejšie nájsť (jeden nejaký) strom odvodenia. Pri preklade budeme mať dané dva formálne systémy, ktoré generujú slová, cieľom bude k slovu vygenerovanému prvým systémom nájsť slovo, ktoré „podobným postupom“ vygeneruje druhý systém. Oba princípy si najľahšie vysvetlíme na jednoduchom príklade.

Majme nasledujúcu jednoduchú bezkontextovú gramatiku: $G = (N, T, P, \sigma)$, kde $N = \{\sigma\}$, $T = \{a, b, c, +, \times\}$ a $P = \{\sigma \to (\sigma) \mid \sigma + \sigma \mid \sigma \times \sigma \mid a \mid b \mid c\}$.

V tejto gramatike vieme odvodiť napríklad slovo $a \times (b + c)$. Keď budeme chcieť k tejto gramatike zostrojiť *parser*, pôjde nám o čo najjednoduchší a najefektívnejší algoritmus, ktorý na vstupe dostane takýto výraz, a na výstupe nám dá jeho strom odvodenia. Všimnite si, že napríklad v našom prípade strom odvodenia zodpovedá možnému postupu vyhodnocovania príslušného aritmetického výrazu.

V praxi sa vhodný parser používa napríklad ako súčasť bežného kompilátora. Gramatika nám popisuje syntax programovacieho jazyka. Slovo, ktoré parsujeme, je program v danom jazyku. Strom odvodenia, ktorý hľadáme, zodpovedá členeniu daného programu na logické celky – napríklad program na funkcie, funkciu na príkazy, príkaz na ďalej nedeliteľné tzv. *syntaktické tokeny*.

Aritmetické výrazy vieme zapisovať aj v *postfixovej notácii*. Tá sa od klasického zápisu líši tým, že operátor nasleduje až po všetkých svojich operandoch. Vyššie uvedený výraz by sme teda vedeli v postfixovej notácii zapísať ako $abc+\times$. Skúste sa zamyslieť, ako by vyzeral algoritmus, ktorý bude prekladať výrazy z klasickej (infixovej) notácie do postfixovej. Takto bude vo všeobecnosti fungovať *preklad*. Veľký význam nadobudla táto oblasť po tom, ako sa začali vo väčšom meradle používať XML dokumenty. Transformácie XML dokumentov totiž predstavujú práve preklad v zmysle, v akom sme ho intuitívne popísali v našom príklade.

8.1 Všeobecné bezkontextové gramatiky

V časti 3.12 sme si ukázali algoritmus od Cockeho, Youngera a Kasamiho, pomocou ktorého vieme k ľubovoľnej bezkontextovej gramatike G a slovu z $L(G)$ zostrojiť jeden strom odvodenia.

Nevýhoda tohto algoritmu je jeho stále ešte veľká časová zložitosť – čas potrebný na zostrojenie stromu odvodenia rastie úmerne tretej mocnine dĺžky vstupného slova. (Ak teda považujeme gramatiku za konštantnú, takto naprogramovaný parser má časovú zložitosť $O(n^3)$.) Toto je pre praktické použitie často priveľa – predstavte si kompilovanie programu, ktorý má desaťtisíce riadkov.

Valiant v roku 1975 ukázal efektívnejší postup parsovania pre všeobecnú bezkontextovú gramatiku, jeho zložitosť pre pevne zvolenú gramatiku bola $O(M(n))$, kde $M(n)$ je čas potrebný na

vynásobenie dvoch boolovských matíc rozmerov $n \times n$. V súčasnosti najefektívnejší algoritmus na násobenie matíc má teoretickú časovú zložitosť $O(n^{2.376})$ a vymysleli ho Coppersmith a Winograd v roku 1990. Pre praktické použitie sú však tieto algoritmy nevhodné, pre rozumne veľké matice nie je známy algoritmus, ktorý by bol rádovo efektívnejší ako triviálny, ktorého časová zložitosť je $O(n^3)$.

Lee v roku 1997 dokázala, že Valiantov výsledok je optimálny – neexistuje efektívnejší postup, ako parsovať všeobecnú bezkontextovú gramatiku.

Našim cieľom teda bude nájsť nejakú čo najväčšiu triedu gramatík, ktoré budeme vedieť efektívne parsovať. Hranica zjavne leží niekde medzi regulárnymi a bezkontextovými gramatikami.

8.2 LR(0) gramatiky

8.2.1 Motivácia a pomocné definície

Prvým kandidátom, s ktorým sme sa už stretli, by mohli byť jednoznačné gramatiky – bezkontextové gramatiky, v ktorých má každé slovo z generovaného jazyka práve jeden strom odvodenia. Ako je to s nimi?

Vo vetách 4.2.8 a 4.2.9 sme ukázali, aký je vzťah medzi DPDA a jednoznačnými bezkontextovými gramatikami – ku všetkým jazykom, ku ktorým existuje DPDA, existuje aj jednoznačná bezkontextová gramatika, naopak to však neplatí.

Toto ale nie je úplne radostná situácia. Pre nás sú totiž významným modelom práve DPDA. Podľa daného DPDA A vieme totiž ľahko naprogramovať jednoduchý a efektívny parser pre jazyk $L(A)$. Skúsme teda nejako vymedziť triedu gramatík, ktorá bude čo najväčšia, ale ku každej gramatike z tejto triedy bude existovať ekvivalentný DPDA.

Celkom dobrým kandidátom sa ukážu byť $LR(0)$ gramatiky, ktoré v nasledujúcom texte definujeme. Pre zaujímavosť, označenie $LR(0)$ znamená, že vieme nájsť pravé krajné odvodenie slova tak, že ho spracúvame zľava doprava (Left to Right) a potrebujeme sa pozerať dopredu na 0 symbolov.

Začneme definíciou niekoľkých pojmov, ktoré budeme potrebovať.

Definícia 8.2.1 *Nech G je bezkontextová gramatika. Potom LR-elementom voláme ľubovoľné pravidlo G, ktorého pravá strana obsahuje navyše jednu bodku.*

Príklad 8.2.1 Uvažujme gramatiku $G = (N, T, P, \sigma)$, kde $N = \{\sigma, \beta, \gamma\}$, $T = \{a, b, c\}$ a
$P = \{\sigma \to \beta c, \quad \beta \to \beta\gamma \mid \gamma, \quad \gamma \to a\beta b \mid ab\}$.
Pravidlu $\sigma \to \beta c$ zodpovedajú LR-elementy $\sigma \to \bullet\beta c$, $\sigma \to \beta \bullet c$ a $\sigma \to \beta c\bullet$.
Keby sme v G mali pravidlo $\sigma \to \varepsilon$, zodpovedal by mu LR-element $\sigma \to \bullet$.

Definícia 8.2.2 *Znakom $\underset{rm}{\Longrightarrow}$ budeme označovať krok odvodenia v pravom krajnom odvodení, skrátene budeme niekedy hovoriť o* pravom kroku odvodenia*. Vetné formy, ktoré vieme dostať zo začiatočného neterminálu na niekoľko pravých krokov odvodenia, voláme* pravé vetné formy*.*

Príklad 8.2.2 V gramatike G z príkladu 8.2.1 platí napríklad:

$$\sigma \underset{rm}{\Longrightarrow} \beta c \underset{rm}{\Longrightarrow} \beta\gamma c \underset{rm}{\Longrightarrow} \beta a\beta bc$$

Preto $\beta a\beta bc$ je pravá vetná forma. V gramatike G síce platí $\beta\gamma c \Rightarrow \gamma\gamma c$, ale neplatí $\beta\gamma c \underset{rm}{\Longrightarrow} \gamma\gamma c$.

Definícia 8.2.3 *Výskyt pravej strany niektorého pravidla gramatiky vo vetnej forme budeme volať* handle*. Prefix pravej vetnej formy voláme* životaschopný*, ak žiadna handle v danej vetnej forme nekončí skôr ako daný prefix. (Najdlhší životaschopný prefix danej vetnej formy teda siaha po koniec najskôr končiacej handle.)*

Príklad 8.2.3 Už vieme, že gramatike G z príkladu 8.2.1 je $\beta a\beta bc$ pravá vetná forma. Jediná handle v nej je $a\beta b$ (čo je pravá strana pravidla $\gamma \to a\beta b$). Životaschopné prefixy tejto vetnej formy sú teda ε, β, βa, $\beta a\beta$ a $\beta a\beta b$.

Keď nájdeme na konci životaschopného prefixu handle, môžeme len dúfať, že naozaj v pravom krajnom odvodení vznikla použitím zodpovedajúceho pravidla, nemusí to vždy byť pravda.

Intuitívne môžeme povedať, že LR(0) gramatiky budú práve tie, v ktorých to naozaj vždy pravda bude.

Pre takéto gramatiky budeme potom jednoducho vedieť zostrojiť pravé krajné odvodenie – v princípe v každom okamihu nájdeme najľavejšiu handle a nahradíme ju zodpovedajúcim neterminálom.

Formálnu definíciu LR(0) gramatiky si ešte odložíme na neskôr, teraz si ukážeme na príklade takéto zostrojenie pravého krajného odvodenia.

Príklad 8.2.4 Zostaneme pri gramatike G z príkladu 8.2.1. Všimnime si slovo abc. Toto slovo je pravou vetnou formou. Najľavejšia (a jediná) handle v ňom je ab. Nahradíme ju ľavou stranou pravidla, dostávame vetnú formu γc. Tam je najľavejšou handle γ, jej nahradením dostávame vetnú formu βc a z nej σ.

Takto sme teda dostali odvodenie $\sigma \underset{rm}{\Longrightarrow} \beta c \underset{rm}{\Longrightarrow} \gamma c \underset{rm}{\Longrightarrow} abc$.

Našim cieľom bude zostrojenie parsera, ktorý nepotrebuje mať v pamäti celú vetnú formu. Budeme slovo, ktoré parsujeme, čítať zľava doprava, a vždy, keď nájdeme handle, nahradíme si ju zodpovedajúcim neterminálom z ľavej strany. Pamätať si teda budeme aktuálnu podobu doteraz spracovanej časti vetnej formy – čiže práve nejaký jej životaschopný prefix.

Aby bol náš parser efektívny, potrebujeme ešte vyriešiť jednu vec – ako efektívne zistiť, že sa na nejakom mieste aktuálnej vetnej formy nachádza handle. Tu prídu k slovu vyššie definované LR-elementy.

Definícia 8.2.4 *Nech p je životaschopný prefix pre danú bezkontextovú gramatiku $G = (N, T, P, \sigma)$. Hovoríme, že LR-element $\varphi \to w_1 \bullet w_2$ je* platný *pre tento životaschopný prefix, ak platí $\sigma \underset{rm}{\Longrightarrow}^* u\varphi v \underset{rm}{\Longrightarrow} uw_1w_2v$ a $uw_1 = p$.*

Platný LR-element tvaru $\varphi \to w\bullet$ voláme úplný.

Poznámka 8.2.1 Všimnime si, že keďže v definícii hovoríme o pravom krajnom odvodení, slovo v musí obsahovať len terminály.

Posledná definícia vlastne hovorí, že uvedený LR-element $\varphi \to w_1 \bullet w_2$ je platný pre životaschopný prefix p, ak p končí na w_1, a navyše sa v G dá odvodiť pravá vetná forma, ktorá má prefix p a obsahuje na správnom mieste celú handle w_1w_2.

Význam tejto definície je nasledovný: Keď poznáme životaschopný prefix p, množina jemu zodpovedajúcich platných LR-elementov hovorí, ktoré handle sa na jeho konci v nejakom platnom odvodení môžu ako posunuté vyskytnúť.

Ak existuje k danému životaschopnému prefixu platný úplný LR-element, znamená to, že sme práve našli handle.

Náš parser si bude spolu s aktuálnym životaschopným prefixom udržiavať množinu platných LR-elementov. V nasledujúcej vete ukážeme, ako si vieme túto množinu upraviť po prečítaní ďalšieho písmena parsovaného slova.

Veta 8.2.1 *Nech $G = (N, T, P, \sigma)$ je bezkontextová gramatika, nech E je množina LR-elementov, ktoré zodpovedajú jej pravidlám. Definujme nedeterministický konečný automat A nasledovne: bude $A = (K, \Sigma, \delta, q_0, F)$, kde $K = \{q_0\} \cup E$, $\Sigma = T$ a δ-funkcia je definovaná nasledovne:*

$$\begin{aligned}
\delta(q_0, \varepsilon) &= \{(\sigma \to \bullet w) \mid (\sigma \to w) \in P\} \\
\delta(\,(\varphi \to w_1 \bullet \xi w_2), \varepsilon) &= \{(\xi \to \bullet w_3) \mid (\xi \to w_3) \in P\} \\
\delta(\,(\varphi \to w_1 \bullet x w_2), x) &= \{(\varphi \to w_1 x \bullet w_2)\}
\end{aligned}$$

Potom platí:

$(q_0, p) \vdash_A^* (\,(\varphi \to w_1 \bullet w_2), \varepsilon) \quad \Longleftrightarrow \quad (\varphi \to w_1 \bullet w_2)$ *je platný LR-element pre životaschopný prefix p*

Dôkaz. Všimnime si, že až na začiatočný stav sú stavy automatu A práve všetky LR-elementy. Veta tvrdí, že konkrétny LR-element je platný pre daný životaschopný prefix p práve vtedy, ak po prečítaní p vie náš automat byť v stave zodpovedajúcom danému LR-elementu.

Neformálne, prvé dva riadky δ-funkcie nám umožňujú v ľubovoľnom okamihu, v ktorom by sa mohol vo vetnej forme vyskytnúť neterminál ξ, rozhodnúť, že (namiesto neho) ideme očakávať výskyt jeho pravej strany. Tretí riadok δ-funkcie zabezpečuje kontrolu výskytu – ak je nasledujúci symbol x zo vstupu (môže to byť terminál aj neterminál) zhodný s očakávaným, posunieme v aktuálnom LR-elemente bodku.

Formálny dôkaz tvrdenia neuvádzame.

Dôsledok 8.2.2 *Keď k vyššie zostrojenému NKA štandardnou konštrukciou zostrojíme ekvivalentný DKA, bude platiť, že stav, do ktorého sa dostane z q_0 prečítaním p bude práve množina platných LR-elementov pre p.*

8.2.2 Definícia LR(0) gramatiky

Definícia 8.2.5 *Hovoríme, že bezkontextová gramatika G je LR(0) gramatika, ak sú splnené nasledujúce dve podmienky:*

1. *Začiatočný neterminál sa nevyskytuje na pravej strane žiadneho pravidla.*
2. *Pre každý životaschopný prefix G platí, že ak je $\xi \to \varphi\bullet$ platný úplný LR-element pre tento prefix, tak nie je platný žiadny iný úplný LR-element, ani žiadny LR-element, v ktorom je bodka pred terminálnym symbolom.*

Poznámka 8.2.2 Veta 8.2.1 a jej dôsledok nám dávajú spôsob ako rozhodnúť, či je daná gramatika LR(0) – stačí v DKA, ktorý nám pre každý životaschopný prefix hovorí množinu platných LR-elementov, preskúmať všetky dosiahnuteľné stavy.

8.2.3 Vzťah LR(0) gramatík a DPDA

Ukážeme, že LR(0) gramatikami vieme generovať presne tú istú triedu jazykov, ktoré vieme rozpoznávať na DPDA **prázdnou pamäťou**.

Na tomto mieste odporúčame čitateľovi, aby si pripomenul výsledky, ktoré uvádzame v časti 4.2.2. Pomocou DPDA vieme prázdnou pamäťou rozpoznávať len jazyky, v ktorých žiadne slovo nie je prefixom iného. Toto sa však ľahko dosiahne technickou úpravou – zavedením nového znaku, ktorý pridáme na koniec každého slova v jazyku.

Veta 8.2.3 *Ku každej LR(0) gramatike G existuje DPDA A taký, že $N(A) = L(G)$.*

Dôkaz. Myšlienku konštrukcie sme už načrtli v predchádzajúcom texte: Na zásobníku si pamätáme aktuálny životaschopný prefix, a navyše pre každý jeho znak sadu LR-elementov, ktoré sú platné pre ním končiaci prefix. Vždy, keď sa dostaneme do situácie, že jeden z platných LR-elementov je úplný, našli sme handle w, zodpovedajúcu nejakému pravidlu $\varphi \to w$. Nájdenú handle odstránime zo zásobníka. Teraz si na vrchu zásobníka pozrieme aktuálnu množinu platných LR-elementov, vložíme na zásobník φ a dopočítame aktuálnu množinu platných LR-elementov.

Tento algoritmus sa v oblasti parserov zvykne označovať *algoritmus shift-reduce*. (Posunutie sa na ďalší znak vstupu je „shift“, nahradenie pravej strany pravidla ľavou je „reduce“.)

Formálnu konštrukciu DPDA neuvádzame.

Dôsledok 8.2.4 *Každá LR(0) gramatika je jednoznačná.*

Veta 8.2.5 *Ku každému DPDA A existuje LR(0) gramatika G taká, že $N(A) = L(G)$.*

Dôkaz. Dá sa dokázať, že gramatika, ktorú dostaneme štandardnou konštrukciou (veta 3.9.2), je pre deterministické PDA vždy LR(0) gramatikou.

Kapitola 9

Riešené úlohy

9.1 Základné pojmy

Úloha 9.1.1 *Rozhodnite, či pre ľubovoľné jazyky L_1, L_2 platí: $L_1^*(L_2L_1^*)^* = (L_1 \cup L_2)^*$.*

Tvrdenie platí. Dokážeme obe inklúzie.

Nech $w \in L_1^*(L_2L_1^*)^*$. Potom $\exists k \geq 0, v_i \in L_1^*, u_i \in L_2; w = v_1u_1v_2 \dots u_kv_{k+1}$. Zjavne každé zo slov u_i, v_i patrí do $(L_1 \cup L_2)^*$. A takisto zjavne $(L_1 \cup L_2)^*$ je uzavretá na zreťazenie, preto aj $w \in (L_1 \cup L_2)^*$.

Naopak, nech $w \in (L_1 \cup L_2)^*$, teda $\exists k \geq 0, w_i \in (L_1 \cup L_2);\ w = w_1 \dots w_k$. Nech i_j sú indexy tých podslov, ktoré patria do L_2. Potom ale:

$$w = \underbrace{w_1 \dots w_{i_1-1}}_{\in L_1^*} \underbrace{w_{i_1}}_{\in L_2} \underbrace{w_{i_1+1} \dots w_{i_2-1}}_{\in L_1^*} \underbrace{w_{i_2}}_{\in L_2} \dots \underbrace{w_{i_n}}_{\in L_2} \underbrace{w_{i_n+1} \dots w_k}_{\in L_1^*}$$

A teda $w \in L_1^*(L_2L_1^*)^*$.

Úloha 9.1.2 *Pomocou konečného počtu operácii zreťazenia, zjednotenia, prieniku, homomorfizmu a inv. homomorfizmu zostrojte z jazyka $L_1 = \{u \mid u \in \{a,b\}^* \ \wedge \ \#_a(u) = \#_b(u)\}$ jazyk $L_2 = \{a^ib^ic^i \mid i \geq 0\}$.*

Dlhšie zrozumiteľnejšie riešenie: Nech h_1 je homomorfizmus taký, že $h(a) = a$ a $h(b) = \varepsilon$. Potom $L_a = h_1(L_1) = \{a^i \mid i \geq 0\}$. Analogicky si vyrobíme L_b a L_c. Potom $L_{ab} = L_a.L_b = \{a^ib^j \mid i,j \geq 0\}$. Všimnime si teraz slová z $L_{a=b} = L_1 \cap L_{ab}$. Sú to práve slová, ktoré majú rovnako a a b a navyše ľubovoľné a je pred ľubovoľným b. To ale znamená, že $L_{a=b} = \{a^ib^i \mid i \geq 0\}$. No a odtiaľto ďalej to bolo ako jedna domáca úloha – vyrobíme jazyk $L_{b=c}$ (homomorfizmom, ktorý a zobrazí na b a b na c). Keď si teraz vezmeme jazyk $(L_{a=b}.L_c) \cap (L_a.L_{b=c})$, tak obsahuje práve slová tvaru $a^ib^jc^k$, v ktorých je rovnako a a b a zároveň je rovnako b a c. To ale znamená, že je to práve L_2, q.e.d.

Kratšie riešenie: Homomorfizmom „preložme“ L_1 na jazyk L_3 všetkých slov, v ktorých je rovnako b a c. Potom $L_2 = (L_1.L_c) \cap (L_a.L_3)$. Rozmyslite si, prečo.

9.2 Regulárne jazyky

Úloha 9.2.1 *Pre dané jazyky L_1, L_2 nad abecedou Σ definujeme operáciu Shuffle nasledovne:*

$$\text{Shuffle}(L_1, L_2) = \Big\{ w \ \Big| \ \exists n \in N, u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^*; \\ w = u_1v_1u_2v_2 \dots u_nv_n \ \wedge \ u_1u_2 \dots u_n \in L_1 \ \wedge \ v_1v_2 \dots v_n \in L_2 \Big\}$$

Neformálne povedané, jazyk $\text{Shuffle}(L_1, L_2)$ *obsahuje všetky slová, ktoré vzniknú zmiešaním slova z* L_1 *a slova z* L_2*. Rozhodnite a dokážte, či je trieda regulárnych jazykov uzavretá na operáciu Shuffle. (Teda či* $\forall L_1, L_2 \in \mathcal{R};\ \text{Shuffle}(L_1, L_2) \in \mathcal{R}$*.)*

Správna odpoveď je áno. Majme deterministické automaty A_1, A_2 také, že $L(A_i) = L_i$. Označme $A_1 = (K_1, \Sigma_1, \delta_1, q_{01}, F_1), A_2 = (K_2, \Sigma_2, \delta_2, q_{02}, F_2)$. Zostrojíme nedeterministický automat A pre $\text{Shuffle}(L_1, L_2)$. Bude $A = (K, \Sigma, \delta, [q_{01}, q_{02}], F)$, kde:
$K = K_1 \times K_2, \quad F = F_1 \times F_2, \quad \Sigma = \Sigma_1 \cup \Sigma_2$
$\forall\, [q_a, q_b] \in K; \forall x \in \Sigma; \quad \delta([q_a, q_b], x) = \Big\{ [\delta_1(q_a, x), q_b], [q_a, \delta_2(q_b, x)] \Big\}$

Neformálna myšlienka konštrukcie: Náš automat si v stave pamätá stavy oboch automatov A_1, A_2. Vždy, keď mu príde písmeno, nedeterministicky sa rozhodne, v ktorom z automatov spraví krok. Ak sa mu podarí natipovať také rozdelenie slova, že A_1 aj A_2 skončí v akceptačnom stave, akceptuje. (Tie písmená, na ktoré robil kroky v A_1 tvoria u, ostatné tvoria v.)

Formálny dôkaz by mal vyzerať tak, že dokážeme obe inklúzie medzi $L(A)$ a $\text{Shuffle}(L_1, L_2)$. Myšlienka dôkazu jednej inklúzie je v predchádzajúcom odseku, druhú inklúziu dokážeme tak, že zoberieme slovo z $\text{Shuffle}(L_1, L_2)$, to sa musí dať rozdeliť na u a v, pre ktoré existujú akc. výpočty v A_1, resp. A_2, z tých zostrojíme akc. výpočet A na w.

Úloha 9.2.2 *Zostrojte deterministický alebo nedeterministický konečný automat akceptujúci jazyk* $L = \{w \mid w \in \{a, b\}^*,\ \#_a(w)$ *je párny* $\wedge$ *vo* w *je tretie písmeno od konca* $b\ \}$.

Náš automat bude deterministický. V stave si bude pamätať paritu počtu doteraz prečítaných písmen a a posledné tri písmená. (Ak ešte neprečítal 3 písmená, príslušné pamätané písmeno bude a, t.j. „nebolo b“.) Uvedieme formálnu konštrukciu.

Bude $A = (K, \Sigma = \{a, b\}, \delta, [0, aaa], F)$. Stavy budú $K = \{[p, xyz] \mid p \in \{0, 1\}, x, y, z \in \Sigma\}$. Akceptačné stavy sú tie, kde sme prečítali párny počet a a tretie písmeno od konca bolo b, teda $F = \{[0, bxy] \mid x, y \in \Sigma\}$.

Prechodová funkcia:

$$\forall p \in \{0, 1\}, \forall x, y, z \in \Sigma; \quad \delta([p, xyz],\ a) = [1 - p, yza]$$

$$\forall p \in \{0, 1\}, \forall x, y, z \in \Sigma; \quad \delta([p, xyz],\ b) = [p, yzb]$$

Dôkaz, že $L(A) = L$:

Každý akceptačný výpočet musí mať zjavne aspoň 3 kroky.

Indukciou od počtu krokov výpočtu dokážeme, že od tretieho kroku výpočtu platí tvrdenie: Automat A je v stave $[p, xyz]$ práve vtedy, ak parita počtu doteraz prečítaných a je p a posledné tri písmená boli x, y, z.

1° Rôznych výpočtov dĺžky 3 je osem, všetky vyhovujú.

2° Z δ-fcie je zjavné, že ak tvrdenie platilo po k krokoch výpočtu, platí aj po $k + 1$ krokoch.

Z práve dokázaného tvrdenia vyplýva, že po dočítaní vstupu je automat v akceptačnom stave iff bol prečítaný párny počet a a posledné 3 písmená boli b, x, y, čiže iff vstup bol z L.

Úloha 9.2.3 *Definujme odmocninu z jazyka nasledovne:* $\sqrt{L} = \{w \mid ww \in L\}$
Rozhodnite a dokážte, či musí byť odmocnina z regulárneho jazyka regulárna.

Majme DFA A pre L, zostrojíme DFA A', ktorý bude čítať vstup a pre každý stav $q \in K$ si pamätať, v akom stave by bol A, keby začal čítať vstup v q. T.j. A' bude mať $K' = K^{|K|}$, $q_0 = \big[q_0, \ldots, q_{|K|-1}\big]$, $\delta'([q_{i_1}, \ldots, q_{i_{|K|-1}}], x) = [\delta(q_{i_1}, x), \ldots, \delta(q_{i_{|K|-1}}, x)]$.

Keď dočítame vstup w, vieme, v akom stave by bol A po prečítaní w a vieme aj to, do akého stavu by prešiel ďalším prečítaním w. Akceptujeme iff tento stav je akceptačný. Teda $F' = \{[q_{i_1}, \ldots, q_{i_{|K|-1}}] \mid q_{i_{i_1}} \in F\}$.

Iná možnosť je zostrojiť nedeterministický A', ktorý si na začiatku výpočtu tipne, v akom stave bude A po prečítaní w. Ďalej simuluje A z q_0 aj z tipnutého stavu q. Na konci slova akceptuje, ak sa z q_0 dostal naozaj do q a z q sa dostal do akceptačného stavu.

Tvrdenie triviálne vyplýva z toho, že dvojsmerné konečné automaty (také, čo môžu ľubovoľne behať po vstupe) sú rovnako silné ako jednosmerné.

Úloha 9.2.4 *Daný je a) determ., b) nedeterm. konečný automat* $A = (K, \Sigma, \delta, q_0, F)$. *Pritom* $\Sigma = \{a, b\}$, $|K| = n$ *a platí:*

$$\forall w \in \Sigma^*;\ |w| \leq n + 3 \ \Rightarrow\ w \in L(A)$$

Rozhodnite a dokážte, či potom musí platiť $L(A) = \Sigma^*$.

a) Nech q je ľub. dosiahnuteľný stav. Do q sa vieme dostať na najviac $n - 1$ krokov. Keďže automat slovo zodpovedajúce tejto ceste akceptuje, musí byť q akceptačný stav. Potom ale $L(A) = \Sigma^*$, lebo všetky dosiahnuteľné stavy A sú akceptačné.

b) Tu to už neplatí. Zoberme jazyk $L = \{a^x \mid x = 0 \vee 15 \text{ nedelí } x\}$. Keď dostaneme vstupné slovo, máme akceptovať, ak dĺžka nie je deliteľná 15. Nedeterministickému automatu stačí 9 stavov. Začiatočný, v tom si tipneme, či dĺžka nie je deliteľná 3 alebo 5, prejdeme do prísl. časti a overíme, či naozaj.

9.3 Bezkontextové jazyky

Úloha 9.3.1 *Je daná gramatika* $G_1 = (\{S, A, B, C, D, E, F\}, \{a, b, c\}, P_1, S)$ *, kde* $P_1 = \{$
$S \to aAS \mid bBC \mid E$
$A \to aA \mid bB \mid a$
$B \to BA \mid BB \mid BD$
$C \to caC \mid acA \mid EB$
$D \to a \mid AD \mid \varepsilon$
$E \to \varepsilon \mid E$
$F \to aAa \mid b\}$
Zostrojte k nej štandardnou konštrukciou redukovanú gramatiku G_2 *takú, že* $L(G_1) = L(G_2)$. *(Redukovaná gramatika je gramatika, v ktorej je každý neterminál dosiahnuteľný a z každého neterminálu vieme v tejto gramatike odvodiť terminálne slovo.)*

Najskôr odstránime neterminály, z ktorých sa nedá odvodiť terminálne slovo, potom nedosiahnuteľné neterminály. Postupne zostrojíme množiny neterminálov, pre ktoré existuje strom odvodenia terminálneho slova, ktorého hĺbka je $\leq k$:

$H_1 = \{A, D, E, F\}$, $H_2 = H_1 \cup \{C\}$, $H_3 = H_2$ a môžeme skončiť.

Množina neterminálov, z ktorých sa nedá odvodiť terminálne slovo je teda $N - H_3 = \{B\}$. Neterminál B a pravidlá, ktoré ho obsahujú, môžeme z gramatiky G vynechať. Zostrojme teraz pre upravenú gramatiku množiny neterminálov, ktoré sú z S dosiahnuteľné na $\leq k$ krokov:

$H_0 = \{S\}$, $H_1 = H_0 \cup \{A, E\}$, $H_2 = H_1$ a končíme.

Neterminály $N' - H_2 = \{C, D, F\}$ sú nedosiahnuteľné, môžeme ich odstrániť. Vo výslednej gramatike teda zostanú neterminály S, A, E a príslušné pravidlá.

Úloha 9.3.2 *Rozhodnite, či je jazyk*

$$L = \{w \mid w \in \{a, b, c\}^* \wedge \#_a(w) > \#_b(w) > \#_c(w)\}$$

bezkontextový. Ak nie, dokážte. Ak áno, zostrojte bezkontextovú gramatiku, ktorá bude tento jazyk generovať. Súčasťou konštrukcie by mal byť aj slovný popis množiny pravidiel a myšlienka dôkazu správnosti.

Dokážeme, že nie je bezkontextový, lebo nespĺňa pumpovaciu lemu. Sporom. Ak by ju spĺňal, existujú p, q s istými vlastnosťami. Zoberme ale slovo $a^{p+q+2}b^{p+q+1}c^{p+q}$. To je dlhšie ako p, teda sa má dať rozdeliť a napumpovať. Ale ak nepumpujeme žiadne c, nevieme napumpovať na nultú, ak nejaké c pumpujeme, nepumpujeme žiadne a, a teda nevieme napumpovať na tretiu.

Úloha 9.3.3 *Rozhodnite, či je jazyk*

$$L = \{u\#v \mid u, v \in \{a, b\}^* \ \wedge \ (u = v \ \vee \ u = v^R)\}$$

(nad abecedou $\{a, b, \#\}$) bezkontextový. Ak áno, zostrojte zásobníkový automat, ktorý ho bude (či už stavom alebo prázdnou pamäťou) akceptovať a zdôvodnite správnosť jeho konštrukcie. Ak nie je bezkontextový, dokážte to.

Intuícia: Bezkontextový nie je, pomocou jedného zásobníka nevieme overiť podmienku $u = v$.

Dokážeme sporom, že L nie je bezkontextový. Nech je, potom preň platí pump. lema, teda existujú nejaké p, q, nech $x = p+q+7$. Zoberme slovo $w = a^x b^x \# a^x b^x$. Toto slovo je zjavne dlhšie ako p. Ukážeme, že ho nevieme prípustne rozdeliť (na u, v, x, y, z) a napumpovať.

Zjavne ani jedna pumpovaná časť nesmie obsahovať $\#$. Ak sú obe pumpované časti na tej istej strane $\#$, po napumpovaní na druhú je „ľavá“ strana dlhšia ako „pravá“, teda to nie je slovo z jazyka. Jediná iná možnosť je, že v je naľavo od $\#$ a y napravo. Potom ale $u = b^i$, $y = a^j$. Po napumpovaní na druhú dostávame slovo $a^x b^{x+i} \# a^{x+j} b^x$, ktoré opäť zjavne nepatrí do L a vyhrali sme.

Iné riešenie s pár dirty trikmi. Opäť sporom. Zbavme sa „druhej časti“ jazyka. Ak $u = v^R$, tak prvé a posledné písmeno slova sú rovnaké. Zoberme $L_2 = L \cap \{awb \mid w \in \{a, b, \#\}^*\}$ – tie slová z L, kde prvé a posledné písmeno sú rôzne. L_2 je tiež bezkontextový (robili sme prienik s reg. jazykom). Ale zjavne $L_2 = \{axb\#axb \mid x \in \{a, b\}^*\}$. To už je takmer náš starý známy jazyk. Môžeme použiť klasický postup s pumpovacou lemou, prípadne sa hrať ďalej.

Nech h je taký homomorfizmus, ktorý x aj $\overline{x}$ zobrazí na x. Potom $h^{-1}(L_2)$ je akoby L_2, pričom v každom slove sú nad niektorými písmenami čiarky (a každé slovo je „očiarkované“ všetkými spôsobmi). Zoberme $L_3 = h^{-1}(L_2) \cap \{\overline{a}x\overline{b}\#\overline{a}y\overline{b} \mid x, y \in \{a, b\}^*\}$. Aj L_3 je bezkontextový. Prienikom s týmto reg. jazykom sme z $h^{-1}(L_2)$ vybrali práve tie slová, kde sú očiarkované písmená, čo nám vadia.

Teraz keď ich máme označené, vieme ich vhodným homomorfizmom g ($g(\overline{x}) = \varepsilon$, $g(x) = x$) zmazať. Aj jazyk $L_4 = g(L_3)$ je bezkontextový. Ale $L_4 = \{w\#w \mid w \in \{a, b\}^*\}$ – spor.

Úloha 9.3.4 *Rozhodnite, či je jazyk $L = \{w \mid w \in \{a, b\}^* \wedge \#_a(w) = 2\#_b(w)\}$ bezkontextový. Ak nie, dokážte. Ak áno, zostrojte bezkontextovú gramatiku, ktorá bude tento jazyk generovať. Súčasťou riešenia by mal byť aj slovný komentár k množine pravidiel a myšlienka dôkazu správnosti.*

Jazyk je bezkontextový. Jedna možná konštrukcia gramatiky je zostrojiť pre L najskôr zásobníkový automat. Ten je pomerne triviálny – na zásobníku si udržuje, čoho a o koľko viac doteraz prečítal. No a na automat už stačí použiť štandardnú konštrukciu.

Samozrejme existuje aj viacero spôsobov, ako zostrojiť priamo gramatiku. Pravidlá hľadanej gramatiky mohli vyzerať napr. nasledovne:

$$\sigma \to a\sigma a\sigma b\sigma \mid a\sigma b\sigma a\sigma \mid b\sigma a\sigma a\sigma \mid \varepsilon$$

V každom kroku buď nevygenerujeme žiaden terminál, alebo vygenerujeme dve a a jedno b, z toho je jedna inklúzia zrejmá. Na druhú treba ukázať, že keď máme ľub. slovo $w \in L$, vieme ho zapísať v jednom z tvarov $auavbx, aubvax, buavax, \varepsilon$ (kde $u, v, x \in L$).

Formálny dôkaz by mohol vyzerať približne nasledovne: Nech $w = w_1 \dots w_n$, položme $f(i) = \#_a(w_1 \dots w_i) - 2\#_b(w_1 \dots w_i)$. Zjavne $f(0) = 0$ a keďže $w \in L$, aj $f(n) = 0$. Prechádzajme teraz s i od 0 do n, v každom kroku sa hodnota f buď zväčší o 1 (ak je príslušný znak a), alebo zmenší o 2 (ak je to b). Ak je $w_1 = b$, je $f_1 = -2$. Zoberme i, kedy je prvýkrát $f(i) = 1$, $j > 0$ kedy je prvýkrát $f(j) = 0$. Zjavne takéto i, j existujú a $i < j$. (Prečo?) Potom ale $w_i = w_j = a$ a tieto dve a rozdelia naše slovo na hľadané 3 časti u, v, x. Ak $w_1 = a$, analogicky.

Trochu iné riešenie. Upravme pravidlá našej gramatiku do tvaru:

$$\sigma \to \sigma a\sigma a\sigma b\sigma \mid \sigma a\sigma b\sigma a\sigma \mid \sigma b\sigma a\sigma a\sigma \mid \varepsilon$$

Jedna inklúzia je naďalej zrejmá. K druhej: Všimnime si, že každé slovo z L obsahuje ako podslovo aspoň jedno zo slov aab, aba, baa (prečo?) a jeho vynechaním dostaneme opäť slovo z L. Indukciou od dĺžky w dokážeme, že ak $w = w_1 \dots w_n \in L$, tak $\sigma \Rightarrow^* \sigma w_1 \sigma w_2 \dots w_n \sigma$. Indukčný krok vyplýva z práve uvedeného pozorovania – keď máme w, nájdeme podslovo, vynecháme ho, z ind. predpokladu vieme odvodiť vetnú formu pre kratšie slovo, ktoré sme vynechaním dostali, no a už len v 1 kroku prepíšeme príslušnú σ.

9.4 Zložitejšie modely

Úloha 9.4.1 *Zistite, či existuje A-prekladač, ktorý preloží jazyk L_1 na jazyk L_2, kde*

$$L_1 = \{w \mid w \in \{a,b,c\}^*,\ \#a = \#b = \#c\}$$
$$L_2 = \{a^n b^{3n} \mid n \geq 0\}$$

Ak áno, zostrojte ho a popíšte, ak nie dokážte.

A-prekladač je vlastne konečný automat, ktorý môže navyše písať na výstup. Netreba zabúdať, že vo výslednom jazyku sú len tie slová, pri ktorých zápise a-prekladač dočítal vstupné slovo a na konci akceptoval. Takto si totiž a-prekladač vie vybrať vhodnú podmnožinu pôvodného jazyka L_1 a preložiť na L_2 len ju.

Myšlienka je jednoduchá: náš a-prekladač bude kontrolovať, či je vstupné slovo tvaru $a^*\{b,c\}^*$ (teda prekladať bude len takéto slová) a pri tom prepisovať a na a, b na b a c na bb.

Formálne bude $A = (K, \Sigma_1, \Sigma_2, H, q_0, F)$, kde podľa zadania $\Sigma_1 = \{a,b,c\}$, $\Sigma_2 = \{a,b\}$. Náš a-prekladač bude mať dva stavy, ktoré budú zároveň akceptačné: $K = F = \{nebolo\_bc, bolo\_bc\}$, začína v stave $q_0 = nebolo\_bc$.

Množina prepisovacích pravidiel bude vyzerať nasledovne:

$$\begin{aligned} H \quad = \quad & \{(nebolo\_bc, a, a, nebolo\_bc) \\ & (nebolo\_bc, b, b, bolo\_bc) \\ & (nebolo\_bc, c, bb, bolo\_bc) \\ & (bolo\_bc, b, b, bolo\_bc) \\ & (bolo\_bc, c, bb, bolo\_bc)\} \end{aligned}$$

Dôkaz správnosti je triviálny, jediná vec, ktorú ešte pri ňom treba spomenúť, je, že ak už a-prekladač A čítal nejaké b alebo c a príde mu a, zasekne sa a nedoprekladá. Z toho je už zrejmé, že $A(L_1) = A(a^*\{b,c\}^*) = L_2$.

Úloha 9.4.2 *Pre dané jazyky L_1, L_2 nad abecedou Σ jazyk* $\mathrm{Shuffle}(L_1, L_2)$ *obsahuje všetky slová, ktoré vzniknú zmiešaním slova z L_1 a slova z L_2. Formálne definujeme operáciu Shuffle nasledovne:*

$$\mathrm{Shuffle}(L_1, L_2) = \Big\{ w \quad \Big| \quad \exists n \in N, u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^*; \\ w = u_1 v_1 u_2 v_2 \dots u_n v_n \ \wedge\ u_1 u_2 \dots u_n \in L_1 \ \wedge\ v_1 v_2 \dots v_n \in L_2 \Big\}$$

K danému regulárnemu jazyku R nad abecedou Σ zostrojte a-prekladač M_R taký, že pre každý jazyk L nad abecedou Σ je $M_R(L) = \mathrm{Shuffle}(L, R)$.

Nech $A = (K_A, \Sigma, \delta, q_0, F_A)$ je NKA pre R, potom $M = (K_A, \Sigma, \Sigma, H, q_0, F_A)$, pričom množina pravidiel $H = \{$

(q, a, a, q) $\forall q \in K_A; \forall a \in \Sigma$ (kopíruje slovo zo vstupu)

(q, ε, a, p) $\forall q \in K_A; \forall a \in \Sigma; \forall p \in \delta(q,a)$ (nedet. generuje slovo z R)

$\}$

Náš a-prekladač teda dostane na vstup slovo z L, to kopíruje na výstup a medzi kopírované písmená nedeterministicky vkladá nejaké ďalšie. Zároveň na týchto nových písmenách simuluje A a akceptuje len vtedy, ak takto primiešané slovo patrilo do R.

Úloha 9.4.3 *Uveďte štandardné 4 definície (čo je to, konfigurácia, výpočet, jazyk) pre* **dvojsmerný nedeterministický zásobníkový automat**. *(Od klasického PDA sa líši tým, že sa môže po vstupnom slove ľubovoľne prechádzať. Teda namiesto čítania písmen zo vstupu ako klasický PDA potrebuje pohyb po páske ako Turingov stroj. Obsah pásky však nesmie meniť!) Slovne porovnajte jeho silu s klasickým PDA.*

Hint: Na vhodnom mieste v definícií pridajte na začiatok a koniec vstupného slova „zarážky", ktoré 2PDA umožnia zistiť, že je na kraji slova a ktoré nesmie prekročiť.

2PDA je 7-ica $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, významy sú rovnaké ako u PDA, len δ nám bude hovoriť aj ako máme pohnúť hlavou:

$$\delta : (K \times (\Sigma \cup \{¢, \$\}) \times \Gamma) \to 2^{K \times \Gamma^* \times \{-1,0,1\}}$$

Nesmieme zabudnúť uviesť, že δ musí byť konečná. Znaky ¢ a \$ budú „zarážky". Pre ne musí platiť, že 2PDA nesmie ísť z ¢ doľava ani z \$ doprava:

$$\forall q \in K, \forall Z \in \Gamma;\ \delta(q, ¢, Z) \subseteq K \times \Gamma^* \times \{0, 1\}$$

$$\forall q \in K, \forall Z \in \Gamma;\ \delta(q, \$, Z) \subseteq K \times \Gamma^* \times \{-1, 0\}$$

Konfigurácia 2PDA je štvorica $(q, z, ¢w\$, i)$, kde $q \in K$ je stav, $z \in \Gamma^*$ slovo na zásobníku (vrch je napravo), $¢w\$$ je vstupné slovo so zarážkami (=obsah pásky) a $i \in \{0, \ldots, |w| + 1\}$ je pozícia, kde sa nachádza hlava.

Krok výpočtu 2PDA je relácia $\vdash$ na jeho konfiguráciách taká, že platí: (Zaveďme značenie, že w_i je i-te písmeno slova w, w_0 je ¢, $w_{|w|+1}$ je \$.)

$$(q, zS, ¢w\$, i) \vdash (p, zT, ¢w\$, i + d) \iff (p, T, d) \in \delta(q, w_i, S)$$

Jazyk akceptovaný 2PDA A je množina

$$L(A) = \{w \mid \exists Z \in \Gamma^*, i \geq 0, q_F \in F;\ (q_0, Z_0, ¢w\$, 1) \vdash^* (q_F, Z, ¢w\$, i)\}$$

2PDA sú silnejšie ako klasické PDA. Klasické PDA ľahko odsimulujeme: podľa toho, či sme simulovali krok na ε alebo písmenko, pohneme hlavou o 0 alebo 1. Vieme navyše akceptovať niektoré nebezkontextové jazyky, napr. $L = \{a^n b^n c^n \mid n \geq 0\}$.

2PDA pre vyššie uvedený jazyk môže vyzerať napr. nasledovne: Prejde zľava doprava a overí tvar $a^i b^j c^k$. Vráti sa späť na začiatok. Prejde na koniec a overí, či $i = j$. Vráti sa späť na začiatok. Prejde na koniec a overí, či $j = k$. Ak sa dostal až sem, akceptuje.

9.5 Kontextové jazyky

Úloha 9.5.1 *Rozhodnite a dokážte, či je trieda $\mathscr{L}_{DLBA}$ jazykov akceptovaných deterministickými lineárne ohraničenými automatmi uzavretá na zjednotenie.*

Je uzavretá. Myšlienka riešenia: Spravím si poschodovú pásku, skopírujem vstupné slovo, teraz paralelne simulujem na jednej kópií DLBA pre jeden jazyk, na druhej DLBA pre druhý jazyk, ak aspoň jeden akceptuje, akceptujem aj ja. Paralelnú simuláciu potrebujeme kvôli tomu, že aj DLBA sa môže zacykliť.

Ukážeme si ešte iné riešenie. DLBA má na danom vstupe w len konečný počet možných konfigurácií, vieme ho zhora odhadnúť $c^{|w|}$ pre vhodné c. K ľubovoľnému DLBA A vieme zostrojiť ekvivalentný, ktoré vždy zastaví. Ten bude fungovať nasledovne: Spraví si viacstopú pásku. Na jednej zo stôp bude simulovať A, na druhej stope si bude v c-čkovej sústave počítať počet krokov A, ktoré už odsimuloval. Keď mu táto páska „pretečie", vie, že A sa zacyklil a zasekne sa.

Ku každému jazyku z $\mathscr{L}_{DLBA}$ teda existuje DLBA, ktoré ho akceptuje a na každom vstupe zastaví. Pomocou takýchto DLBA ľahko dokážeme viacero uzáverových vlastností $\mathscr{L}_{DLBA}$. Zjednotenie: Skopírujeme si vstup na inú stopu pásky, aby sme si ho nezničili. Spustíme DLBA (v

našom normálnom tvare) pre prvý jazyk, určite zastaví, ak sme neakceptovali, z pomocnej pásky obnovíme vstup a spustíme DLBA pre druhý jazyk.

Úloha 9.5.2 *Zadefinujte (uveďte štandardné 4 definície) „skáčúce LBA“, ktoré na rozdiel od normálneho LBA nevie hýbať hlavou doľava, namiesto toho ale vie skočiť hlavou na ľavý endmarker (cent). Porovnajte jeho silu so štandardným LBA.*

Skáčúce LBA je n-tica $A = (K, \Sigma, \Gamma, \delta, q_0, F)$, kde K je konečná množina stavov, $q_0 \in K$ počiatočný stav, $F \subseteq K$ množina akceptačných stavov, Σ je konečná vstupná a $\Gamma \supseteq \Sigma \cup \{¢, \$\}$ konečná pracovná abeceda. δ je prechodová funkcia, $\delta : K \times \Gamma \to 2^{K \times \Gamma \times \{-\infty, 0, 1\}}$.

Konfigurácia skáčúceho LBA je trojica (q, w, n), kde $q \in K$ je stav, $w \in \Gamma^*$ je aktuálna pracovná páska a $n \in \{0, 1, \ldots, |w|, |w| + 1\}$ je pozícia hlavy.

Krok výpočtu skáčúceho LBA je relácia $\vdash$ na konfiguráciách taká, že platí:[1]

$(q, w, n) \vdash (p, w_1 \ldots w_{n-1} x w_{n+1} \ldots w_{|w|}, d(n))$ pre $(p, x, d) \in \delta(q, w_n)$, $n \in \{1, \ldots, |w|\}$

$(q, w, n) \vdash (p, w, d(n))$ pre $(p, x, d) \in \delta(q, w_n)$, $n \in \{0, |w| + 1\}$

Jazyk akceptovaný skáčúcim LBA je množina
$L(A) = \{w \mid \exists q_F \in F, v \in \Gamma^{|w|}, n \in N_0;\ (q_0, w, 1) \vdash^* (q_F, v, n)\}$.

Aby sme ukázali, že je ekvivalentné so štandardným LBA, ukážeme, že sa vedia navzájom simulovať. Majme najskôr skáčúce LBA, zostrojíme k nemu ekvivalentné štandardné. Kroky, kde stojíme na mieste alebo ideme doprava sa nezmenia. Krok, pri ktorom skočíme na začiatok pásky, odsimuluje LBA tak, že si v stave poznačí, že ide na ľavý kraj pásky a kým tam nie je, len ide doľava a nič iné nerobí.

Pri simulácií opačným smerom to bude trochu ťažšie. Opäť kroky, kde stojíme alebo ideme doprava, simulovať vieme. Ako ale odsimulovať krok doľava? Takto: Skáčúce LBA si na páske spraví čiernu bodku, že tu stojí. Skočí na cent. Teraz ide doprava, až kým sa nedeterministicky nerozhodne, že je práve o políčko naľavo od čiernej bodky. Ešte ale potrebuje overiť, či sa rozhodlo správne. Preto si tu spraví bielu bodku a pohne sa o políčko doprava. Ak našlo čiernu bodku, uhádlo a môže pokračovať – čiernu bodku zmaže, skočí na začiatok a ide doprava, kým nenarazí na bielu bodku. Tú tiež zmaže a môže ísť simulovať ďalší krok.

9.6 Rekurzívne vyčísliteľné a rekurzívne jazyky

Úloha 9.6.1 *Daný je PDA A akceptujúci prázdnou pamäťou. Zostrojte nedet. Turingov stroj M taký, že $L(M) = N(A)$. Zdôvodnite správnosť svojej konštrukcie.*

Použijeme klasický NTS s oboma smermi nekonečnou páskou. Myšlienka konštrukcie: Naľavo od vstupného slova si udržiavame (reverznutý) zásobník. Na odsimulovanie jedného kroku A si M pozbiera do stavu písmeno čítané zo vstupu (resp. ε), písmeno na vrchu zásobníka, nedeterministicky sa rozhodne, ktorý riadok δ_A použije, príslušne zmení stav a vloží správne slovo na zásobník. Stavy M budú tvaru q_{index}, kde q je stav, v ktorom je simulovaný PDA A a $index$ nám hovorí, čo práve robíme.

Formálna konštrukcia: Nech $A = (K_A, \Sigma_A, \Gamma_A, \delta_A, qsA, Z_{0A}, \emptyset)$. Označme k dĺžku najdlhšieho slova, ktoré A vie v jednom kroku zapísať na zásobník. Nech $L^{\leq x} = L^0 \cap L^1 \cap \ldots \cap L^k$, nech $\overline{L} = \{\overline{x} \mid x \in L\}$. BUNV nech Γ_A, Σ_A a $\overline{\Sigma_A}$ sú po dvoch disjunktné.

Potom $M = (K, \Sigma, \Gamma, \delta, q_0, F)$, kde K je konečná množina stavov, pre ktoré je definovaná δ-fcia uvedená nižšie, $\Sigma = \Sigma_A$, $\Gamma = \Sigma_A \cup \overline{\Sigma_A} \cup \Gamma_A \cup \{\#, \$\}$ (kde $\#, \$$ sú nové znaky).

Na začiatku pásku prepíšeme tak, aby na nej bolo namiesto slova w slovo $\#Z_{0A}w\$$, M bol v stave qsA_{citaj} a hlava bola na prvom znaku slova w. Táto časť δ-fcie je triviálna, preto ju nebudeme uvádzať. „Simulačný cyklus“ vyzerá nasledovne:

$$\delta(q_{citaj}, x) \quad = \quad \{(q_{mam\_x}, \overline{x}, -1), (q_{mam\_\varepsilon}, x, -1)\} \quad (\forall q \in K_A, \forall x \in \Sigma) \tag{9.1}$$

[1] Pre jednoduchší zápis nech w_i je i-ty znak slova w, špeciálne $w_0 = ¢$, $w_{|w|+1} = \$$. Ďalej keď $d = -\infty$ definujeme $d(n) = 0$, inak $d(n) = \min(n + d, |w| + 1)$ (teda $d(n)$ hovorí, kam sa pohneme, ak sme na n a δ-fcia vráti pohyb d).

$$
\begin{aligned}
\delta(q_{citaj},\$) &= \{(q_{mam\_\varepsilon},\$,-1)\} \quad (\forall q \in K_A) && (9.2)\\
\delta(q_{mam\_c},x) &= \{(q_{mam\_c},x,-1)\} \quad (\forall q \in K_A, \forall x \in \Gamma \setminus \{\#\}, \forall c \in \Sigma \cup \{\varepsilon\}) && (9.3)\\
\delta(q_{mam\_c},\#) &= \{(q_{krok\_c},\#,1)\} \quad (\forall q \in K_A, \forall c \in \Sigma \cup \{\varepsilon\}) && (9.4)\\
\delta(q_{krok\_c},Z) &= \{(p_{pis\_w},Z,0) \mid (p,w) \in \delta_A(q,c,Z)\} \quad (\forall q \in K_A, \forall c \in \Sigma \cup \{\varepsilon\}, \forall Z \in \Gamma) && (9.5)\\
\delta(q_{pis\_aw},Z) &= \{(q_{pis\_w},a,-1)\} \quad (\forall q \in K_A, \forall Z \in \Gamma, \forall a \in \Gamma_A, \forall w \in \Gamma_A^{\leq k}) && (9.6)\\
\delta(q_{pis\_\varepsilon},Z) &= \{(q_{naspat},\#,1)\} \quad (\forall q \in K_A, \forall Z \in \Gamma) && (9.7)\\
\delta(q_{naspat},x) &= \{(q_{naspat},x,1)\} \quad (\forall q \in K_A, \forall x \in \Gamma_A \cup \overline{\Sigma_A}) && (9.8)\\
\delta(q_{naspat},x) &= \{(q_{citaj},x,0)\} \quad (\forall q \in K_A, \forall x \in \Sigma \cup \{\$\}) && (9.9)
\end{aligned}
$$

Zjavne až na $(1),(5)$ je celý „simulačný cyklus“ deterministický. V (1) sa rozhodneme, či simulovaný automat A číta písmeno alebo ε (ak už sme na konci vstupu, tak sa použije (2) a čítame ε). V $(3),(4)$ prejdeme na koniec zásobníka. V (5) sa podľa symbolu zo vstupu (ktorý si pamätáme v stave) a z vrchu zásobníka (ktorý čítame z pásky) rozhodneme, do akého stavu p by A prešiel a aké slovo w by zapísal na zásobník. V (6) toto slovo na zásobník zapíšeme, v (7) pripíšeme na koniec zásobníka zarážku, v $(8),(9)$ sa vrátime na aktuálnu pozíciu vo vstupnom slove – a sme pripravení simulovať ďalší krok A.

K správnosti: Zjavne každý krok A dokážeme odsimulovať. Z toho, že simulačný cyklus je takmer deterministický (až na miesto, kde sa rozhodujeme práve medzi možnosťami z δ_A), nevieme robiť nič iné.

Ešte ostáva dať nášmu automatu možnosť akceptovať vstupné slovo – vtedy, ak má prázdny zásobník a dočítal vstupné slovo. Pridáme mu možnosť skončiť v okamihu, keď zistí, že má prázdny zásobník (t.j. keď ide čítať písmeno zo zásobníka, nájde tam prvé písmeno vstupu) a práve nič nečítal zo vstupu:

$$
\begin{aligned}
\delta(q_{krok\_\varepsilon},x) &= \{(over,x,0)\} \quad (\forall q \in K_A, \forall x \in \Gamma \setminus \Gamma_A)\\
\delta(over,x) &= \{(over,x,1)\} \quad (\forall x \in \Gamma \setminus (\Sigma \cup \{\$\})\\
\delta(over,\$) &= \{(akceptuj,\$,0)\}
\end{aligned}
$$

Konštrukcia sa značne zjednoduší, ak použijeme dvojpáskový Turingov stroj.

Úloha 9.6.2 *Zostrojte (deterministický alebo nedeterministický) Turingov stroj, ktorý bude akceptovať jazyk:*

$$L = \{a^{x_1}\#a^{x_2}\#\ldots\#a^{x_n} \mid x_i \geq 0, \text{čísla } x_i \text{ sa dajú rozdeliť na dve časti s rovnakým súčtom}\}$$

Váš stroj teda dostane na vstupe n-ticu čísel $x_1,\ldots,x_n$ zapísaných v unárnej sústave a má rozhodnúť, či $\exists P \subseteq \{1,\ldots,n\} = N_n$; $\sum_{i\in P} x_i = \sum_{i\in N_n/P} x_i$. Súčasťou konštrukcie by mal byť aj slovný popis δ-funkcie. Zdôvodnite správnosť svojej konštrukcie. (Pozn.: L je jazyk nad abecedou $\Sigma = \{a,\#\}$.)

Začneme tým, že prejdeme pásku a nedeterministicky niektoré úseky a prepíšeme na b.
$\delta(q_0,x) = \{(q_a,x,0),(q_b,x,0)\}$ $\quad (x \in \{a,\#\})$
$\delta(q_x,a) = \{(q_x,x,1)\}$ $\quad (x \in \{a,b\})$ (v stave q_x prepisujeme a na x)
$\delta(q_x,\#) = \{(q_y,\#,1)\}$ $\quad (x,y \in \{a,b\})$ (na # sa môžeme rozhodnúť o ďalšom úseku)
$\delta(q_x,B) = \{(h_a,B',-1)\}$ $\quad (x \in \{a,b\})$ (sme na konci slova)

Teraz potrebujeme overiť, či máme rovnako veľa a a b. To ide najjednoduchšie tak, že na striedačku odškrtávame a a b. Budeme mať stavy h_a, h_b (hľadaj a, b). Keď sme v stave h_a, znamená to, že sme doteraz odškrtli rovnako a a b, vtedy sa môžeme rozhodnúť (a overiť), či je už všetko vyškrtané. Ak áno, akceptujeme.

$\delta(h_x,y) = \{(h_x,y,d)\}$ $\quad (x \in \{a,b\}, y \in \{a,b,\#\}, d \in \{-1,0,1\})$ (nedeterministicky beháme po páske)

$\delta(h_x, x) = \{(h_y, \#, 0)\} \quad (x, y \in \{a, b\}, x \neq y)$ (ak sme našli, prepíšeme na #)
$\delta(h_a, \#) = \{(q_{-1}, \#, -1)\})$ (ideme overiť, či už máme iba #)
(prebehneme po páske najskôr doľava, potom doprava)
$\delta(q_d, \#) = \{(q_d, \#, d)\}) \quad (d \in \{-1, 1\})$ (v q_{-1} ideme po # doľava, v q_1 doprava)
$\delta(q_{-1}, B) = \{(q_1, B', 1)\})$ (sme na začiatku, ešte musíme ísť doprava)
$\delta(q_1, B) = \{(q_F, B', 0)\})$ (prešli sme až na koniec, akceptujeme)

Úloha 9.6.3 *Nech L_1, L_2 sú rekurzívne vyčísliteľné jazyky, pričom $L_1 \cup L_2$ aj $L_1 \cap L_2$ sú rekurzívne. Rozhodnite a dokážte, či potom musia byť aj jazyky L_1, L_2 rekurzívne.*

Musia. Zostrojíme TS A pre L_1, ktorý na každom vstupe zastane. Nech A_1, A_2 sú TS pre L_1, L_2, Z, P sú TS pre $L_1 \cup L_2$, $L_1 \cap L_2$, ktoré zastanú na každom vstupe. Potom A na w bude pracovať nasledovne: Spustí P na w, ak P akceptuje, A akceptuje (w je z prieniku, teda aj z L_1). Spustí Z na w, ak Z neakceptuje, A neakceptuje (w nie je zo zjednotenia, teda ani z L_1). Ak sme sa dostali až sem, vieme, že w patrí práve do jedného z jazykov L_1, L_2. Spustíme naraz na w stroje A_1, A_2. Jeden z nich v konečnom čase zastane a akceptuje. V tom okamihu zastaví aj A a akceptuje iff akceptoval A_1.

Iné riešenie: Ukážeme, že $L_1^C \in \mathscr{L}_{RE}$, z toho vyplýva $L_1 \in \mathscr{L}_{rec}$. Pre L_2 dôkaz vyzerá analogicky. Platí $L_1^C = (L_1 \cup L_2)^C \cup (L_2 - L_1) = (L_1 \cup L_2)^C \cup (L_2 \cap (L_1 \cap L_2)^C)$. Vieme, že $L_2 \in \mathscr{L}_{RE}$. Navyše:
$L_1 \cup L_2 \in \mathscr{L}_{rec} \Rightarrow (L_1 \cup L_2)^C \in \mathscr{L}_{rec} \Rightarrow (L_1 \cup L_2)^C \in \mathscr{L}_{RE}$
$L_1 \cap L_2 \in \mathscr{L}_{rec} \Rightarrow (L_1 \cap L_2)^C \in \mathscr{L}_{rec} \Rightarrow (L_1 \cap L_2)^C \in \mathscr{L}_{RE}$
A keďže $\mathscr{L}_{RE}$ je uzavretá na prienik a zjednotenie, aj $L_1^C \in \mathscr{L}_{RE}$, q.e.d.

Úloha 9.6.4 *Daná je inštancia MPKP (prvé domino je začiatočné). Štandardnou konštrukciou k nej zostrojte ekvivalentnú inštanciu PKP.*

cab	*e*	*cb*	*ab*	*dd*
c	*a*	*bc*	*ab*	*cdd*

Ekvivalentnú sadu domín uvádzame nižšie. Zjavne každé riešenie PKP s novou sadou domín musí začínať prvým z nich, a teda zodpovedá nejakému riešeniu MPKP s pôvodnou sadou domín. A naopak, podľa každého riešenia pôvodnej inštancie MPKP vieme zostrojiť riešenie PKP s novou sadou domín.

#c#a#b#	c#a#b#	e#	c#b#	a#b#	d#d#	#
#c	#c	#a	#b#c	#a#b	#c#d#d	##

Úloha 9.6.5 *Ukážte, či je nasledujúci problém rozhodnuteľný: Pre daný NTS A, slovo w a tri stavy q_1, q_2, q_3 povedať, či existuje akceptačný výpočet A na w, prechádzajúci cez tieto stavy tesne po sebe v danom poradí. Formálne: keď zapisujeme konfigurácie A ako trojice (stav,slovo na páske,pozícia hlavy), pýtame sa, či existujú $v_i \in \Gamma^*$, $n_i \in N$ také, že $(q_0, w, 1) \vdash^* (q_1, v_1, n_1) \vdash (q_2, v_2, n_2) \vdash (q_3, v_3, n_3) \vdash^* (q_F, v_4, n_4)$, pričom q_F je akceptačný stav.*

Nie je. Ukážeme, že ak by bol rozhodnuteľný, vedeli by sme rozhodovať aj príslušnosť slova do jazyka akceptovaného TS. To by znamenalo, že $L_u \in \mathcal{L}_{rec}$ a máme spor.

Nech M je DTS (ktorý vždy zastaví) rozhodujúci problém zo zadania. Zostrojíme DTS U ktorý vždy zastaví a akceptuje L_u nasledovne: U dostane na vstupe dvojicu (A, w). Upraví δ-fciu A tak, že pridá tri nové stavy q_X, q_Y, q_Z, ktorými A prejde na začiatku výpočtu bez toho, aby sa hýbal a prepisoval pásku. Takto U vyrobil kód TS A', ktorý je zjavne ekvivalentný s A a navyše každý jeho akceptačný výpočet prechádza cez pridané stavy. Preto keď U chce rozhodnúť, či A akceptuje w, stačí mu spustiť M so vstupom (A', w, q_X, q_Y, q_Z).

Iná možnosť konštrukcie U bola taká, že U vôbec nezmení δ-fciu A, len postupne zavolá M pre všetky trojice stavov A. Takto zistí, či v A existuje akceptačný výpočet dĺžky aspoň 3, kratšie výpočty potom všetky vyskúša.

Úloha 9.6.6 *Rozhodnite, či existuje deterministický Turingov stroj, ktorý na každom vstupe zastaví a ak na vstupe dostane trojicu* $\Big[$*kód (ľubovoľného deterministického) Turingovho stroja <A>, kód vstupného slova w, kód znaku* $x \in \Gamma_A\Big]$*, tak akceptuje práve vtedy, ak TS A počas výpočtu na slove w niekedy zapíše na pásku znak x. Ak takýto stroj existuje, stručne popíšte jeho konštrukciu. Ak neexistuje, dokážte.*

Taký TS neexistuje, inými slovami tento problém nie je rozhodnuteľný.

Sporom. Nech taký TS M existuje. Zostrojme TS M', ktorý bude využívať M, vždy zastaví a bude akceptovať univerzálny jazyk $L_U = \{<A>\#w \mid w \in L(A)\}$. To bude spor so skutočnosťou, že $L_U \notin \mathscr{L}_{rec}$, ktorá bola dokázaná na prednáške.

(Dohodnime sa, že TS akceptuje, keď zastane v akc. stave. Ak by sme chceli, nech akceptuje akonáhle sa dostane do akc. stavu, nasledujúca konštrukcia by vyzerala trochu ináč.) Náš stroj M' má o stroji A rozhodnúť, či akceptuje slovo w. Preto upraví kód stroja A nasledovne:

- do pracovnej abecedy pridá nový symbol ♡
- do množiny stavov pridá nový stav $q_\heartsuit$, množinu akc. stavov zmení na $\{q_\heartsuit\}$
- do δ-fcie z každého bývalého akc. stavu na každé chýbajúce písmeno (t.j. na tie, na ktoré by bol zastal) pridá pravidlo, v ktorom stroj prejde do $q_\heartsuit$, zapíše na pásku ♡ a hlavou zostane stáť na mieste

Takto dostane kód nejakého stroja A'. Zjavne $L(A) = L(A')$. Takisto zjavne vie M' urobiť všetky tieto zmeny v kóde A v konečnom čase.

Všimnime si, že A' akceptuje w práve vtedy, keď sa počas výpočtu A' na w objaví na páske ♡. Ak teda M' chce povedať, či A akceptuje w, stačí mu vyššie popísaným spôsobom vyrobiť kód A' a opýtať sa M na trojicu $[<A'>, w, \heartsuit]$.

Úloha 9.6.7 *Je rozhodnuteľné, či pre dané homomorfizmy* $h_1, h_2 : \Sigma_1^* \to \Sigma_2^*$ *existuje slovo* $w \neq \varepsilon$ *také, že* $h_1(w) = h_2(w)$*?*

Ukážeme, že tento problém nie je rozhodnuteľný, lebo keby bol rozhodnuteľný, potom by aj PKP bol rozhodnuteľný.

Nech je daná inštancia PKP (X, Y), kde $|X| = |Y| = n$, nad abecedou Σ. Nech $\Sigma_1 = \{1, \ldots, n\}$, $\Sigma_2 = \Sigma$ a nech $\forall k \in \Sigma_1;\ h_1(k) = x_k$ a $h_2(k) = y_k$. Zjavne každé slovo $w \in \Sigma_1^*$ predstavuje postupnosť indexov domín, $h_1(w)$ je slovo, ktoré pri tejto postupnosti dostaneme na hornom poschodí a $h_2(w)$ slovo na dolnom poschodí. Slová, na ktorých dajú oba tieto homomorfizmy rovnaký obraz teda zjavne zodpovedajú práve všetkým riešeniam príslušnej inštancie PKP. Teda keby sme pre ľubovoľné dva homomorfizmy vedeli rozhodnúť, či existuje neprázdne slovo, ktoré má v oboch rovnaký obraz, vedeli by sme rozhodovať aj PKP. To ale nevieme.

Úloha 9.6.8 *Majme triedu Turingovych strojov, ktoré nemôžu zapisovať na pásku blank. Je pre takéto stroje rozhodnuteľný problém: Povedať, či pre daný kód TS <A>, vstupné slovo w a konfiguráciu K existuje výpočet A na w, ktorý prechádza cez konfiguráciu K?*

Uvedomme si, že takýto Turingov stroj nikdy nedokáže skrátiť slovo zapísané na páske. Preto vo výpočte pred konfiguráciou K nebol v žiadnej konfigurácií, v ktorej je slovo na páske dlhšie ako v K. Takýchto konfigurácii je ale len konečne veľa. Stačí teda skúmať všetky možné výpočty A a prestať v okamihu, keď je dĺžka popísanej časti pásky pridlhá alebo sa niektorá konfigurácia zopakuje.

Odpoveď na otázku by sa prudko zmenila, keby sme sa pýtali, či existuje **akceptačný** výpočet A na w, prechádzajúci cez K. Zjavne túto úlohu ľahko zredukujeme na L_U (stačí zobrať $K = q_0 w$).

9.7 Úvod do teórie zložitosti

Úloha 9.7.1 *Rozhodnite, či je trieda* $NSPACE(n^3)$ *uzavretá na ľubovoľný homomorfizmus. Svoje tvrdenie dokážte. (Hint:* $\mathscr{L}_{ECS} \subseteq NSPACE(n^3) \subseteq \mathscr{L}_{rec} \subsetneq \mathscr{L}_{RE}$*).*

Tvrdenie neplatí. Nech L je ľubovoľný rekurzívne vyčísliteľný, ale nerekurzívny jazyk. (Napr. univerzálny jazyk L_U.) Nech A je DTS s jednosmerne nekonečnou páskou, ktorý ho akceptuje. Tento jazyk zjavne nepatrí do $NSPACE(n^3)$, lebo nie je ani len rekurzívny. Ukážeme, že je vhodným homomorfným obrazom vhodného jazyka z $NSPACE(n^3)$.

BUNV nech L je nad abecedou $\{0,1\}$. Zoberme homomorfizmus h taký, že $h(0) = 0$, $h(1) = 1$, $h(B) = \varepsilon$. Zostrojíme TS A', ktorý bude pracovať v priestore n (a teda aj n^3) a akceptovať jazyk L' taký, že $h(L') = L$.

TS A' bude pracovať nasledovne: Na začiatku skontroluje, či má na páske slovo z $\{0,1\}^*\{B\}^*$, ak nie, neakceptuje. Inak začne na tomto vstupe simulovať A, pričom znaky B berie ako blanky pre A. Ak by počas simulácie potreboval písať za koniec vstupného slova, zasekne sa.

Zjavne ak A' akceptuje nejaké slovo wB^k ($w \in \{0,1\}^*$), tak A akceptoval w — A' sa podarilo odsimulovať A. On the other hand, nech $w \in L(A)$. Potom A pri výpočte na w použil len konečne veľa políčok pásky, say x. Ale potom A' zjavne akceptuje slovo wB^x.

Zjavne $L(A') \in DSPACE(n)$, a teda $L(A') \in NSPACE(n^3)$. Ale keď zoberieme vyššie uvedený homomorfizmus h, tak $h(L(A')) = L \notin NSPACE(n^3)$.

Našli sme teda jazyk z $NSPACE(n^3)$ a homomorfizmus také, že homomorfný obraz nášho jazyka nepatrí do $NSPACE(n^3)$. Preto $NSPACE(n^3)$ nie je uzavretá na ľubovoľný homomorfizmus, q.e.d.