

# Teória paralelných výpočtov

verzia 0.03 (nekompletná)

26.6.2000

# Obsah

<b>1</b>	<b>Paralelné gramatiky</b>	<b>3</b>
1.1	Lindenmayerove systémy ( <i>L</i> -systémy)	3
1.1.1	<i>OL</i> -systémy	3
1.1.2	Porovnanie $\mathcal{L}_{OL}$ s Chomského hierarchiou	7
1.1.3	Rozšírené <i>OL</i> -systémy ( <i>EOL</i> -systémy)	10
1.1.4	<i>TOL</i> -systémy (table)	12
1.2	Ďalšie paralelné gramatiky	12
1.2.1	Indické paralelné gramatiky	12
1.2.2	Ruské paralelné gramatiky	13
1.2.3	Absolútne paralelné gramatiky	13
<b>2</b>	<b>Generatívne systémy</b>	<b>14</b>
2.1	Definície a označenia	14
2.2	Porovnanie generatívnej sily <i>g</i> -systémov s gramatikami Chomského hierarchie	15
2.3	Modelovanie známych gramatík pomocou <i>g</i> -systémov	17
2.4	Miery zložitosti	18
2.5	Porovnanie sekvenčných a paralelných <i>g</i> -systémov	19
2.6	Normálové tvary <i>g</i> -systémov	23
2.7	Charakterizácia triedy $TIME_G(f(n))$ pomocou sekvenčného priestoru	24
2.8	Niektoré vlastnosti “rýchlo generovateľných” jazykov	29
2.9	Záverom o <i>g</i> -systémoch	29
<b>3</b>	<b>Kooperujúce distribuované systémy gramatík (<i>CDGS</i>)</b>	<b>30</b>
3.1	Definície a označenia	30
3.2	Porovnanie akcečných štýlov	31
3.3	Niektoré otázky popisnej zložitosti	34
<b>4</b>	<b>Paralelné komunikujúce systémy gramatík (<i>PCGS</i>)</b>	<b>35</b>
4.1	Definície a označenia	35
4.2	Parametre uvažované na <i>PCGS</i>	36
4.3	Generatívna sila <i>PCGS</i>	37
4.4	Porovnanie <i>PCGS</i> so sekvenčnými triedami	42
4.5	Niektoré ďalšie vlastnosti <i>PCGS</i>	45
<b>5</b>	<b>Alternujúce stroje</b>	<b>46</b>
5.1	Definície a označenia	46
5.2	Miery zložitosti	48
5.3	Alternujúce vs. sekvenčné triedy zložitosti	49

5.4	Alternujúce konečné automaty ( <i>AFSA</i> ) . . . . .	56
5.5	Alternujúce zásobníkové automaty ( <i>APDA</i> ) . . . . .	58
5.6	Synchronizované alternujúce stroje . . . . .	59
<b>6</b>	<b>Booleovské obvody (<i>BO</i>)</b>	<b>61</b>
6.1	Definície a označenia . . . . .	61
6.2	Miery zložitosti . . . . .	62
6.3	<i>BC</i> -uniformné booleovské obvody . . . . .	62
6.4	Porovnanie <i>BO</i> a <i>TS</i> . . . . .	63
6.5	Iné uniformity pre booleovské obvody . . . . .	66
6.6	Porovnanie modelov <i>BO</i> a <i>ATS</i> . . . . .	68
<b>7</b>	<b>Parallel Random Access Machine (<i>PRAM</i>)</b>	<b>71</b>
7.1	Definície a označenia . . . . .	71
	7.1.1 <i>RAM</i> . . . . .	71
	7.1.2 <i>PRAM</i> . . . . .	72
7.2	Miery zložitosti . . . . .	74
7.3	Generatívna sila modelu <i>PRAM</i> . . . . .	75
7.4	Porovnanie modelov <i>BO</i> a <i>PRAM</i> . . . . .	77

# Kapitola 1

## Paralelné gramatiky

Skúsme sa trochu zamyslieť nad tým, aký druh paralelizmu by sme v gramatikách mohli zaviesť. Jednou z možností by bolo, na rozdiel od gramatík Chomského hierarchie, kde sa v jednom kroku odvodenia prepisuje iba jeden neterminál, prepísať naraz všetky neterminály vo vetnej forme. Inou by mohlo byť prepisovanie rovnakých neterminálov na jeden krok vybraným pravidlom a ako uvidíme, existuje množstvo ďalších modifikácií.

### 1.1 Lindenmayerove systémy (*L*-systémy)

(: Pôvodná motivácia, ktorá dala vzniknúť teoretickému modelu, o ktorom si teraz niečo bližšie povieme, nebola ani zďaleka taká blízka teórii jazykov, ako by si niekto mohol chybné myslieť. Pán Lindenmayer, podľa ktorého je tento model pomenovaný, skúmal správanie sa istého druhu fylamentózných organizmov, ktoré vyzerali asi takto: tvorila ich reťaz buniek, pričom každá bunka (okrem krajných, ktoré majú po jednom) mala práve dvoch susedov, ktorí ju mohli, resp. nemuseli v jej správaní ovplyvňovať. Celý mechanizmus fungoval akoby v taktach tak, že sa niekedy pár buniek rozhodlo, že sa rozmnoží (presnejšie, že sa každá z nich rozmnoží), niekedy zasa niektoré bunky odumreli, a teda z reťazca akoby zmizli a inokedy sa s nimi nič nedialo :)

#### 1.1.1 *OL*-systémy

**Definícia 1.1.1.** *OL*-systém je trojica  $G = (V, P, w)$ , kde  $V$  je abeceda symbolov,  $P \subseteq V \times V^*$  je množina prepisovacích pravidiel taká, že  $\text{proj}_1(P) = V$  (teda musí existovať pravidlo pre každý symbol abecedy *L*-systému),  $w \in V^+$  nazývame axiom, alebo počiatočné slovo<sup>1</sup>

**Definícia 1.1.2.** Krok odvodenia je relácia  $\Rightarrow$  na  $V^*$  definovaná nasledovne:  $u \Rightarrow v$  práve vtedy, ak  $u = a_1 \dots a_n$ ,  $\forall i \ a_i \in V$ ,  $v = b_1 \dots b_n$ ,  $\forall i \ b_i \in V^*$  a  $\forall i \ a_i \rightarrow b_i \in P$ , teda v jednom kroku odvodenia sa prepisuje každý symbol vetnej formy

**Definícia 1.1.3.** Jazyk<sup>2</sup> generovaný *OL*-systémom  $G$  je  $L(G) = \{u \in V^* \mid w \xRightarrow{*} u\}$

<sup>1</sup>Je dobré si hneď na začiatku uvedomiť rozdiel medzi *L*-systémami a gramatikami, s ktorými sme sa stretli v Chomského hierarchii, počiatočné slovo *L*-systému nemusí tvoriť iba jeden symbol, ba dokonca tu nerozlišujeme medzi terminálmi a neterminálmi

<sup>2</sup>teda prakticky každá vetná forma, ktorú z počiatočného slova dostaneme (vrátane jeho samého), patrí do jazyka  $L(G)$ , ak  $G$  je *L*-systém

**Definícia 1.1.4.** *DOL-systém je taký OL-systém, kde  $\forall a \in V$  existuje práve jedno pravidlo  $a \rightarrow u \in P$  také, že  $u \in V^*$  (takémuto OL-systému hovoríme aj deterministický)*

**Definícia 1.1.5.** *POL-systém (propagating-pokračujúci) je taký OL-systém, ktorý je bez- $\varepsilon$*

Nasledujúce príklady poskytnú čitateľovi náhľad na generatívnu silu OL-systémov.

**Príklad 1.1.1.**  $G_1 = (\{a\}, \{a \rightarrow a^2\}, a)$  potom  $L(G_1) = \{a^{2^n} \mid n \geq 0\}$

Ako vidieť, OL-systém  $G_1$  nám úplne jednoducho (použitím jediného pravidla) umožňuje generovať jazyk, na ktorý by nám v Chomského hierarchii nestačili ani bezkontextové prostriedky. Sila OL-systémov spočíva v paralelnom prepisovaní, kedy v jednom kroku odvodenia musíme použiť prepisovacie pravidlá na všetky symboly. Skúsme ale do  $G_1$  pridať jedno pravidlo:

$G'_1 = (\{a\}, \{a \rightarrow a, a \rightarrow a^2\}, a)$  potom  $L_{G'_1} = a^+$  a veľká sila je razom preč. Je tomu tak preto, lebo pridaním pravidla  $a \rightarrow a$  sme umožnili akoby rozsynchronizovať odvodenie, a teda symboly sa teraz neprepisujú naraz, ale každý v inom čase.

**Príklad 1.1.2.**  $G_2 = (\{a, b\}, \{a \rightarrow b, b \rightarrow ab\}, a)$  potom jazyk  $L(G_2)$  budú tvoriť slová, ktorých dĺžky sú členy Fibonacciho postupnosti, teda opäť máme jeden dosť zložitý, zrejme nie bezkontextový jazyk, ktorý dokážeme OL-systémom pomerne jednoducho generovať.

**Príklad 1.1.3.**  $G_3 = (\{a, b, c\}, \{a \rightarrow abcc, b \rightarrow bcc, c \rightarrow c\}, a)$  potom dĺžky slov jazyka  $L(G_3)$  tvoria postupnosť štvorcov (druhých mocnín) prirodzených čísel.

**Definícia 1.1.6.** *AF $\mathcal{L}$  (Abstract Family of Languages) je každá trieda jazykov obsahujúca nejaký neprázdny jazyk a uzavretá na  $\cup, \cdot, +, h_\varepsilon, h^{-1}, \cap \mathcal{R}$*

**Veta 1.1.1.**  *$\mathcal{L}_{OL}$  je anti-AF $\mathcal{L}$  (t.j. nie je uzavretá na žiadnu AF $\mathcal{L}$  operáciu)*

**Dôkaz:** Pre každú AF $\mathcal{L}$  operáciu ukážeme, že trieda  $\mathcal{L}_{OL}$  na ňu nie je uzavretá

- $\cup : \{a\}, \{a^2\} \in \mathcal{L}_{OL}$  a sporom predpokladajme, že  $L = \{a, a^2\} \in \mathcal{L}_{OL}$ . Potom existuje nejaký OL-systém  $G$ , ktorý jazyk  $\{a, a^2\}$  generuje. Ten ale musí mať nejaký axiom, môžu nastať dve možnosti:

1. axiom je  $a$  - potom ale v  $G$  existuje pravidlo  $a \rightarrow a^2$ , lebo keby neexistovalo, tak by sme slovo  $a^2$  nikdy nevyrobili, resp. by sme vyrobili iné slová, ktoré do  $L$  nepatria. Keďže máme toto pravidlo, tak môžeme vyrobiť aj slová, ktoré do  $L$  nepatria (napr.  $a^4$ ), čo je spor s tým, že  $G$  generuje  $L$
2. axiom je  $a^2$  - potom ak z neho chceme vyrobiť  $a$ , tak musíme v  $G$  mať pravidlo  $a \rightarrow \varepsilon$ , ale aj  $a \rightarrow a$ , lebo keby sme ho nemali, tak vďaka paralelnému prepisovaniu symbolov by sme dostali  $\varepsilon \in L$ , ale my vieme, že  $\varepsilon \notin L$ . Ale ak tu už máme tieto dve pravidlá, tak  $\varepsilon$  chciac-nechtiac niekedy vyrobíme, čo je opäť v spore s tým, že  $\varepsilon \notin L$

Teda dostávame  $L \notin \mathcal{L}_{OL}$  <sup>3</sup>

- $\cdot$  : Zvolíme  $L_1 = \{a\} \in \mathcal{L}_{OL}, L_2 = \{\varepsilon, a\} \in \mathcal{L}_{OL}$  ale  $L_1 \cdot L_2 = \{a, a^2\} \notin \mathcal{L}_{OL}$  ako sme v predchádzajúcej časti ukázali
- $+$  : Definujme  $L = \{aa\} \cup \{b^{2^n} \mid n \geq 2\} \in \mathcal{L}_{OL}$ , dôkaz urobíme sporom, nech  $G$  je OL-systém pre  $L^+$ :

<sup>3</sup>Dostávame sa teda k možno trochu prekvapujúcemu výsledku. Ukazuje sa, že i keď  $L$ -systémy v predchádzajúcom texte zvládli taký krkolomný jazyk ako bol  $L(G_1)$ , neporadia si s evidentne regulárnym jazykom obsahujúcim iba dve slová

1. uvedomme si, že axiomom  $G$  môže byť jedine  $aa$ , ak by tomu tak nebolo, axiomom by muselo byť  $b^i$  pre nejaké  $i \geq 4$ , potom by sme ale nutne museli mať pravidlo  $b \rightarrow a$  alebo  $b \rightarrow a^2$ , ale potom by sme vyrobili aj slovo tvaru  $b^i a^j$ ,  $i, j \neq 0$ , ktoré do jazyka  $L^+$  nepatrí
  2.  $\varepsilon \notin L^+ \rightsquigarrow G$  je  $POL$ -systém
  3.  $a^4 \in L^+ \rightsquigarrow a \rightarrow a^2 \in P$  (nesmú tu byť pravidlá tvaru  $a \rightarrow a$ ,  $a \rightarrow a^3$ , pretože by sme mohli vyrobiť aj slovo  $ab^i \notin L^+$ )
  4.  $a^2 \stackrel{*}{\Rightarrow} b^4 \rightsquigarrow a \rightarrow b^i \in P$  pre  $1 \leq i \leq 3$ . Ak teraz použijeme  $a \rightarrow a^2$  a  $a \rightarrow b^i$  dostaneme  $a^2 \Rightarrow a^2 b^i \notin L^+$
- $h_\varepsilon$  : Uvažujme jazyk  $L = \{\varepsilon, a, a^2\} \in \mathcal{L}_{OL}$  a definujme homomorfizmus  $h$  nasledovne:  $h(a) = a^2$ , potom  $h(L) = \{\varepsilon, a^2, a^4\} \notin \mathcal{L}_{OL}$ , čo sa dokáže rozbitím na prípady podobne ako pre  $\cup$
  - $h^{-1}$  : Uvažujme jazyk  $L = \{a\} \in \mathcal{L}_{OL}$  a homomorfizmus  $h$  daný predpisom  $h(a) = a^2$ . Potom  $h^{-1}(L) = \emptyset \notin \mathcal{L}_{OL}$
  - $\cap \mathcal{R}$  : Uvažujme jazyk  $L_1 = \{\varepsilon, a, a^2\} \in \mathcal{L}_{OL}$  a regulárny jazyk  $L_2 = \{a^3\}^*$ , dostávame rovnosť  $L_1 \cap L_2 = \{\varepsilon\} \notin \mathcal{L}_{OL}$ <sup>4</sup>

□

**Dôsledok 1.1.1.**  $\mathcal{L}_{OL}$  nie je uzavretá na substitúciu ani na zobrazenie  $a$ -prekladačom a nie je uzavretá ani na  $\cap$  a  $^C$  (komplement)

**Dôkaz:** Čo sa týka uzavretosti tejto triedy na zobrazenie  $a$ -prekladačom, princíp dôkazu je podobný ako pri predchádzajúcich uzáverových vlastnostiach. Keďže  $\mathcal{L}_{OL}$  nie je uzavretá na  $\cap \mathcal{R}$ , tak nemôže byť uzavretá ani na  $\cap$  všeobecne. Uzavretosť na  $^C$  nechávame na čitateľa □

**Veta 1.1.2.**  $\mathcal{L}_{OL}$  je uzavretá na zrkadlový obraz

**Dôkaz:** Idea je rovnaká ako pre bezkontextové gramatiky. Je daný jazyk  $L$ . Nech  $G = (V, P, w)$  je  $OL$ -systém taký, že  $L(G) = L$ . Zostrojíme  $OL$ -systém  $G' = (V', P', w')$  tak, že  $V = V'$ , ak  $a \rightarrow b_1 \dots b_n \in P$ , potom  $a \rightarrow b_n \dots b_1 \in P'$  a ak  $w = w_1 \dots w_m, \forall i w_i \in V$ , tak  $w' = w_m \dots w_1$ . Je zrejmé, že  $L(G') = L^R$  □

Veta 1.1.1 nám poskytla šokujúci pohľad na triedu  $\mathcal{L}_{OL}$  ako na anti- $\mathcal{AF}\mathcal{L}$ , čo v abstraktnej teórii jazykov znamená veľmi zlé vlastnosti. Trochu lepšia je situácia pri  $OL$ -jazykoch nad jednopísmenkovou abecedou Vyslovíme niekoľko tvrdení o týchto  $OL$ -jazykoch, ktoré nám v dostatočnej miere umožnia ich charakterizáciu.

**Veta 1.1.3.** Nech  $L \in \mathcal{L}_{OL}$  je jazyk  $L \subseteq a^*$ , potom  $L^* \in \mathcal{L}_{OL}$

**Dôkaz:** Nech  $L = L(G)$  kde  $G = (\{a\}, P, a^m), m \geq 1$

1.  $L$  je konečný:

(a)  $L = \{a\}$  alebo  $L = \{\varepsilon, a\} \rightsquigarrow L^* = a^* \in \mathcal{L}_{OL}$

(b) nech  $L = \{w_1, \dots, w_n\}$ , kde  $w_1 \neq \varepsilon, w_1 \neq a$  (jazyk musí obsahovať slovo dlhšie ako  $|a|$ ), potom  $L^* = L(G')$ , kde  $G' = (\{a\}, \{a \rightarrow w_1, \dots, a \rightarrow w_n, a \rightarrow \varepsilon\}, w_1)$

<sup>4</sup> $\{\varepsilon\} \notin \mathcal{L}_{OL}$ , pretože každý  $OL$ -systém, ktorý by tento jazyk generoval, by musel mať ako axiom  $\varepsilon$ , čo je z definície axiому nemožné

2.  $L$  je nekonečný:  $\forall i \ 1 \leq i \leq m$  nech  $v_i$  je najkratšie slovo v  $L$  také, že  $|v_i| \cong i \pmod m$  ak existuje, inak ho nedefinujeme. Nech  $G'$  je  $OL$ -systém tvaru  $G' = (\{a\}, P', a^m)$ , kde  $P' = \{a \rightarrow \varepsilon\} \cup \{a \rightarrow u \mid a^m \xrightarrow[G]{\Rightarrow} u \text{ alebo } \exists i \ v_i \xrightarrow[G]{\Rightarrow} u\}$ . Tvrdíme, že  $L(G') = L^*$

$\subseteq$ : Každé  $u$  v pravých stranách  $P'$  patrí do  $L$  a teda všetky slová generované  $OL$ -systémom  $G'$  patria do  $L^*$ , pretože začíname z  $a^m \in L$ , v ďalšom kroku prepíšeme každý symbol buď na  $\varepsilon$  (a teda sa ho zbavíme), alebo ho prepíšeme na slovo  $\in L$ , vetná forma má teda v každom kroku tvar  $w = w_1 \dots w_n$ ,  $\forall i \ i = 1 \dots n \ w_i \in L \rightsquigarrow w \in L^*$

$\supseteq$ : Opäť budeme kvôli lepšej zrozumiteľnosti rozlišovať dva prípady:

- A)  $L \subseteq L(G')$  ukážeme MI vzhľadom na počet krokov odvodu nasledovne:

1°  $a^m \in L$ , súčasne  $a^m \in L(G')$

2° ak  $x \in L$  a  $x \xrightarrow[G]{\Rightarrow} y$ , potom  $x \xrightarrow[G']{\Rightarrow} y$ , lebo  $x = (a^m)^j v_i$  pre nejaké  $i, j$

$$\underbrace{\underbrace{aa \dots aa}_m \underbrace{aa \dots aa}_m \dots \underbrace{aa \dots aa}_m \underbrace{aa \dots aa}_m \dots \underbrace{aa \dots aa}_m \underbrace{aa \dots aa}_i}_{v_i}^x$$

Pri prepisovaní  $x$  na  $y$  budeme postupovať nasledovne: prvé písmenko zo skupiny  $a^m$  prepíšeme na  $\varepsilon$ , na čo by sa to prepísalo celé v  $G$ , zvyšok prepíšeme na  $\varepsilon$ , to isté urobíme v časti  $v_i$ , inak povedané  $y = u_1 u_2 \dots u_j u_{j+1}$ , kde  $a^m \xrightarrow[G]{\Rightarrow} u_i, 1 \leq i \leq j, v_i \xrightarrow[G]{\Rightarrow} u_{j+1}$ , teda  $x \xrightarrow[G']{\Rightarrow} y$

- B) Nech  $w \in L^*$ : ak  $w = \varepsilon, w \in L$ , tak je to zrejmé z A). Nech  $w = w_1 \dots w_k, w_i \in L, \forall i \ w_i \neq \varepsilon, k \geq 2, \forall i \ a^m \xrightarrow[G']{*} w_i$

$a^m \xrightarrow[G']{*} (a^m)^k x$  pre nejaké  $x \in a^*$  teda  $a^m \xrightarrow[G']{*} a^{mk} \xrightarrow[G']{*} w_1 \dots w_k$  teda sme našli nejaké odvodenie slova  $w$ , no ešte musíme zabezpečiť, aby slová  $w_1, \dots, w_k$  vznikali synchronne na rovnaký počet krokov (musíme akosi nťahnuť odvodenie tých  $w_i$ , ktoré vznikajú skôr ako ostatné. Urobíme to nasledovne: ak  $a \xrightarrow[G']{\Rightarrow} y$ , tak odvodenie natiahneme<sup>5</sup>  $a^m \xrightarrow[G']{\Rightarrow} a^m a^t \xrightarrow[G']{\Rightarrow} y\varepsilon$ . Našli sme teda (existuje) odvodenie, kde odvodenia  $w_i$  majú rovnakú dĺžku  $\forall i$

□

**Poznámka 1.1.1.** Keď  $L \subseteq a^*$  je konečný, tak  $L^* \in \mathcal{L}_{OL}$

**Definícia 1.1.7.**  $M$ -množina je množina prirodzených čísel  $M(n, m_1, \dots, m_s)$ , ktorej tvar je  $M(n, m_1, \dots, m_s) = \{n\} \cup \{k_1 m_1 + \dots + k_s m_s \mid k_i \geq 0, 1 \leq i \leq s\}$

**Veta 1.1.4.** (Charakterizácia nekonečných  $OL$ -jazykov obsahujúcich  $\varepsilon$ )

Nech  $L \subseteq a^*$  je nekonečný a obsahuje  $\varepsilon$ . Potom  $L \in \mathcal{L}_{OL} \iff$  keď existuje  $M$ -množina  $M_L$  taká, že  $L = \{a^i \mid i \in M_L\}$

**Dôkaz:** Dokážeme obe implikácie:

“ $\Rightarrow$ ” Keďže  $L \in \mathcal{L}_{OL}$ , tak existuje nejaký  $OL$ -systém  $G$ , ktorý ho generuje, určite musí obsahovať (keďže  $\varepsilon \in L$ ) pravidlo  $a \rightarrow \varepsilon$ . Bez ujmy na všeobecnosti môžeme predpokladať, že pravidlá v  $G$  majú nasledovný tvar:  $P = \{a \rightarrow \varepsilon, a \rightarrow a^{m_1}, \dots, a \rightarrow a^{m_s}\}$ , pričom  $m_1 < \dots < m_s$  a  $G = (\{a\}, P, a^n)$ .  $M$ -množinu navrhne nasledovne  $M_L = \{n\} \cup \{k_1 m_1 + \dots + k_s m_s\}$ . Tvrdíme, že  $L(G) = L(M_L)$

<sup>5</sup>toto máme existenciou neepsilonových pravidiel zabezpečené

⊆: pre axiom  $a^n$  máme v  $M_L$  charakterizačný prvok  $n$ , vezmeme si teraz nejaké odvodenie v  $G$  a k nemu nájdeme príslušný prvok  $m \in M_L$ , ktorý ho charakterizuje. Odvodenie má tvar<sup>6</sup>  $a^n \xrightarrow{G} a^{m_{i_1}} \dots a^{m_{i_k}} (k \leq n) \xrightarrow{G} \dots \xrightarrow{G} a^{k_1 m_1} a^{k_2 m_2} \dots a^{k_s m_s}$ , pričom niektoré  $k_i$  (možno aj všetky, vďaka  $a \rightarrow \varepsilon$ ) sa môže rovnáť 0, stačí zvoliť  $m = k_1 m_1 + \dots + k_s m_s$ , pričom  $\forall i$  je  $k_i$  rovnaké ako v odvodení

⊇: zoberme prvok  $m \in M_L$  a k nemu hľadáme odvodenie slova  $w \in L(G)$  takého, že  $|w| = m$ . Ak  $m = n$ , tak  $w = a^n$  je axiom jazyka  $L(G)$  a sme hotoví, majme teda  $m = k_1 m_1 + \dots + k_s m_s \in M_L$ . Odvodenie nájdeme nasledovným spôsobom: najskôr si vyrobíme dostatočne veľa symbolov  $a$ , konkrétne toľko, aby ich bolo viac ako  $\sum_{i=1}^s k_i$  (to ide, pretože  $L$  je nekonečný jazyk a tak v  $G$  musí existovať pravidlo s počtom symbolov na pravej strane väčším ako 1), keď tieto symboly máme vyrobené, v jedinom ďalšom kroku vyrobíme slovo  $w$  tak, že prvých  $\sum_{i=1}^s k_i$  symbolov prepíšeme na  $a^{k_1 m_1} \dots a^{k_s m_s}$  (to ide jednoducho tak, že na prvých  $k_1$  symbolov aplikujeme pravidlo<sup>7</sup>  $a \rightarrow a^{m_1}$ , na ďalších  $k_2$  symbolov pravidlo  $a \rightarrow a^{m_2}$  a tak ďalej, až na posledných  $k_s$  symbolov aplikujeme pravidlo  $a \rightarrow a^{m_s}$ ), na zvyšné symboly aplikujeme pravidlo  $a \rightarrow \varepsilon$  a sme hotoví, pretože sme našli odvodenie  $w$  v  $G$

“ $\Leftarrow$ ” Je daná  $M$ -množina  $M_L(n, m_1, \dots, m_s)$  taká, že  $L(M_L) = \{a^i \mid i \in M_L\}$ , potom  $L(M_L) \in \mathcal{L}_{OL}$ . Našou úlohou je teda nájsť  $OL$ -systém  $G'$  taký, že  $L(G') = L(M_L)$ . Zvoľme  $G' = G$ . Dôkaz je potom úplne identický dôkazu prvej implikácie.

□

**Poznámka 1.1.2.**  $L_1 = \{a^{2^n} \mid n \geq 0\} \in \mathcal{L}_{OL}$ , ale  $L_1 \cup \{\varepsilon\} \notin \mathcal{L}_{OL}$ . Tu sa ukazuje, aká dôležitá je existencia, resp. neexistencia prázdneho slova v jazyku

Konečné jazyky nad jednopísmenkovou abecedou  $\{a\}$  v  $\mathcal{L}_{OL}$  sú tvaru:

1.  $\{a^m\}$ , kde  $m \geq 1$
2.  $\{\varepsilon, a^m\}$ , kde  $m \geq 1$
3.  $\{a^m, a^{m-1}, \dots, \varepsilon\}$ , kde  $m \geq 1$

### 1.1.2 Porovnanie $\mathcal{L}_{OL}$ s Chomského hierarchiou

Ako už predchádzajúce výsledky naznačili, vzhľadom k jazykom Chomského hierarchie, netvorí trieda  $\mathcal{L}_{OL}$  žiadnu “peknú” podmnožinu. Toto pozorovanie zhrnieme v nasledujúcom tvrdení.

**Veta 1.1.5.** Každá z tried  $\mathcal{R}, \mathcal{L}_{CF} - \mathcal{R}, \mathcal{L}_{CS} - \mathcal{L}_{CF}$  obsahuje aj jazyky, ktoré sú v  $\mathcal{L}_{OL}$ , aj jazyky, ktoré nie sú v  $\mathcal{L}_{OL}$

**Dôkaz:**  $OL$ -jazyky v jednotlivých triedach sú:

1.  $\{a\} \in \mathcal{R}$
2.  $\{a^i b a^i \mid i \geq 0\} \in \mathcal{L}_{CF} - \mathcal{R}$
3.  $\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}_{CS} - \mathcal{L}_{CF}$

<sup>6</sup>takto musí vyzeráť každé odvodenie, lebo v každom kroku sa  $a$  prepíše na nejaké  $a^{m_i}$  alebo na  $\varepsilon$

<sup>7</sup>nesmieme zabúdať na to, že pracujeme v  $OL$ -systéme, a teda v jednom kroku odvodenia musíme pravidlá aplikovať na všetky symboly vo vetnej forme



Jazyky v príslušných triedach, ktoré nie sú v  $\mathcal{L}_{OL}$ :

1.  $\{a, a^2\} \in \mathcal{R}$
2.  $\{a^i b^i \mid i \geq 0\} \in \mathcal{L}_{CF} - \mathcal{R}$
3.  $\{a^{2^n} \mid n \geq 0\} \cup \{a^3\} \in \mathcal{L}_{CS} - \mathcal{L}_{CF}$

□

**Veta 1.1.6.** Každý  $OL$ -systém (jazyk) obsahujúci  $\varepsilon$ , ktorý je v abecede  $\{a\}$  je regulárny

**Dôkaz:** Nech  $L \in \mathcal{L}_{OL}$ . Rozlíšime dva prípady:

1.  $L$  je konečný: vieme, že každý konečný jazyk je regulárny
2.  $L$  je nekonečný: podľa vety 1.1.4 vieme, že ak  $L \in \mathcal{L}_{OL}$ , tak existuje  $M$ -množina  $M_L$  taká, že  $L = \{a^i \mid i \in M_L\}$ . Nech  $M_L = \{n, m_1, \dots, m_k\}$ . Potom regulárna gramatika pre jazyk  $L$  bude:  $G = (\{\sigma, A\}, \{a\}, \{\sigma \rightarrow a^n, \sigma \rightarrow A, A \rightarrow a^{m_1} A, \dots, A \rightarrow a^{m_k} A, A \rightarrow \varepsilon\}, \sigma)$

□

**Poznámka 1.1.3.** Každý jazyk generovaný  $OL$ -systémom, v ktorom  $a \rightarrow a \in P$  pre každé  $a \in V$ , je bezkontextový<sup>8</sup>

**Poznámka 1.1.4.** Každý jazyk  $L \in \mathcal{L}_{CF}$  je tvaru  $L' \cap R$  pre  $L' \in \mathcal{L}_{OL}, R \in \mathcal{R}$

Nasledujúca lema pre  $OL$ -jazyky je akousi obdobou vety o lineárnom priestore platnej pre triedu jazykov  $\mathcal{L}_{ECS}$ .

**Lema 1.1.1.** Nech  $G = (V, P, w_0)$  je  $OL$ -systém, potom  $\forall w \in L(G)$  existuje odvodenie, ktoré používa maximálne  $k|w|$  priestoru, kde  $k$  je konštanta závislá od  $G$

**Dôkaz:** Definujme najskôr  $OL$ -systém  $G'$  nasledovne:  $G' = (V \cup \{x_0\}, P \cup \{x_0 \rightarrow w_0\}, x_0)$ ,  $x_0 \notin V$ , platí  $L(G') = L(G) \cup \{x_0\}$ . Teraz ku každému odvodeniu v  $G'$  uvažujme jeho strom<sup>9</sup>. Hovoríme, že odvodenie je redukované, keď jemu zodpovedajúci strom spĺňa nasledujúcu podmienku: neexistuje podstrom  $T$  taký, že súčasne platí:

1. všetky listy  $T$  sú  $\varepsilon$
2.  $T$  obsahuje vetvu, v ktorej majú dva vrcholy rovnaké návěstie

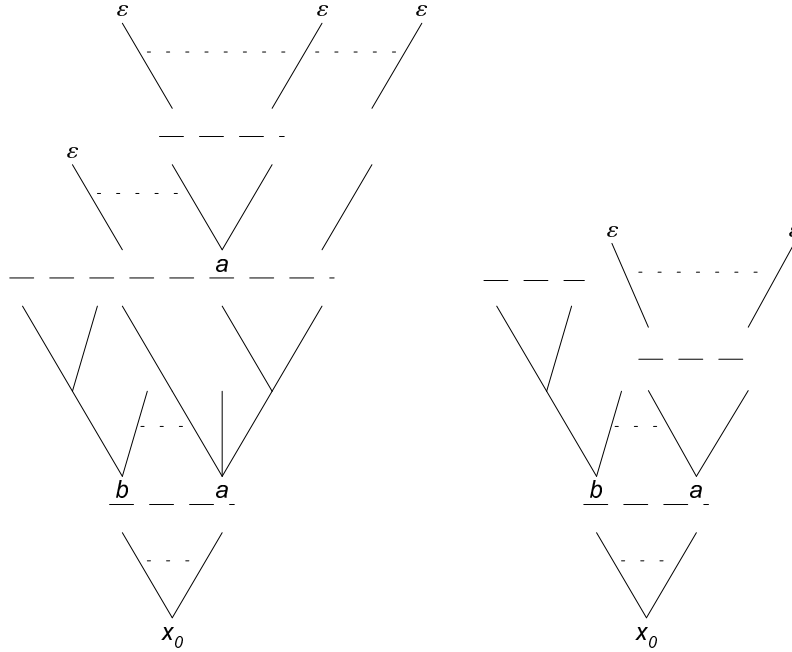
Pre každé odvodenie slova  $w$  existuje redukované odvodenie (obr. 1.1), toto získame jednoducho tak, že ztotožníme rovnaké uzly v jednej vetve v podstrome  $T$  (ktorý obsahuje ako listy iba  $\varepsilon$ ) originálneho stromu odvodenia.

Označme  $m$  dĺžku najdlhšej pravej strany spomedzi všetkých pravidiel  $G'$  a  $n = |V \cup \{x_0\}|$ . Tvrdíme, že stačí zvoliť:

$$k = 3m^n$$

<sup>8</sup>pravidlo  $a \rightarrow a, \forall a \in V$  nám umožní si pri každom kroku vybrať jeden symbol a nahradiť ho príslušnou pravou stranou pravidla a na ostatné symboly použiť pravidlo  $a \rightarrow a$

<sup>9</sup>ako pre bezkontextové gramatiky - koreň tvorí  $x_0$ , návěstia vrcholov sú symboly  $\in V \cup \{x_0\}$ , hrany sú orientované a množina orientovaných hrán vychádzajúca z vrchola  $v$  má tvar  $E(v) = \{v_1, \dots, v_k\} \iff v \rightarrow v_1 \dots v_k \in P \cup \{x_0 \rightarrow w_0\}$



Obrázok 1.1: Vytváranie redukovaného odvodenia

Pre  $\forall w \neq \varepsilon, w \in L(G')$  s odvodením  $x_0 \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n \equiv w$  bez újmy na všeobecnosti predpokladajme, že toto je redukované. Výskyt symbolu  $a$  v jednom zo slov  $w_i$  nazveme neproduktívny  $\iff$  ak tento výskyt je návěstím koreňa podstromu, ktorého všetky listy majú návěstie  $\varepsilon$ . Inak výskyt nazveme produktívny. Pokiaľ  $w \neq \varepsilon$ , tak  $w_i$  obsahuje najmenej jeden produktívny symbol, navyše počet produktívnych symbolov vo  $w_i \leq |w|$ . Stačí nám ukázať, že  $\forall i$ , keď  $Q$  je podslovo  $w_i$  a platí  $|Q| = 3m^n$ , potom  $Q$  obsahuje najmenej jeden produktívny symbol. Pre  $i < n$  žiadne podslovo  $w_i$  nie je dlhšie ako  $3m^n$ . Pre  $i = n + j, j \geq 0$  predpokladajme, že  $Q$  je podslovo  $w_i, |Q| = 3m^n$ .  $Q$  môžeme zapísať nasledovne:

$$Q = Q_1 Q_2 Q_3, |Q_1| = |Q_2| = |Q_3| = m^n$$

Predpokladajme, že  $Q$  obsahuje iba neproduktívne symboly. Uvažujme nejaký výskyt symbolu  $a$  v  $Q_2$ . Existuje jediný výskyt symbolu  $b$  vo  $w_{i-n}$  taký, že  $a$  je návěstie v strome  $T$ , ktorého koreň má návěstie  $b$ . Ďalej, voľbou  $Q, m, n$  všetky listy v  $T$  majú návěstia  $\varepsilon$ . To je spor, lebo potom existuje v  $T$  vetva s dvoma uzlami s rovnakým návěstím  $\square$

**Veta 1.1.7.** (*O lineárnom priestore*)

Nech  $A$  je Turingov stroj<sup>10</sup> (TS) taký, že existuje  $k$  také, že pre každé slovo  $w$  tento TS použije pri práci na  $w$  najviac  $k|w|$  políčok. Potom  $L(A) \in \mathcal{L}_{ECS}$

**Veta 1.1.8.**  $\mathcal{L}_{OL} \subseteq \mathcal{L}_{ECS}$

**Dôkaz:** Použijeme lemu 1.1.1 a na základe vety o lineárnom priestore, ktorá potom pre OL-jazyky platí, dostávame  $L(G) \in \mathcal{L}_{ECS}$   $\square$

<sup>10</sup>definície, popis modelu a iné v [1]

### 1.1.3 Rozšírené OL-systémy (EOL-systémy)

Ako sa pri OL-systémoch ukázalo, paralelné prepisovanie symbolov nám istú silu pridalo, no fakt, že do jazyka sme museli zahrnúť všetko, čo sme vyrobili (každú vetnú formu), nám veľkú časť nášho optimizmu odobral. EOL-systémy nám dovoľia opäť (ako pri gramatikách Chomského hierarchie) filtrovať vetné formy rozdelením množiny symbolov na terminály a neterminály. Do akej miery sa nám tým náš model zosilní (malo by byť jasné, že jeho sila bude minimálne taká, ako sila OL-systémov) si ukážeme na príklade neskôr.

**Definícia 1.1.8.** EOL-systém je štvorica  $G = (N, T, P, w)$ ,  $P \subseteq (N \cup T) \times (N \cup T)^*$  a  $\forall a \in (N \cup T)$  existuje v  $P$  pravidlo

**Definícia 1.1.9.** Krok ododenia je relácia  $\Rightarrow$  na  $(N \cup T)^*$  definovaná nasledovne:  $u \Rightarrow v$  práve vtedy, keď  $u = a_1 \dots a_n$ ,  $\forall i \ a_i \in (N \cup T)$ ,  $v = b_1 \dots b_n$ ,  $\forall i \ b_i \in (N \cup T)^*$  a  $\forall i \ a_i \rightarrow b_i \in P$

**Definícia 1.1.10.** Jazyk generovaný EOL-systémom  $G$  je  $L(G) = \{x \in T^* \mid w \xRightarrow{*} x\}$

**Príklad 1.1.4.**  $G_1 = (\{\sigma\}, \{a, b\}, \{\sigma \rightarrow a, \sigma \rightarrow b, a \rightarrow a^2, b \rightarrow b^2\}, \sigma)$ , potom zjavne jazyk  $L(G_1) = \{a^{2^n} \mid n \geq 0\} \cup \{b^{2^n} \mid n \geq 0\} \notin \mathcal{L}_{OL}$

Tento príklad ukazuje, že EOL-systémy sú silnejšie ako obyčajné OL-systémy a tým sme vlastne dali odpoveď na otázku, ktorú sme si položili na začiatku kapitoly, v tomto prípade sa sila neterminálu prejavila v tom, že nám umožnila spraviť zjednotenie dvoch jazykov.

**Príklad 1.1.5.**  $G_2 = (\{A\}, \{a, b\}, \{A \rightarrow A, A \rightarrow a, a \rightarrow a^2, b \rightarrow b\}, AbA)$  potom jazyk generovaný  $G$  je  $L(G_2) = \{a^{2^n} b a^{2^m} \mid m, n \geq 0\} \notin \mathcal{L}_{OL}$

**Príklad 1.1.6.**  $G_3 = (\{S, A, B, C, \bar{A}, \bar{B}, \bar{C}, F\}, \{a, b, c\}, P, S)$

$$P = \left\{ \begin{array}{llll} S \rightarrow ABC & a \rightarrow F & b \rightarrow F & c \rightarrow F \\ A \rightarrow A\bar{A} & A \rightarrow a & \bar{A} \rightarrow \bar{A} & \bar{A} \rightarrow a \\ B \rightarrow B\bar{B} & B \rightarrow b & \bar{B} \rightarrow \bar{B} & \bar{B} \rightarrow b \\ C \rightarrow C\bar{C} & C \rightarrow c & \bar{C} \rightarrow \bar{C} & \bar{C} \rightarrow c \\ F \rightarrow F \end{array} \right\}$$

Možno niekoho prekvapí fakt, že  $L(G_3) = \{a^n b^n c^n \mid n \geq 1\}$ , no ak sa nad systémom  $G_3$  trošku zamyslíme, tak sa nám táto skutočnosť ihneď vyjasní. Ide tu totiž o rafinované využitie neterminálu  $F$  na to, aby terminálne slová vznikali naraz (teda slová z jazyka  $L(G_3)$  vznikajú v jednom kroku ododenia prepísaním všetkých neterminálov na terminálne symboly). Ako si v nasledovnej vete ukážeme, tento jav nie je ani zďaleka taký zriedkavý, ako by sa mohlo zdať

**Veta 1.1.9.** (Synchronizovaný tvar EOL-systému)

Nech  $G = (N, T, P, w)$  je EOL-systém pre jazyk  $L$ . Potom existuje taký EOL-systém  $G'$ ,  $L(G') = L(G)$ , že všetky jeho terminálne slová vznikajú z neterminálnych vetných foriem v jednom kroku ododenia<sup>11</sup>

**Dôkaz:** Máme daný EOL-systém  $G = (N, T, P, w)$ . K nemu zostrojíme synchronizovaný EOL-systém  $G'$  v nasledujúcich štyroch krokoch

1.  $\forall a \in T$  vytvoríme nový neterminál  $\xi_a$  a všetky výskyty terminálov v pravidlách z  $P$  nahradíme príslušnými novými neterminálmi (teda  $a$  nahradíme  $\xi_a$ )

<sup>11</sup>teda keď sa vo vetnej forme objaví prvý terminál, tak nutnou podmienkou, aby sme niekedy vygenerovali terminálne slovo je, aby v kroku ododenia, kedy sa do vetnej formy dostal, "zterminálnela" vetná forma celá

2. Pridáme pravidlá  $\xi_a \rightarrow a$  pre  $\forall a \in T$
3. Pridáme pravidlá  $a \rightarrow F$  pre  $\forall a \in T$ ,  $F$  zahrnieme do množiny neterminálov
4. Axiom  $G'$  bude  $w'$ , dostaneme ho z  $w$  tak, že v ňom všetky terminály  $a \in T$  nahradíme príslušnými neterminálmi  $\xi_a$
5. Pridáme pravidlá  $\xi_a \rightarrow \xi_a$  pre  $\forall \xi_a$  novovytvorené, aby sme mohli “naťahovať” odvedenie

Malo by byť zrejmé, že vetná forma, ktorá je počas generovania slova celá zložená iba z neterminálov, sa stáva v jednom kroku celá terminálnou, ak chceme, aby niekedy vygenerovala terminálne slovo, takisto by malo byť intuitívne jasné, že  $L(G) = L(G')$   $\square$

**Veta 1.1.10.** Každý konečný jazyk je v  $\mathcal{L}_{EOL}$

**Dôkaz:** Nech  $L$  je konečný, potom existuje regulárna gramatika  $G$  taká, že  $L = L(G)$ . Táto regulárna gramatika je ale zároveň (po doplnení pravidiel  $a \rightarrow a \forall a \in (N \cup T)$ ) aj hľadaným  $EOL$ -systémom  $\square$

**Lema 1.1.2.** Nech  $G$  je  $EOL$ -systém, potom existuje  $EOL$ -systém  $G'$ , ktorého všetky pravidlá obsahujúce  $a \in T$  na ľavej strane sú tvaru  $a \rightarrow a$

**Dôkaz:** Zkonštruujeme  $EOL$ -systém  $G'$  nasledovne: pre každé  $a \in T$  zavedieme nový neterminál  $\xi_a$  a upravíme pravidlá:

1. tie pravidlá, kde sa vyskytujú terminály, nahradíme novými tak, že všetky terminály (na ľavej i pravej strane) zmeníme na príslušné nové neterminály ( $a \in T$  nahradíme  $\xi_a \in N$ )
2. pre každé  $a \in T$  pridáme pravidlo  $a \rightarrow a$
3. pre každé nové  $\xi_a \in N$  pridáme pravidlo  $\xi_a \rightarrow a$

$\square$

**Veta 1.1.11.** Trieda  $\mathcal{L}_{EOL}$  je uzavretá na  $\cup, \cdot, +, \cap, \mathcal{R}, h_\varepsilon$  a nie je uzavretá na  $h^{-1}$

**Dôkaz:** Dokážeme iba dve vlastnosti, väčšina ostatných dôkazov sa až na malé technické detaily veľmi nelíši od známych konštrukcií pre bezkontextové gramatiky:

- $\cup$  : Máme  $EOL$ -systémy  $G_1$  pre  $L_1$  a  $G_2$  pre  $L_2$ . Ku  $G_1, G_2$  spravíme normálne tvary  $G'_1, G'_2$  podľa konštrukcie z lemy 1.1.2. Bez ujmy na všeobecnosti môžeme predpokladať, že  $N'_1 \cap N'_2 = \emptyset$ . Vytvoríme  $G_3$  pre  $L_1 \cup L_2$  nasledovne: zavedieme nový neterminál  $\sigma_3$  tak, že  $\sigma_3 \notin N'_1$  a  $\sigma_3 \notin N'_2$ . Ďalej  $N_3 = N'_1 \cup N'_2 \cup \{\sigma_3\}$ ,  $T_3 = T'_1 \cup T'_2$ ,  $P_3 = P'_1 \cup P'_2 \cup \{\sigma_3 \rightarrow \sigma'_1 | \sigma'_2\}$  a nakoniec  $G_3 = (N_3, T_3, P_3, \sigma_3)$
- $h^{-1}$  : Zoberme jazyk  $L \in \mathcal{L}_{EOL}$ ,  $L = \{a^{2^n} \mid n \geq 0\}$  a definujme homomorfizmus  $h$  nasledovne:  $h(a) = a, h(b) = \varepsilon$ . Potom  $h^{-1}(L) = \{w \in \{a, b\}^* \mid \#_a w = 2^n, n \geq 0\}$ . Ukážeme, že  $h^{-1}(L) \notin \mathcal{L}_{EOL}$ . Nech  $G$  je  $EOL$ -systém pre  $h^{-1}(L)$ . Zoberme  $w_0 \in h^{-1}(L)$ , nech  $\#_a w_0 = n$ . Zoberme ľubovoľné dve  $a$ , medzi ktorými sú iba  $b$ . Ak je medzi týmito dvoma  $a$  “príliš veľa”<sup>12</sup>  $b$ , tak takéto slovo nedokážeme vyrobiť (nemáme na to dostatok času), čo je spor s tým, že  $G$  je  $EOL$ -systém pre  $h^{-1}(L)$

$\square$

<sup>12</sup>tento počet vieme určiť, závisí napr. od dĺžky najdlhšej pravej strany pravidla v  $G$ , bližšie v [2]

**Veta 1.1.12.**  $\mathcal{L}_{CF} \subsetneq \mathcal{L}_{EOL}$

**Dôkaz:** Každá bezkontextová gramatika vyhovuje definícii  $EOL$ -systému (po doplnení pravidiel  $a \rightarrow a \ \forall a \in (N \cup T)$ ), teda platí nevlastná inklúzia  $\mathcal{L}_{CF} \subseteq \mathcal{L}_{EOL}$ . Že platí aj vlastná inklúzia, sme ukázali v príklade 1.1.6, kde sme našli  $EOL$ -systém pre jazyk  $L = \{a^n b^n c^n \mid n \geq 1\}$ , čo je jazyk z  $\mathcal{L}_{CS} - \mathcal{L}_{CF}$   $\square$

**Poznámka 1.1.5.**  $1L$  a  $2L$ -systémy sú akousi analógiou kontextových gramatík, i keď o ich sile by sa dali viesť podobné diskusie ako pri  $OL$ -systémoch. Snáď len toľko, že obojstranný kontext je silnejší ako jednostranný a oba systémy sú silnejšie ako  $OL$ -systémy

### 1.1.4 $TOL$ -systémy (table)

Tieto systémy nebudeme striktné definovať, povieme si len základné veci, ktorými sa od klasických  $OL$ -systémov odlišujú.  $TOL$ -systém  $G = (V, P_1, \dots, P_k, w_0)$  má niekoľko (konkrétne  $k$ ) sád pravidiel, v danom kroku odvodenia použijeme na vetnú formu iba jednu sadu pravidiel.

**Príklad 1.1.7.**  $G = (\{a\}, \{a \rightarrow a^2\}, \{a \rightarrow a^3\}, a)$ , potom  $L(G) = \{a^{2^n} a^{3^m} \mid n, m \geq 0\} \notin \mathcal{L}_{OL}$

**Poznámka 1.1.6.** Trieda  $\mathcal{L}_{TOL}$  zdieľa "zlé" uzáverové vlastnosti triedy  $\mathcal{L}_{OL}$ . Podobne ako pre  $\mathcal{L}_{OL}$  platí aj pre  $\mathcal{L}_{TOL}$  vzťah  $\mathcal{L}_{TOL} \subseteq \mathcal{L}_{ECS}$

## 1.2 Ďalšie paralelné gramatiky

V tejto časti sa letmo budeme venovať paralelným gramatikám s iným prepisovacím systémom, ako mali  $L$ -systémy a ukážeme o nich pár, možno zaujímavých, výsledkov.

### 1.2.1 Indické paralelné gramatiky

**Definícia 1.2.1.** Indická paralelná gramatika ( $IP$ ) je štvorica  $G = (N, T, P, \sigma)$ , pričom  $\sigma \in N \cap T = \emptyset, P \subseteq N \times (N \cup T)^*$

**Definícia 1.2.2.** Krok odvodenia  $IP$   $G$  je relácia  $\Rightarrow$  keď sa všetky výskyty zvoleného neterminálu prepíšu naraz tým istým pravidlom

**Príklad 1.2.1.**  $G_1 = (\{\sigma\}, \{a\}, \{\sigma \rightarrow \sigma\sigma, \sigma \rightarrow a\}, \sigma)$  potom  $L(G_1) = \{a^{2^n} \mid n \geq 0\}$

**Príklad 1.2.2.** Dyckov jazyk nad dvoma písmenkami  $D_1$  (jazyk správne uzátvorkovaných výrazov) nie je  $\mathcal{L}_{IP}$

**Dôkaz:** Sporom predpokladajme, že  $D_1 \in \mathcal{L}_{IP}$ , potom existuje  $IP$  gramatika  $G$  taká, že  $L(G) = D_1$ . Ukážeme, že by musela mať nekonečne veľa neterminálov, čo nie je možné. Pre jednoduchosť označme  $L = D_1$ . Vytvoríme v  $L$  hierarchiu tvarov slov nasledovne: definujeme rekurentne podmnožiny  $L$  a uvedomme si, koľko najmenej neterminálov treba na generovanie slov z každej z nich:

$$L_1 = \{a^n b^n \mid n \geq 1\}^+ \text{ treba aspoň 2 neterminály}$$

$$L_{i+1} = \{a^n w b^n \mid w \in L_i, n \geq 1\}^+ \text{ treba aspoň } i + 2 \text{ neterminálov}$$

Ako vidno, táto hierarchia je nekonečná, teda na generovanie  $(\bigcup_{i=1}^{\infty} L_i) \subseteq L$  je treba nekonečne veľa neterminálov  $\square$

**Veta 1.2.1.**  $\mathcal{L}_{IP} \cap \mathcal{L}_{CF} = \mathcal{L}_{DBL}$

**Poznámka 1.2.1.**  $\mathcal{L}_{DBL}$  je trieda Derivation Bounded Languages, ku každému jazyku z tejto triedy existuje  $k$  také, že počet neterminálov v každej vetnej forme je menší ako  $k$

**Veta 1.2.2.**  $\mathcal{L}_{IP} \subseteq \mathcal{L}_{ECS}$

**Veta 1.2.3.**  $\mathcal{L}_{IP}$  je uzavretá na  $\cup, \cdot, *, h$

**Veta 1.2.4.**  $\mathcal{L}_{IP}$  je neporovnateľná s  $\mathcal{L}_{EOL}$

**Veta 1.2.5.**  $\mathcal{L}_{IP} \subseteq \mathcal{L}_{ETOL}$  (extended TOL)

## 1.2.2 Ruské paralelné gramatiky

**Definícia 1.2.3.** Ruská paralelná gramatika je päťica  $G = (N, T, P_1, P_2, \sigma)$ , pričom  $\sigma \in N$ ,  $N \cap T = \emptyset$ ,  $P_1, P_2$  sú konečné množiny pravidiel,  $P_1, P_2 \subseteq N \times (N \cup T)^*$

**Definícia 1.2.4.** Krok odvodenia: všetky výskyty zvoleného neterminálu sa prepíšu naraz istým pravidlom z  $P_1$ , alebo sa prepíše práve jeden neterminál pravidlom z množiny  $P_2$

## 1.2.3 Absolútne paralelné gramatiky

**Definícia 1.2.5.** Absolútne paralelná gramatika (AP) je štvorica  $G = (N, T, P, \sigma)$ , kde  $\sigma \in N$ ,  $N \cap T = \emptyset$ ,  $P$  je konečná množina pravidiel tvaru  $(A_1, \dots, A_n) \rightarrow (w_1, \dots, w_n)$ , kde  $\forall A_i \in N$ ,  $w_i \in (N \cup T)^*$

**Definícia 1.2.6.** Krok odvodenia je relácia  $w \Rightarrow v$  práve vtedy, keď  $w = u_1 A_1 u_2 A_2 \dots u_n A_n u_{n+1}$  a  $v = u_1 w_1 u_2 w_2 \dots u_n w_n u_{n+1}$ , pričom  $u_1, \dots, u_{n+1} \in T^*$  (t.j. naraz prepisujeme všetky neterminály vo vetnej forme, na každú možnosť výskytu neterminálov musíme mať pravidlo)

**Príklad 1.2.3.** Pozrime sa na AP gramatiku:

$G = (\{S\}, \{a, b, c\}, \{(S) \rightarrow (SSS), (S, S, S) \rightarrow (aS, bS, cS), (S, S, S) \rightarrow (a, b, c)\}, S)$  potom  $L(G) = \{a^n b^n c^n \mid n \geq 1\}$

**Veta 1.2.6.**  $\mathcal{L}_{AP}$  je  $\mathcal{AFL}$

**Veta 1.2.7.**  $\mathcal{L}_{AP} \cap 2^{a^*} \subseteq \mathcal{R}$  (t.j. jednopísmenkové AP jazyky sú regulárne)

**Dôkaz:** Nech  $L \in \mathcal{L}_{AP}$  a  $G = (N, T, P, A_p)$  je AP gramatika taká, že  $L(G) = L$ . Bez újmy na všeobecnosti môžeme predpokladať, že  $T = \{a\}$ . Nech  $N = \{A_1, \dots, A_n\}$ . Definujme  $N'$  nasledovne:  $N' = \{A_p\} \cup \{A_{i_1, \dots, i_k} \mid \forall j A_{i_j} \in N, A_{i_1}, \dots, A_{i_k} \text{ sa nachádzajú na pravej strane niektorého (rovnakého) z pravidiel } P \text{ v tomto poradí}\}$ . Bez újmy na všeobecnosti môžeme predpokladať, že všetky pravidlá z  $P$  sú tvaru  $(A_{i_1}, \dots, A_{i_m}) \rightarrow (a^{k_1} \pi_1, \dots, a^{k_m} \pi_m)$ , kde  $\forall i \pi_i \in N^*$ . Definujme  $\varphi_i = i_1, \dots, i_l \iff \pi_i = A_{i_1} \dots A_{i_l}$ . Vytvoríme množinu pravidiel  $P'$  nasledovne:

$$A_{i_1, \dots, i_k} \rightarrow a^{w_1 + \dots + w_k} A_{\varphi_{i_1}, \dots, \varphi_{i_k}} \in P' \iff (A_{i_1}, \dots, A_{i_k}) \rightarrow (a^{w_1} \pi_1, \dots, a^{w_k} \pi_k) \in P$$

Teraz definujme  $G' = (N', \{a\}, P', A_p)$ .  $G'$  je zrejme regulárna gramatika a z konštrukcie vyplýva, že  $L(G') = L(G)$   $\square$

**Veta 1.2.8.**  $\mathcal{L}_{AP} = \mathcal{L}_{2DAP}$

**Poznámka 1.2.2.**  $\mathcal{L}_{2DAP}$  je trieda jazykov, ktoré sú generované dvojsmernými deterministickými  $a$ -prekladačmi

Existuje množstvo ďalších paralelných gramatík, nie sú však zaujímavé do takej miery, aby malo význam sa nimi zaoberať.

## Kapitola 2

# Generatívne systémy

Doteraz sme sa zaoberali iba paralelnými modelmi, ktorých spoločnou črtou bola akási “gramatiková filozofia”, teda mali sme množinu symbolov, počiatočný symbol, resp. počiatočné slovo a akúsi množinu prepisovacích pravidiel, ktoré striktne riadili odvodenie a celý mechanizmus generoval nejaký jazyk. Ukážeme si trochu iný pohľad na vec, nový z hľadiska spôsobu prepisovania, kedy zostane zachovaná “viditeľná gramatiková časť”, teda symboly, počiatočný symbol, zmení sa však prepisovací mechanizmus. Ako uvidíme neskôr, pôjde o akúsi kombináciu gramatikového a automatového pohľadu na jazyky. Pomocou tohto mechanizmu bude možné simulovať už známe gramatiky, či už z Chomského hierarchie, alebo skôr spomenuté paralelné gramatiky. Zaujímavý (a veľmi dôležitý) na tomto modeli je fakt, že pomerne jednoducho umožní porovnať paralelné a sekvenčné formalizmy z hľadiska zložitosti, a tým ukázať, v ktorej oblasti nám paralelizmus pomôže a kde naopak nemá zmysel sa ním veľmi zaoberať.

### 2.1 Definície a označenia

**Definícia 2.1.1.** *Abstraktný generatívny systém je štvorica  $G = (N, T, f, \sigma)$  kde  $N$  je množina neterminálov,  $T$  je množina terminálov,  $f$  je konečne zadaná funkcia  $(N \cup T)^* \rightarrow 2^{(N \cup T)^*}$ ,  $\sigma$  je počiatočný neterminál pričom  $N, T$  nie sú nutne disjunktné abecedy.*

**Definícia 2.1.2.** *Krok odvodenia abstraktného generatívneho systému je relácia  $\Rightarrow$  na  $(N \cup T)^*$  definovaná takto:  $u \Rightarrow v$  práve vtedy keď  $v \in f(u)$ .*

**Definícia 2.1.3.** *Jazyk generovaný abstraktným generatívnym systémom  $G$  je množina  $L(G)$ , pričom  $L(G) = \{w \in T^* \mid \sigma \xrightarrow{*} w\}$ .*

V ďalšom sa budeme zaoberať špeciálnym typom abstraktných generatívnych systémov, a to generatívnymi systémami.

**Definícia 2.1.4.** *1-a-prekladač je a-prekladač  $M = (K, \Sigma_1, \Sigma_2, H, q_0, F)$ , v ktorom množina štvoríc je tvaru<sup>1</sup>  $H \subseteq K \times \Sigma_1 \times \Sigma_2^* \times K$ .*

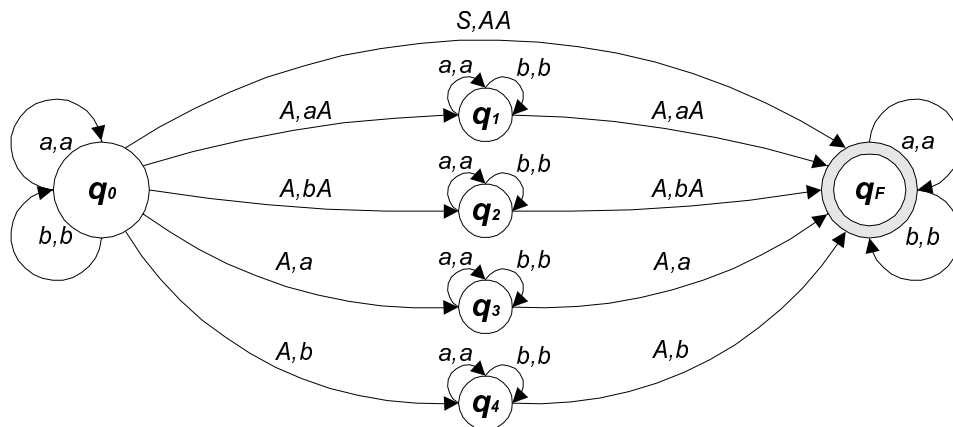
**Definícia 2.1.5.** *Generatívny systém (g-systém) je abstraktný generatívny systém, v ktorom  $f$  je zobrazenie 1-a-prekladačom. Zapisujeme  $G = (N, T, M, \sigma)$  kde  $M$  je 1-a-prekladač.*

---

<sup>1</sup>1-a-prekladač nemá pružnú čítaciu hlavu t.j. môže čítať práve jeden symbol

**Definícia 2.1.6.** *Krok odvodenia  $g$ -systému je relácia  $\Rightarrow$  na  $(N \cup T)^*$  definovaná takto:  $u \Rightarrow v$  práve vtedy<sup>2</sup> keď  $v \in M(u)$ .*

**Príklad 2.1.1.** Zostrojíme  $g$ -systém  $G = (N, T, M, S)$  pre jazyk  $L = \{ww \mid w \in \{a, b\}^*\}$ . Neterminály budú  $N = \{S, A\}$ , terminály  $T = \{a, b\}$ ,  $a$ -prekladač  $M$  bude (obr.2.1).



Obrázok 2.1: 1- $a$ -prekladač  $g$ -systému pre jazyk  $L$  z príkladu 2.1.1

**Označenie:**  $\mathcal{G}$  trieda všetkých  $g$ -systémov.

$\mathcal{G}_\varepsilon$  trieda všetkých bez- $\varepsilon$   $g$ -systémov, t.j. 1- $a$ -prekladač nedáva na výstup  $\varepsilon$ .

## 2.2 Porovnanie generatívnej sily $g$ -systémov s gramatikami Chomského hierarchie

**Veta 2.2.1.**  $\mathcal{L}_{\mathcal{G}} = \mathcal{L}_{RE}$

**Dôkaz:** Dokážeme obe inklúzie:

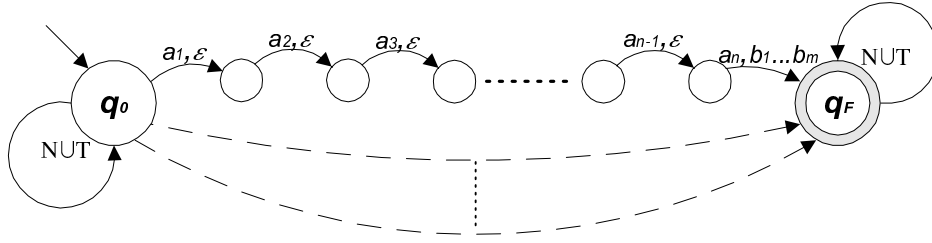
$\subseteq$ : Táto inklúzia vyplýva z Turingovej tézy, ale iste by sme si vedeli predstaviť aj TS, ktorý dostane na pásku slovo  $a$  na druhej páske simuluje odvodenie tohto slova  $g$ -systémom od počiatočného neterminálu a porovnáva, či je slovo z druhej páske zhodné so vstupom na prvej páske.

$\supseteq$ : Pomocou  $g$ -systémov chceme simulovať frázové gramatiky, t.j. ku každej frázovej gramatike  $G$  treba zostrojiť  $g$ -systém  $G'$  taký, že  $L(G') = L(G)$ .  $G'$  zostrojíme nasledovne: ku každému pravidlu  $a_1 a_2 \dots a_n \rightarrow b_1 b_2 \dots b_m$  urobíme v  $g$ -systéme cestu ako na obrázku 2.2.

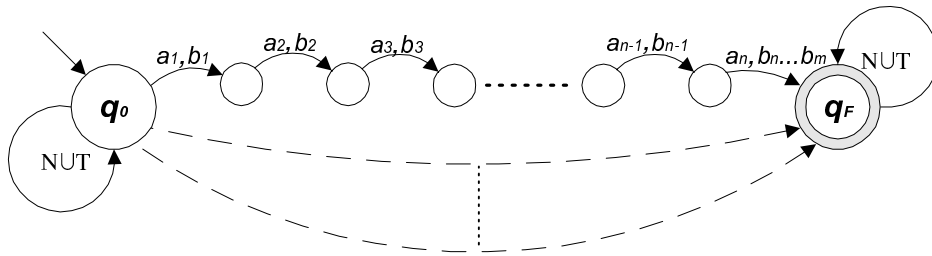
□

<sup>2</sup>pre vstupné slovo  $u$  1- $a$ -prekladač  $M$  vygeneruje výstup  $v$





Obrázok 2.2: Konštrukcia  $g$ -systému ku frázovej gramatike



Obrázok 2.3: Konštrukcia  $g$ -systému ku kontextovej gramatike

**Veta 2.2.2.**  $\mathcal{L}_{g_\varepsilon} = \mathcal{L}_{CS}$

**Dôkaz:** Dokážeme obe inklúzie:

$\subseteq$ : K bez- $\varepsilon$   $g$ -systému zostrojíme LBA akceptujúci ten istý jazyk. LBA si, rovnako ako TS v predchádzajúcej vete, vyrobí druhú pásku, na ktorej bude simulovať odvodenie vstupného slova simulovaným  $g$ -systémom. Keďže  $g$ -systém je bez  $\varepsilon$ , nemôže sa stať, že pri odvodení nejakého slova  $w$  vznikne vetná forma dĺžky väčšej ako je  $|w|$ . Preto na simuláciu stačí priestor ohraničený dĺžkou vstupného slova, a teda vystačíme s LBA.

$\supseteq$ : Konštrukcia je podobná ako v predchádzajúcej vete (obr.2.3) s využitím toho, že v kontextových gramatikách nie je pravá strana pravidla kratšia ako ľavá, takže v konštrukcii 1- $a$ -prekladača nepotrebujeme pravidlá s  $\varepsilon$  výstupom.

□

**Definícia 2.2.1.** *Kopírovací cyklus v 1- $a$ -prekladači je štvorica z  $H$  tvaru  $(q, a, a, q)$ .*

**Definícia 2.2.2.**  *$g$ -systém je sekvenčný, ak jediné cykly v 1- $a$ -prekladači sú kopírovacie cykly v počiatočnom alebo koncovom stave.*

**Označenie:**  $\mathcal{S}$  trieda všetkých sekvenčných  $g$ -systémov.

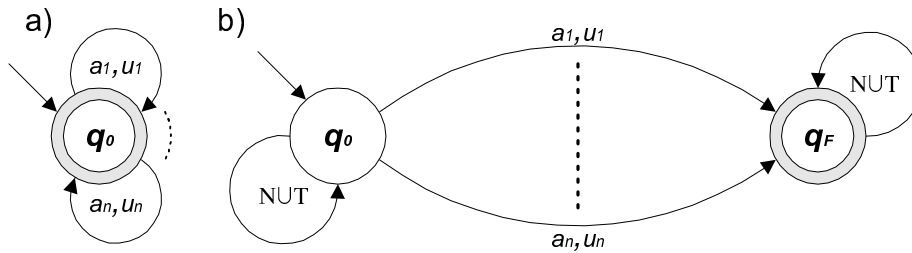
$\mathcal{S}_\varepsilon$  trieda všetkých bez- $\varepsilon$  sekvenčných  $g$ -systémov.

**Veta 2.2.3.**  $\mathcal{L}_S = \mathcal{L}_{RE}, \mathcal{L}_{S_e} = \mathcal{L}_{CS}$

**Dôkaz:** Konštrukcia je tá istá ako vo vete 2.2.1, resp. 2.2.2, lebo ľahko vidno, že zostrojený  $g$ -systém je sekvenčný.  $\square$

## 2.3 Modelovanie známych gramatík pomocou $g$ -systémov

**Príklad 2.3.1.** *EOL*: Nech množina pravidiel *EOL* je  $P = \{a_1 \rightarrow u_1, a_2 \rightarrow u_2, \dots, a_n \rightarrow u_n\}$  potom príslušný  $g$ -systém bude vyzeráť ako na obrázku 2.4a.



Obrázok 2.4: Modelovanie *EOL*-systémov a *CF*-gramatík pomocou  $g$ -systémov

*CF*: Nech množina pravidiel *CF* je  $P = \{a_1 \rightarrow u_1, a_2 \rightarrow u_2, \dots, a_n \rightarrow u_n\}$  potom príslušný  $g$ -systém bude vyzeráť ako na obrázku 2.4b.

**Poznámka 2.3.1.** Gramatika  $G$  je bezkontextová práve vtedy, keď existuje sekvenčný  $g$ -systém  $G'$  s dvoma stavmi taký, že  $L(G) = L(G')$ . Implikácia " $\Rightarrow$ " vyplýva z konštrukcie na obrázku 2.4b, dôkaz opačnej implikácie nie je triviálny a presahuje rámec tohto textu.

**Veta 2.3.1.** Absolutne paralelné gramatiky sú ekvivalentné s  $g$ -systémami spĺňajúce nasledujúce podmienky (ozn.  $\tilde{\mathcal{G}}$  trieda takýchto  $g$ -systémov):

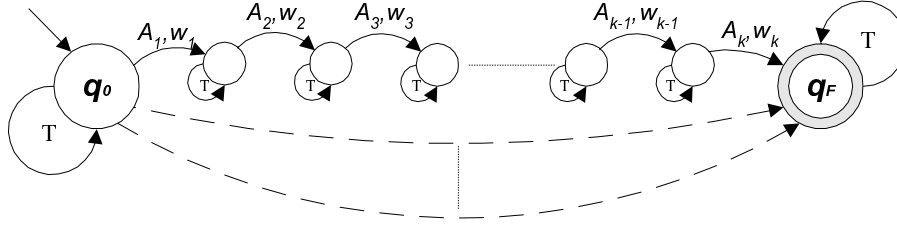
1. každý cyklus je kopírovací cyklus pre terminál
2. v každom stave je kopírovací cyklus pre každé  $a \in T$
3. terminálne symboly sú prepisované len v kopírovacích cykloch

**Dôkaz:** Máme dokázať, že  $\mathcal{L}_{AP} = \mathcal{L}_{\tilde{\mathcal{G}}}$

$\subseteq$ : K danej *AP* gramatike  $G$  zkonštruujeme ekvivalentný  $g$ -systém  $G' \in \tilde{\mathcal{G}}$  nasledovne. Každému pravidlu v  $G$  tvaru  $(A_1, \dots, A_k) \rightarrow (w_1, \dots, w_k)$  zodpovedá v 1- $a$ -prekladači  $g$ -systému  $G'$  práve jedna cesta znázornená na obrázku 2.5.

$\supseteq$ : Pre každú cestu v 1- $a$ -prekladači z  $q_0$  do  $q_F$  (mimo kopírovacích cyklov) v *AP* gramatike urobíme pravidlo:  $(A_1, \dots, A_k) \rightarrow (w_1, \dots, w_k)$ .

$\square$



Obrázok 2.5: Konštrukcia  $g$ -systému k  $AP$  gramatike

## 2.4 Miery zložitosti

- $STATE(G)$  = počet stavov 1- $a$ -prekladača  $g$ -systému  $G$
- $ARC(G)$  = počet pravidiel 1- $a$ -prekladača  $g$ -systému  $G$
- $STATE_U(L) = \min\{STATE(G) \mid L = L(G), G \in U\}$
- $ARC_U(L) = \min\{ARC(G) \mid L = L(G), G \in U\}$
- $TIME(G, w) = \min\{k \mid \sigma \xrightarrow[k]{G} w\}; w \in L(G)$
- $TIME(G, n) = \max\{TIME(G, w) \mid w \in L(G), |w| \leq n\}$
- $TIME_U(f(n)) = \{L \mid \exists G \in U, L = L(G), TIME(G, n) = O(f(n))\}$
- $\overline{TIME}_U(f(n)) = \{L \mid \forall G \in U, L = L(G), TIME(G, n) = \Omega(f(n))\}$
- $SPACE(G, w) = \min\{SPACE(\alpha)\}$ ; kde  $\alpha : \sigma \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_n$  je odvodenie  $w$  v  $G$  a  $SPACE(\alpha) = \max\{|u_i| \mid 0 \leq i \leq n\}$

**Veta 2.4.1.** Pre každý nekonečný jazyk  $L$  platí:

1.  $L \in \overline{TIME}_S(n)$
2.  $L \in \overline{TIME}_G(\log n)$

**Dôkaz:**

1. Nech  $G \in \mathcal{S}$ . Zmyslime sa nad tým, aké najdlhšie slovo môže vygenerovať  $G$  na  $n$  krokov.  $G$  dokáže v jednom kroku prepisovať len jeden súvislý úsek vetnej formy. Túto vetnú formu dokáže  $G$  predĺžiť o najviac  $m = \max\{\text{súčet dĺžok výstupov na jednej ceste v 1- $a$ -prekladači} - \text{dĺžka tejto cesty}\}$   
Majme odvodenie  $\sigma = w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_i \Rightarrow w_{i+1} \Rightarrow \dots \Rightarrow w_n$ . Potom platí  $|w_{i+1}| \leq |w_i| + m$ . Keďže  $|w_0| = |\sigma| = 1$ , tak  $|w_n| \leq 1 + nm$ . Z toho teda nakoniec plynie, že  $G$  potrebuje na vygenerovanie slova dĺžky  $n$  aspoň lineárny čas.
2. Nech  $G \in \mathcal{G}$ .  $G$  môže v každej vetnej forme prepísať všetky symboly. Označme  $m = \max\{\text{výstup v 1- $a$ -prekladači}\}$   
Potom platí  $|w_{i+1}| \leq |w_i| \cdot m$ . Keďže  $|w_0| = |\sigma| = 1$ , tak  $|w_n| \leq m^n$ . Z toho plynie, že  $G$  potrebuje na vygenerovanie slova dĺžky  $n$  aspoň logaritmický čas

□

## 2.5 Porovnanie sekvenčných a paralelných $g$ -systémov

**Veta 2.5.1.**  $\mathcal{L}_G = \mathcal{L}_S, \mathcal{L}_{G_\varepsilon} = \mathcal{L}_{S_\varepsilon}$

**Dôkaz:** Tvrdenie priamo vyplýva z viet 2.2.1, 2.2.2 a 2.2.3 a hovorí nám, že paralelné a sekvenčné  $g$ -systémy sú z hľadiska generatívnej sily ekvivalentné, no nedáva nám žiadny návod, ako ku paralelnému  $g$ -systému nájsť sekvenčný, rovnako nič nehovorí o tom, aký vplyv bude mať táto konštrukcia na popisnú zložitosť.

Ukážeme si, ako k ľubovoľnému  $g$ -systému  $G \in \mathcal{G}$  zkonštruujeme  $g$ -systém  $G'' \in \mathcal{S}$  s ním ekvivalentný (t.j.  $L(G) = L(G'')$ ) a potom si povieme, ako sa “zosekvenčnenie” odrazí na zložitosti.

Nech  $G = (N, T, M, \sigma)$  je ľubovoľný  $g$ -systém,  $L(G) \in \mathcal{G}$ , počiatkový stav  $M$  je  $q_0$  a koncový  $q_F$ . Uvedomme si, čo potrebujeme na  $G$  prepracovať, aby sme ho “zosekvenčili”. V prvom rade potrebujeme mať v počiatkovom aj koncovom stave kopírovacie cykly pre všetky<sup>3</sup> symboly, aby sme sa vo vetnej forme mohli presunúť ľubovoľne ďaleko, potom niečo  $a$ -prekladačom prepísať a vetnú formu dokopírovať do konca. V druhom rade potrebujeme odstrániť z  $G$  vnútorné cykly (teda všetky také, ktoré nie sú kopírovacími cyklami v počiatkovom, resp. koncovom stave). Keď sa nám toto podarí, budeme s konštrukciou hotoví.  $G''$  budeme konštruovať v dvoch krokoch:

1. Zostrojíme  $G' = (N', T, M', \bar{\sigma})$  ekvivalentný s  $G$  s užitočnými technickými vlastnosťami, ktoré neskôr využijeme:

- (a) zavedieme nový počiatkový stav  $q'_0$  a nový koncový stav  $q'_F$  a v týchto stavoch zkonštruujeme kopírovacie cykly<sup>4</sup>  $\forall a \in (N' \cup T)$ ,  $N'$  definujeme neskôr, označme množinu týchto kopírovacích cyklov  $C = \{(q, a, a, q) \mid q \in \{q'_0, q'_F\}, a \in (N \cup T)\}$ .
- (b)  $G'$  bude udržiavať informáciu o prvom a poslednom symbole vetnej formy tak, že ich označí: prvý symbol horným prúžkom, posledný dolným prúžkom. Označené symboly budú nové neterminály, v  $G'$  musíme vytvoriť nové podčiarknuté aj nadčiarknuté symboly, teda  $N' = N \cup \{\bar{a} \mid a \in (N \cup T)\} \cup \{\underline{a} \mid a \in (N \cup T)\} \cup \{\bar{\sigma}\}$ . Počiatkový neterminál  $G'$  bude  $\bar{\sigma}$ . Vetná forma bude vyzerať nasledovne:  $\bar{a}_1 a_2 a_3 \dots a_{n-1} a_n$ .

Teraz poďme konštruovať  $G'$  tak, aby sme zabezpečili obe “dobré” vlastnosti, ktoré sme uviedli a zároveň aby boli  $G$  a  $G'$  ekvivalentné. Zrejme bude potrebné upraviť prechodovú množinu  $1$ - $a$ -prekladača. Ak  $H$  bola prechodová množina  $M$ , tak  $H_{M'}$  bude prechodová množina  $M'$ . Zatiaľ definujme  $H'$  nasledovne (nech  $a, c, A, B \in (N \cup T)$ ,  $b, d \in (N \cup T)^*$ ):

$$H' = H \cup C \cup \{(q'_0, \bar{A}, \bar{a}b, q_1) \mid (q_0, A, ab, q_1) \in H\} \cup \{(q, \underline{B}, \underline{c}d, q'_F) \mid (q, B, cd, q_F) \in H\}$$

Ak sa nad  $H'$  trochu zamyslíme, tak je jasné, že v  $q'_0$ , resp. v  $q'_F$  síce máme kopírovacie cykly, ale nebudeme ich používať. Keby sme totiž použili ľubovoľný kopírovací cyklus v  $q'_0$ , tak sa už z tohto stavu nikdy nedostaneme, pretože z neho vedú iba šípky na nadčiarknutý, teda prvý, symbol vetnej formy. Čo sa týka stavu  $q'_F$ , tak do neho sa možno dostať iba na podčiarknutý, teda posledný, symbol vetnej formy.

Aby sme našu konštrukciu príliš nepretechnizovali, nebudeme sa zaoberať detailami ohľadom počiatkového neterminálu  $\bar{\sigma}$  a vynecháme aj prípady, kedy  $M$  “príliš veľa” vymazáva, teda ak sa v odvodení môže vyskytnúť vetná forma obsahujúca jediný symbol, prípadne  $\varepsilon$ . Pozrime sa teraz na  $g$ -systém, ktorý sme zkonštruovali. Keby sme uvažovali  $G \in \mathcal{G}_\varepsilon$ , tak už teraz máme (až na prvý a posledný symbol, s ktorým budeme pracovať neskôr)

<sup>3</sup>ako sa neskôr ukáže, tak nie nutne pre úplne všetky, stačí pre niektoré

<sup>4</sup>Zrejme by nestačilo dodať kopírovacie cykly do  $q_0$ , resp.  $q_F$ . Mohlo by sa totiž ľahko stať, že by sme takto zkonštruovaným  $1$ - $a$ -prekladačom akceptovali aj to, čo  $1$ - $a$ -prekladač v  $G$  neakceptoval

$G'$  ekvivalentný s  $G$  a splnené (a) aj (b). Ale my uvažujeme  $G \in \mathcal{G}$ . Uvedomme si, aké nepríjemnosti nám môže spôsobiť skutočnosť, že 1- $a$ -prekladač môže zapisovať na výstup  $\varepsilon$ . Môže sa stať, že v  $H$  existuje  $(q_0, A, \varepsilon, q_1)$ . Podľa doterajšej konštrukcie by sme do  $H'$  pridali  $(q'_0, \bar{A}, \varepsilon, q_1)$ . Tým by sme ale vymazali prvý symbol vetnej formy a 1- $a$ -prekladač by nepracoval v ďalších krokoch odvodenia správne. Podobne sa nám môže stať, že v  $H$  existuje  $(q, B, \varepsilon, q_F)$ , potom by sme do  $H'$  pridali  $(q, \underline{B}, \varepsilon, q'_F)$ , teda by sme si zmazali posledný (podčiarknutý) symbol vetnej formy a v ďalšom kroku odvodenia by nastali problémy s akceptovaním. Ukazuje sa teda, že predchádzajúcu konštrukciu možno použiť iba na tie  $(q, a, u, p)$ , kde  $u \neq \varepsilon$  (zámerné sme do tretej komponenty vždy písali reťazec  $w \in (N' \cup T)^+$ ).

Pre  $\varepsilon$ -výstupy 1- $a$ -prekladača použijeme nasledovnú konštrukciu:

- (a) 1- $a$ -prekladaču  $M'$  pridáme nové stavy tak, že pre každý stav  $q$  1- $a$ -prekladača  $M$  vyrobíme nový stav  $\bar{q}$ . V týchto stavoch si budeme pamätať, že sme si zmazali prvý symbol vetnej formy a keď budeme robiť v danom kroku odvodenia prvý  $\varepsilon$ -výstup, tak prvý symbol z neho bude zároveň aj prvým symbolom vetnej formy, a tak mu pridáme čiaru hore. Teraz to všetko formálne zapíšeme. Označme  $\bar{H}$  prechodovú množinu, ktorú definujeme nasledovne ( $u \in (N \cup T)^*$ ):

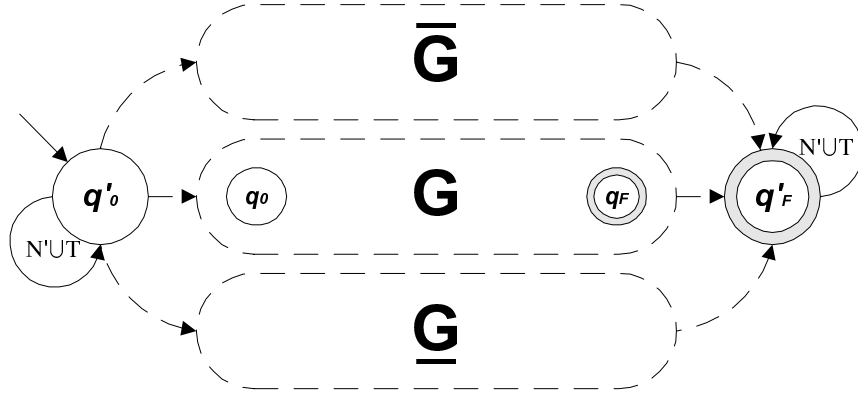
$$\begin{aligned} \bar{H} = & \{(q'_0, \bar{A}, \varepsilon, \bar{q}_1) \mid (q_0, A, \varepsilon, q_1) \in H\} \cup && \text{pamätáme si, že sme zmazali } \bar{A} \\ & \cup \{(\bar{q}, a, \varepsilon, \bar{p}) \mid (q, a, \varepsilon, p) \in H\} \cup && \text{stále sme nezapísali prvý symbol} \\ & \cup \{(\bar{q}, a, \bar{b}u, p) \mid (q, a, bu, p) \in H\} && \text{zapíšeme } \bar{b}u \end{aligned}$$

- (b) 1- $a$ -prekladaču  $M'$  pridáme nové stavy tak, že pre každý stav  $q$  1- $a$ -prekladača  $M$  vyrobíme nový stav  $\underline{q}$ . Tieto stavy budeme používať na to, aby sme si nezmazali posledný (podčiarknutý) symbol vetnej formy. Tu to však nebude také jednoznačné ako pri prvom symbole. Pretože 1- $a$ -prekladač sa nemôže vracieť, tak po tom, ako zmaže posledný symbol, už nemôže označiť čiarou dole nejaký iný. Na to, že raz zmaže posledný symbol vetnej formy, musí prísť “dostatočne skoro”, využijeme možnosť nedeterminizmu. Uhádneme, že istý symbol vo vetnej forme je posledný, ktorý reálne zapisujeme, a že to, čo vznikne prepísaním symbolov za ním budú iba  $\varepsilon$ , a potom len kontrolujeme, či sme hádali správne. Formálne zapísané v prechodovej množine  $\underline{H}$  ( $w \in (N \cup T)^*$ ):

$$\begin{aligned} \underline{H} = & \{(q, a, w\underline{b}, \underline{p}) \mid (q, a, wb, p) \in H\} \cup && \text{nedet. zapíše nový posledný symbol} \\ & \cup \{(\underline{q}, a, \varepsilon, \underline{p}) \mid (q, a, \varepsilon, p) \in H\} \cup && \text{stále maže všetky symboly} \\ & \cup \{(\underline{q}, \underline{a}, \varepsilon, \underline{q}'_F) \mid (q, a, \varepsilon, q_F) \in H\} && \text{ak zmaže posledný symbol, akceptuje} \end{aligned}$$

Vytvorili sme akoby virtuálne  $g$ -systémy  $G, \underline{G}, \bar{G}$  (obr.2.6), pričom v jednom kroku odvodenia sa môžeme medzi nimi pohybovať. Neexistujú však žiadne šípky  $G \rightarrow \bar{G}$ ,  $\underline{G} \rightarrow G$ ,  $\underline{G} \rightarrow \bar{G}$ . Na to, aby sme mali  $L(G) = L(G')$  ešte potrebujeme v istom kroku odznačiť prvý aj posledný označený symbol. Avšak keďže máme v 1- $a$ -prekladači  $g$ -systému  $G'$  šípky z  $q'_0$  do iného stavu, resp. šípky do  $q'_F$  z iného stavu (teda nie kopírovacie cykly v týchto stavoch) iba na označený symbol, tak s odznačenou vetnou formou už ďalej nepohneme. Z toho plynie, že symboly možno odznačiť len v poslednom kroku odvodenia, teda vtedy, ak všetky symboly vetnej formy, okrem prvého a posledného, sú terminály a aj prvý a posledný symbol (odhliadnuc od označenia) sú terminály. Teda opäť nedeterministicky hádame posledný krok. Formálne definujeme:

$$H_T = \{(q'_0, \bar{a}, a, q_T), (q_T, a, a, q_T), (q_T, \underline{a}, a, q'_F) \mid a \in T\}$$



Obrázok 2.6: 1- $a$ -prekladač  $g$ -systému  $G'$

$q_T$  je nový stav 1- $a$ -prekladača  $g$ -systému  $G'$ . Zachovali sme konzistentnosť v tom zmysle, že z  $q'_0$  vychádzajú, okrem kopírovacích cyklov, iba šípky na prvý (nadčiarknutý) symbol a do  $q'_F$  vchádzajú, okrem kopírovacích cyklov, iba šípky na posledný (podčiarknutý) symbol vetnej formy, teda máme  $L(G) = L(G')$ , pričom  $G' = (N', T, M', \underline{\sigma})$ . Čo sa týka  $M'$  pre nás je zaujímavé, ako sa zmenila prechodová množina: keď v  $M$  to bola  $H$ , tak v  $M'$  to bude  $H_{M'} = H' \cup \overline{H} \cup \underline{H} \cup H_T$ . Pokiaľ ide o stavy a abecedy, ich konštrukcia by mala byť z už uvedeného zrejماً.

2. Ku  $G' = (N', T, M', \underline{\sigma})$  zostrojíme ekvivalentný  $g$ -systém  $G''$  tak, že prerušíme všetky vnútorné cykly (teda tie, ktoré nie sú kopírovacie v počiatocnom alebo koncovom stave). Vonkajšou šípkou 1- $a$ -prekladača nazveme takú, ktorá buď vychádza z počiatocného stavu, alebo vchádza do akceptačného stavu (v našom prípade ide o stavy  $q'_0, q'_F$ ). Ostatné šípky nazveme vnútorné. Pri odstraňovaní cyklov budeme postupovať nasledovne:

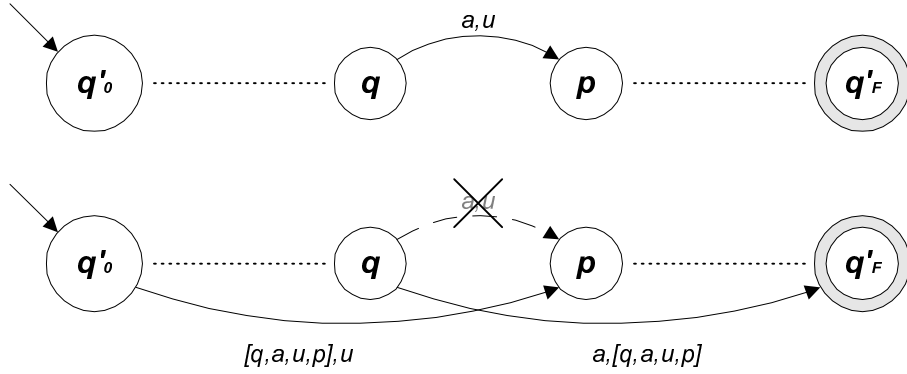
- (a) stavy budú rovnaké ako v  $G'$
- (b) všetky vonkajšie šípky necháme tak ako sú
- (c) každú vnútornú šípku nahradíme dvoma novými nasledovne: ak  $(q, a, u, p) \in H_{M'}$ , tak  $(q, a, u, p) \notin H_{M''}$ ,  $(q, a, [q, a, u, p], q'_F) \in H_{M''}$ ,  $(q'_0, [q, a, u, p], u, p) \in H_{M''}$ , pričom  $[q, a, u, p]$  budú nové neterminály<sup>5</sup> (obr.2.7)

Môže sa naskytnúť otázka: prečo sme nenahradzovali aj vonkajšie šípky? Odpoveď je veľmi jednoduchá: pretože ony žiadne cykly vytvárať nemohli, tak sme  $G'$  konštruovali. Malo by byť zrejmé, že takto sme odstránili všetky cykly, okrem kopírovacích v počiatocnom a koncovom stave, lebo teraz všetky šípky vychádzajú z  $q'_0$  alebo vchádzajú do  $q'_F$  a toto sú jediné šípky v  $G''$ . Rovnako zrejmé by malo byť  $L(G') = L(G'')$ , v  $G''$  sme iba nahradili jeden paralelný krok  $n$  sekvenčnými, kde  $n$  je počet šípok v jednom paralelnom kroku.

Máme zkonštruovaný  $g$ -systém  $G''$  taký, že  $L(G) = L(G'')$  a  $G'' \in \mathcal{S}$ , teda naše “zosekvenčovanie”  $g$ -systému  $G$  je na konci.  $\square$

Nasledujúce tvrdenia hovoria niečo o zložitosti  $g$ -systému ktorý sme práve zostrojili v predchádzajúcej konštrukcii a priamo z nej vyplývajú.

<sup>5</sup>Ak sa čitateľ trochu zamyslí, malo by mu byť jasné, že v každom kroku môže byť vo vetnej forme najviac jeden takýto neterminál



Obrázok 2.7: Nahrádzanie vnútorných šípok v  $G''$

**Veta 2.5.2.** (Vzťah popisnej zložitosti sekvenčných a paralelných  $g$ -systémov)

- $\forall L \text{ STATE}_S(L) \leq 3 \cdot \text{STATE}_G(L) + 3$
- $\text{ARC}_S(L) \leq 32 \cdot \text{STATE}_G(L) + 22 \cdot \#\Sigma_L$

Je dobré si uvedomiť, že zosekvenčením  $g$ -systému sme nepoužili žiadny priestor navyše, o čom hovorí nasledujúca veta.

**Veta 2.5.3.**  $\text{SPACE}_S(f(n)) = \text{SPACE}_G(f(n))$  pre  $\forall f(n)$

**Veta 2.5.4.** Pre ľubovoľný jazyk  $L$  a ľubovoľný  $G \in \mathcal{G}$  taký, že  $L = L(G)$  platí:

$$L \in \text{TIME}_S(\text{SPACE}_G(G, n) \cdot \text{TIME}_G(G, n))$$

Veta 2.5.4 hovorí o akomsi hornom odhade počtu krokov sekvenčných  $g$ -systémov v porovnaní s paralelnými. Na bližšie pochopenie tohto tvrdenia si stačí uvedomiť, že sekvenčný  $g$ -systém môže v jednom kroku vďaka tomu, že nemá vnútorné cykly, prepísať len istý, vo všeobecnosti malý, úsek vetnej formy na rozdiel od paralelného  $g$ -systému, ktorý môže v jednom kroku prepísať celú vetnú formu. Teda ak chce sekvenčný  $g$ -systém odsimulovať jeden krok paralelného  $g$ -systému, musí urobiť { rádovo dĺžka vetnej formy } krokov. Ak chceme sekvenčne odsimulovať celý výpočet paralelného  $g$ -systému, dostávame spomínané tvrdenie.

Keď sa na tento problém pozrieme z opačnej strany (keď chceme jazyk generovaný sekvenčným  $g$ -systémom, generovať paralelným  $g$ -systémom), podobná úvaha nám umožňuje pretransformovať dolný odhad počtu krokov pre sekvenčné  $g$ -systémy na dolný odhad počtu krokov pre paralelné  $g$ -systémy. Dostávame teda nasledujúce tvrdenie.

**Veta 2.5.5.**  $\overline{\text{TIME}}_{S_\varepsilon}(f(n)) \subseteq \overline{\text{TIME}}_{G_\varepsilon}(\frac{f(n)}{n})$

Toto tvrdenie nám ukazuje, že vzťah paralelných a sekvenčných  $g$ -systémov je lineárny a teda, že paralelizmus nám vo všeobecnosti veľmi nepomôže (aspoň nie nejak dramaticky), ako si však neskôr ukážeme, existujú tzv. "rýchlo generovateľné" jazyky, kde je rozdiel výraznejší.

**Príklad 2.5.1.** Je známe, že  $L = \{w c w \mid w \in \{a, b\}^*\} \in \overline{\text{TIME}}_{S_\varepsilon}(n^2)$ , teda podľa vety 2.5.5  $L \in \text{TIME}_{G_\varepsilon}(n)$ . Paralelizmus teda tomuto jazyku veľmi nepomôže, nie je totiž exponenciálny

## 2.6 Normálové tvary $g$ -systémov

**Definícia 2.6.1.** Hovoríme, že  $g$ -systém je v normálovom tvare, ak spĺňa nasledujúce podmienky:

1. existuje kopírovací cyklus pre  $\forall a \in N \cup T$  v  $q_0$  aj v  $q_F$
2. iba kopírovacie cykly vstupujú do  $q_0$  a vystupujú z  $q_F$
3. terminálne symboly sa iba kopírujú
4.  $g$ -systém je bez  $\varepsilon$

**Označenie:**  $\mathcal{N}$  je trieda všetkých  $g$ -systémov v normálnom tvare

**Veta 2.6.1.**  $\mathcal{L}_{\mathcal{N}} = \mathcal{L}_{\mathcal{G}_\varepsilon}$

**Dôkaz:** Inklúzia  $\subseteq$  je zrejmá vďaka štvrtej podmienke. Dokážeme opačnú inklúziu:

Nech  $G = (N, T, M, \sigma) \in \mathcal{G}_\varepsilon$ . Chceme zostrojiť  $G'' \in \mathcal{N}$  taký, že  $L(G'') = L(G)$ , teda chceme, aby  $G''$  spĺňal všetky štyri podmienky z definície  $\mathcal{N}$ . Keďže  $G \in \mathcal{G}_\varepsilon$  tak podmienka 4 je automaticky splnená.  $G''$  zostrojíme v dvoch krokoch:

1. Najskôr zostrojíme  $G' = (N', T, M', \sigma)$  taký, že  $L(G') = L(G)$  a  $G'$  bude spĺňať tretiu podmienku z definície  $\mathcal{N}$ :
  - $N' = N \cup \{\xi_a \mid a \in T\}$
  - $\forall a \in T$  : všetky výskyty terminálu  $a$  v  $H$  nahradíme novým neterminálom  $\xi_a$
  - do  $H'$  pridáme štvorce  $(q, \xi_a, u, p)$  ak  $(q, a, u, p) \in H$
2. Na  $G'$  použijeme konštrukciu podobnú s prvou časťou konštrukcie z kapitoly 2.5, ktorá nám zabezpečí splnenie podmienok 1 a 2. Všimnime si, že keďže pracujeme s bez  $\varepsilon$   $g$ -systémami, tak nemusíme stroj násobovať stavy, a teda vychádzajú aj lepšie odhady popisnej zložitosti.

□

**Dôsledok 2.6.1.** Pre každý jazyk  $L \in \mathcal{L}_{\mathcal{G}_\varepsilon}$  platí:

1.  $STATE_{\mathcal{N}}(L) \leq STATE_{\mathcal{G}_\varepsilon}(L) + 3$
2.  $ARC_{\mathcal{N}}(L) \leq 12 \cdot ARC_{\mathcal{G}_\varepsilon}(L) + 14 \cdot \#\Sigma_L$
3.  $SPACE_{\mathcal{N}}(n) = SPACE_{\mathcal{G}_\varepsilon}(n) = \mathcal{L}_{CS}$
4.  $TIME_{\mathcal{N}}(f(n)) = TIME_{\mathcal{G}_\varepsilon}(f(n))$

**Označenie:**  $STATE_{\mathcal{N}}(n) = \{L \mid \exists G \in \mathcal{N}, STATE(G) \leq n, L = L(G)\}$

**Veta 2.6.2.**  $STATE_{\mathcal{N}}(n)$  je  $\mathcal{AFL}$  pre<sup>6</sup>  $\forall n > 1$

**Označenie:**  $\mathcal{N}_{i,j}$  - trieda všetkých sekvenčných  $g$ -systémov v normálnom tvare, ktoré majú najviac  $i$  stavov a najviac  $j$  neterminálov, pričom sú dovolené  $\varepsilon$  prechody v 1- $a$ -prekladači.

<sup>6</sup>pre  $n = 1$  by boli problémy s  $h^{-1}$



**Veta 2.6.3.**  $\mathcal{L}_{\mathcal{N}_{6,2}} = \mathcal{L}_{\mathcal{N}_{5,3}} = \mathcal{L}_{\mathcal{N}_{4,4}} = \mathcal{L}_{RE}$

Pre  $\mathcal{N}_{6,2}$  tvrdenie platí, ak  $g$ -systém začína svoju prácu z počiatočného slova, otvoreným problémom je, či platí aj pri použití počiatočného neterminálu.

Dôsledkom tejto vety sú niektoré normálové tvary pre frázové gramatiky:

**Dôsledok 2.6.2.** K ľubovoľnému jazyku  $L \in \mathcal{L}_{RE}$  existuje taká frázová gramatika, že všetky jej pravidlá sú tvaru:

1.  $\sigma \rightarrow u$  alebo  $AB \rightarrow \varepsilon$  a  $CD \rightarrow \varepsilon$  kde  $N = \{\sigma, A, B, C, D\}$  a  $u \in (N \cup T)^*$
2.  $\sigma \rightarrow u$  alebo  $ABBBAA \rightarrow \varepsilon$  kde  $N = \{\sigma, A, B\}$  a  $u \in (N \cup T)^*$
3.  $\sigma \rightarrow u$  alebo  $ABC \rightarrow \varepsilon$  kde  $N = \{\sigma, A, B, C\}$  a  $u \in (N \cup T)^*$

Otvorenými problémami zostávajú:

- $\mathcal{N}_{5,2} \stackrel{?}{=} \mathcal{L}_{RE}$
- $\mathcal{N}_{4,2} \stackrel{?}{=} \mathcal{L}_{RE}$
- $\mathcal{N}_{4,3} \stackrel{?}{=} \mathcal{L}_{RE}$

## 2.7 Charakterizácia triedy $TIME_G(f(n))$ pomocou sekvenčného priestoru

**Definícia 2.7.1.** Nech  $D$  je odvodenie slova  $w \in L(G)$   $g$ -systémom  $G = (N, T, M, \sigma)$ , kde  $M = (K, \Sigma, \Sigma, H, q_0, q_F)$ . Strom odvodenia (ozn.  $T_D$ ) nazývame strom, ktorého vrcholy sú označené ako štvorce z  $H$ , pričom označenie koreňa  $T_D$  je tvaru  $(q_0, \sigma, u, q_F)$ , kde  $u \in (N \cup T)^*$ . Vrchol s označením  $(q, a, b_1 b_2 \dots b_k, p)$  má  $k$  synov  $(p_1, b_1, u_1, p_2)(p_2, b_2, u_2, p_3) \dots (p_k, b_k, u_k, p_{k+1})$ , kde  $\forall i u_i \in (N \cup T)^*$ . Postupnosť štvorcí na každej úrovni stromu je výpočet 1- $a$ -prekladača. Zreľazením tretích komponent štvorcí na  $k$ -tej úrovni dostaneme  $k$ -tu vetnú formu výpočtu  $G$ . Teda zreľazením tretích komponent štvorcí na poslednej úrovni<sup>7</sup> dostaneme slovo  $w$ .

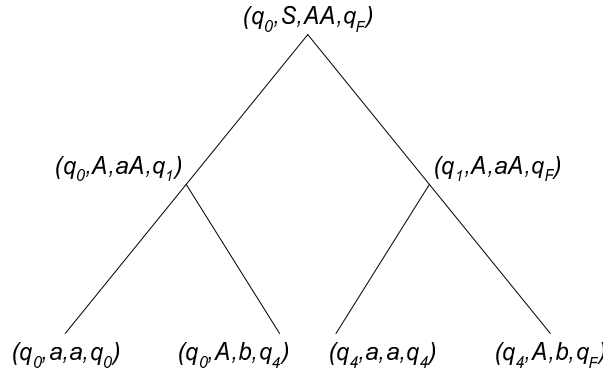
**Príklad 2.7.1.** Zoberme si  $g$ -systém a jeho 1- $a$ -prekladač pre jazyk  $L = \{ww \mid w \in \{a, b\}^*\}$  z príkladu 2.1.1. Zoberme si jedno konkrétne odvodenie  $S \Rightarrow AA \Rightarrow aAaA \Rightarrow abab$ . Jednotlivým krokom tohto odvodenia zodpovedá v 1- $a$ -prekladači výpočet:

- $S \Rightarrow AA \rightsquigarrow (q_0, S, AA, q_F)$
- $AA \Rightarrow aAaA \rightsquigarrow (q_0, A, aA, q_1)(q_1, A, aA, q_F)$
- $aAaA \Rightarrow abab \rightsquigarrow (q_0, a, a, q_0)(q_0, A, b, q_4)(q_4, a, a, q_4)(q_4, A, b, q_F)$

Strom tohto odvodenia je na obrázku 2.8.

---

<sup>7</sup>hĺbka stromu odvodenia  $\geq TIME(G, w)$



Obrázok 2.8: Strom zodpovedajúci odvodeniu  $S \Rightarrow AA \Rightarrow aAaA \Rightarrow abab$

**Lema 2.7.1.**  $TIME_G(f(n)) \subseteq 1NSPACE(f(n))$  pre  $\forall f(n)$ .

**Dôkaz:** Nech  $G = (N, T, M, \sigma)$  je  $g$ -systém pracujúci v čase  $O(f(n))$ . Chceme skonštruovať Turingov stroj  $A$  s jednosmernou vstupnou páskou, ktorý bude simulovať  $G$  v priestore  $O(f(n))$ . Uvažujme slovo  $w \in L(G)$  a jeho príslušný strom odvedenia  $T$ . Chceme ukázať, že  $A$  akceptuje  $w$  práve vtedy, keď  $w \in L(G)$ .  $A$  bude na svojej pracovnej páske zapisovať cesty z  $T$  vedúce od koreňa k listom (obr.2.9).  $A$  bude hádať tieto cesty a overovať, či jednotlivé štvorice na rovnakej úrovni  $T$  tvoria výpočet 1- $a$ -prekladača  $M$  a či zreťazenie výstupov štvoríc v listoch  $T$  tvoria  $w$ .

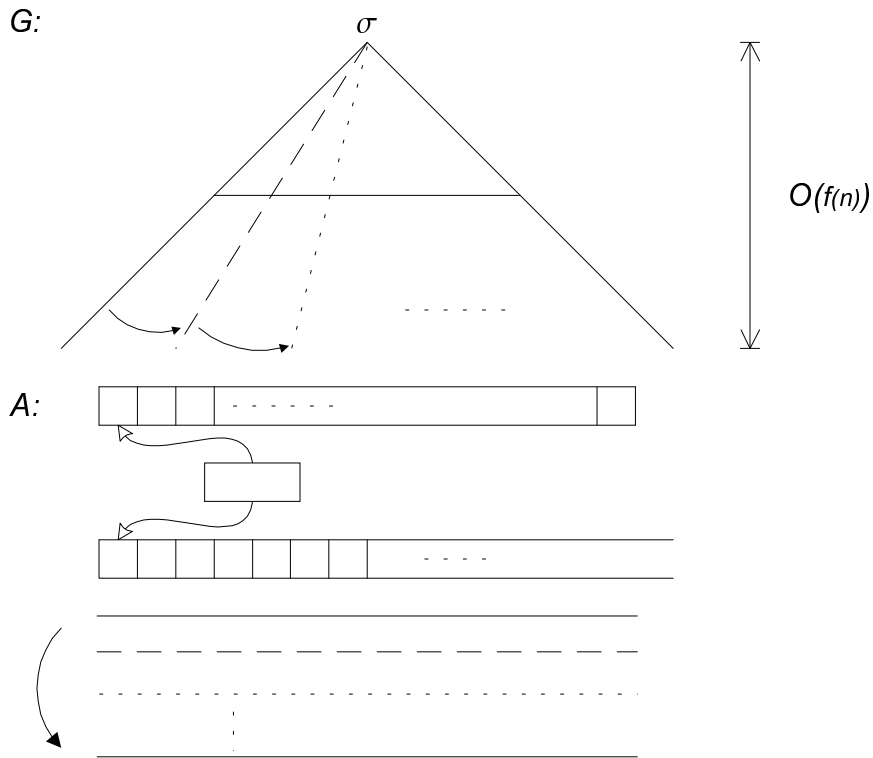
$A$  najprv uhádne a zapíše na pracovnú pásku cestu z koreňa do najľavejšieho listu  $\alpha$ , pričom musí skontrolovať či prvá štvorica je tvaru  $(q_0, \sigma, u, q_F)$  a ostatné štvorice musia začínať v stave  $q_0$  a prepisovať prvý symbol výstupu predchádzajúcej štvorice. Navyše výstup listu  $\alpha$  musí byť prefixom slova  $w$ . Ak nejaká z týchto podmienok neplatí, tak  $A$  neuhádol najľavejšiu cestu správne a zasekne sa. Ak  $A$  uhádol, tak nahradí štvoricu  $\alpha$  jeho najľavejším bratom  $\beta$  (to samozrejme tiež uhádne (obr.2.10a) a zaznačí si, že  $\beta$  je druhým synom ich spoločného otca  $\gamma$ .  $A$  overí, či  $\beta$  je dobre uhádnutý t.j. počiatkový stav  $\beta$  je rovnaký ako koncový stav  $\alpha$ ,  $\beta$  prepisuje druhý symbol výstupu  $\gamma$  a výstup  $\beta$  je rovnaký ako ďalšia časť  $w$  (bez prefixu, ktorý bol vo výstupe  $\alpha$ ). Takto  $A$  pokračuje až kým neuhádne a overí posledného syna  $\gamma$ . Potom  $A$  nahradí  $\gamma$  jej najľavejším bratom  $\delta$  (podobne ako bol  $\alpha$  nahradený  $\beta$ ). Teda  $A$  robí prehľadávanie do hĺbky, pričom si treba uvedomiť, že pri návrate na vyššiu úroveň v strome, si  $A$  musí pamätať koncové stavy jednotlivých vrcholov, aby mohol pri hádaní ďalších vrcholov overiť správnu následnosť.

Týmto spôsobom  $A$  pokračuje až kým vo výstupoch listov nenájde celé slovo  $w$ . Potom už  $A$  vie, že všetky ostatné neoverené listy musia mať na výstupe  $\varepsilon$  (obr.2.10b). Teda zvyšné cesty bude  $A$  hľadať s tým, že v listoch musí byť na výstupe  $\varepsilon$ . Keď  $A$  dosiahol najpravejší list (to je, ako inak, opäť uhádnuté), tak  $A$  overí či všetky koncové stavy vo všetkých štvoriciach na celej pracovnej páske sú  $q_F$ . Ak je to tak, potom  $A$  akceptuje  $w$ .

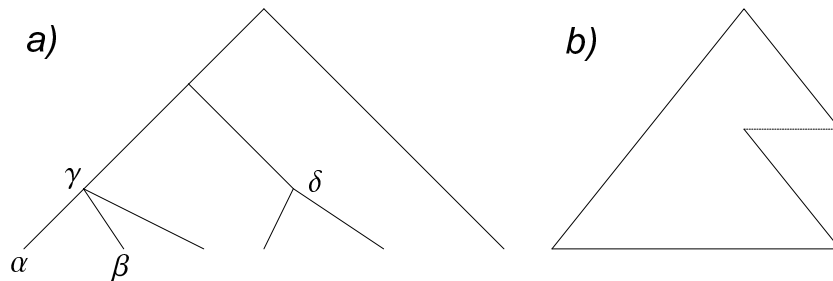
Z konštrukcie  $A$  je zrejme, že  $A$  akceptuje  $w$  práve vtedy, keď  $w \in L(G)$ . Keďže  $w \in L(G)$ , tak hĺbka stromu odvedenia nie je väčšia ako  $c \cdot f(|w|)$  kde  $c$  je konštanta nezávislá od  $w$ , teda hĺbka stromu odvedenia je  $O(f(n))$ , a teda  $A$  na akceptovanie slova  $w$  nepotrebuje viac políčok na páske ako  $O(f(n))$  z čoho konečne plynie, že  $L(G) \in 1NSPACE(f(n))$ .  $\square$

**Lema 2.7.2.**  $1NSPACE(f(n)) \subseteq TIME_G(f(n))$  pre  $f(n) = \Omega(\log n)$ .

**Dôkaz:** Nech  $A$  je nedeterministický Turingov stroj s jednosmernou vstupnou páskou, ktorý akceptuje v priestore  $O(f(n))$ . Bez ujmy na všeobecnosti predpokladajme, že  $A$  má len jednu jednosmerne nekonečnú pracovnú pásku.



Obrázok 2.9: Simulácia  $g$ -systému  $G$  Turingovým strojom  $A$



Obrázok 2.10: Nahrádzanie hrán v 1- $a$ -prekladači  $M$

Očíslujme políčka pracovnej pásky  $0,1,2 \dots$ . Chceme skonštruovať  $g$ -systém  $G$ , ktorý simuluje  $A$  v čase  $O(f(n))$ . Jedna vetná forma  $G$  obsahuje informáciu o políčku pracovnej pásky  $A$  počas celého výpočtu. Nasledujúca vetá forma obsahuje informáciu o nasledujúcom políčku atď. Keďže  $A$  pracuje na najviac  $O(f(n))$  políčkach, tak  $G$  potrebuje na vygenerovanie slova dĺžky  $n$  najviac  $O(f(n))$  vetných foriem (t.j.  $G$  pracuje v čase  $O(f(n))$ ). Samozrejme  $G$  musí zaručiť konzistentnosť medzi jednotlivými políčkami pásky, stavmi  $A$  a symbolmi na vstupnej páske podľa  $\delta$ -funkcie  $A$ .

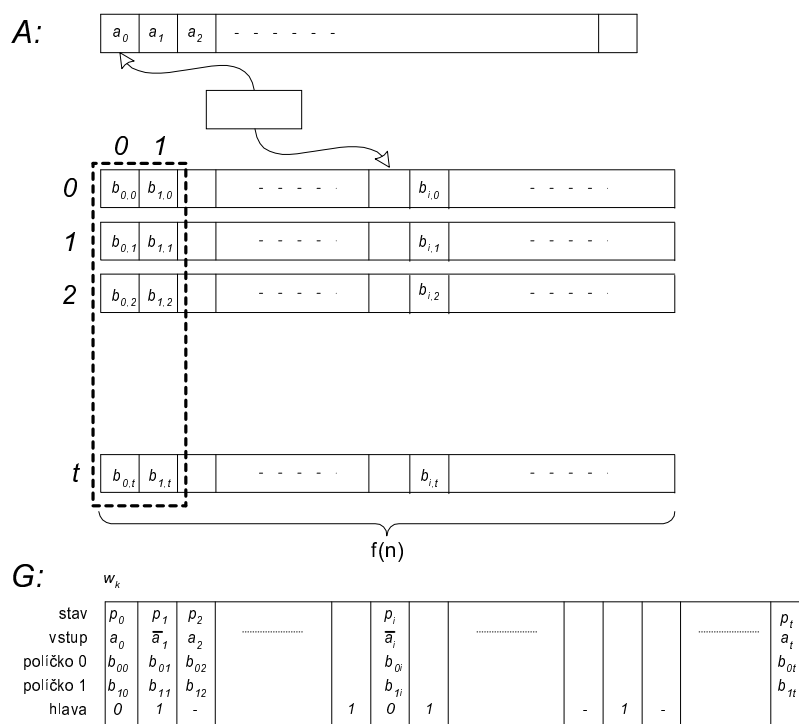
Uvažujme jeden výpočet  $A$  na vstupnom slove  $w \in L(A)$ . Nech  $s$  je priestor a  $t$  je čas potrebný na výpočet  $w$ . Odvodenie slova  $w$   $g$ -systémom  $G$  je tvaru:

$$\sigma = w_0 \xRightarrow{k} w_k \Rightarrow w'_k \Rightarrow w_{k+1} \Rightarrow w'_{k+1} \Rightarrow \dots \Rightarrow w_{k+s} \Rightarrow w'_{k+s} \Rightarrow w$$

kde vetná forma  $w_{k+j}$  drží obsah  $j$ -teho a  $j + 1$ . políčka  $A$  v celom výpočte  $A(w)$  a vetná forma  $w'_{k+j}$  je použitá na overenie či uhádnuté obsahy sú legálne vzhľadom na  $\delta$ -funkciu  $A$ .  
 $i$ -ty symbol vetnej formy  $w_{k+j}$  je päťposchodový symbol obsahujúci:

- stav  $p_i$ , v ktorom je  $A$  v čase  $i$  ( $p_{t+1}$  je akceptujúci)
- symbol  $a_i$ , ktorý v čase  $i$  číta vstupná hlava  $A$ , pričom tento symbol si označíme, ak v čase  $i$   $A$  posúva hlavu na vstupe
- symbol  $b_{j,i}$ , ktorý je na  $j$ -tom políčku pracovnej pásky  $A$  v čase  $i$
- symbol  $b_{j+1,i}$ , ktorý je na  $j + 1$ . políčku pracovnej pásky  $A$  v čase  $i$
- špeciálny symbol 0, ak hlava bola na  $j$ -tom políčku, špeciálny symbol 1, ak hlava bola na  $j + 1$ . políčku a v ostatných prípadoch špeciálny symbol -

V prvých  $k$ -krokoch  $G$  odvodí (uhádne<sup>8</sup>) vetnú formu  $w_k$  dĺžky  $t + 1$  t.j. postupnosť stavov, ktorými  $A$  prechádza počas výpočtu na  $w$ , vstupné slovo a časy, v ktorých  $A$  pohne hlavou na vstupe, obsahy nultého a prvého políčka pracovnej pásky  $A$  počas celého výpočtu a časy výskytu hlavy na nultom políčku pracovnej pásky. Schématicky vyzerá vetná forma ako na obrázku 2.11.



Obrázok 2.11: Simulácia TS  $A$   $g$ -systémom  $G$

V kroku  $k + 1$   $G$  overí či, to čo uhádol v predchádzajúcom kroku je v súlade s  $\delta$ -funkciou  $A$  a vygeneruje<sup>9</sup>  $w'_k$ . Ak to bolo uhádnuté dobre, tak  $G$  vygeneruje  $w_{k+1}$  t.j. prvé a druhé poschodie

<sup>8</sup>to sa dá na  $O(f(|w|))$  krokov keďže  $A$  pracuje v priestore  $O(f(|w|))$  a teda v čase  $O(|w| \cdot c^{f(|w|)})$  pre nejakú konštantu  $c$  t.j. počet všetkých možných konfigurácií  $A$  pri výpočte  $w$

<sup>9</sup>tento krok je naozaj iba overovací, to znamená, že ak  $G$  uhádol dobre, tak  $w'_k = w_k$  inak sa  $G$  zasekne

bude rovnaké ako v  $w'_k$ , štvrté poschodie  $w'_k$  bude vo  $w_{k+1}$  tretím poschodím, a kde bol v piatom poschodí  $w'_k$  špeciálny symbol 1, tam bude v piatom poschodí  $w_{k+1}$  špeciálny symbol 0, ostatné  $G$  uhádne<sup>10</sup> (obr.2.12). V ďalšom kroku  $G$  overí či to čo uhádol teraz je legálne vzhľadom na  $\delta$ -funkciu  $A$  a vygeneruje  $w'_{k+1} \dots$

Overovanie vo všeobecnosti vyzerá tak, že  $G$  akoby sa naraz pozeral na dva susedné päťposchodové symboly, takže vidí isté malé okolie (2 symboly) pracovnej pásky v istom malom časovom úseku (2 takty TS) a k zmenám na pracovnej páske hľadá príslušnú časť  $\delta$ -funkcie, ktorá je schopná takéto zmeny spôsobiť. Ak takúto časť  $\delta$ -funkcie  $A$  nájde, tak tieto dva päťposchodové symboly môžu stáť vedľa seba a  $G$  sa posunie o jeden päťposchodový symbol. Takto môže  $G$  overiť celú vetnú formu.

$G$  pokračuje v striedaní hádacích a overovacích krokov, až kým neodvodí vetnú formu bez špeciálnych symbolov výskytu hlavy na pracovnej páske  $A$  t.j. v piatom poschodí vetnej formy sa nevyskytuje 0 ani 1. To znamená, že  $G$  odsimuloval celý výpočet  $A$ .

**G:**

	$w_{k+1}$											
stav	$p_0$	$p_1$	$p_2$	.....		$p_i$	.....					$p_t$
vstup	$a_0$	$\bar{a}_1$	$a_2$	.....		$\bar{a}_i$	.....					$a_t$
políčko 1	$b_{10}$	$b_{11}$	$b_{12}$	.....		$b_{1i}$	.....					$b_{1t}$
políčko 2	$b_{20}$	$b_{21}$	$b_{22}$	.....		$b_{2i}$	.....					$b_{2t}$
hlava	-	0	1	.....		0	1	-	-	-	0	

Obrázok 2.12: Posun symbolov vo vetnej forme

V poslednom kroku  $G$  prepíše vetnú formu tak, že každý päťposchodový symbol sa prepíše na symbol z druhého poschodia (t.j. symbol zo vstupnej pásky  $A$ ) ak bol tento symbol označený, inak sa celý päťposchodový symbol prepíše na  $\varepsilon$ .

Zrejme  $w \in L(G) \iff$  ak  $w$  je akceptované  $A$ . Naviac  $w$  je vygenerované v čase  $O(f(n)) + O(2f(n)) = O(f(n))$ . Samozrejme dva kroky ododenia  $G w_l \Rightarrow w'_l \Rightarrow w_{l+1}$  sa dajú nahraďiť jedným, potom  $w$  je vygenerované v čase  $O(f(n)) + O(f(n)) = O(f(n))$ . Tento spôsob ododenia sme zvolili iba pre lepšiu čitateľnosť dôkazu.  $\square$

Predchádzajúce dve lemy nám umožňujú vysloviť nasledujúce tvrdenie:

**Veta 2.7.1.**  $TIME_G(f(n)) = 1NSPACE(f(n))$  pre  $f(n) \geq \log n$ .

Ak si uvedomíme, že pre  $f(n) \geq n$  je pracovná páska Turingovho stroja dosť dlhá, aby sme si na nej zapamätali vstup, potom  $1NSPACE(f(n)) = NSPACE(f(n))$ .

Ďalším jednoduchým pozorovaním zistíme, že ak  $f(n) \geq n$ , tak priestor na  $TS$  a na  $g$ -systémoch je ekvivalentný, teda  $NSPACE(f(n)) = SPACE_G(f(n))$ .

Môžeme teda vysloviť nasledovné dôsledky predchádzajúcej vety:

**Dôsledok 2.7.1.**  $TIME_G(f(n)) = NSPACE(f(n))$  pre  $f(n) \geq n$ .

**Dôsledok 2.7.2.**  $TIME_G(f(n)) = SPACE_G(f(n))$  pre  $f(n) \geq n$ .

<sup>10</sup>t.j.  $G$  uhádne symboly na druhom políčku pracovnej pásky  $A$  počas celého výpočtu a časy výskytu hlavy na prvom políčku

## 2.8 Niektoré vlastnosti “rýchlo generovateľných” jazykov

“Rýchlo generovateľné” jazyky nazývame také jazyky z triedy  $TIME_G(f(n))$ , pre ktoré  $f(n) < n$ .

**Veta 2.8.1.**  $TIME_G(\log^p n)$  je  $\mathcal{AFL}$  pre<sup>11</sup>  $p \geq 1$ .

**Veta 2.8.2.**  $TIME_G(\log^p n) \subsetneq TIME_G(\log^q n)$  pre  $q > p > 1$ .

**Dôsledok 2.8.1.** Hierarchia  $TIME_G(\log^p n)$  pre  $n > 1$  je nekonečná.

**Veta 2.8.3.** Pre každý jazyk  $L \in \mathcal{L}_{RE}$  existuje  $L' \in TIME_G(\log n)$  a existuje homomorfizmus  $h$  taký, že  $L = h(L')$ .

**Dôkaz:** Táto veta nám hovorí, že ku každému jazyku  $L \in \mathcal{L}_{RE}$  vieme nájsť  $g$ -systém  $G$  pracujúci v logaritmickej čase a homomorfizmus  $h$  taký, že  $h(L(G)) = L$ . Veta 2.2.1 nám hovorí, že vieme k  $L$  nájsť príslušný  $g$ -systém  $G$ , ale nezaručuje, že bude pracovať v logaritmickej čase.  $G$  upravíme na  $G'$  nasledovne:

- Do 1- $a$ -prekladača  $G$  zavedieme nový terminál  $\gamma$
- Každú štvoricu tvaru  $(q_0, \sigma, u, p)$  nahradíme štvoricou  $(q_0, \sigma, \gamma u, p)$
- Pridáme štvoricu  $(q_0, \gamma, \gamma\gamma, q_0)$

Lahko vidno, že takto upravený  $g$ -systém  $G'$  generuje jazyk

$$L' = \{\gamma^{2^m} w \mid w \in L(G) \text{ a } m \text{ je počet krokov odvedenia } w \text{ v } G\}$$

Zamyslime sa teraz nad časovou zložitou  $G'$ . Zoberme si nejaké slovo  $u = w\gamma^{2^k} \in L'$  pre nejaké  $k$ . Zrejme  $|u| \geq 2^k$ , ale  $G'$  toto slovo vygeneruje v  $k$  krokoch, teda v logaritmickej čase, a teda  $L' \in TIME_G(\log n)$ . Homomorfizmus  $h$  zvolíme takto:

- $\forall a \in T : h(a) = a$
- $h(\gamma) = \varepsilon$

To znamená, že  $h$  vymaže všetky  $\gamma$  zo slov  $w \in L'$ , ktoré sme zaviedli  $g$ -systémom  $G'$ . Teda  $h(L') = L$ . Ak sa nad touto konštrukciou ešte raz zamyslíme, zistíme, že my sme  $g$ -systém nejakým spôsobom “nezrýchľovali”, ale to, že sme jazyk dostali do logaritmickej časovej zložitosti sme dosiahli tým, že dĺžku slov z tohto jazyka sme natoľko zväčšili, že pri zachovaní počtu krokov odvedenia bude tento jazyk vygenerovaný v logaritmickej čase vzhľadom na túto zväčšenú dĺžku slova.  $\square$

**Dôsledok 2.8.2.** Trieda  $TIME_G(f(n))$  nie je uzavretá na ľubovoľný homomorfizmus.

## 2.9 Záverom o $g$ -systémoch

Je vhodné si uvedomiť niekoľko významných faktov, ktoré nám model generatívnych systémov priniesol.

Pre známe paralelné gramatiky, ktoré dokážeme simulovať na  $g$ -systémoch, dostaneme priestorové ohraničenie najviac  $1NSPACE(f(n))$ . Podobne, ak navrhne nový paralelný model, ktorý vieme “tesne” simulovať na  $g$ -systémoch, dostaneme priestorové ohraničenie  $1NSPACE(f(n))$ . Navyše pre každý jazyk  $L \in 1NSPACE(f(n))$  existuje nejaký typ paralelnej gramatiky, ktorá  $L$  dokáže generovať v čase  $f(n)$ .

---

<sup>11</sup>rýchlejšie ako v logaritmickej čase  $g$ -systém nedokáže pracovať

## Kapitola 3

# Kooperujúce distribuované systémy gramatík (CDGS)

Dostávame sa k ďalšiemu paralelnému modelu pracujúcemu na báze gramatík. Ide konkrétne o systém gramatík nejakého typu (napr. často gramatiky Chomského hierarchie, my sa budeme zaoberať iba prípadom, keď sú jednotlivé gramatiky bezkontextové), ktoré pristupujú k spoločnej vetnej forme podľa istých pravidiel tak, že v jednom kroku s ňou pracuje iba jedna gramatika.

### 3.1 Definície a označenia

**Definícia 3.1.1.** CDGS je  $(n + 2)$ -tica  $\Gamma = (T, G_1, G_2, \dots, G_n, S)$ , kde  $\forall i G_i = (N_i, T_i, P_i)$  je "bezkontextová gramatika bez počiatočného symbolu".  $T \subseteq \bigcup_i T_i$ ,  $S \in \bigcup_i N_i$  je počiatočný symbol

**Definícia 3.1.2.** Nech  $\Gamma = (T, G_1, G_2, \dots, G_n, S)$ . Krok odvodenia je relácia  $\xrightarrow[\Gamma]{\leq k}$ , kde  $\xrightarrow[\Gamma]{\leq k} = \xrightarrow[\Gamma]{\leq k} i \in \{1, 2, \dots, n\}$  definovaná nasledovne:  $\xrightarrow[\Gamma]{\leq k} = \left( \bigcup_{j=1}^k \xrightarrow[\Gamma]{j} \right)$ , podobne definujeme aj:  $\xrightarrow[\Gamma]{\geq k}$ ,  $\xrightarrow[\Gamma]{=}$ . Definujeme  $x \xrightarrow[\Gamma]{\tilde{r}} y^1$ :  $x \xrightarrow[\Gamma]{t} y = x \xrightarrow[\Gamma]{t} y$   $i \in \{1, 2, \dots, n\}$  platí práve vtedy, ak:  $x \xrightarrow[\Gamma]{*} y$  a  $\nexists z \neq y : y \xrightarrow[\Gamma]{*} z$

**Definícia 3.1.3.** Nech  $f \in \{t, *, = 1, = 2, \dots, \leq 1, \leq 2, \dots, \geq 1, \geq 2, \dots\}$  a nech  $\Gamma$  je CDGS. Potom jazyk definovaný systémom pri spôsobe prepisovania  $f$  je

$$L_f(\Gamma) = \{w \in T^* \mid \exists r, i_1, i_2, \dots, i_r \ S \xrightarrow[\Gamma_{i_1}]{f} w_1 \xrightarrow[\Gamma_{i_2}]{f} w_2 \xrightarrow[\Gamma_{i_3}]{f} \dots \xrightarrow[\Gamma_{i_r}]{f} w_r \equiv w\}$$

**Príklad 3.1.1.**  $\Gamma = (\{a, b, c\}, G_1, G_2, S)$

$G_1 = (\{A, B\}, \{A', B', a, b, c\}, \{A \rightarrow aA'b, B \rightarrow cB', A \rightarrow ab, B \rightarrow c\})$  a

$G_2 = (\{S, S', A', B'\}, \{A, B\}, \{S \rightarrow S', S' \rightarrow AB, A' \rightarrow A, B' \rightarrow B\})$ , potom

$$L_{=1}(\Gamma) = L_*(\Gamma) = L_{\leq k}(\Gamma) = L_{\geq 1}(\Gamma) = L_t(\Gamma) = \{a^n b^n c^m \mid n, m \geq 1\}, k \geq 1$$

---

<sup>1</sup>ďalej budeme písať len  $x \xrightarrow[\Gamma]{t} y$

$$L_{=2}(\Gamma) = L_{\geq 2}(\Gamma) = \{a^n b^n c^n \mid n \geq 1\}$$

$$L_{=k}(\Gamma) = L_{\geq k}(\Gamma) = \emptyset \text{ pre } k \geq 3$$

**Príklad 3.1.2.**  $\Gamma = (\{a\}, G_1, G_2, G_3, S)$

$$G_1 = (\{S\}, \{A\}, \{S \rightarrow AA\})$$

$$G_2 = (\{A\}, \{S\}, \{A \rightarrow S\}) \text{ a}$$

$$G_3 = (\{A\}, \{a\}, \{A \rightarrow a\}). \text{ Potom } L_t(\Gamma) = \{a^{2^n} \mid n \geq 1\}$$

**Príklad 3.1.3.**  $\Gamma = (\{a, b, c\}, G_1, G_2, G_3, S)$

$$G_1 = (\{S, A, A'\}, \{a, b, c\}, \{S \rightarrow S, S \rightarrow AcA, A' \rightarrow A\})$$

$$G_2 = (\{S, A, A'\}, \{a, b, c\}, \{A \rightarrow aA', a \rightarrow a\}) \text{ a}$$

$$G_3 = (\{S, A, A'\}, \{a, b, c\}, \{A \rightarrow bA', A \rightarrow b\}). \text{ Potom } L_{=2}(\Gamma) = L_{\geq 2}(\Gamma) = \{wcv \mid w \in \{a, b\}^+\}$$

Skúmali sa viaceré možnosti voľby  $T$ :

**Definícia 3.1.4.** Akceptačný štýl definujeme nasledovne:

- *arb*  $T \subseteq \bigcup_i T_i$  (*arbitrary*)
- *ex*  $T = \bigcup_i T_i$  (*exactly*)
- *all*  $T = \bigcap_i T_i$
- *one*  $T = T_i$  pre nejaké  $i$

**Označenie:**  $f \in \{*, t, =, 1, =, 2, \dots, \leq 1, \leq 2, \dots, \geq 1, \geq 2, \dots\} = D$   $D' = \{*, =, 1, \geq 1, \leq 1, \leq 2, \dots\}^2$   $A \in \{\text{arb}, \text{ex}, \text{all}, \text{one}\}$ .  $(CD_n CF, f, A)$  označuje triedu s bezkontextovými komponentami (najviac  $n$ ) s akceptačným štýlom  $A$ .  $(CD_* CF, f, A)$  označuje triedu s ľubovoľným počtom bezkontextových komponent s akceptačným štýlom  $A$ .

## 3.2 Porovnanie akcečných štýlov

V tejto časti ukážeme, že je prakticky jedno, aký akceptačný štýl  $CDGS$  použije, genetatívnu silu to ani v najmenšom neovplyvní.

**Veta 3.2.1.**  $\mathcal{L}(CD_* CF, f, \text{arb}) = \mathcal{L}(CD_* CF, f, \text{ex}) = \mathcal{L}(CD_* CF, f, \text{all}) = \mathcal{L}(CD_* CF, f, \text{one})$

**Dôkaz:**

$$\mathcal{L}(CD_* CF, f, \text{arb}) = \mathcal{L}(CD_* CF, f, \text{all})$$

$\subseteq$ : Nech  $\Gamma \in \mathcal{L}(CD_* CF, f, \text{arb})$ . Zostrojíme ekvivalentnú  $\Gamma' \in \mathcal{L}(CD_* CF, f, \text{all})$ :  $\Gamma = (T, G_1, \dots, G_n, S)$   $\Gamma' = (T, G'_1, \dots, G'_n, G_{n+1}, S')$

(a)  $f = t$   $G'_i = (N'_i, T'_i, P'_i)$ , kde  $N'_i = \{A' \mid A \in N_i\}$ ,  $T'_i = \{a' \mid a \in T_i\} \cup T$  -potrebujeme dosiahnuť, aby  $T$  bolo prienikom  $T'_i$ -čiek - môžu tam byť nejaké navyše.  $P'_i = \{A' \rightarrow w' \mid A \rightarrow w \in P_i\}$ , kde  $w' = a'_1, \dots, a'_n$  ak  $w = a_1, \dots, a_n$ .  $G_{n+1} = (N_{n+1}, T_{n+1}, P_{n+1})$ , kde  $N_{n+1} = \{a' \mid a \in (\bigcup_i N_i \cup \bigcup_i T_i)\} \cup \{F\}$   $T_{n+1} = T^3$   $P_{n+1} = \{a' \rightarrow a \mid a \in T\} \cup \{a' \rightarrow F \mid a \notin T\} \cup \{F \rightarrow FF\}$

<sup>2</sup> $v$   $D'$  nie sú tie, ktoré nás nútia robiť viac ako 1 krok

<sup>3</sup>Týmto sme zabezpečili, že  $(\bigcap_i T'_i \cap T_{n+1}) = T$



(b)  $f \neq t$  Rovnaká konštrukcia ako v prípade (a), ale  $P_{n+1} = \{a' \rightarrow a' \mid a \in T\} \cup \{a' \rightarrow a \mid a \in T\}$

⊇: Táto inklúzia triviálne platí, lebo ak  $T = \bigcap_i T_i$ , tak potom aj  $T \subseteq \bigcup_i T_i$

$$\mathcal{L}(CD_*CF, f, arb) = \mathcal{L}(CD_*CF, f, ex)$$

⊆: Nech  $\Gamma \in \mathcal{L}(CD_*CF, f, arb)$ . Zostrojíme ekvivalentnú  $\Gamma' \in \mathcal{L}(CD_*CF, f, ex)$ :  $\Gamma = (T, G_1, \dots, G_n, S)$   $\Gamma' = (T, G'_1, \dots, G'_n, G_{n+1}, S')$

(a)  $f = t$   $G'_i = (N'_i, T'_i, P'_i)$ , kde  $N'_i = \{A' \mid A \in N_i\} \cup \{a' \mid a \in T_i\}$ ,  $T'_i = T$  - potrebujeme dosiahnuť, aby  $T$  bolo rovné zjednoteniu  $T'_i$ -čiek.  $P'_i = \{A' \rightarrow w' \mid A \rightarrow w \in P_i\}$ , kde  $w' = a'_1, \dots, a'_n$  ak  $w = a_1, \dots, a_n$ .  
 $G_{n+1} = (N_{n+1}, T_{n+1}, P_{n+1})$ , kde  $N_{n+1} = \{a' \mid a \in (\bigcup_i N_i \cup \bigcup_i T_i)\} \cup \{F\}$   $T_{n+1} = T$   $P_{n+1} = \{a' \rightarrow a \mid a \in T\} \cup \{a' \rightarrow F \mid a \notin T\} \cup \{F \rightarrow FF\}$

(b)  $f \neq t$  Rovnaká konštrukcia ako v prípade (a), ale  $P_{n+1} = \{a' \rightarrow a' \mid a \in T\} \cup \{a' \rightarrow a \mid a \in T\}$

⊇: Táto inklúzia triviálne platí, lebo ak  $T = \bigcup_i T_i$ , tak potom aj  $T \subseteq \bigcup_i T_i$

$$\mathcal{L}(CD_*CF, f, arb) = \mathcal{L}(CD_*CF, f, one)$$

⊆: Nech  $\Gamma \in \mathcal{L}(CD_*CF, f, arb)$ . Zostrojíme ekvivalentnú  $\Gamma' \in \mathcal{L}(CD_*CF, f, one)$ :  $\Gamma = (T, G_1, \dots, G_n, S)$   $\Gamma' = (T, G'_1, \dots, G'_n, G_{n+1}, S')$

$G'_i = (N'_i, T'_i, P'_i)$ , kde  $N'_i = N_i$ ,  $T'_i = T_i$ ,  $P'_i = P_i$ .  $G_{n+1} = (N_{n+1}, T_{n+1}, P_{n+1})$ , kde  $N_{n+1} = \emptyset$ ,  $T_{n+1} = T$  - gramatikou  $G_{n+1}$  sme dosiahli to, že určite existuje  $i$  také, že platí:  $T = T_i$  pre nejaké  $i$

⊇: Táto inklúzia triviálne platí, lebo ak  $T = T_i$  pre nejaké  $i$ , tak potom aj  $T \subseteq \bigcup_i T_i$

□

V ďalšej časti tejto kapitoly platí:  $A=all$  a nebudeme ho explicitne písať.

**Veta 3.2.2.** *Platia nasledovné tvrdenia:*

- $\mathcal{L}(CD_*CF, f) = \mathcal{L}_{CF} \quad \forall f \in D'$
- $\mathcal{L}_{CF} = \mathcal{L}(CD_1CF, f) \subsetneq \mathcal{L}(CD_2CF, f) \subseteq \mathcal{L}(CD_nCF, f) \subseteq \mathcal{L}(CD_*CF, f) \subseteq \mathcal{L}_{CFMatrix}$ <sup>4</sup>  
 $\forall f \in D - D', n \geq 3$
- $\mathcal{L}(CD_nCF, = k) \subseteq \mathcal{L}(CD_nCF, = s.k) \quad \forall k, n, s \geq 1$ <sup>5</sup>
- $\mathcal{L}(CD_nCF, \geq k) \subseteq \mathcal{L}(CD_nCF, \geq k+1) \quad \forall n, k \geq 1$
- $\mathcal{L}(CD_*CF, \geq) \subseteq \mathcal{L}(CD_*CF, =)$ , kde  $\mathcal{L}(CD_*CF, \geq) = \mathcal{L}(CD_*CF, \geq 1) \cup \mathcal{L}(CD_*CF, \geq 2) \cup \dots$  a  $\mathcal{L}(CD_*CF, =) = \mathcal{L}(CD_*CF, = 1) \cup \mathcal{L}(CD_*CF, = 2) \cup \dots$
- $\mathcal{L}_{CF} = \mathcal{L}(CD_1CF, t) = \mathcal{L}(CD_2CF, t) \subsetneq \mathcal{L}(CD_3CF, t) = \mathcal{L}(CD_*CF, t) = \mathcal{L}(ETOL)$ <sup>6</sup>

<sup>4</sup>Maticové bezkontextové gramatiky - istým spôsobom sa reguluje, akým spôsobom sa používajú pravidlá.  $P$ : množina -tíc; vyberieme jednu z nich a už musíme použiť všetky pravidlá, ktoré sú v nej

<sup>5</sup>toto tvrdenie sa nepodarilo doposiaľ dokázať všeobecnejšie, len pre násobky

<sup>6</sup>tabuľkové rozšírené 0L - systémy

**Dôkaz:** Za všetky len jeden príklad:  $\mathcal{L}(CD_*CF, t) \subseteq \mathcal{L}(CD_3CF, t)$ :

Toto tvrdenie hovorí o tom, že bezkontextové  $CDGS$  s ľubovoľným počtom komponent možno simulovať na  $CDGS$  s tromi komponentami. Celá simulácia funguje tak, že v jednej gramatike nového  $CDGS$  si držíme informáciu o všetkých gramatikách pôvodného  $CDGS$  a zvyšné dve gramatiky<sup>7</sup> slúžia na prepínanie medzi pôvodnými gramatikami.

Nech  $\Gamma \in \mathcal{L}(CD_*CF, t)$ . Zostrojíme  $\Gamma' \in \mathcal{L}(CD_3CF, t)$ .

- $\Gamma = (T, G_1, G_2, \dots, G_n, S)$ <sup>8</sup>, kde  $G_i = (N_i, T_i, P_i)$
- $\Gamma' = (T, G'_1, G'_2, G'_3, [S, 1])$

Jednotlivé gramatiky  $\Gamma'$  budú vyzeráť nasledovne:

- $G'_1 = (N'_1, T'_1, P'_1)$ , kde
  - $N'_1 = \{[A, i] \mid A \in N_i\}$
  - $T'_1 = \bigcup_{i=1}^n T_i$
  - $P'_1 = \{[A, i] \rightarrow [w', i] \mid A \rightarrow w \in P_i, 1 \leq i \leq n\}$

kde  $w'$  je vlastne  $w$ , ibaže všetky staré neterminály sú nahradené novými.

- $G'_2 = (N'_2, T'_2, P'_2)$ , kde
  - $N'_2 = \{[A, i] \mid A \in \bigcup_{j=1}^n N_j, i = 1, \dots, n\}$
  - $T'_2 = \emptyset$
  - $P'_2 = \{[A, i] \rightarrow [A, i + 1] \mid i \equiv 1(\text{mod } 2)\}$
- $G'_3 = (N'_3, T'_3, P'_3)$ , kde
  - $N'_3 = N'_2$
  - $T'_3 = \emptyset$
  - $P'_3 = \{[A, i] \rightarrow [A, i + 1] \mid i \equiv 0(\text{mod } 2)\} \cup \{[A, n] \rightarrow [A, 1] \mid [A, n] \in N'_3\}$

Malo by byť zrejmé, že  $L(\Gamma) = L(\Gamma')$   $\square$

**Príklad 3.2.1.**

$$\begin{array}{ll}
G_1: S \rightarrow aAB|... & [S, 1] \rightarrow a[A, 1][B, 1]|... \\
G_2: & \\
G_3: A \rightarrow bAS|... & [A, 3] \rightarrow b[A, 3][S, 3]|... \\
\\
S \xrightarrow{G_1} aAB \xrightarrow{G_3} abASB \Rightarrow \dots & [S, 1] \xrightarrow{G'_1} a[A, 1][B, 1] \xrightarrow{G'_2} a[A, 2][B, 1] \xrightarrow{G'_2} a[A, 2][B, 2] \xrightarrow{G'_3} \\
& \xrightarrow{G'_3} a[A, 3][B, 2] \xrightarrow{G'_3} a[A, 3][B, 3] \xrightarrow{G'_1} ab[A, 3][S, 3][B, 3] \Rightarrow \dots
\end{array}$$

<sup>7</sup>Musia byť dve, lebo keby sme mali iba jednu a keďže sa nachádzame v mode  $t$ , táto jedna gramatika by sa nám zacyklila

<sup>8</sup>Tu je nutné predpokladať, že  $n$  je párne. Ak by tomu tak nebolo, pridáme gramatiku, v ktorej bude  $P = \emptyset$

### 3.3 Niektoré otázky popisnej zložitosti

**Definícia 3.3.1.** *Definujeme miery:*

- $Var(\Gamma) = \#(\bigcup_i N_i)$  - počet neterminálov
- $Prod(\Gamma) = \sum_i \#P_i$  - suma počtu pravidiel
- $Symb(\Gamma) = \sum_i (\sum_{A \rightarrow w \in P_i} (|w| + 2))$

**Definícia 3.3.2.** *Pre miery  $M \in \{Var, Prod, Symb\}$  a triedu gramatík  $X$  a jazyk  $L$  definujeme:  $M_X(L) = \min\{M(\Gamma) \mid \Gamma \in X, L = L(\Gamma)\}$*

**Definícia 3.3.3.** *Pre mieru  $M$  a triedy gramatík  $X$  a  $Y$  a triedu jazykov  $\mathcal{L} = \mathcal{L}(X) \cap \mathcal{L}(Y)$  takú, že  $M_Y(L) \leq M_X(L) \forall L \in \mathcal{L}$  označíme:*

- $Y \stackrel{M}{=} X \Leftrightarrow M_Y(L) = M_X(L) \forall L \in \mathcal{L}$
- $Y \stackrel{M}{<}_1 X \Leftrightarrow \exists L \in \mathcal{L} M_Y(L) < M_X(L)$
- $Y \stackrel{M}{<}_2 X \Leftrightarrow \forall n \exists L_n \in \mathcal{L} M_X(L_n) - M_Y(L_n) > n$
- $Y \stackrel{M}{<}_3 X \Leftrightarrow \exists L_n \in \mathcal{L}, n \geq 1$  také, že  $\lim_{n \rightarrow \infty} \frac{M_Y(L_n)}{M_X(L_n)} = 0$
- $Y \stackrel{M}{<}_4 X \Leftrightarrow \exists p$  a  $\exists L_n \in \mathcal{L}, n \geq 1$  také, že  $M_X(L_n) > n$  a  $M_Y(L_n) \leq p$

**Veta 3.3.1.** *Porovnanie  $(CD_*CF, f, A)$  a  $CFG$  :*

	*	t	$\leq k$	$= k$	$\geq k$
VAR	=	$<_4$	=	$<_4$	$<_4$
PROD	=	$<_3$	=	$<_4$	$<_4$
SYMB	=	$<_3$	=	$<_3$	$<_3$

**Dôkaz:** Příklad:  $(CD_*CF, t) \stackrel{Var}{<}_4 CFG$  (VAR)

Uvažujme  $L_n = \bigcup_{i=1}^n b(a^i b)^+$

Potom  $Var_{CFG}(L_n) = n + 1$

$P_n = \{S_0 \rightarrow bS_i \mid 1 \leq i \leq n\} \cup \bigcup_{i=1}^n \{S_i \rightarrow a^i b S_i, S_i \rightarrow a^i b\}$

S menej neterminálmi to neide, lebo pomiešaním pravidiel by sme dostali zlé slová.

$Var_{CD_*CF, t}(L_n) \leq 3$

$\Gamma = (\{a, b\}, G_1, G_2, \dots, G_{n+1}, S)$ , kde

$G_i = (\{A\}, \{a, b\}, \{A \rightarrow a^i b\}); 1 \leq i \leq n$

$G_{n+1} = (\{S, S', A\}, \{a, b\}, \{S \rightarrow bS', S' \rightarrow AS', S' \rightarrow A\})$  - táto gramatika pracuje ako prvá.  $\square$

Čitateľ si iste všimol, že sme tomuto modelu nevenovali veľa pozornosti. Je tomu tak z dôvodu jeho postavenia medzi teoretickými modelmi paralelizmu. V súčasnej dobe totiž nie sú známe o ňom žiadne dôležité výsledky. Jediné veci, ktoré sú ukázané, sú vzťahy medzi jednotlivými podtriedami.

## Kapitola 4

# Paralelné komunikujúce systémy gramatík (*PCGS*)

Dostávame sa k ďalšiemu typu paralelizmu. Paralelný komunikujúci systém gramatík v sebe integruje viacero gramatík nejakého typu, ktoré sú zosynchronizované podľa akýchsi globálnych hodín, takže pracujú v taktach. Označme si tieto gramatiky  $G_1, G_2, \dots, G_n$ . Každá z týchto gramatík pracuje na svojej vetnej forme podľa svojich pravidiel. Navyiac týmto gramatikám dodáme špeciálny neterminál  $Q$  (*query* symbol). Ak sa vo vetnej forme nejakej gramatiky vyskytne symbol  $Q_i$ , znamená to, že v ďalšom takte sa  $Q_i$  zmení na vetnú formu vygenerovanú gramatikou  $G_i$ , pričom  $G_i$  začne generovať odznova. Toto presunutie sa vykoná len vtedy, keď vetná forma  $G_i$  neobsahuje, žiadny symbol *query*. Teraz pristúpime k formálnemu zadefinovaniu tohto modelu.

### 4.1 Definície a označenia

**Definícia 4.1.1.** *PCGS (Parallel Communicating Grammar Systems) stupňa  $n$  je  $(n + 3)$ -ica  $\Gamma = (N, K, T, G_1, \dots, G_n)$  kde  $N$  je množina neterminálov,  $K$  je množina komunikačných (*query*) symbolov štandardne označovaných  $K = \{Q_1, \dots, Q_n\}$ ,  $T$  je množina terminálov, pre každé  $i$   $G_i = (N \cup K, T, P_i, S_i)$  sú bez- $\varepsilon$  gramatiky ľubovoľného typu<sup>1</sup>, pričom  $S_i \in N$  je počiatočný neterminál a v  $P_i$  nie sú pravidlá obsahujúce na ľavej strane *query*.*

**Označenie:**  $V_\Gamma = N \cup K \cup T$

**Definícia 4.1.2.**  *$n$ -vetná forma (konfigurácia) je  $n$ -tica slov  $(x_1, \dots, x_n)$  kde  $x_i \in V_\Gamma^*$ .*

**Definícia 4.1.3.** *Krok odvodenia je relácia  $\Rightarrow_\Gamma$  na  $n$ -vetných formách definovaná nasledovne:  $(x_1, \dots, x_n) \Rightarrow_\Gamma (y_1, \dots, y_n)$  práve vtedy keď nastane jeden z prípadov<sup>2</sup>:*

1.  $x_i$  neobsahuje *query* symbol a

- $x_i$  obsahuje neterminál, potom  $x_i \xrightarrow{G_i} y_i$
- $x_i$  je terminálne slovo, potom  $y_i = x_i$

<sup>1</sup>väčšinou sa používajú regulárne alebo bezkontextové typy gramatík, lebo pri zložitejších typoch už máme veľké problémy sledovať, čo taký systém vôbec robí

<sup>2</sup>prípád 1 nazývame prepisovací krok, prípad 2 nazývame komunikačný krok

2.  $x_i$  obsahuje query symboly  $Q_{j_1}, \dots, Q_{j_s}$  potom

- ak  $x_{j_k}$  neobsahuje query symbol, tak v  $x_i$  nahradíme  $Q_{j_k}$  vetnou formou  $x_{j_k}$  a  $y_{j_k} = S_{j_k}$
- ak  $x_{j_k}$  obsahuje nejaký query symbol tak v  $x_i$  necháme  $Q_{j_k}$

pre všetky ostatné  $x_i$  neobsahujúce query symboly platí  $y_i = x_i$

Práve vyslovená definícia je trochu komplikovaná, pretože neberie do úvahy nejaký konkrétny typ gramatiky, ale je použiteľná pre akýkoľvek typ gramatiky Chomského hierarchie. Keďže v ďalšom sa budeme zaoberať hlavne *PCGS* s regulárnymi komponentami, vyslovíme teraz definíciu kroku odvodenia pre tieto *PCGS*, ktorá je o niečo jednoduchšia.

**Definícia 4.1.4.** Krok odvodenia je relácia  $\xRightarrow{\Gamma}$  na  $n$ -vetných formách definovaná nasledovne:  $(x_1, \dots, x_n) \xRightarrow{\Gamma} (y_1, \dots, y_n)$  práve vtedy keď nastane jeden z prípadov:

1.  $x_i$  neobsahuje query symbol, teda

- $x_i = w_i A$  kde  $w_i \in T^*$  a  $A \in N$ , potom  $x_i \xRightarrow{G_i} y_i$
- $x_i = w_i$  je terminálne slovo, potom  $y_i = x_i$

2.  $x_i$  obsahuje query symbol, teda  $x_i = w_i Q_j$ , a potom

- ak  $x_j$  neobsahuje query symbol, tak  $y_i = w_i x_j$  a  $y_j = S_j$
- ak  $x_j$  obsahuje nejaký query symbol, tak  $y_i = x_i$

pre všetky ostatné  $x_i$  neobsahujúce query symboly platí  $y_i = x_i$

Uvedomme si, že odvodenie v *PCGS* pozostáva z prepisovacích a komunikačných krokov. Prepisovací krok nastane vtedy, ak sa v  $n$ -vetnej forme nevyskytuje ani jeden komunikačný symbol, a potom všetky gramatiky spravia jeden krok odvodenia na svojich vetných formách. Ak nejaká komponenta  $n$ -vetnej formy je terminálne slovo, tak sa v ďalšom nemení až kým nejaká gramatika nepožiadá o jej obsah. Ak sa v nejakej vetnej forme vyskytne neterminál, ktorý sa nedá prepísať, tak odvodenie sa zasekne. V komunikačnom kroku sa nahradia všetky *query* príslušnými vetnými formami, ak tie neobsahujú *query*. Je zrejmé, že komunikačných krokov môže po sebe nasledovať viac, až kým sa celá  $n$ -vetná forma "nevyčistí" od *query*. Môže sa stať, že komunikácia sa zacyklí a nebude možné vykonať žiaden prepisovací krok. Vtedy sa odvodenie zasekne.

**Definícia 4.1.5.** Jazyk generovaný *PCGS* systémom  $\Gamma$  je množina terminálnych slov vygenerovaných gramatikou  $G_1$ . Teda  $L(\Gamma) = \{x \in T^* \mid (S_1, \dots, S_n) \xRightarrow{\Gamma} (x, v_2, \dots, v_n), v_i \in V_{\Gamma}^*\}$

## 4.2 Parametre uvažované na *PCGS*

1. komunikačná štruktúra - Môžeme si predstaviť orientovaný graf, ktorého vrcholy sú gramatiky a hrana z  $G_i$  vedie do  $G_j$  ak  $G_i$  môže generovať  $Q_j$ . Komunikačné štruktúry delíme na dva základné typy:

centralizované - *query* môže generovať len gramatika  $G_1$

necentralizované - všetky ostatné napr. dag(directed acyclic graph), tree, two-way array, one-way array, two-way ring, one-way ring, ...

2. typ gramatík v komponentách
3. počet komponent (gramatík)
4. počet komunikačných krokov
5. systémy s resetom resp. bez resetu - t.j. či sa vetná forma po presunutí svojho obsahu do inej vetnej formy zmení na počiatočný neterminál alebo nie

**Označenie:**  $xPCGS_nX$  - trieda  $PCGS$ -systémov kde  $x \in \{centr, tree, dag, \dots\}$  je typ komunikačnej štruktúry<sup>3</sup>,  $n$  je počet komponent<sup>4</sup> (gramatík) a  $X \in \{REG, LIN, CF, \dots\}$  je typ komponent<sup>5</sup>

### 4.3 Generatívna sila $PCGS$

Na bližšie pochopenie sily  $PCGS$  si uvedieme najskôr zopár príkladov

**Príklad 4.3.1.**  $\Gamma_1 = (\{\sigma_1, \sigma_2, \sigma_3\}, K, \{a, b, c\}, G_1, G_2, G_3)$  kde množiny pravidiel príslušných gramatík sú:

$$P_1 = \{\sigma_1 \rightarrow a\sigma_1, \sigma_1 \rightarrow aQ_2, \sigma_2 \rightarrow bQ_3, \sigma_3 \rightarrow c\}$$

$$P_2 = \{\sigma_2 \rightarrow b\sigma_2\}$$

$$P_3 = \{\sigma_3 \rightarrow c\sigma_3\}$$

Skúsme si napísať pár krokov odvodenia:

$$\begin{aligned} (\sigma_1, \sigma_2, \sigma_3) &\Rightarrow (a\sigma_1, b\sigma_2, c\sigma_3) \Rightarrow \dots k - \text{krokov} \dots \Rightarrow (a^k\sigma_1, b^k\sigma_2, c^k\sigma_3) \Rightarrow \\ &(a^{k+1}Q_2, b^{k+1}\sigma_2, c^{k+1}\sigma_3) \Rightarrow (a^{k+1}b^{k+1}\sigma_2, \sigma_2, c^{k+1}\sigma_3) \Rightarrow (a^{k+1}b^{k+2}Q_3, b\sigma_2, c^{k+2}\sigma_3) \Rightarrow \\ &(a^{k+1}b^{k+2}c^{k+2}\sigma_3, b\sigma_2, \sigma_3) \Rightarrow (a^{k+1}b^{k+2}c^{k+3}, b^2\sigma_2, c\sigma_3) \end{aligned}$$

Teraz už vidíme, že  $L(\Gamma_1) = \{a^n b^{n+1} c^{n+2} \mid n \geq 1\}$ , a teda  $PCGS$  s troma regulárnymi komponentami vygeneroval jazyk, ktorý nie je regulárny, ba dokonca ani bezkontextový

**Príklad 4.3.2.**  $\Gamma_2 = (\{\sigma_1, \sigma_2\}, K, \{a, b, c\}, G_1, G_2)$  kde

$$P_1 = \{\sigma_1 \rightarrow a\sigma_1, \sigma_1 \rightarrow aQ_2, \sigma_2 \rightarrow a\sigma_1, \sigma_2 \rightarrow c\}$$

$$P_2 = \{\sigma_2 \rightarrow b\sigma_2\}$$

Napišme si pár krokov odvodenia:

$$\begin{aligned} (\sigma_1, \sigma_2) &\stackrel{*}{\Rightarrow} (a_1^k\sigma_1, b_1^k\sigma_2) \Rightarrow (a^{k+1}Q_2, b^{k+1}\sigma_2) \Rightarrow (a^{k+1}b^{k+1}\sigma_2, \sigma_2) \Rightarrow \\ &(a^{k+1}b^{k+1}a\sigma_1, b\sigma_2) \Rightarrow \dots \end{aligned}$$

Ľahko vidno, že  $L(\Gamma_2) = \{a^n b^n \mid n \geq 1\}(\{a^n b^n \mid n \geq 2\})^*c$ , čo je opäť jazyk, ktorý nie je ani bezkontextový

**Príklad 4.3.3.**  $\Gamma_3 = (\{\sigma_1, \sigma_2\}, K, \{a, b, c, d\}, G_1, G_2)$

$$P_1 = \{\sigma_1 \rightarrow c\sigma_1 d, \sigma_1 \rightarrow cQ_2 d\}$$

$$P_2 = \{\sigma_2 \rightarrow a\sigma_2 b, \sigma_2 \rightarrow ab\}$$

Všimnime si niektoré možné odvodenia:

1.  $(\sigma_1, \sigma_2) \stackrel{*}{\Rightarrow} (c^{k-1}\sigma_1 d^{k-1}, a^{k-1}\sigma_2 b^{k-1}) \Rightarrow (c^k Q_2 d^k, a^k \sigma_2 b^k) \Rightarrow (c^k a^k \sigma_2 b^k d^k, \sigma_2)$  a tu sa  $\Gamma_3$  zasekne
2.  $(\sigma_1, \sigma_2) \stackrel{*}{\Rightarrow} (c^{k-1}\sigma_1 d^{k-1}, a^{k-1}\sigma_2 b^{k-1}) \Rightarrow (c^k Q_2 d^k, a^k b^k) \Rightarrow (c^k a^k b^k d^k, \sigma_2)$

<sup>3</sup>ak nie je uvedený typ komunikačnej štruktúry, máme na mysli triedu všetkých  $PCGS$ -systémov bez ohľadu na štruktúru

<sup>4</sup> $n = *$  označuje ľubolný počet komponent

<sup>5</sup>lineárne gramatiky (LIN) sú bezkontextové gramatiky, ktoré majú na pravej strane pravidiel najviac jeden neterminál

$$3. (\sigma_1, \sigma_2) \xrightarrow{*} (c^{k-1}\sigma_1 d^{k-1}, a^{k-1}\sigma_2 b^{k-1}) \Rightarrow (c^k \sigma_1 d^k, a^k b^k) \xrightarrow{*} (c^{k+i} Q_2 d^{k+i}, a^k b^k) \Rightarrow (c^{k+i} a^k b^k d^{k+i}, a^k b^k)$$

Teda  $L(\Gamma_3) = \{c^l a^k b^k d^l \mid l \geq k \geq 1\}$ . *PCGS* s tromi lineárnymi komponentami vygeneroval jazyk mimo bezkontextovú triedu jazykov

**Príklad 4.3.4.**  $\Gamma_4 = (\{\sigma_1, \sigma_2\}, K, \{a, b, c\}, G_1, G_2)$

$$P_1 = \{\sigma_1 \rightarrow \sigma_1, \sigma_1 \rightarrow Q_2 c Q_2\}$$

$$P_2 = \{\sigma_2 \rightarrow a \sigma_2, \sigma_2 \rightarrow b \sigma_2, \sigma_2 \rightarrow a, \sigma_2 \rightarrow b\}$$

Ľahko vidno, že  $L(\Gamma_4) = \{w c w \mid w \in \{a, b\}^+\}$ , a teda *PCGS* s dvoma bezkontextovými komponentami vygeneroval jazyk, ktorý nie je bezkontextový

**Príklad 4.3.5.**  $\Gamma_5 = (\{\sigma_1, \sigma_2, \sigma_3\}, K, \{a, b, c, d\}, G_1, G_2, G_3)$

$$P_1 = \{\sigma_1 \rightarrow a \sigma_1, \sigma_2 \rightarrow a Q_2, \sigma_3 \rightarrow d\}$$

$$P_2 = \{\sigma_2 \rightarrow b \sigma_2, \sigma_2 \rightarrow b Q_3\}$$

$$P_3 = \{\sigma_3 \rightarrow c \sigma_3\}$$

Všimnime si odvodenie, v ktorom nasleduje po sebe viac komunikačných krokov:

$$(\sigma_1, \sigma_2, \sigma_3) \xrightarrow{*} (a^k \sigma_1, b^k \sigma_2, c^k \sigma_3) \Rightarrow (a^{k+1} Q_2, b^{k+1} Q_3, c^{k+1} \sigma_3) \Rightarrow (a^{k+1} Q_2, b^{k+1} c^{k+1} \sigma_3, \sigma_3) \Rightarrow (a^{k+1} b^{k+1} c^{k+1} \sigma_3, \sigma_2, \sigma_3) \Rightarrow (a^{k+1} b^{k+1} c^{k+1} d, b \sigma_2, c \sigma_3)$$

Ľahko vidno, že  $L(\Gamma_5) = \{a^n b^n c^n d \mid n \geq 1\} \in \mathcal{L}_{CS} - \mathcal{L}_{CF}$

**Príklad 4.3.6.** V tomto príklade ukážeme ako *PCGS* naplno využije silu svojej komunikácie a vyrobíme jazyk  $L(\Gamma) = \{w^{2^n} c \mid w \in \{a, b\}^*, n \geq 1\}$ :

$$\Gamma = (\{S_1, S_2, S_3\}, K, \{a, b\}, G_1, G_2, G_3)$$

$$P_1 = \{\sigma_1 \rightarrow a \sigma_1, \sigma_1 \rightarrow b \sigma_1, \sigma_1 \rightarrow Q_2, \omega_1 \rightarrow Q_3, \omega_2 \rightarrow \sigma'_1, \sigma'_1 \rightarrow c\}$$

$$P_2 = \{\sigma_2 \rightarrow \sigma_2, \sigma_2 \rightarrow Q_1, \sigma_1 \rightarrow \omega_1, \sigma'_1 \rightarrow \omega_1\}$$

$$P_3 = \{\sigma_3 \rightarrow \sigma_3, \sigma_3 \rightarrow Q_1, \sigma_1 \rightarrow \omega_1, \omega_1 \rightarrow \omega_2, \sigma'_1 \rightarrow \omega_1\}$$

Pozrime sa na to ako funguje odvodenie v  $\Gamma$ :

$$(\sigma_1, \sigma_2, \sigma_3) \xrightarrow{*} (w \sigma_1, \sigma_2, \sigma_3) \Rightarrow (w \sigma_1, Q_1, Q_1) \Rightarrow (\sigma_1, w \sigma_1, w \sigma_1) \Rightarrow (Q_2, w \omega_1, w \omega_1) \Rightarrow (w \omega_1, \sigma_2, w \omega_1) \Rightarrow (w Q_3, \sigma_2, w \omega_2) \Rightarrow (w w \omega_2, \sigma_2, \sigma_3) \Rightarrow (w w \sigma'_1, Q_1, Q_1) \xrightarrow{*} (w^4 \sigma'_1, Q_1, Q_1) \xrightarrow{*} (w^{2^n} \sigma'_1, \sigma_2, \sigma_3) \Rightarrow (w^{2^n} c, \dots)$$

Pravidlá v  $\Gamma$  sú zostavené tak dômyselne, že gramatiky  $G_2, G_3$  naraz vygenerujú komunikačný symbol  $Q_1$ , teda prenesú si vetnú formu vygenerovanú prvou gramatikou a následne  $G_1$  požiadajú o vetné formy  $G_2$  a  $G_3$ , teda obsah vetnej formy  $G_1$  sa zdvojnásobí. Ak by gramatiky takto “nespolupracovali”, tak sa  $\Gamma$  zasekne.

Všimnime si ešte, že  $\Gamma$  vygeneruje slovo  $w^{2^n} c$  v čase  $O(n)$ .  $G_1$  na začiatku vygeneruje  $w$  a potom pri každom zdvojnásobení  $\Gamma$  používa už len konštantný počet krokov odvodenia. Treba si uvedomiť, že je to možné len vďaka tomu, že pri modeli *PCGS* máme komunikáciu v podstate zadarmo, lebo v jednom kroku dokážeme preniesť ľubovoľne veľké slovo

**Veta 4.3.1.**  $\mathcal{L}(PCGS_n REG) - \mathcal{L}_{LIN} \neq \emptyset$  pre  $n \geq 2$

**Dôkaz:** pozri príklad 4.3.2  $\square$

**Veta 4.3.2.**  $\mathcal{L}(PCGS_n REG) - \mathcal{L}_{CF} \neq \emptyset$  pre  $n \geq 3$

**Dôkaz:** pozri príklad 4.3.1  $\square$

**Veta 4.3.3.**  $\mathcal{L}(PCGS_n LIN) - \mathcal{L}_{CF} \neq \emptyset$  pre  $n \geq 2$

**Dôkaz:** pozri príklad 4.3.3  $\square$

**Veta 4.3.4.**  $\mathcal{L}_{LIN} - \mathcal{L}(centr PCGS_* REG) \neq \emptyset$

**Dôkaz:** Uvažujme jazyk  $L = \{a^n b^m c b^m a^n \mid n, m \geq 1\}$

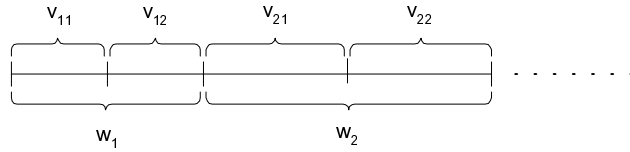
Zrejme  $L \in \mathcal{L}_{LIN}$ . Ukážeme, že  $L \notin \mathcal{L}(\text{centrPCGS}_*REG)$

Bez ujmy na všeobecnosti môžeme predpokladať, že  $G_1$  negeneruje  $a$ -čka, lebo ak by generovala tak to isté dokáže aj iná komponenta a  $G_1$  môže požiadať o jej výstup. Teda  $a$ -čka generujú dve iné gramatiky ( $G_2, G_3$ ), lebo potrebujeme rovnaký počet  $a$ -čiek na začiatku aj na konci vetnej formy. Podobne musia existovať aj dve gramatiky ( $G_4, G_5$ ) na generovanie  $b$ -čiek. Ak po nejakom počte krokov  $G_1$  vygeneruje  $Q_2$ , tak v konečnom (dost' malom) počte krokov musí vygenerovať aj  $Q_3$ , lebo  $G_3$  sa nemá ako dozvedieť, že už nemá generovať  $a$ -čka, keďže uvažujeme centralizovanú komunikačnú štruktúru. Z toho ale plynie, že v tomto konečnom počte krokov musí  $G_1$  vygenerovať  $Q_4$  a  $Q_5$ , lebo uvažujeme regulárne gramatiky. Teda  $G_4$  a  $G_5$  majú obmedzený čas na generovanie  $b$ -čiek, a teda počet  $b$ -čiek nemôže byť oveľa väčší<sup>6</sup> ako počet  $a$ -čiek, čo je spor s tým, že potrebujeme vygenerovať aj slová s ľubovoľne veľkým rozdielom medzi  $m$  a  $n$   $\square$

**Veta 4.3.5.**  $\mathcal{L}(\text{centrPCGS}_2REG) \subsetneq \mathcal{L}_{CF}$

**Dôkaz:** Podľa vety 4.3.4 stačí dokázať už len inklúziu  $\subseteq$ :

Majme  $\Gamma = (N, K, T, G_1, G_2) \in \text{centrPCGS}_2REG$ . Chceme zostrojiť bezkontextovú gramatiku  $G = (N', T, P, \sigma)$  takú, že  $L(\Gamma) = L(G)$ . Rozšírime množinu neterminálov takto:  $N' = N \cup \{[A, B] \mid A \in N, B \in N\} \cup \{\bar{A} \mid A \in N\}$ , neskôr si ukážeme ako tieto nové neterminály budeme využívať. Najskôr sa zamyslime nad tým, ako bude vyzeráť výsledné slovo  $w \in L(\Gamma)$ .  $w$  bude mať nasledujúce vlastnosti (obr.4.1):



Obrázok 4.1: Tvar slova generovaného  $\text{centrPCGS}_2REG$

1.  $w$  sa dá dekomponovať  $w = w_1 w_2 \dots w_s$  pre nejaké  $s \geq 1$
2.  $\forall i w_i = v_{i_1} v_{i_2}$  pričom  $v_{i_1}$  generuje  $G_1$  a  $v_{i_2}$  generuje  $G_2$
3.  $v_{i_1}$  a  $v_{i_2}$  sú generované na rovnaký počet krokov

$G$  musí byť schopná generovať slová s takýmito vlastnosťami. Na zabezpečenie podmienky 3 bude  $G$  generovať jednotlivé podslová  $w_i$  odstredu t.j. nejaký špeciálny neterminál sa bude postupne obklopovať terminálmi podľa pravidiel  $G_1, G_2$ .

V  $P$  budú pravidlá:

$$\sigma \rightarrow [\sigma_1, B]B \quad \forall B \in N$$

Teda máme špeciálnu sadu neterminálov  $[\sigma_1, B]$ , ktoré určujú, že podslovo  $v_{1_1}$  sa začína generovať z počiatočného neterminálu  $\sigma_1$  a podslovo  $v_{1_2}$  bude mať posledný neterminál  $B$ .

To v podstate znamená, že  $G$  sa nedeterministicky rozhodne pre neterminál  $B$

$$[A, B] \rightarrow a[A', B']b \quad \text{ak } A \rightarrow aA' \in P_1 \text{ a } B' \rightarrow bB \in P_2$$

Tieto pravidlá nám zabezpečujú postupné simulovanie odvodenia slov  $v_{i_1}$  odpredu a slov  $v_{i_2}$  odzadu, pričom toto odvodenie bude legálne vzhľadom na pravidlá  $G_1, G_2$

<sup>6</sup> $m$  môže byť väčšie od  $n$  najviac lineárne v závislosti od pravidiel v  $G_2, G_3$  a v  $G_4, G_5$



$[Q_2, \sigma_2] \rightarrow \varepsilon$

Toto pravidlo zabezpečuje, že ak  $G$  na začiatku dobre uhádla posledný neterminál  $v_{i_2}$ , tak po konečnom počte krokov simulácia  $G_1$  dospela ku komunikačnému neterminálu  $Q_2$  a simulácia  $G_2$  v spätnom odvodení dospela k počiatočnému neterminálu  $\sigma_2$ , teda neterminál  $[Q_2, \sigma_2]$  sa už nebude ďalej rozvíjať, a teda ho vymažeme

$A \rightarrow [A, B]B \forall A, B \in N$

Tieto pravidlá slúžia na správne nadväzovanie podslov  $w_i, w_{i+1}$ . To znamená, že týmto pravidlom  $G$  uhádne akým neterminálom bude končiť ďalšie podslovo

Ukážme si teraz ako funguje simulácia. Zoberme si nejaké odvodenie v  $\Gamma$ .

$(\sigma_1, \sigma_2) \Rightarrow (u_1 A_1, v_1 B_1) \xRightarrow{*} (u_1 u_2 \dots u_k Q_2, v_1 v_2 \dots v_k B_k) \Rightarrow (u_1 \dots u_k v_1 \dots v_k B_k, \sigma_2) \xRightarrow{*}$

Príslušná časť odvodenia v  $G$  bude vyzeráť takto:

$\sigma \Rightarrow [\sigma_1, B_k]B_k \Rightarrow u_1[A_1, B_{k-1}]v_k B_k \Rightarrow u_1 u_2[A_2, B_{k-2}]v_{k-1}v_k B_k \xRightarrow{*}$

$u_1 \dots u_k [Q_2, \sigma_2] v_1 \dots v_k B_k \Rightarrow u_1 \dots u_k v_1 \dots v_k B_k \xRightarrow{*}$

$G$  najskôr nedeterministicky vybrala pravidlo  $\sigma \rightarrow [\sigma_1, B_k]B_k$  tak, aby správne uhádla neterminál, ktorým bude ďalej pokračovať  $G_1$  v odvodení po komunikačnom kroku (t.j. neterminál, ktorý bol prenášaný v komunikačnom kroku). Potom postupne simulovala odvodenie  $G_1, G_2$  a nakoniec vymazala neterminál  $[Q_2, \sigma_2]$  z vetnej formy. Takto ostal vo vetnej formy len neterminál  $B_k$ , ktorým začne ďalšiu simuláciu  $G_1$ . Teraz  $G$  použije nejaké z pravidiel  $B_k \rightarrow [B_k, B]B$  na uhádnutie neterminálu, ktorý bude prenášaný v nasledujúcom komunikačnom kroku atď.

Ešte sa musíme zamyslieť nad tým, ako simuláciu ukončiť. Môžu nastať dva prípady:

1.  $G_1$  už nepožiadala  $G_2$  o jej vetnú formu (teda nevyprodukuje  $Q_2$ ) a sama vyprodukuje terminálne slovo. Pre túto možnosť dodáme do  $P$  tieto pravidlá:

- $B \rightarrow \bar{B} \forall B \in N$
- $\bar{A} \rightarrow u\bar{B}$  ak  $A \rightarrow uB \in P_1$

Tým, že sme použili “pruhované” neterminály a nemáme pravidlo typu  $\bar{A} \rightarrow A$  sme zmarili ďalšiu<sup>7</sup> komunikáciu, a teda simulujeme  $G_1$  až kým nevygeneruje terminálne slovo

2.  $G_2$  skončí skôr (t.j. vygeneruje terminálne slovo) ako  $G_1$  požiadala o jej vetnú formu. V  $\Gamma$  sa vetná forma  $G_2$  nemení a čaká, kým  $G_1$  vygeneruje  $Q_2$ . Pre túto možnosť dodáme do  $P$  tieto pravidlá:

- $A \rightarrow [A, \varepsilon] \forall A \in N$
- $[A, \varepsilon] \rightarrow u[A', \varepsilon]$  ak  $A \rightarrow uA' \in P_1$
- $[A, \varepsilon] \rightarrow u[A', B] \forall B \in N$  ak  $A \rightarrow uA' \in P_1$

Teda pravidlá typu  $[A, \varepsilon] \rightarrow u[A', \varepsilon]$  simulujú čakaknie  $G_2$  na komunikáciu a pravidlá typu  $[A, \varepsilon] \rightarrow u[A', B]$  opäť slúžia na uhádnutie posledného neterminálu, ktorý sa objavil vo vetnej forme  $G_2$

□

**Lema 4.3.1.** (*Pumpovacia lema pre centrPCGS<sub>n</sub>REG*)

Nech  $L \in \mathcal{L}(\text{centrPCGS}_n\text{REG})$ . Potom existuje prirodzené číslo  $N$ , že  $\forall w \in L$  také, že  $|w| \geq N \exists m : 1 \leq m \leq N$  a  $\exists x_i, y_i$  tak, že sú splnené nasledovné podmienky:

<sup>7</sup>k dispozícii máme samozrejme aj pravidlo  $\sigma \rightarrow \bar{\sigma}$  teda žiadna komunikácia nemusí nastať

1.  $w = x_1 y_1 x_2 y_2 \dots x_m y_m x_{m+1}$
2.  $y_i \neq \varepsilon$
3.  $\forall k \geq 0 : x_1 y_1^k x_2 y_2^k \dots x_m y_m^k x_{m+1} \in L$

**Dôkaz:** Nech  $\Gamma = (N, K, T, G_1, \dots, G_n) \in \text{centrPCGS}_n \text{REG}$ . Každá gramatika má vo svojej vetnej forme najviac jeden neterminál a ten je posledným symbolom tejto vetnej formy. Teda konfigurácia  $\Gamma$  je tvaru  $c = (x_1 A_1, x_2 A_2, \dots, x_n A_n)$ . Budeme hovoriť, že dve konfigurácie  $c_1 = (x_1 A_1, x_2 A_2, \dots, x_n A_n)$  a  $c_2 = (y_1 B_1, y_2 B_2, \dots, y_n B_n)$ , kde  $\forall i x_i, y_i \in T^+$  a  $\forall i A_i, B_i \in N \cup \{\varepsilon\}$  sú ekvivalentné (ozn.  $c_1 \equiv c_2$ ), ak  $\forall i : A_i = B_i$ .

Uvažujme slovo  $w \in L(\Gamma)$  a jeho odvodenie minimálnej dĺžky. Ak  $N$  je počet<sup>8</sup> všetkých možných rôznych výskytov neterminálov vo vetných formách, tak za predpokladu, že  $|w| > N$ , existujú v tomto odvodení dve konfigurácie  $c_1$  a  $c_2$  spĺňajúce nasledovné podmienky:

1.  $c_1 \equiv c_2$
2. ak je v odvodení medzi konfiguráciami  $c_1$  a  $c_2$  použitý komunikačný symbol  $Q_i$ ,  $2 \leq i \leq n$ , tak  $x_i = y_i$

Teda odvodenie je tvaru:

$$(S_1, S_2, \dots, S_n) \xrightarrow{*} (x_1 A_1, x_2 A_2, \dots, x_n A_n) \xrightarrow{*} (x_1 z_1 A_1, x_2 z_2 A_2, \dots, x_n z_n A_n) \xrightarrow{*} (w, \dots)$$

Ak je v odvodení medzi konfiguráciami  $c_1$  a  $c_2$  použitý komunikačný symbol  $Q_i$ ,  $2 \leq i \leq n$ , tak podľa podmienky (2) platí, že  $z_i = \varepsilon$ . Pre ostatné komponenty konfigurácie nastáva jedna z nasledujúcich možností:

1.  $z_1 \in T^+$  t.j.  $z_1$  je neprázdne terminálne slovo
2. existuje  $j$ ,  $2 \leq j \leq n$  také, že  $Q_j$  nie je použité v odvodení medzi konfiguráciami  $c_1$  a  $c_2$  ale je použité v odvodení, ktoré začína konfiguráciou  $c_2$ , navyiac  $z_j \in T^+$  teda  $z_j$  je neprázdne terminálne slovo

Predpokladajme, že ani jedna z týchto možností nenastane. Potom jednotlivé komponenty konfigurácií  $c_1$  a  $c_2$  sú totožné. Z toho vyplýva, že časť odvodenia ( $c_1 \xrightarrow{*} c_2$ ) medzi konfiguráciami  $c_1$  a  $c_2$  môžeme z odvodenia vynechať, čím dostaneme kratšie odvodenie slova  $w$ , čo je spor s predpokladom, že sme uvažovali najkratšie odvodenie  $w$ .

Teda jedna z uvedených možností určite nastane. V tomto prípade môžeme postupnosť krokov odvodenia medzi  $c_1$  a  $c_2$  opakovat'  $k$  krát pre ľubovoľnú<sup>9</sup>  $k$ . Po týchto  $k$  iteráciách sa komponenta  $j$ , pre ktorú  $z_j$  bolo neprázdne terminálne slovo, zmení na  $x_j z_j^k A_j$ . Ak po týchto  $k$  iteráciách dokončíme odvodenie časťou pôvodného odvodenia  $c_2 \xrightarrow{*} (w, \dots)$ , tak dostaneme nové slovo  $w' \in L(\Gamma)$ , ktoré sa od pôvodného  $w$  líši napumpovaním časti  $z_j$ .

Ak si uvedomíme, že počet podslov, ktoré môžu byť takto pumpované, môže byť najviac  $n$ , tak sme s dôkazom tejto lemy hotoví  $\square$

**Poznámka 4.3.1.** *Podobné pumpovacie lemy platia aj pre triedy  $\text{treePCGS}_n \text{REG}$  a  $\text{dagPCGS}_n \text{REG}$ , avšak pre všeobecnú necentralizovanú komunikačnú štruktúru  $\text{PCGS}$  sa pumpovať nedá*

**Veta 4.3.6.**  $\mathcal{L}(\text{centrPCGS}_{n-1} \text{REG}) \subsetneq \mathcal{L}(\text{centrPCGS}_n \text{REG})$  pre  $n \geq 2$

<sup>8</sup>ak  $K$  je množina stavov, tak za  $N = |K|^n$

<sup>9</sup>ak túto postupnosť krokov úplne vynecháme, tiež dostaneme legálne odvodenie v  $\Gamma$ , teda pripúšťame aj  $k = 0$

**Dôkaz:** Inklúzia  $\subseteq$  je zrejmalá.

Treba ukázať, že existuje jazyk  $L$  taký, že  $L \in \mathcal{L}(\text{centrPCGS}_n\text{REG})$  a  $L \notin \mathcal{L}(\text{centrPCGS}_{n-1}\text{REG})$ . Uvažujme jazyky  $L_n = \{a_1^{k+1}a_2^{k+2}a_3^{k+3} \dots a_n^{k+n} \mid k \geq 0\}$ . Zostrojme  $\text{centrPCGS}_n\text{REG}$   $\Gamma_n = (\{S_1, \dots, S_n\}, K, \{a_1, \dots, a_n\}, G_1, \dots, G_n)$  kde  $P_1 = \{S_1 \rightarrow a_1S_1, S_n \rightarrow a_n\} \cup \{S_i \rightarrow a_iQ_{i+1} \mid 2 \leq i \leq n-1\}$   
 $P_j = \{S_j \rightarrow a_jS_j\}$  pre  $2 \leq j \leq n$   
 Zrejme<sup>10</sup>  $L(\Gamma_n) = L_n$ . Chceme ukázať, že  $L_n \notin \mathcal{L}(\text{centrPCGS}_{n-1}\text{REG})$ . Sporom, predpokladáme, že  $L_n \in \mathcal{L}(\text{centrPCGS}_{n-1}\text{REG})$ . Podľa lemy 4.3.1 pre tento jazyk existuje prirodzené číslo  $N$ . Zoberme si slovo  $w = a_1^{N+1}a_2^{N+2}a_3^{N+3} \dots a_n^{N+n} \in L_n$ . Zjavne  $|w| \geq N$ . Slovo  $w$  môže pumpovať najviac na  $n-1$  miestach. Aby toto slovo po pumpovaní ostalo v  $L_n$  potrebujeme ho pumpovať na  $n$  miestach, a to je spor s predpokladom  $\square$

**Dôsledok 4.3.1.** Hierarchia  $\mathcal{L}(\text{centrPCGS}_n\text{REG})$ ,  $n \geq 1$  je nekonečná

Podobnými dokazovacími technikami (t.j. využitím pumpovacích lemm pre nejaký jazyk) by sme sa dopracovali k nasledujúcim dvom tvrdeniam:

**Veta 4.3.7.** Hierarchie nasledujúcich tried sú nekonečné:

- $\mathcal{L}(\text{treePCGS}_n\text{REG})$ ,  $n \geq 1$
- $\mathcal{L}(\text{dagPCGS}_n\text{REG})$ ,  $n \geq 1$

Hoci pre triedy  $PCGS$  s komunikačnými štruktúrami *array* a *ring* nie sú známe pumpovacie lemy, inými spôsobmi sa dajú dokázať nasledujúce tri tvrdenia:

**Veta 4.3.8.** Hierarchie nasledujúcich tried sú nekonečné:

- $\mathcal{L}(\text{two-way-arrayPCGS}_n\text{REG})$ ,  $n \geq 1$
- $\mathcal{L}(\text{two-way-ringPCGS}_n\text{REG})$ ,  $n \geq 1$
- $\mathcal{L}(\text{one-way-ringPCGS}_n\text{REG})$ ,  $n \geq 1$

**Veta 4.3.9.**  $\mathcal{L}(\text{PCGS}_{n-1}\text{REG}) \subsetneq \mathcal{L}(\text{PCGS}_n\text{REG})$  pre  $n \geq 2$

**Dôkaz:** Uvažujme jazyky  $L_n = \{a_1^k a_2^k \dots a_{2n-2}^k \mid k \geq 1\}$ . Dá sa skonštruovať  $\Gamma \in \text{PCGS}_n\text{REG}$  tak, že  $L(\Gamma) = L_n$ . Potom zrejme  $L_n \in \mathcal{L}(\text{PCGS}_n\text{REG})$ . Treba ukázať, že  $L_n \notin \mathcal{L}(\text{PCGS}_{n-1}\text{REG})$ . Tento dôkaz je však dosť technický, a preto ho tu nebudeme uvádzať<sup>11</sup>  
 $\square$

**Dôsledok 4.3.2.** Hierarchia  $\mathcal{L}(\text{PCGS}_n\text{REG})$ ,  $n \geq 1$  je nekonečná

**Poznámka 4.3.2.** Otvorenými problémami zostávajú nekonečnosť hierarchií  $\mathcal{L}(\text{centrPCGS}_nX)$ ,  $\mathcal{L}(\text{PCGS}_nX)$  pre  $n \geq 1$  a  $X \in \{CF, CS\}$

## 4.4 Porovnanie $PCGS$ so sekvenčnými triedami

**Veta 4.4.1.** Nech  $\Gamma$  je  $\text{treePCGS}_m\text{REG}(f(n))$ , kde  $f(n)$  je počet komunikačných krokov potrebných na vygenerovanie slova dĺžky  $n$ . Potom existuje<sup>12</sup> nedeterministický  $(m-1)$  počítadlový automat  $M$  pracujúci v lineárnom čase a akceptujúci jazyk  $L(\Gamma)$  pričom vykoná  $2f(n)$  obrátov<sup>13</sup> a  $f(n)$  testov na nulu<sup>14</sup>

<sup>10</sup>čitateľovi to môže byť zrejmejšie, ak sa pozrie na príklad 4.3.1

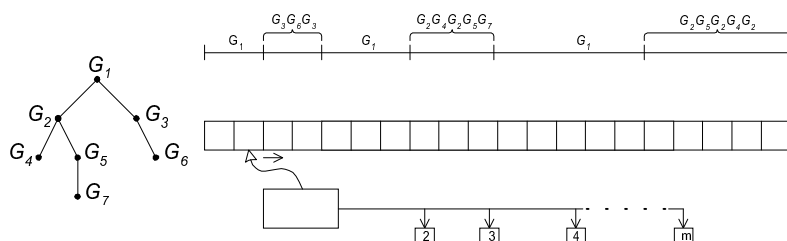
<sup>11</sup>podrobnosti možno nájsť v [5]

<sup>12</sup> $n$  počítadlový automat je zásobníkový automat s  $n$  zásobníkmi, ktoré pracujú nad jednopísmenkovou abecedou

<sup>13</sup>zmena smeru hlavy v zásobníku resp. zmena inkrementácie counteru na dekrementáciu a opačne

<sup>14</sup>dosiahnutie dna zásobníka resp. counter dosiahol nulu

**Dôkaz:** Nech  $\Gamma$  má komponenty  $G_1, \dots, G_m$ . Chceme zostrojiť nedeterministický  $(m - 1)$  počítadlový automat  $M$  simulujúci  $\Gamma$ . Celá simulácia  $\Gamma$  nám bude jasnejšia, ak sa zamyslíme nad tým, ako môže vyzeráť slovo  $w \in L(\Gamma)$  pri stromovej komunikačnej štruktúre. Slovo  $w = w_{i_1} w_{i_2} \dots w_{i_k}$ ,  $\forall j \ i_j \in \{1, \dots, m\}$  pozostáva z podslov  $w_{i_j}$  generovaných gramatikami  $G_{i_j}$ , pričom dĺžky týchto podslov závisia od konkrétnej komunikačnej štruktúry (obr.4.2).



Obrázok 4.2: Viacpočítadlový automat a tvar slova generovaného *treePCGS*

Vieme, že každá regulárna gramatika sa dá simulovať pomocou nedeterministického konečného automatu<sup>15</sup>. Podobne budú v pravidlách nášho automatu  $M$  simulované pravidlá regulárnych komponent  $G_1, \dots, G_m$ .  $M$  bude používať svoje počítadlá  $C_2, \dots, C_m$  na zabezpečenie toho, aby jednotlivé gramatiky  $G_1, \dots, G_m$  neboli simulované dlhšie ako to vyžaduje daná situácia (konfigurácia). V každej konfigurácii  $M$  bude číslo  $c(C_i) \forall i \in \{2, \dots, m\}$  uložené v  $C_i$  uchovávať rozdiel medzi počtom prepisovacích krokov  $G_i$  simulovaných  $M$  a počtom prepisovacích krokov otca  $G_i$  v stromovej komunikačnej štruktúre (to znamená, že ak  $G_i$  je požiadaná svojím otcom o vyprodukovanú vetnú formu, tak túto vetnú formu môže  $G_i$  generovať najviac  $c(C_i)$  krokov).  $M$  nedeterministicky striedavo simuluje gramatiky  $G_1, \dots, G_m$  a zároveň overuje, či slovo generované gramatikami je zhodné so slovom na vstupnej páske. Na začiatku  $M$  simuluje  $G_1$  a priebežne overuje vstupnú pásku. Zároveň po každom odsimulovanom prepisovacom kroku  $G_1$  inkrementuje počítadlá všetkým synom  $G_1$  a ostatné počítadlá zostanú nezmenené. Simulácia  $G_1$  skončí, keď  $G_1$  vygeneruje komunikačný symbol  $Q_i$  pre nejaké  $i$ .  $M$  začne simulovať gramatiku  $G_i$  z počiatočného neterminálu  $S_i$ . Počas tejto simulácie po každom prepisovacom kroku  $G_i$  sa dekrementuje počítadlo  $C_i$  a inkrementujú sa počítadlá všetkých synov  $G_i$ . Ak  $G_i$  vyprodukuje terminálnu vetnú formu, tak  $M$  zastaví a akceptuje vstupné slovo práve vtedy, keď bol vstup prečítaný až do konca. Ak  $C_i$  je prázdne ( $c(C_i) = 0$ ) a  $G_i$  v poslednom kroku vygeneroval na konci neterminál  $A$ , tak  $M$  začne simulovať otca  $G_i$  (v tomto prípade  $G_1$ ) pričom pokračuje v prepisovaní z neterminálu  $A$ . Ak  $G_1$  nemôže prepísať  $A$  (t.j. nemá pravidlá na  $A$ ), tak  $M$  neakceptuje vstupné slovo. Ak  $G_i$  vygeneruje komunikačný symbol  $Q_j$  pre nejaké  $j$ , tak  $M$  začne simulovať  $G_j$  (kde  $G_j$  je synom  $G_i$ ) z počiatočného neterminálu  $S_j$ .  $M$  od tohto momentu po každom prepisovacom kroku  $G_j$  dekrementuje  $C_j$  a inkrementuje počítadlá všetkých svojich synov.  $M$  takto rekurzívne simuluje gramatiky až kým posledná gramatika nevygeneruje terminálnu vetnú formu.

Z popisu práce  $M$  by malo byť zrejmé, že počet obrátov je  $2f(n)$  a počet testov na nulu je  $f(n)$ , lebo počítadlo  $C_i$  začne klesať práve vtedy, keď je vygenerovaný komunikačný symbol  $Q_i$ .

Ak sa v pravidlách gramatík nenachádzajú žiadne *chainrules*<sup>16</sup>, tak  $M$  pracuje v čase  $n$ . Ak sú v gramatikách takéto pravidlá, tak  $M$  pracuje v čase  $O(n)$ , pretože existuje konštanta  $d$  taká, že pre každé  $w \in L(\Gamma)$  existuje odvodenie, ktoré v každých  $d$  krokoch vygeneruje aspoň jeden terminál  $\square$

<sup>15</sup>pozri [1]

<sup>16</sup>pravidlá typu  $A \rightarrow B$  kde  $A, B$  sú neterminály

Uvedomme si, že  $m - 1$  počítadlový automat vieme simulovať viacpáskovým TS pracujúcim v rovnakom čase, a že počítadlá uchovávajú čísla veľkosti najviac  $n$ , takže v binárnom kódovaní sa zmestia do priestoru  $O(\log n)$ . Z týchto dvoch poznatkov plynie nasledujúce tvrdenie

**Veta 4.4.2.**  $\mathcal{L}(\text{treePCGS}_m \text{REG}(f(n))) \subseteq \text{NTIME}(n) \cap \text{NSPACE}(\log n)$

**Poznámka 4.4.1.** *Predchádzajúca veta nehovorí, že vieme zostrojiť ekvivalentný TS pracujúci v lineárnom čase a logaritmickom priestore. Hovorí len, že vieme zostrojiť ekvivalentný TS pracujúci v lineárnom čase a (iný) ekvivalentný TS pracujúci v logaritmickom priestore*

Skúsme sa zamyslieť nad simuláciou  $\text{PCGS}_m \text{REG}(f(n))$  priamočiaro pomocou  $m$ -páskového TS. Ten by na každej páске simuloval prepisovanie vetnej formy jednej gramatiky. Komunikácia medzi gramatikami v tomto prípade znamená presunutie obsahu jednej pásky na druhú. Keďže veľkosť pásky je  $O(n)$  a komunikácií je  $f(n) \in O(n)$ , tak časová zložitosť je  $O(n^2)$ . Pri takejto simulácii sa nám ľahko môže stať, že nejaký kúsok vetnej formy bol presúvaný medzi páskami  $O(n)$  krát a nakoniec sa aj tak nedostal do výsledného terminálneho slova. V nasledujúcej vete budeme toto “zbytočné” presúvanie eliminovať a dostaneme dobrý výsledok pre  $\text{dagPCGS}$

**Veta 4.4.3.**  $\mathcal{L}(\text{dagPCGS}_* \text{REG}) \subseteq \text{NTIME}(n)$

**Dôkaz:** Nech  $\Gamma = (N, K, T, G_1, \dots, G_m) \in \text{dagPCGS}_m \text{REG}$ . Chceme zostrojiť  $m$ -páskový<sup>17</sup> nedeterministický TS  $A$  pracujúci v lineárnom čase a akceptujúci jazyk  $L(\Gamma)$ .

V  $\delta$ -funkcii si  $A$  uchováva pravidlá všetkých  $m$  gramatík a v stavoch si drží aktuálne neterminály<sup>18</sup> všetkých gramatík. Na páskach  $T_1, \dots, T_m$  sú vetné formy generované gramatikami  $G_1, \dots, G_m$  alebo  $T_i$  obsahuje blank symbol  $B$ , ak sa  $A$  nedeterministicky rozhodne, že vetnú formu, ktorú generuje  $G_i$  sa neobjaví vo výslednom terminálnom slove generovanom  $\Gamma$  t.j. eliminovanie “zbytočných” presúvaní z jednej pásky na druhú.

Na začiatku bude  $A$  v stave so začiatočným neterminálom pre každú gramatiku. Pre každé  $i = 1..m$   $A$  uhádne, či sa vetná forma generovaná gramatikou  $G_i$  bude nachádzať vo výslednom terminálnom slove. Ak  $A$  rozhodne, že vetná forma generovaná  $G_i$  bude vo výslednom slove, tak  $A$  bude simulovať odvodenie gramatiky  $G_i$  na páске  $T_i$ . Ak  $A$  rozhodne, že nebude, tak  $A$  zapíše na pásku  $T_i$  symbol  $B$  a ďalej nesimuluje odvodenie  $G_i$  na páске  $T_i$ . Takto  $A$  simuluje iba to, čo sa vo výslednom slove naozaj objaví.

Jeden krok simulácie  $A$  pozostáva z jedného prepisovacieho kroku každej z gramatík, pre ktoré sa  $A$  rozhodol, že ich bude simulovať. To môžeme, lebo ako sme si povedali  $A$  si v stave uchováva aktuálne neterminály každej z gramatík. Takto  $A$  pokračuje, až kým sa na páске  $T_1$  neobjaví terminálne slovo, alebo až kým sa aspoň na jednej páске neobjaví *query* symbol.

Ak sa na  $T_1$  objaví terminálne slovo, tak  $A$  porovná obsah pásky  $T_1$  so vstupom a ak sa rovnajú, tak  $A$  akceptuje vstupné slovo.

Ak aktuálny neterminál  $G_i$  je  $Q_j$  a

- ani  $T_i$  ani  $T_j$  neobsahuje  $B$ , tak  $A$  prekopíruje obsah  $T_j$  na miesto  $Q_j$  na páске  $T_i$  a celý obsah pásky  $T_j$  prepíše na  $S_j$  (počiatočný neterminál gramatiky  $G_j$ ). Po tomto pokračuje  $A$  v simulácii  $G_i$  z neterminálu, ktorý bol posledným symbolom pri kopírovaní.  $A$  uhádne, či ďalšia vetná forma generovaná  $G_j$  bude časťou výsledného slova alebo nie. Podľa toho začne simulovať  $G_j$  z počiatočného neterminálu  $S_j$  alebo na  $T_j$  zapíše  $B$ .
- buď  $T_i$  alebo  $T_j$  obsahuje  $B$ , tak to znamená, že  $A$  uhádol nesprávne<sup>19</sup> a výpočet sa zasekne.

<sup>17</sup>to znamená, že TS  $A$  má  $m$  hláv

<sup>18</sup>keďže gramatiky sú regulárne, vo vetnej forme danej gramatiky sa vyskytuje najviac jeden neterminál

<sup>19</sup>Ak  $T_j$  obsahuje  $B$ , tak  $T_i$  sa objaví vo výsledku, a teda aj  $T_j$  bude vo výsledku, keďže si ju  $G_i$  vyžiadala, teda  $A$  zle uhádol. Ak  $T_i$  obsahuje  $B$ , tak  $T_j$  sa má objaviť vo výsledku, ale nemá sa ako dostať cez  $T_i$ , takže  $A$  zle uhádol

- $T_i$  aj  $T_j$  obsahuje  $B$ , tak  $A$  nezmení obsah  $T_i$  a nedeterministicky uhádne, či sa ďalšia vetná forma generovaná  $G_j$  objaví vo výslednom terminálnom slove. Podľa toho  $A$  začne na  $T_j$  simulovať  $G_j$  od počiatočného neterminálu  $S_j$  alebo na  $T_j$  nechá  $B$ .

Ak sa naraz na viacerých páskach objavia komunikačné symboly, tak  $A$  kopíruje vetné formy v určitom poradí tak, aby nekopíroval nejaký komunikačný symbol. To sa určite dá, pretože uvažujeme komunikačnú štruktúru *dag*, ktorá nepripúšťa cykly.

Pre každé slovo  $w \in L(\Gamma)$  existuje postupnosť správnych nedeterministických rozhodnutí TS  $A$ , ktorá vedie k odvodeniu tohto slova na páske  $T_1$ . Keďže komunikačná štruktúra nepripúšťa cykly, tak každý symbol akceptovaného slova  $w$  bol kopírovaný z jednej pásky na druhú najviac  $m - 1$  krát, čo je konštantne veľa. Teda na prekopírovanie všetkých symbolov slova  $w$  potrebujeme čas  $O(n)$ . Na páskach generujeme iba symboly, ktoré sa objavia vo výslednom slove  $w$ , teda na vygenerovanie všetkých symbolov slova  $w$  potrebujeme čas  $O(n)$ . Konečne na porovnanie terminálneho slova na páske  $T_1$  so vstupom potrebujeme čas  $O(n)$ . Teda celkovo pracuje  $A$  v čase  $O(n)$   $\square$

Uvedieme ešte niekoľko tvrdení, ktoré hovoria o tom, že ani zvýšenie počtu gramatík, ani zvýšenie počtu komunikačných liniek v stromovej štruktúre nedokáže vykompenzovať obmedzenie počtu komunikačných krokov:

**Veta 4.4.4.**  $\mathcal{L}(\text{centrPCGS}_2\text{REG}(n)) - \mathcal{L}(\text{treePCGS}_*\text{REG}(f(n))) \neq \emptyset$  pre  $f(n) \notin \Omega(n)$

**Dôkaz:** Uvedieme<sup>20</sup> len, že dôkaz uvažuje jazyk

$$L = \{a^{i_1}b^{i_1}a^{i_2}b^{i_2} \dots a^{i_k}b^{i_k}c \mid k \geq 1, i_j \geq 1 \text{ pre } j \in \{1, \dots, n\}\}$$

o ktorom sa ukáže, že  $L \in \mathcal{L}(\text{centrPCGS}_2\text{REG}(n))$  a  $L \notin \mathcal{L}(\text{treePCGS}_*\text{REG}(f(n)))$ . Využíva sa pri tom simulácia  $\text{treePCGS}_m\text{REG}(f(n))$  pomocou  $m - 1$  počítadlového automatu ako to bolo uvedené vo vete 4.4.1  $\square$

**Veta 4.4.5.** •  $\mathcal{L}(\text{one-way-arrayPCGS}_m(f(n))) \subsetneq \mathcal{L}(\text{one-way-arrayPCGS}_m(f(n)))$  pre  $m \geq 2$  a  $f(n) \notin \Omega(n)$

•  $\mathcal{L}(\text{centrPCGS}_m(f(n))) \subsetneq \mathcal{L}(\text{centrPCGS}_m(n))$  pre  $m \geq 2$  a  $f(n) \notin \Omega(n)$

•  $\mathcal{L}(\text{treePCGS}_m(f(n))) \subsetneq \mathcal{L}(\text{treePCGS}_m(n))$  pre  $m \geq 2$  a  $f(n) \notin \Omega(n)$

**Veta 4.4.6.**  $\mathcal{L}(\text{centrPCGS}_{k+1}(k)) - \mathcal{L}(\text{treePCGS}_*(k-1)) \neq \emptyset$  pre ľubovoľnú konštantu  $k$

**Veta 4.4.7.** Pre ľubovoľné  $k \geq 1$  a ľubovoľné  $x \in \{\text{centr}, \text{tree}, \text{one-way-array}\}$  platí:

•  $\mathcal{L}(x\text{PCGS}_{k+1}(k-1)) \subsetneq \mathcal{L}(x\text{PCGS}_{k+1}(k))$

•  $\mathcal{L}(x\text{PCGS}_*(k-1)) \subsetneq \mathcal{L}(x\text{PCGS}_*(k))$

## 4.5 Niektoré ďalšie vlastnosti PCGS

**Veta 4.5.1.**  $\mathcal{L}(\text{PCGS}_*CF)$  je úplná AFL

**Veta 4.5.2.**  $\text{centrPCGS}_*CF <_4^{\text{Var}} CF$

**Veta 4.5.3.**  $\text{centrPCGS}_*CF <_4^{\text{Prod}} CF$

---

<sup>20</sup>podrobnosti možno nájsť v [6]

# Kapitola 5

## Alternujúce stroje

V predchádzajúcich kapitolách sme si ukázali viaceré paralelné modely. I keď šlo o rôzne pohľady na paralelizmus, všetky mali jedno spoločné: pozerali sa na vec z “gramatikového” pohľadu, teda vo všetkých prípadoch sme mali jednu, prípadne viacero gramatík, ktoré čosi generovali. Teraz si ukážeme jednu automatovú charakterizáciu a porovnáme ju so sekvenčnými triedami. Pre čitateľa, ktorý sa s pojmom alternovania ešte nestretol, povedzme nasledovných pár viet. Modely alternujúcich strojov sa uvažujú pre všetky známe zariadenia, my si ich popíšeme na najvšeobecnejšom prípade, teda na Turingových strojoch, pre čitateľa iste nebude problémom analogicky si zdefinovať alternujúce konečné automaty, alternujúce zásobníkové automaty, prípadne alternujúce lineárne ohraničené automaty. O niektorých týchto zariadeniach si v závere kapitoly ukážeme pár zaujímavých vecí, napr. ako alternovanie ovplyvní, resp. neovplyvní ich generatívnu silu a podobne. Zariadenie (alternujúci stroj) má tzv. existenčné a tzv. univerzálne stavy, ktoré môže ľubovoľne kombinovať, a ktoré si ďalej bližšie popíšeme. Alternujúci Turingov stroj sa v existenčných stavoch správa rovnako, ako už známy nedeterministický model Turingovho stroja. Odlišné je správanie sa v univerzálnych stavoch. Zariadenie sa akoby rozdelí na  $n$  nových strojov (kde  $n$  je počet konfigurácií dosiahnuteľných na 1 krok z jeho momentálnej konfigurácie), ktoré dostanú novú pamäť (rovnakú ako pôvodný stroj) a ďalej nezávisle od seba pracujú, táto operácia sa nazýva FORK a čitateľovi môže byť trochu v reálnejšej podobe známa z niektorých operačných systémov (UNIX), kde jeden proces vytvára nové a prideluje im pamäť.

### 5.1 Definície a označenia

**Definícia 5.1.1.** *Alternujúci Turingov stroj (ATS) je 6-tica  $A = (K, \Sigma, \Gamma, \delta, q_0, F)$ , kde  $K$  je konečná množina stavov rozdelená na dve disjunktné časti  $K = K_1 \cup K_2$ ,  $K_1$  je množina existenčných stavov,  $K_2$  je množina univerzálnych stavov,  $\Sigma$  je abeceda vstupných symbolov,  $\Gamma$  je pracovná (pásková) abeceda,  $q_0$  je počiatočný stav a  $F$  je množina akceptačných stavov,  $\delta$  je prechodová funkcia*

$$\delta : K \times \Gamma \rightarrow 2^{K \times \Gamma \times \{-1, 0, 1\}}$$

Niektoré definície alternujúcich strojov uvažujú okrem už spomenutých existenčných a univerzálnych stavov ešte dva špeciálne stavy ACCEPT a REJECT, do ktorých sa môže stroj dostať iba na konci výpočtu pri akceptovaní vstupného slova, resp. pri jeho odmietnutí.

**Definícia 5.1.2.** *Konfiguráciu ATS  $A$  definujeme rovnako ako pre nedeterministické Turingove stroje (NTS), bližšie napr. [1]*

**Definícia 5.1.3.** *Krok výpočtu ATS  $A$  je relácia  $\vdash$  na konfiguráciách definovaná rovnako ako pre NTS*

Čitateľ si možno spomenie na pojem stromu konfigurácií, definovaný pre nedeterministický Turingov stroj, ktorý prehľadne reprezentoval jeho výpočet na konkrétnom slove<sup>1</sup>. Podobný strom definujeme aj pre alternujúce stroje. Okrem toho, že nám tento pojem pomôže pri definovaní výpočtu ATS, mal mi výraznou mierou prispieť k pochopeniu alternovania samotného.

**Definícia 5.1.4.** *Úplný strom konfigurácií pre ATS  $A$  a vstupné slovo  $w$  je strom, ktorého vrcholy sú označené konfiguráciami taký, že:*

1. *koreň je počiatočná konfigurácia na slove  $w$*
2. *každý vrchol má za priamych nasledovníkov práve tolko vrcholov, koľko konfigurácií je v relácii  $\vdash$  s konfiguráciou označujúcou daný vrchol*
3. *tieto vrcholy sú označené týmito konfiguráciami*

**Poznámka 5.1.1.** *Úplný strom konfigurácií nemusí byť konečný*

**Definícia 5.1.5.** *Výpočet ATS  $A$  (na slove  $w$ ) je podstrom úplného stromu konfigurácií na slove  $w$  taký, že:*

1. *obsahuje koreň*
2. *s každým univerzálnym vrcholom ( $q \in K_2$ ) obsahuje všetkých jeho priamych nasledovníkov*
3. *s každým existenčným vrcholom obsahuje práve jedného jeho priameho nasledovníka, ak existuje*

**Definícia 5.1.6.** *Akceptujúci výpočet ATS  $A$  na slove  $w$  je taký výpočet na  $w$ , ktorý je konečný a každá listová konfigurácia je akceptačná*

**Definícia 5.1.7.** *Jazyk akceptovaný ATS  $A$  je*

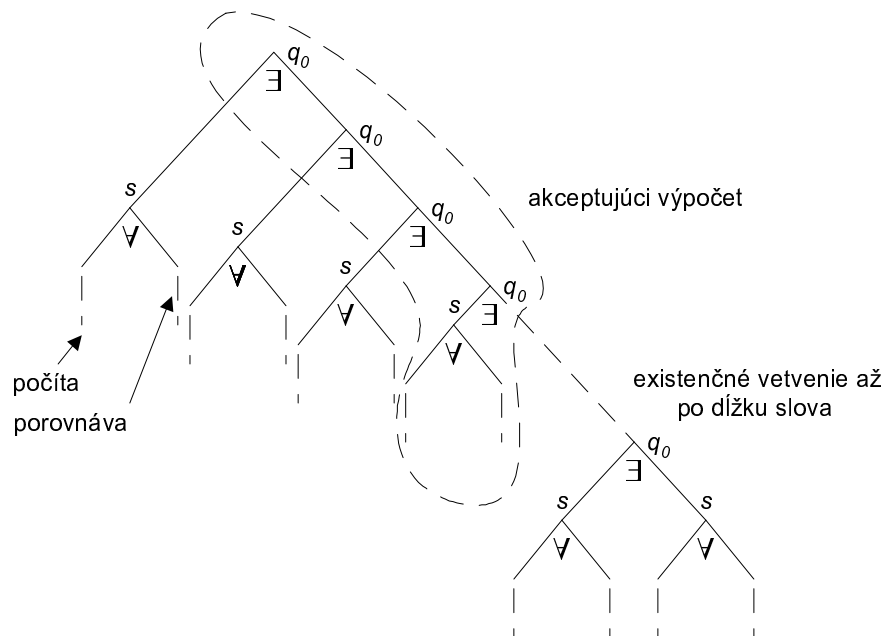
$$L(A) = \{w \in \Sigma^* \mid \text{existuje akceptačný výpočet } A \text{ na } w\}$$

**Príklad 5.1.1.** Ukážeme si, ako alternovanie pomôže dvojhlavým konečným automatom<sup>2</sup> v generatívnej sile. Zoberme si jazyk  $L = \{ww \mid w \in \{a, b\}^*\}$ . Dalo by sa ukázať, že ho nie je možné akceptovať dvojhlavým konečným automatom, no zostrojíme preň alternujúci dvojhlavý konečný automat, ktorý bude pracovať nasledovne:

1. dostane na vstup slovo, nedeterministicky uhádne jeho polovicu
2. nastane FORK, automat alternuje v univerzálnom stave
  - 1. proces overuje, či vstup je tvaru  $ww$  tak, že hlavy sa synchronne pohybujú po vstupe, v jednom kroku každá číta symbol, ak obe prečítajú to isté a keď sa jedna z nich dostane na koniec vstupu, dôjde k akceptovaniu
  - 2. proces overuje, či automat uhádol polovicu správne tak, že prvá hlava sa hýbe o jedno políčko vpravo, zatiaľ čo za ten istý čas sa druhá hlava hýbe o dve políčka vpravo, k akceptovaniu dôjde, ak obe hlavy dočítajú naraz vstupné slovo

Výpočet automatu možno vidieť na (obr.5.1).





Obrázok 5.1: Úplný strom konfigurácií automatu pre jazyk  $L = \{ww \mid w \in \{a, b\}^*\}$  z príkladu 5.1.1

**Veta 5.1.1.**  $\mathcal{L}_{ATS} = \mathcal{L}_{RE}$

**Dôkaz:** Povieme si pár slov o oboch inklúziách:

⊆: vyplýva z Turingovej tézy, keby sme mali zkonštruovať TS, ktorý by simuloval ATS, tak by sme zrejme prechádzali strom konfigurácií ATS do šírky, pamätali si niečo o výpočte a charaktere stavov a podobne

⊇: na NTS sa možno pozerať aj ako na špeciálny prípad ATS, keď akoby všetky stavy NTS boli existenčné

□

## 5.2 Miery zložitosti

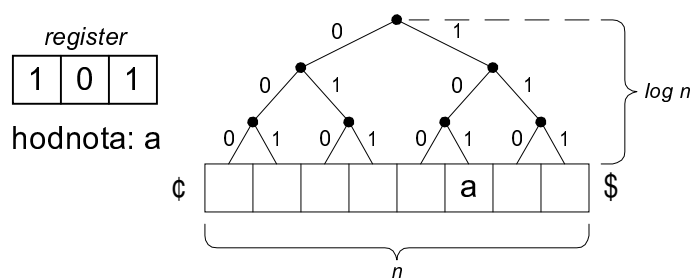
Pre ATS si zdefinujeme dve miery, ktoré sú analógiou mier  $NSPACE$ ,  $NTIME$  definovaných pre NTS:

- $ASPACE$  = najväčší počet použitých políčok na páske v niektorej konfigurácii akceptačného výpočtu
- $ATIME$  = hĺbka akceptačného výpočtu v strome konfigurácií

<sup>1</sup>bol napr. dobrou pomôckou pri dôkaze ekvivalencie deterministických a nedeterministických Turingových strojov

<sup>2</sup>budeme uvažovať zariadenie s možnosťou pohybu hlavy iba jedným smerom

V niektorých simuláciách budeme pracovať s triedou jazykov  $ATIME(f(n))$ ,  $\log n < f(n) < n$ . Klasický model ATS by sme tu nemohli použiť, lebo na prečítanie vstupu dĺžky  $n$  potrebujeme aspoň  $n$  krokov. V takýchto prípadoch sa používa model ATS s rýchlym prístupom na vstup. Môžeme si ho reprezentovať nasledovne: nad páskou má binárny vyhľadávací strom, každý list je práve jedno políčko, strom má výšku  $\log n$ , máme register dĺžky  $\log n$ , keď do neho binárne zapíšeme číslo  $k$ , tak za čas  $\log n$  sa dostaneme ku  $k$ -temu vstupnému políčku (obr.5.2).



Obrázok 5.2: Model ATS s rýchlym prístupom na vstup

### 5.3 Alternujúce vs. sekvenčné triedy zložitosti

Keďže vo väčšine simulácií budeme požadovať predpoklad páskovej konštruovateľnosti nejakej funkcie, bude dobré, keď si tento pojem zdefinujeme.

**Definícia 5.3.1.** *Funkcia  $f(n)$  sa nazýva páskovo konštruovateľná, ak existuje deterministický Turingov stroj (DTS), ktorý je  $f(n)$  páskovo ohraničený a vyznačí na páske  $f(n)$  políčko*

Ukazuje sa, že takmer všetky funkcie, ktoré si možno reálne predstaviť, sú páskovo konštruovateľné, problémy pri konštruovateľnosti však nastávajú, keď sa pozrieme na triedu funkcií menších ako  $\log n$ , teda  $O(\log n)$ . Keby sme chceli nahliadnuť medzi nekonštruovateľné funkcie, dobrým prostriedkom na ich zostrojovanie by bola metóda diagonalizácie.

V tejto chvíli sme už pripravení na ukázanie základného výsledku o alternujúcich Turingových strojoch.

**Veta 5.3.1.**  $NSPACE(S(n)) \subseteq ATIME(S^2(n))$  pre  $S(n) \geq \log n$

**Dôkaz:** Ak  $S(n) \geq n$ , nie je problém v lineárnom čase prečítať vstup, ak by však bolo  $S(n) < n$ , na prečítanie vstupu našim ATS  $A'$  by sme potrebovali aspoň lineárny čas a celá simulácia by ztroskotala už na začiatku. Preto použijeme model ATS s rýchlym prístupom na vstup, v logaritmickom čase  $O(\log n)$  sa vieme dostať na ľubovoľné políčko vstupnej pásky, využívame tu ideu paralelnej práce procesov, nie každý proces musí vidieť celý vstup, aby akceptoval, resp. každému procesu bude k práci stačiť malý úsek vstupu. K danému NTS  $A$  pracujúcemu v priestore  $S(n)$  zostrojíme ATS  $A'$  pracujúci v čase  $S^2(n)$  taký, že  $L(A) = L(A')$ .

Akceptujúci výpočet  $A$  na slove  $w$  dĺžky  $n$  má (zmysluplne) dĺžku najviac  $c^{S(n)}$  pre vhodné<sup>3</sup>  $c$ . Položme  $m = c^{S(n)}$ ,  $\forall i$   $k_i$  je konfigurácia  $A$ , akceptujúci výpočet má tvar:

$$k_1 \vdash k_2 \vdash \dots \vdash k_m$$

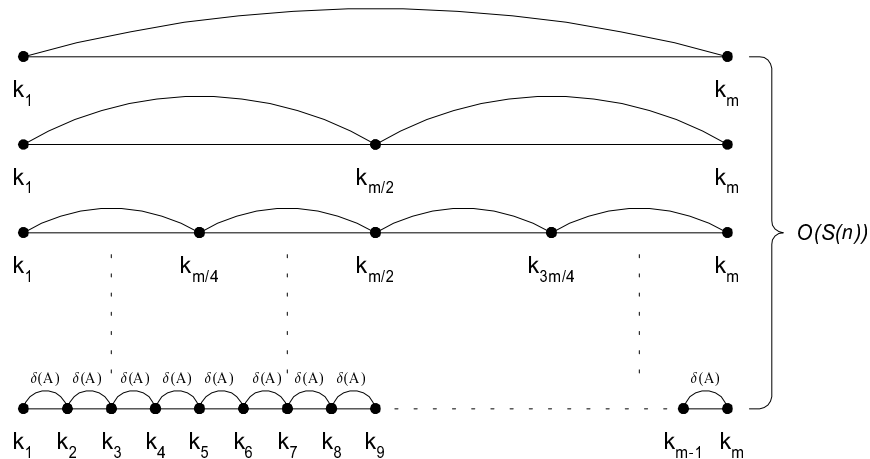
<sup>3</sup>ako už neraz,  $c^{S(n)}$  je počet rôznych konfigurácií  $A$  na slove dĺžky  $n$ , keď máme k dispozícii priestor  $S(n)$

pričom  $k_1$  je počiatková konfigurácia,  $k_m$  je akceptačná konfigurácia (ak je náhodou akceptujúci výpočet kratší, dodefinujeme  $\delta$ -funkciu v akceptačnom stave tak, aby sa nič nedialo, ale aby sme mohli “naťahovať” výpočet).

ATS  $A'$  bude pracovať nasledovne:

1. výpočet začne v  $k_1$
2. uhádne akceptačnú konfiguráciu  $k_m$
3. overí, či sa z  $k_1$  do  $k_m$  dá dostať na  $m$  krokov nasledovne:
  - (a) uhádne  $k_{\frac{m}{2}}$
  - (b) overí, či sa z  $k_1$  dá dostať do  $k_{\frac{m}{2}}$  na  $\frac{m}{2}$  krokov
  - (c) overí, či sa z  $k_{\frac{m}{2}}$  dá dostať do  $k_m$  na  $\frac{m}{2}$  krokov

Body b) a c) tretieho kroku sú vlastne rekurzívne volania kroku 3 s parametrami  $k_1, k_{\frac{m}{2}}, \frac{m}{2}$ , resp.  $k_{\frac{m}{2}}, k_m, \frac{m}{2}$ . ATS  $A'$  sa do rekurzie bude vnašať<sup>4</sup> dovtedy, kým sa nedostane na úroveň, kedy bude overovať, či sa dá z  $k_i$  dostať do  $k_{i+1}$  na jeden krok  $\forall i = 1, \dots, m-1$  (obr.5.3). Táto úroveň je z hľadiska rekurzie elementárna, na nej stačí overiť, či v  $\delta$ -funkcii NTS  $A$  existoval taký prechod, ktorý umožnil prepísanie konfigurácie  $k_i$  na  $k_{i+1}$  (teda simulujeme jeden krok pôvodného NTS  $A$ ). Je nutné si uvedomiť, že rekurzia sa vykonáva paralelne.



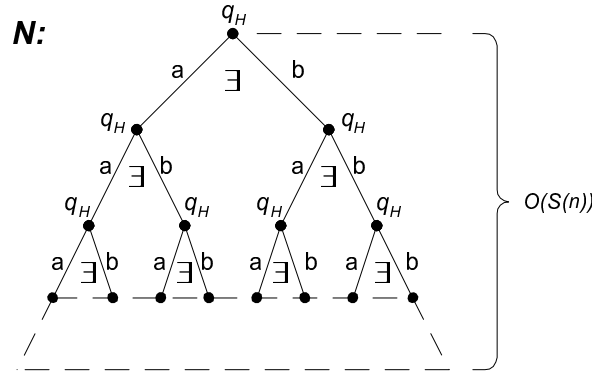
Obrázok 5.3: Overovanie návaznosti konfigurácií v ATS  $A'$

Pozrime sa na proces hľadania konfigurácií a overovania dosiahnuteľnosti konfigurácie  $k_j$  z konfigurácie  $k_i$  na  $m$  krokov podrobnejšie, z hľadiska alternovania a rozdelenia množiny stavov ATS  $A'$  na existenčné a univerzálne. Na začiatku výpočtu je  $A'$  v stave  $q_0$  a na vstupe má slovo  $w$ , teda  $k_1 = q_0w$ , tento stav je existenčný. Teraz  $A'$  prejde do stavu  $q_H$  a háda  $k_m$ , pod  $q_0w$  v strome konfigurácií visí “veľmi košatý” strom (označme ho  $N$ ), je na (obr.5.4), výšky  $S(n)$ , ktorý ako všetky svoje vrcholy, vrátane listov, obsahuje všetky možné konfigurácie NTS  $A$  v priestore

<sup>4</sup>každé rekurzívne volanie si môžeme reprezentovať z hľadiska alternovania tak, že  $A'$  vytvorí nový proces, ktorý ako parametre dostane  $k_i, k_j, r$  a za úlohu má overiť, či sa z  $k_i$  dá dostať do  $k_j$  na  $r$  krokov (obr.5.5)

$S(n)$ . Kvôli jednoduchosti si ho môžeme popísať nasledovne<sup>5</sup> (na čitateľa nechávame domyslenie ukončenia generovania):

$$\delta(q_H, 1) = \{(q_H, a, 1), (q_H, b, 1)\}$$



Obrázok 5.4: Hádanie konfigurácie dĺžky  $S(n)$  ATS  $A'$

Keď má  $A'$  uhádnutú akceptačnú konfiguráciu  $k_m$ , overí, či sa z  $k_1$  do  $k_m$  dá dostať na  $m$  krokov. To spraví tak, že uhádne  $k_{\frac{m}{2}}$  rovnako, ako hádal  $k_m$  (teda pod každým vrcholom stromu  $N$  visí nový strom (opäť rovnaký ako  $N$ , má odlišné stavy, všetky sú existenčné), v ktorom  $A'$  háda  $k_{\frac{m}{2}}$ ). Keď ju uhádne, overuje, či sa z  $k_1$  dá dostať do  $k_{\frac{m}{2}}$  na  $\frac{m}{2}$  krokov a či sa z  $k_{\frac{m}{2}}$  dá dostať do  $k_m$  na  $\frac{m}{2}$  krokov tak, že v univerzálnom stave spraví FORK dvoch nových procesov<sup>6</sup>, ktoré pracujú paralelne a každý overuje polovicu toho, čo mal overiť materský proces, atď.

Z konštrukcie by malo byť zrejmé (nebudeme to dokazovať), že  $L(A) = L(A')$ . Poďme sa teraz pozrieť na časovú zložitosť ATS  $A'$ , ktorý sme práve zkonštruovali: máme  $O(S(n))$  rekurzívnych volaní a v každom hádame strednú konfiguráciu, resp. na začiatku musíme uhádnúť akceptačnú konfiguráciu  $k_m$ . Hádanie ľubovoľnej konfigurácie  $k_i$  trvá čas  $O(S(n))$ , čo je presne výška stromu  $N$ , lebo ju treba zapamätať na páske. Celkový počet krokov (alebo inak výška akceptačných vetiev úplného stromu konfigurácií)  $A'$  je  $O(S^2(n))$  a teda  $L(A') \in ATIME(S^2(n))$   $\square$

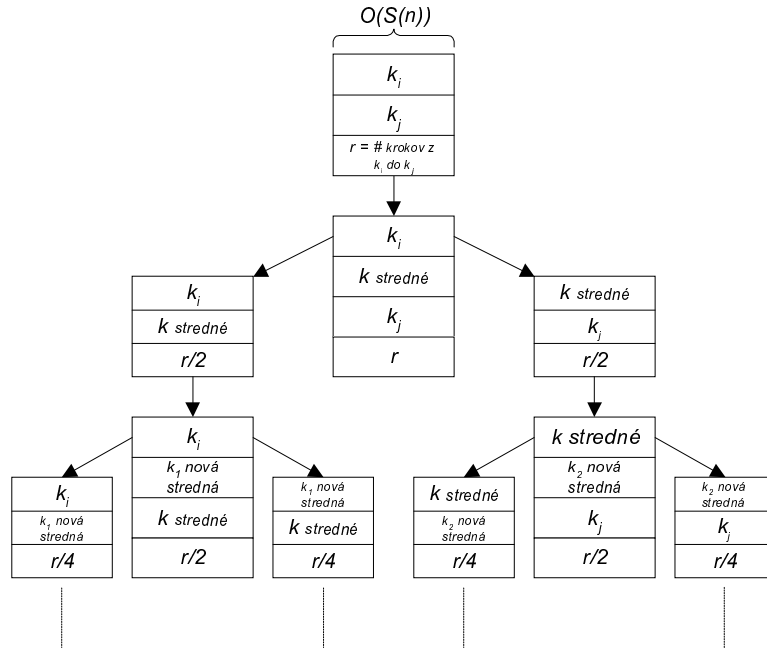
**Poznámka 5.3.1.** V predchádzajúcej vete sme použili “paralelnú verziu” konštrukcie, ktorá bola (v sekvencnej podobe) použitá pri dôkaze vety  $NSPACE(S(n)) \subseteq DSPACE(S^2(n))$  (Savitch)

Pri väčšine známych konštrukcií, keď máme dané zariadenie (Turingov stroj) a požadujeme k nemu zostrojiť zariadenie ekvivalentné, ale rýchlejšie, urýchlenie ide na úkor použitého priestoru, resp. analogicky, keď požadujeme zmenšenie použitého priestoru, ide to na úkor času potrebného na výpočet. Ako naznačila konštrukcia z vety 5.3.1, pri alternujúcich strojoch je situácia trochu iná. Pozornému čitateľovi iste neunikol fakt, že priestorové ohraňenie zostrojeného ATS zostalo nezmenené.

V niektorých konštrukciách bude kôli jednoduchosti vhodné uvažovať binárne vetvenie úplného stromu konfigurácií ATS, teda fakt, že ľubovoľný prechod  $\delta$ -funkcie má najviac dva prvky. Ukážeme si preto nasledovné dve tvrdenia o normálnych tvaroch pre ATS.

<sup>5</sup>konfigurácia ako ju popisujeme, neobsahuje stav, ani pozíciu hlavy, čitateľ si tieto veci iste rád domyslí sám

<sup>6</sup>to je práve jedno rekurzívne volanie



Obrázok 5.5: Schématické vytváranie nových procesov ATS  $A'$

**Lema 5.3.1.** *K ľubovoľnému ATS  $A$  pracujúcemu v čase  $T(n)$  s ľubovoľnou veľkosťou vetvenia<sup>7</sup> v úplnom strome konfigurácií existuje ATS  $A'$  pracujúci v čase  $T(n)$ , ktorého úplný strom konfigurácií je binárny a  $L(A) = L(A')$*

**Dôkaz:** Iba neformálne naznačíme spôsob konštrukcie  $A'$ : nech je vetvenie vo vrchole  $v$  úplného stromu konfigurácií  $A$  stupňa  $n$ . Binárne vetvenie vytvoríme tak, že v ľavej vetve vychádzajúcej z vrchola  $v$  bude jeho najľavejší nasledovník a v pravej všetci ostatní nasledovníci tak, že priamy nasledovník  $v$  bude  $v'$  a jeho nasledovníci budú zvyšnými nasledovníkmi  $v$ . Konfiguráciu v tomto vrchole dostaneme tak, že v konfigurácii vo  $v$  zmeníme stav na nový, ktorý ešte nepatrí do množiny stavov  $A$ , vetvenie vrchola  $v'$  bude stupňa  $n - 1$ , teda o 1 menšieho ako vetvenie  $v$ . Na vrchol  $v'$  aplikujeme algoritmus rekurzívne a v rekuzii skončíme až vtedy, keď bude vrchol stupňa  $\leq 2$ . Na čitateľa nechávame premyslieť, ako algoritmus aplikovať na celý strom konfigurácií<sup>8</sup>. Dodajme ešte, že ak vrchol  $v$  bol existenčný, tak aj všetky vrcholy, ktoré pri vytváraní binárneho vetvenia vzniknú, budú existenčné, podobne ak  $v$  bol univerzálny, tak všetky vzniknuté vrcholy budú univerzálne. Uvedomme si, že hĺbka akceptačného výpočtu v úplnom strome konfigurácií  $A'$  bude iba konštantným násobkom hĺbky úplného stromu konfigurácií  $A$ , teda  $A'$  pracuje v čase  $T(n)$  a akceptuje rovnaký jazyk ako  $A$   $\square$

**Lema 5.3.2.** *K ľubovoľnému ATS  $A$  pracujúcemu v priestore  $S(n)$  s ľubovoľnou veľkosťou vetvenia v úplnom strome konfigurácií existuje ATS  $A'$  pracujúci v priestore  $S(n)$ , ktorého úplný strom konfigurácií je binárny a  $L(A) = L(A')$*

<sup>7</sup>veľkosťou vetvenia rozumieme maximálny počet konfigurácií dosiahnuteľný z ľubovoľnej konfigurácie na jeden krok

<sup>8</sup>pomôcka:  $\delta$ -funkcia  $A$  má iba konečne veľa prvkov a vetvenie v úplnom strome konfigurácií je iba grafické znázornenie  $\delta$ -funkcie, resp. výpočtu

**Dôkaz:** V konštrukcii z lemy 5.3.1 sme nikde nemenili priestor, ktorý využíval  $A$  pri práci, a teda tvrdenie je jej priamym dôsledkom  $\square$

Nasledujúce tvrdenie hovorí, ako efektívne z hľadiska priestorového ohraničenia vieme simulovať alternujúce Turingove stroje pracujúce v čase  $T(n)$  na deterministických Turingových strojoch.

**Veta 5.3.2.**  $ATIME(T(n)) \subseteq DSPACE(T(n))$  pre  $T(n) \geq \log n, T(n)$  je páskovo konštruovateľná

**Dôkaz:** V celom dôkaze budeme pod pojmom strom konfigurácii rozumieť podstrom úplného stromu konfigurácii, ktorý dostaneme tak, že v úplnom strome konfigurácii “odrežeme” všetky vetvy v hĺbke  $T(n)$ .

K danému ATS  $A$  pracujúcemu v čase  $T(n)$  zostrojíme DTS  $A'$  pracujúci v priestore  $T(n)$ . Podľa lemy 5.3.1 môžeme kvôli jednoduchosti predpokladať, že úplný strom konfigurácii  $A$  je binárny.  $A'$  musí overiť, či sa v strome konfigurácii  $A$  nachádza podstrom akceptujúceho výpočtu.

$A'$  bude prehľadávať strom konfigurácii  $A$  algoritmom PREORDER do hĺbky a priradovať vrcholom hodnoty 0, 1 nasledovne:

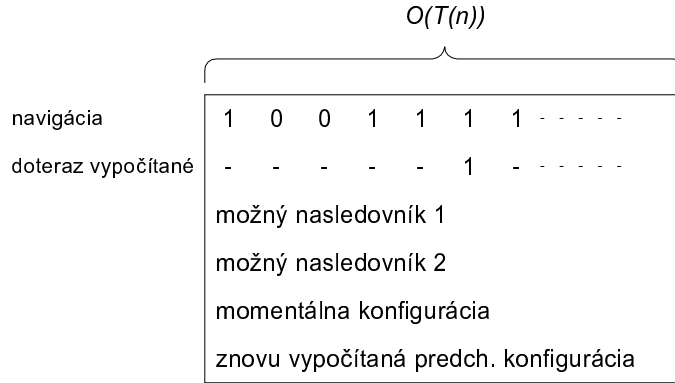
- vrcholu sa môže priradiť hodnota iba vtedy, ak sú už priradené hodnoty všetkým jeho nasledovníkom v strome konfigurácii alebo ak je to list<sup>9</sup>
- listom priradujeme hodnoty podľa toho, či sú akceptujúcimi, alebo neakceptujúcimi konfiguráciami, akceptujúcim priradíme hodnotu 1, neakceptujúcim hodnotu 0
- ak je vrchol existenčný (a nie je to list) a aspoň jeden z jeho nasledovníkov v strome konfigurácii má hodnotu 1, tak sa mu priradí hodnota 1, inak sa mu priradí 0
- ak je vrchol univerzálny (a nie je to list), tak sa mu priradí hodnota 1 práve vtedy, keď obaja jeho nasledovníci v strome konfigurácii majú hodnotu 1, inak sa mu priradí 0

$A'$  akceptuje svoj vstup práve vtedy, keď bude koreňu priradená hodnota 1. Keby sme chceli dosiahnuť dobrý pomer čas/priestor, asi by sme postupovali tak, že by sme si pamätali informáciu o celom “lúči” konfigurácii, teda všetky konfigurácie, ktorými sme od koreňa prechádzali až k listom, resp. ku vrcholom v hĺbke  $T(n)$ , aby sme nestrácali čas pri vracaní sa v strome smerom nahor. Dosiahli by sme však priestorové ohraničenie  $T^2(n)$  (pretože by sme si museli pamätať až  $T(n)$  konfigurácií, každú dĺžky  $T(n)$ ), čo v našom prípade nie je žiadúce, preto budeme musieť použiť trochu rafinovanejšiu konštrukciu, vzhľadom na to, že nám nezáleží na čase. Nebudeme si pamätať celý “lúč” konfigurácii, ale iba momentálnu konfiguráciu a navigáciu k nej v strome konfigurácii. Navigácia bude dĺžky najviac  $T(n)$  a bude to postupnosť z  $\{0, 1\}^*$ , kde 0 znamená pohyb v strome vľavo smerom dole a 1 znamená pohyb vpravo smerom dole. Keď sa budeme v strome musieť vracat smerom hore, tak si predchodcu konfigurácie, v ktorej práve sme, podľa tejto navigácie vypočítame, na páske  $A'$  (obr.5.6) si budeme potrebovať pamätať navigáciu, momentálnu konfiguráciu a pár pomocných konfigurácií, teda máme priestorové ohraničenie  $T(n)$  pre  $A'$   $\square$

V teórii formálnych jazykov a automatov, ale aj v iných informatických disciplínach, majú významné postavenie tzv. polynomiálne triedy zložitosti, najmä z hľadiska riešiteľnosti problémov do nich patriacich v “rozumnom” čase, resp. v “rozumnom” priestore (množstvo pamäte potrebnej na vyriešenie problému).

---

<sup>9</sup>listom sa v tomto prípade myslí aj taký vrchol, ktorý síce nie je listovým v úplnom strome konfigurácii  $A$ , ale stáva sa listovým vrcholom v strome konfigurácii  $A$



Obrázok 5.6: Páska DTS  $A'$  (druhá stopa by sa dala pamätať aj v stave)

**Definícia 5.3.2.** *Polynomiálne triedy zložitosti definujeme nasledovne:*

- $NSPACE(Poly) \stackrel{def}{=} \bigcup_{k \geq 1} NSPACE(n^k)$
- $DSPACE(Poly) \stackrel{def}{=} \bigcup_{k \geq 1} DSPACE(n^k)$
- $ATIME(Poly) \stackrel{def}{=} \bigcup_{k \geq 1} ATIME(n^k)$

**Dôsledok 5.3.1.**  $NSPACE(Poly) = DSPACE(Poly) = ATIME(Poly)$

**Dôkaz:** Tvrdenie vyplíva zo Savitchovej vety, vety 5.3.1 a vety 5.3.2  $\square$

Vety 5.3.1 a 5.3.2 nám poskytli porovnanie alternujúceho času a sekvenčného priestoru, nasledujúce tvrdenia nám umožnia porovnať alternujúci priestor so sekvenčným časom.

**Veta 5.3.3.**  $ASPACE(S(n)) \subseteq \bigcup_{c > 0} DTIME(c^{S(n)})$  pre  $S(n) \geq \log n$

**Dôkaz:** Ukážeme si, že keď máme k ATS  $A$  pracujúcemu v priestore  $S(n)$  zostrojiť DTS  $A'$  pracujúci v exponenciálnom čase  $c^{S(n)}$  pre nejaké nezáporné  $c$ , tak nám na simuláciu postačí aj veľmi hrubá sila, akou je bezosporu vygenerovanie všetkých možných konfigurácií  $A$  na páske a práca na týchto konfiguráciách. Opäť budeme bez újmy na všeobecnosti predpokladať, že úplný strom konfigurácií  $A$  je binárny.

$A'$  bude pracovať nasledovne:

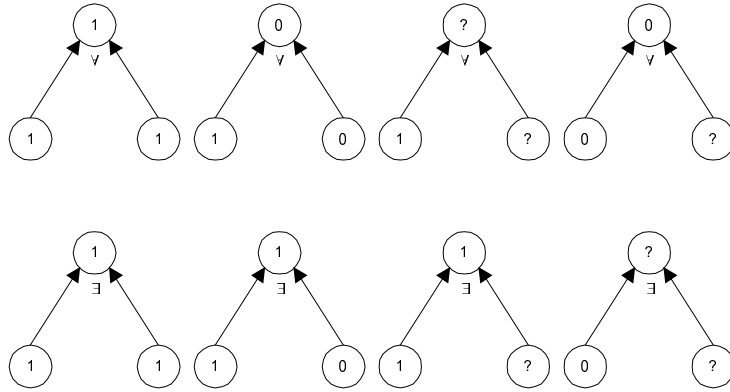
- na pásku zapíše všetkých  $|\Gamma_A|^{S(n)} \cdot S(n) \cdot |K_A| < r^{S(n)}$  konfigurácií  $A$  v lexikografickom usporiadaní, za každou konfiguráciou si nechá priestor (jedno políčko), ktorý ďalej využije<sup>10</sup>, na to potrebuje čas  $S(n) \cdot r^{S(n)}$
- do vyznačeného priestoru bude každej konfigurácii priradovať hodnoty 0, 1, ? prechodom stromu konfigurácií  $A$  tak, že v každom prechode priradí hodnoty 0, 1 rodičom tých detí,

<sup>10</sup>priestor si na páske vyznačíme napr.  $\#\#\#$ , pričom  $\#$  slúži ako oddeľovač konfigurácií, nie je prvkom páskovej abecedy  $\Gamma$ , rovnako ani  $?$ , ktorý hovorí, že o tomto políčku zatiaľ nič nevieme

ktoré už sú vyhodnotené a hodnotu ? rodičom nevyhodnotených detí (podľa (obr.5.7)), akceptuje, ak bude počiatočnej konfigurácii priradená hodnota 1, časová zložitosť bude nasledovná:

1.  $r^{S(n)}$ -krát prejde pásku a “spracuje” každú konfiguráciu
2. “spracovať” konfiguráciu znamená zistiť hodnoty jej nasledovníkov a ak sa dá, tak jej priradiť hodnotu 0,1, toto je časovo ohraničené  $\approx r^{S(n)}.S(n)$

Keď si celú prácu  $A'$  zosumarizujeme, dostávame časovú zložitosť približne  $c^{S(n)}$  pre nejaké  $c$  (stačí zvoliť napr.  $c = r^{10}$ )  $\square$



Obrázok 5.7: Priradovanie hodnôt rodičom v strome konfigurácii ATS  $A$

**Veta 5.3.4.**  $DTIME(T(n)) \subseteq ASPACE(\log T(n))$  pre  $T(n) \geq n$

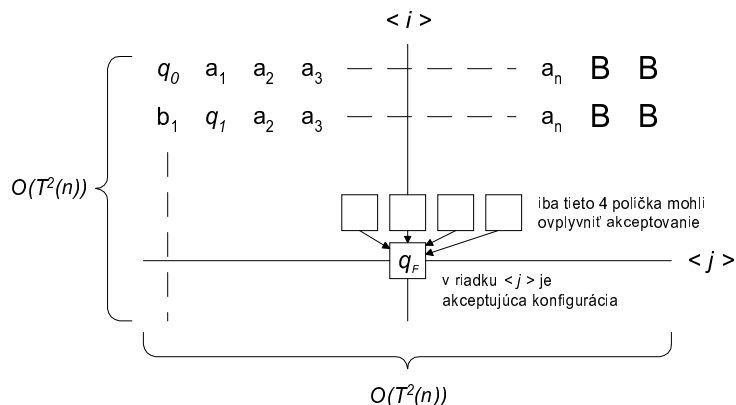
**Dôkaz:** Nech  $A_1$  je  $k$ -páskový DTS pracujúci v čase  $T(n)$ . K nemu existuje jednopáskový DTS  $A = (K_A, \Sigma, \Gamma, \delta, q_0, F_A)$  pracujúci v čase  $T^2(n)$  taký, že  $L(A_1) = L(A)$ . K nemu zkonštruujeme ATS  $A'$  pracujúci v priestore  $\log T(n)$  taký, že  $L(A) = L(A')$ . Najskôr si zapíšme konfigurácie  $A$  pod seba podľa (obr.5.8) tak, že v prvom riadku bude počiatočná konfigurácia  $q_0 w$  a keď označíme  $\langle qw_k \rangle_l$  konfiguráciu v  $k$ -tom riadku pre nejaký stav  $q \in K_A$  a pozíciu stavu (hlavy) v tejto konfigurácii  $l$ , tak platí  $\langle qw_k \rangle_l \vdash \langle pw_{k+1} \rangle_m$ , pričom  $m \in \{l-1, l, l+1\}$ . Ak  $w \in L(A)$ , tak máme v tabuľke zapísanú postupnosť konfigurácií akceptujúceho výpočtu (vzhľadom na to, že  $A$  je deterministický), teda existuje  $i$  také, že pre  $\langle q_F w_i \rangle_j$  je  $q_F \in F_A$ .

$A'$  bude pracovať nasledovne:

- uhádne pozíciu  $\langle \text{riadok}, \text{stĺpec} \rangle$  akceptačného stavu  $q_F$  v akceptujúcej konfigurácii tvaru  $\langle q_F w_i \rangle_j$ , teda háda  $\langle i, j \rangle$ , toto zaberie  $\log T^2(n)$  políček (môžeme zvoliť napr. binárnu reprezentáciu čísel  $i, j$ ), to je  $2 \cdot \log T(n) \approx \log T(n)$  políček
- uhádne, ktorý akceptujúci stav sa na pozícii  $\langle i, j \rangle$  nachádza
- overí, či dobre hádal pozíciu  $\langle i, j \rangle$  a akceptačný stav nasledovne:
  1. uhádne (zmysluplne<sup>11</sup>) obsahy políček  $\langle i-1, j-1 \rangle, \langle i-1, j \rangle, \langle i-1, j+1 \rangle, \langle i-1, j+2 \rangle$ , pretože týmito obsahmi je obsah  $\langle i, j \rangle$  jednoznačne určený
  2. v univerzálnom vetvení overí, či hádal správne (každá vetva overí jedno políčko)

<sup>11</sup>podľa  $\delta$ -funkcie  $A$





Obrázok 5.8: Tabuľka konfigurácií jednopáskového DTS  $A$

Takto sa bude  $A'$  vracaf v tabuľke až do prvého riadku (zrejme si bude dekrementovať  $i$  a pracovať s  $j$  podľa pohybu hlavy  $A$ ), keď bude na pozícii  $\langle 1, 1 \rangle$  a bude v počiatocnom stave, tak akceptuje

Pri prechode tabuľkou konfigurácií smerom zdola nahor sa nám nemôže stať, že by sme našli viac ako jednu cestu vedúcu k akceptovaniu, pretože  $A$  je deterministický a teda akceptačný výpočet pre dané slovo je vždy jednoznačný, ak sme v niektorej vetve niečo zle uhádli, výpočet sa určite zablokuje.

Keď sa máme baviť o priestorovej zložitosti  $A'$ , jediná vec, ktorá ju ovplyvňuje, je dĺžka  $i$ , resp.  $j$ , pretože okrem týchto dvoch čísel si pamätáme iba konštantne veľa informácií (dokonca veľmi málo). Ale o dĺžke sme si už povedali, že je  $\log T(n)$   $\square$

**Dôsledok 5.3.2.**  $SPACE(S(n)) = \bigcup_{c>0} DTIME(c^{S(n)})$  pre  $S(n) \geq n$

## 5.4 Alternujúce konečné automaty (AFSA)

Čitateľ sa iste stretol s viacerými modifikáciami pôvodného modelu deterministických konečných automatov. Nedeterminizmus im nepomohol, rovnako ako nepomohlo napr. pridanie možnosti obojsmerného pohybu po vstupnej páske. Mohlo by sa zdať, že taká sila, akou je alternovanie, by mohlo pomôcť týmto zariadeniam vyjsť z triedy  $\mathcal{R}$  a akceptovať aj nejaké nie regulárne jazyky. Ako však hovorí nasledujúca veta, opak je pravdou. Faktom totiž zostáva, že keď procesom v jednotlivých vetvách neumožníme komunikáciu, resp. synchronizáciu, tak sa sila zariadenia nezväčší.

**Veta 5.4.1.**  $\mathcal{L}_{AFSA} = \mathcal{R}$

**Dôkaz:** Inklúzia  $\supseteq$  je triviálna a preto ju nebudeme ukazovať. Na dôkaz  $\subseteq$  zostrojíme k AFSA  $A$  ekvivalentný nedeterministický konečný automat (NKA)  $A'$  taký, že  $L(A) = L(A')$ . Najskôr podobnou konštrukciou ako pre NKA zostrojíme k  $A$  ekvivalentný ATS  $A_1$  (teda platí  $L(A) = L(A_1)$ ) taký, že v  $A_1$  neexistujú  $\varepsilon$ -prechody (konštrukcii sa nebudeme bližšie venovať, čitateľ ju nájde napr. v [1]). Zavedieme pár označení, ktoré sa nám neskôr budú hodiť:

- $all(q, a) = \{\text{množina všetkých stavov dosiahnuteľných v } A_1 \text{ na jeden krok zo stavu } q \text{ pri čítaní vstupného symbolu } a\}$
- $ex_i(q, a) = p$ , pričom  $p \in \delta'(q, a)$  a keď máme prvky  $\delta'(q, a)$  očíslované, resp. usporiadané, tak  $\delta'(q, a) = p$  je  $i$ -ty v poradí

Teraz k bez- $\varepsilon$  ATS  $A_1 = (K, \Sigma, \delta, q_0, F)$  zostrojíme NKA  $A' = (K', \Sigma', \delta', q'_0, F')$  nasledovne:

- Vezmime si úplný strom konfigurácií  $A_1$  a pozrime sa naň po úrovniach. Keďže v  $A_1$  nie sú  $\varepsilon$ -prechody, tak na  $n$ -tej úrovni je zo vstupu prečítaných práve  $n$ -symbolov
- $K'$  bude nová množina stavov, pričom jej prvky budú podmnožiny množiny stavov  $K$ , teda počet stavov  $|K'| = 2^{|K|}$ , formálne

$$[q_{i_1}, \dots, q_{i_k}] \in K' \xLeftrightarrow{def} q_{i_1}, \dots, q_{i_k} \in K$$

- $\Sigma' = \Sigma$ , teda vstupná abeceda sa nemení
- $\delta'$  definujeme nasledovne:
  - ak  $q$  je univerzálny a  $\delta(q, a) = \{q_{i_1}, \dots, q_{i_k}\}$ , tak  $\delta'([q], a) = \{[q_{i_1}, \dots, q_{i_k}]\}$
  - ak  $q$  je existenčný a  $\delta(q, a) = \{q_{i_1}, \dots, q_{i_k}\}$ , tak  $\delta'([q], a) = \{[q_{i_1}], \dots, [q_{i_k}]\}$
  - rekurzívne definujeme  $\delta'([q_{i_1}, \dots, q_{i_j}, q_{i_{j+1}}, \dots, q_{i_k}], a)$  ako  $\{[all(q_{i_1}, a), \dots, all(q_{i_j}, a), ex_1(q_{i_{j+1}}, a)], \dots, [all(q_{i_1}, a), \dots, all(q_{i_j}, a), ex_n(q_{i_k})])\}$  pričom  $q_{i_1}, \dots, q_{i_j}$  sú univerzálne stavy a  $q_{i_{j+1}}, \dots, q_{i_k}$  sú existenčné stavy a platí  $|\delta'(q_{i_k})| = n$

V stavoch  $A'$  udržujeme informácie o všetkých vetvách úplného stromu konfigurácií  $A_1$ , je dobré si uvedomiť, že ak už máme množinu stavov  $[p, q]$ , a napr.  $\delta'([p], a) = [r]$  a súčasne  $\delta'([q], a) = [r]$ , tak si túto informáciu nemusíme pamätať dva krát, ako stav si budeme pamätať iba  $[r]$

- prirodzene definujeme  $q'_0 = [q_0]$
- $F' = \{[q_{i_1}, \dots, q_{i_k}] \mid q_{i_1}, \dots, q_{i_k} \in F\}$ , teda  $A'$  akceptuje, ak všetky vetvy  $A$ , o ktorých si pamätá informáciu, akceptujú

Nemalo by byť až také ťažké pochopiť, že  $L(A_1) = L(A')$ , a teda platí aj  $L(A) = L(A')$   $\square$

V niektorých konštrukciách je potrebné a zmysluplné požadovať, aby zostrojený konečný automat k alternujúcemu konečnému automatu bol deterministický. Potom jeho stavy budú opäť množiny stavov AFSA, keď tieto budú univerzálne, no keď prídu do hry existenčné stavy, tak sa stavy rozpadnú na množiny, akceptačné stavy potom budú také, ktorých aspoň jedna zložka je akceptačná v už definovanom nedeterministickom zmysle. Stavov môže byť až  $2^{2^{|K|}}$ , čo nie je zanedbateľné číslo.

**Príklad 5.4.1.** Ku AFSA  $A = (K, \Sigma, \delta, q_0, F)$ , ktorého úplný strom konfigurácií na  $w = aba$  je na (obr.5.9), pričom  $\{q_0, \dots, q_4, p_1, p_2, r_1, \dots, r_4\} \subseteq K$ ,  $\Sigma = \{a, b\}$ ,  $\{r_1, r_2, r_3\} \subseteq F$  a  $\delta$  je zrejmá z obrázku<sup>12</sup>, zostrojíme NKA  $A' = (K', \Sigma', \delta', q'_0, F')$  (jeho časť) podľa konštrukcie z vety 5.4 nasledovne:

1.  $\{[q_0], [p_1, p_2], [q_1, q_3], [q_1, q_4], [q_2, q_3], [q_2, q_4], [r_1, r_2, r_3], [r_2, r_3, r_4]\} \subset K'$

<sup>12</sup>definovali sme iba časť AFSA  $A$  potrebnú pre výpočet na  $w$

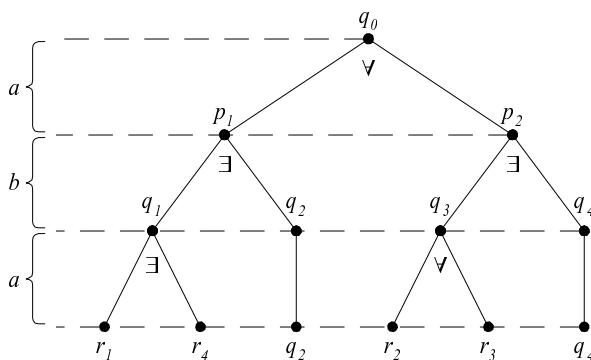
2.  $\Sigma' = \Sigma$
3.  $q'_0 = [q_0]$
4.  $[r_1, r_2, r_3] \subseteq F'$
5.  $\delta'$  definujeme schématicky nasledovne:

$$[q_0] \xrightarrow{a} [p_1, p_2] \xrightarrow{b} \begin{cases} [q_1, q_3] \xrightarrow{a} \begin{cases} [r_1, r_2, r_3] \\ [r_2, r_3, r_4] \end{cases} \\ [q_1, q_4] \xrightarrow{a} \begin{cases} [r_1, q_4] \\ [r_4, q_4] \end{cases} \\ [q_2, q_3] \xrightarrow{a} [q_2, r_2, r_3] \\ [q_2, q_4] \xrightarrow{a} [q_2, q_4] \end{cases}$$

Potom zrejme výpočet

$$[q_0]aba \vdash_{A'} [p_1, p_2]ba \vdash_{A'} [q_1, q_3]a \vdash_{A'} [r_1, r_2, r_3]$$

je akceptujúcim výpočtom NTS  $A'$  na  $w$ .



Obrázok 5.9: Úplný strom konfigurácií AFSA  $A$  na slove  $w = aba$

## 5.5 Alternujúce zásobníkové automaty (APDA)

Ako sme ukázali, konečným automatom alternovanie v generatívnej sile vôbec nepomohlo. Lepšia je situácia v oblasti bezkontextových jazykov, ktoré rozpoznávajú zásobníkové automaty (PDA). Tu alternovanie zvýši možnosť rozpoznávania jazykov až na úroveň, o ktorej hovorí nasledujúce tvrdenie.

**Veta 5.5.1.**  $\bigcup_c \text{DTIME}(c^n) \subseteq \mathcal{L}_{APDA}$

**Dôkaz:** Tvrdenie nebudeme dokazovať priamo, využijeme tvrdenie vety 5.3.4, podľa ktorého platí  $\bigcup_c \text{DTIME}(c^n) \subseteq \text{SPACE}(n)$ . My ukážeme, že platí  $\text{SPACE}(n) \subseteq \mathcal{L}_{APDA}$ , potom bude z tranzitívnosti relácie  $\subseteq$  platiť aj  $\bigcup_c \text{DTIME}(c^n) \subseteq \mathcal{L}_{APDA}$ .

Na dôkaz  $ASPACE(n) \subseteq \mathcal{L}_{APDA}$  potrebujeme k ATS  $A$ , ktorý pre vstup dĺžky  $n$  používa  $n$  políček na pracovnej páske, zostrojiť APDA  $A'$  taký, že  $L(A) = L(A')$ . Bez újmy na všeobecnosti budeme predpokladať, že úplný strom konfigurácií  $A$  je binárny<sup>13</sup>.

$A'$  bude pracovať nasledovne<sup>14</sup>:

- najskôr v existenčnom vetvení uhádne počiatočnú konfiguráciu (bude hádať  $n+1$  symbolov, háda aj stav)  $q_0w$ , pričom  $w = a_1 \dots a_n$ , v obrátenom poradí, teda zásobník bude mať po  $n+1$  krokoch tvar ako na (obr.5.10a)
- v univerzálnom vetvení v jednej vetve overí, či konfiguráciu hádal správne, v druhej vetve háda nasledovníkov konfigurácie  $q_0w$  vo vetvení, ktorého typ zodpovedá vetvení v úplnom strome konfigurácií  $A$ , každej konfigurácii priradí aj vetvu, v ktorej sa nachádzala v úplnom strome konfigurácií  $A$  (teda L,R), konfigurácie ďalej overí a háda ďalšie, až kým neakceptuje, resp. neodmietne (REJECT) vstup
- do overenia, či konfigurácie na seba nadväzujú, spadá:
  - overiť, či sme hádali správnu vetvu (prvky  $\delta_A$  si usporiadame L,R)
  - $A'$  musí po jednotlivých symboloch prejsť konfigurácie a testovať ich na rovnosť (obr.5.10b), resp. možnosť prechodu v  $\delta_A$ , univerzálné sa rozvetví: v jednej vetve overí, či sa 1. symbol zhoduje s  $(n+k)$ -tým symbolom<sup>15</sup>, resp. či existuje v  $\delta_A$  prechod taký, aby sa symboly na seba mohli prepísať, tak, že zo zásobníka zmaže  $n+k$  symbolov (je dôležité uvedomiť si, že do tejto chvíle sme v tejto vetve vstup nečítali, tu ho používame na počítanie  $n$ , zrejme bude dobre vždy si v stave pamätať 4 symboly z vrchu pôvodného obsahu zásobníka pred vymazávaním), v ďalšej overí, či 2. až  $(n+1)$ -vý symbol z prvej konfigurácie sedí podľa  $\delta_A$  s 2. až  $(n+1)$ -vým symbolom druhej konfigurácie podobne ako pre 1. symbol

Čitateľovi odporúčame podrobnejšie rozpracovať overovanie konfigurácii, najmä overenie, či sa konfigurácia nachádzala v ľavej, alebo pravej vetve vzhľadom na svojho predka, v úplnom strome konfigurácií ATS  $A$ . Na pochopenie, že  $L(A) = L(A')$  by však úroveň detailu v našej konštrukcii mala byť dostačujúca  $\square$

**Poznámka 5.5.1.** Platí aj obrátená inklúzia, jej dôkaz však presahuje rozsah tohto textu, navyše na ukážku, že alternovanie zásobníkovým automatom pomáha v generatívnej sile, stačí veta 5.5.1

## 5.6 Synchronizované alternujúce stroje

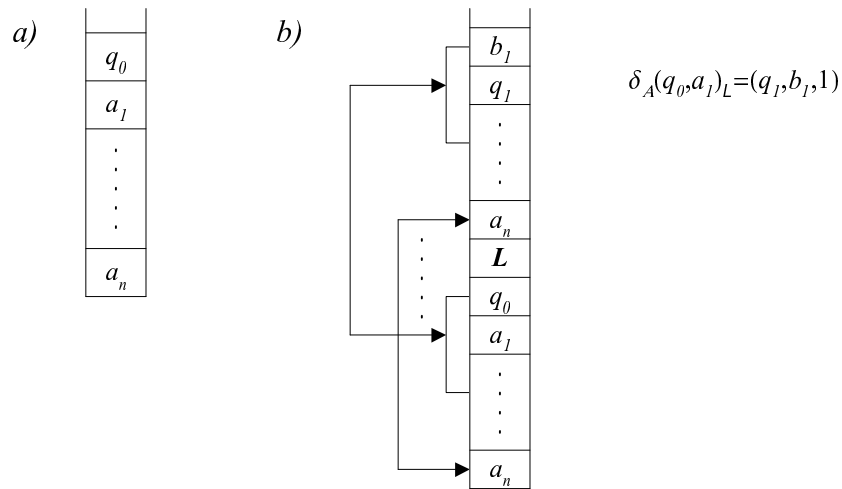
Ukážeme si jednu modifikáciu pôvodného modelu alternujúcich strojov, keď umožníme jednoduchú komunikáciu medzi paralelnými procesmi. Zavedieme nové delenie stavov alternujúceho stroja, nezávisle od delenia na existenčné a univerzálné stavy, na:

- obyčajné
- synchronizačné - dvojica  $(q, S) \rightarrow (\text{stav}, \text{symbol})$

<sup>13</sup>čitateľ si iste vie predstaviť podobnú konštrukciu ako v leme 5.3.2 pre APDA

<sup>14</sup>idea je nasledovná: v zásobníku bude simulovať výpočet ATS  $A$  tak, že do neho bude postupne pridávať konfigurácie  $A$  a overovať ich návaznosť vzhľadom na  $\vdash$ , keď pridá do zásobníka akceptačnú konfiguráciu, ktorá bude nadväzovať na predchádzajúce, akceptuje

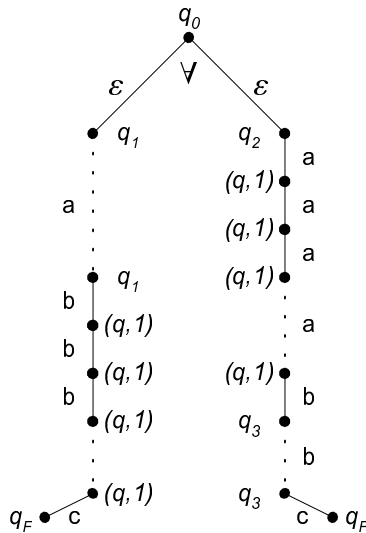
<sup>15</sup> $k$  je konštanta, jej určenie prenechávame na čitateľa



Obrázok 5.10: Zásobník APDA  $A'$  a) po uhádnutí  $q_0 w$  b) pri overovaní náväznosti konfigurácií

Keď proces prejde do synchronizačného stavu, čaká, kým všetky ostatné prejdú do synchronizačného stavu. Keď majú všetky rovnaký symbol v druhej komponente synchronizačného stavu, pokračujú, inak sa zariadenie zablokuje.

**Príklad 5.6.1.** Ukážeme si, že keď umožníme synchronizáciu konečným automatom, tak sa nám ich podarí posunúť z triedy regulárnych jazykov. Na (obr.5.11) je znázornený synchronizovaný konečný automat, ktorý akceptuje bezkontextový jazyk  $L = \{a^n b^n c \mid n \geq 1\}$



Obrázok 5.11: Synchronizovaný konečný automat pre jazyk  $L = \{a^n b^n c \mid n \geq 1\}$

**Poznámka 5.6.1.** *Dá sa ukázať tvrdenie, ktoré hovorí o tom, že trieda jazykov rozpoznávaných synchronizovanými konečnými automatmi je presne  $\mathcal{L}_{CS}$*

## Kapitola 6

# Booleovské obvody ( $BO$ )

### 6.1 Definície a označenia

**Definícia 6.1.1.** *Booleovský obvod ( $BO$ ) je konečný acyklický orientovaný graf, v ktorom každému vrcholu  $v$  priradíme typ  $\tau(v) \in \{I\} \cup \{B_0\} \cup \{B_1\} \cup \{B_2\}$  a hodnotu  $\mathcal{V}(v) \in \{0, 1\}$ . Vrchol  $v$ , pre ktorý typ  $\tau(v) \in \{I\}$ , má vstupný stupeň 0 a nazývame ho vstupný vrchol. Vstupom pre booleovský obvod je  $n$ -tica rôznych vstupných vrcholov označených  $\langle x_1, \dots, x_n \rangle$ . Vrchol  $v$ , pre ktorý typ  $\tau(v) \in \{B_i\}$ , má vstupný stupeň  $i$  a nazývame ho hradlo. Medzi hradlá typu  $B_0$  patria konštanty "0" a "1", do  $B_1$  patria hradlá "I" a "¬" reprezentujúce booleovské funkcie identita a negácia a do  $B_2$  patria hradlá "∧" a "∨" reprezentujúce booleovské funkcie AND a OR. Vrcholy s výstupným stupňom 0 nazývame výstupné. Výstupom pre booleovský obvod je  $m$ -tica rôznych výstupných vrcholov označených  $\langle y_1, \dots, y_m \rangle$ .*

**Definícia 6.1.2.** *Booleovský obvod so vstupom  $\langle x_1, \dots, x_n \rangle$  a výstupom  $\langle y_1, \dots, y_m \rangle$  počíta funkciu  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  nasledovne. Každý vstupný vrchol  $x_i$  má danú hodnotu  $\mathcal{V}(x_i) \in \{0, 1\}$ . Každé hradlo  $h$  jednoznačne vyhodnotí hodnotu  $\mathcal{V}(h)$  aplikovaním elementárnej booleovskej funkcie  $\tau(h)$  na hodnoty vstupných vrcholov. Výsledná hodnota funkcie  $f$  je daná  $m$ -ticou hodnôt výstupných vrcholov  $\langle \mathcal{V}(y_1), \dots, \mathcal{V}(y_m) \rangle$ .*

Všimnime si, že v definícii sme ohraničili počet vstupov vrchola, ale nie počet výstupov. Takisto sme nezakázali, aby vstupný vrchol bol zároveň výstupným (pozri obrázok).

Ak chceme pomocou modelu  $BO$  definovať jazyky, je rozumné sa obmedziť na  $BO$  s jedným výstupným vrcholom, pričom slovo na vstupe z  $\{0, 1\}^*$  akceptujeme práve vtedy, keď hodnota výstupného vrchola bude 1. Toto so sebou prináša jednu nepríjemnosť, pretože  $BO$  má konečný počet vstupných vrcholov, a teda je schopný akceptovať len konečné jazyky. Preto jazyky budeme definovať pomocou triedy booleovských obvodov.

**Definícia 6.1.3.** *Nech  $\{C_n\}_{n=0}^\infty$  je postupnosť booleovských obvodov, kde  $BO C_n$  s  $n$  vstupnými a  $m(n)$  výstupnými vrcholmi počíta funkciu  $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{m(n)}$ . Túto postupnosť nazývame trieda booleovských obvodov  $\{C_n\}$  počítajúca funkciu  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  definovanú takto:  $f(w) \equiv f_n(w)$  práve vtedy, keď  $|w| = n$ .*

**Definícia 6.1.4.** *Nech  $\{C_n\}$  je trieda (postupnosť) booleovských obvodov počítajúca funkciu  $f : \{0, 1\}^* \rightarrow \{0, 1\}$ , teda každý  $BO$  má jeden výstupný vrchol. Jazyk akceptovaný triedou  $\{C_n\}$  definujeme takto:  $L(\{C_n\}) = \{w \in \{0, 1\}^* \mid f(w) = 1\}$ .*

## 6.2 Miery zložitosti

Pre booleovské obvody definujeme nasledujúce miery zložitosti:

- $SIZE(C_n)$  = počet hradiel obvodu  $C_n$
- $DEPTH(C_n)$  = dĺžka najdlhšej cesty v obvode  $C_n$

Ak predpokladáme, že čas, ktorý potrebuje hradlo na vyhodnotenie výstupu je jedna časová jednotka, a čas potrebný na prenos informácie medzi hradlami neuvažujeme, potom miera  $DEPTH$  je ekvivalentná časovej náročnosti výpočtu na  $BO$ .

**Príklad 6.2.1.** Chceme vypočítať skalárny súčin dvoch  $n$ -bitových vektorov  $(x_1, \dots, x_n)$  a  $(y_1, \dots, y_n)$ . Vieme, že ich skalárny súčin vypočítame takto:  $(x_1 \wedge y_1) \vee \dots \vee (x_n \wedge y_n)$ . Príslušný  $BO$  realizujúci tento výpočet je znázornený na obrázku XXX, z ktorého ľahko vidieť, že  $SIZE(C_n) = 2n - 1 = O(n)$  a  $DEPTH(C_n) = n = O(n)$ .

**Príklad 6.2.2.** Opäť chceme vypočítať skalárny súčin dvoch  $n$ -bitových vektorov, tentoraz však použijeme iný  $BO$   $C_n$  (obrázok XXX). Pri zachovaní rovnakého počtu hradiel sme znížili hĺbku obvodu na  $DEPTH(C_n) = O(\log n)$

**Príklad 6.2.3.** Chceme vynásobiť dve booleovské matice  $A$  a  $B$  rozmeru  $n \times n$ . Výsledok je matica rovnakého rozmeru  $C$ , pričom jej prvky vypočítame nasledovne:  $c_{i,j} = \bigvee_{k=1}^n a_{ik} \wedge b_{kj}$ . To je  $n^2$  nezávislých skalárnych súčinov. Ak budeme každý takýto súčin počítat podľa príkladu 6.2.2 dostaneme príslušný obvod  $C_n$ , pre ktorý platí, že  $SIZE(C_n) = O(n^3)$  a  $DEPTH(C_n) = O(\log)$ .

**Príklad 6.2.4.** Teraz sa pokúsime vyrobiť reflexívny a tranzitívny uzáver booleovskej matice  $M$  rozmeru  $n \times n$ . Výsledkom je matica  $M^*$ , ktorú dostaneme nasledovným výpočtom:

$M^* = I \vee M \vee M^2 \vee \dots \vee M^n$ , kde  $M^i = \bigwedge_{k=1}^i M$ , čo znamená, že  $M^* = (I \vee M)^n$ . Ak na výpočet súčinu matíc použijeme obvod z príkladu 6.2.3 a štruktúru úplného binárneho stromu (obrázok XXX) dostaneme obvod  $C_n$  počítajúci  $M^*$ , pre ktorý platí  $SIZE(C_n) = O(n^4)$  a  $DEPTH(C_n) = O(\log^2 n)$ .

## 6.3 $BC$ -uniformné booleovské obvody

Postupnosť booleovských obvodov, ako neskôr uvidíme, dokáže akceptovať triedu jazykov  $\mathcal{L}_{RE}$ , ale tak ako sme ju zatiaľ zdefinovali dokáže akceptovať ešte viac, pretože postupnosť  $BO$  môžeme určiť tak, že pre rôzne dĺžky vstupu bude používať úplne iný mechanizmus spracovania vstupného slova. Preto doterajší model  $BO$  trochu zoslabíme zavedením akejsi "pravidelnosti" (uniformity) do postupnosti  $BO$ .

**Definícia 6.3.1.** *Postupnosť booleovských obvodov je uniformná, ak existuje nejaký deterministický stroj (DTS, ATS), ktorý na vstupe  $1^n$  vygeneruje  $BO$   $C_n$  resp. jeho kód.*

**Definícia 6.3.2.** (Štandardný kód  $BO$ )

Kód booleovského obvodu  $C_n$  (ozn.  $\langle C_n \rangle$ ) je postupnosť štvoric  $\langle g, t, a, b \rangle$ , kde  $g \in \{0, 1\}^+$  je jednoznačné číslo vrchola,  $t \in \{0, 1, I, \neg, \wedge, \vee, x\}$  je typ vrchola  $g$  ( $x$  označuje vstup),  $a, b \in \{0, 1\}^+$  sú čísla ľavého a pravého vstupu vrchola  $g$ . Vstupným vrcholom  $x_1, \dots, x_n$  štandardne priradíme čísla  $1, \dots, n$  a výstupnému vrcholu (ak je len jeden) priradíme číslo 0.

**Definícia 6.3.3.** *Postupnosť booleovských obvodov  $\{C_n\}$  je BC-uniformná (BC-uniformne konštruovateľná<sup>1</sup>), ak existuje DTS, ktorý pre každé  $n$  na vstupe  $1^n$  vygeneruje kód booleovského obvodu  $\langle C_n \rangle$  v priestore<sup>2</sup>  $\log(\text{SIZE}(C_n))$ .*

**Poznámka 6.3.1.** *Čísla vrcholov v kóde obvodu  $C_n$  kódujeme binárne, takže pre veľkosť kódu dostávame  $|\langle C_n \rangle| = O(\text{SIZE}(C_n) \cdot \log \text{SIZE}(C_n))$ , čo nám dáva časovú zložitosť DTS z definície BC-uniformity.*

*Ďalším predpokladom na tento DTS je topologické usporiadanie výstupu t.j. kód vrchola  $C_n$  sa na výstupe neobjaví skôr ako kódy jeho vstupov.*

**Poznámka 6.3.2.** *BC-uniformita zabezpečuje len to, aby sme s jazykmi nevyšli z triedy  $\mathcal{L}_{RE}$ , teda v návrhoch postupností BO nás príliš neobmedzuje. Preto aj každá "rozumne" navrhnutá postupnosť BO je BC-uniformná.*

**Označenie:**  $\mathcal{U}_{BC} \text{SIZEDEPTH}(S(n), D(n))$  - trieda jazykov. Pre každý jazyk z tejto triedy existuje BC-uniformná postupnosť booleovských obvodov  $\{C_n\}$  akceptujúca daný jazyk pričom  $\text{SIZE}(C_n) = O(S(n))$  a  $\text{DEPTH}(C_n) = O(D(n))$ .

## 6.4 Porovnanie BO a TS

**Veta 6.4.1.** *Ak  $L$  je jazyk akceptovaný jednopáskovým DTS v čase  $T(n)$ , potom existuje BC-uniformná postupnosť BO  $\{C_n\}$  taká, že  $L(\{C_n\}) = L$  a  $\text{SIZE}(C_n) = O(T^2(n))$ .*

**Dôkaz:** Uvažujme DTS  $A$  a jazyk  $L$  zo znenia vety. Zoberme si nejaké slovo  $w = a_1 \dots a_n \in L$ . Ukážeme si ako bude vyzeráť BO  $C_n$  akceptujúci slová dĺžky  $n$  z jazyka  $L$ .

Najskôr binárne zakódujeme všetky symboly vstupnej a pracovnej abecedy a všetky stavy DTS  $A$  t.j. všetkým jednoznačne priradíme nenulový binárny vektor. Obvod  $C_n$  bude mať  $T(n)$  úrovní, pričom na  $i$ -tej úrovni bude "udržiavať" informáciu o  $i$ -tej konfigurácii  $A$  pri výpočte na slove  $w$  nasledovným spôsobom. Každá úroveň bude pozostávať z elementárnych obvodov ( $eBO$ ) reprezentujúcich jedno políčko pracovnej pásky  $A$ , pričom jeden takýto obvod dostane na vstup kód znaku, ktorý je na danom políčku, a kód stavu, v ktorom je  $A$ , ak sa hlava  $A$  nachádza na tomto políčku (obrázok XXXa).

Z mechanizmu výpočtu Turingovho stroja vieme, že zmeny konfigurácií sú len lokálneho charakteru t.j. v jednom kroku výpočtu sa môže zmeniť len jedнопísmenkové okolie pozície hlavy. Podobne aj v našom obvode  $C_n$  jeden elementárny obvod môže ovplyvniť len svojho nasledovníka a jeho susedov, preto sú jednotlivé elementárne obvody pospájané ako na obrázku XXXb.

Každý elementárny obvod bude pracovať nasledovne:

1. ak na vstup dostane kód nejakého znaku a nulový vektor ako kód stavu, znamená to, že hlava  $A$  nie je na políčku, ktoré je reprezentované týmto obvodom a na výstup pošle kód prijatého znaku a nulový vektor ako kód stavu (obrázok XXXa)
2. ak na vstup dostane kód znaku  $a$  a kód stavu  $q$ , tak podľa definície  $\delta$ -funkcie  $A$  pošle na výstup kód nového znaku  $b$ , a podľa pohybu hlavy v  $\delta$ -funkcii pošle príslušnému obvodu v nasledujúcej úrovni kód nového stavu, čím ho informuje o tom, že v nasledujúcom kroku bude hlava nad jeho políčkou a ostatným dvom pošle ako kód stavu nulový vektor (obrázok XXXb...)

<sup>1</sup>BC - Borodin, Cook

<sup>2</sup>pracovný priestor neurčujeme podľa veľkosti vstupu, ale na základe veľkosti vygenerovaného výstupu



Takže celý obvod  $C_n$  bude mať  $T^2(n)$  elementárnych obvodov ( $T(n)$  úrovni pre každú konfiguráciu a v každej úrovni  $T(n)$  obvodov pre každé políčko<sup>3</sup>). Vstupom pre  $C_n$  bude počiatková konfigurácia  $A$ , teda prvý obvod v prvej úrovni dostane na vstup kód počiatkového stavu a kód prvého písmenka slova  $w$ , ďalších  $n - 1$  obvodov dostane na vstup kódy zvyšných  $n - 1$  písmenok slova  $w$  a nulové vektory ako kódy stavov, a ostatné obvody dostanú na vstup nulové vektory ako kódy znakov aj stavov (obrázok XXX). Ďalej bude každý elementárny obvod pracovať ako sme už uviedli, takže jednotlivé úrovne obvodu  $C_n$  budú krok po kroku zodpovedať konfiguráciám vo výpočte  $A$ . Na úrovni  $T(n)$  už iba skontrolujeme, či nejaký  $eBO$  je v akceptačnom stave, ak áno, na výstup dáme jednotku inak nulu.

Z uvedeného vyplýva, že  $DEPTH(C_n) = T(n)$ , a keďže uvažujeme konečný počet stavov, symbolov abecedy a konečný zápis  $\delta$ -funkcie  $DTS A$ , tak dokážeme realizovať elementárny obvod z konečného počtu hradl, z čoho nakoniec plynie, že  $SIZE(C_n) = O(T^2(n))$ .

Týmto sme ukázali, že  $L \subseteq L(\{C_n\})$ . Opačná inklúzia (t.j. že takto zostrojená postupnosť  $BO$  neakceptuje žiadne slovo mimo jazyka  $L$ ) je z konštrukcie zrejmé.  $\square$

**Veta 6.4.2.** *Ak  $L$  je jazyk akceptovaný  $BC$ -uniformnou postupnosťou  $BO \{C_n\}$ , pričom  $SIZE(C_n) = S(n)$ , potom existuje  $DTS$  akceptujúci jazyk  $L$  v čase  $S^3(n)$ .*

**Dôkaz:** Nech  $\{C_n\}$  je postupnosť  $BO$  zo znenia vety. Chceme zostrojiť  $DTS A$  taký, že  $L(A) = L$ . Postupnosť  $\{C_n\}$  je  $BC$ -uniformná, teda poznáme  $DTS A'$ , ktorý vygeneruje kód  $\langle C_n \rangle$ .  $DTS A$  bude na vstupnom slove dĺžky  $n$  pracovať nasledovne:

1.  $A$  si na pásku napíše kód  $BO C_n$  simulovaním  $A'$  so vstupom  $1^n$ . To dokáže v čase  $O(S(n) \cdot \log S(n))$ , lebo kód  $\langle C_n \rangle$  je takejto dĺžky.
2.  $A$  bude postupne ohodnocovať<sup>4</sup> vrcholy  $BO C_n$  tak, že vstupné vrcholy ohodnotí podľa príslušných hodnôt vstupu a hradlá ohodnotí podľa hodnôt vstupov hradla. Ohodnocovanie musí samozrejme prebiehať v topologickom usporiadaní. Na ohodnotenie jedného vrchola potrebuje  $A$  v najhoršom prípade prejsť celú pásku trikrát (dvakrát na nájdenie hodnôt vstupov a tretíkrát na nájdenie a ohodnotenie samotného vrchola). Vrcholov je  $S(n)$ , veľkosť pásky je  $O(S(n) \cdot \log S(n))$ , teda na ohodnotenie všetkých vrcholov  $C_n$  potrebuje  $A$  čas  $O(S^2(n) \cdot \log S(n))$ .

$A$  akceptuje vstupné slovo práve vtedy, keď výstupný vrchol ohodnotí jednotkou. Teda ukázali sme, že  $L \subseteq L(A)$ . Opačná inklúzia je z konštrukcie zrejmé. Obidva kroky výpočtu vykoná  $A$  v čase  $O(S^2(n) \cdot \log S(n))$ , čo je samozrejme v  $O(S^3(n))$ .  $\square$

**Dôsledok 6.4.1.**  $DTIME(Poly) = \mathcal{U}_{BC} SIZE(Poly)$

**Dôkaz:** Z vety 6.4.1 sme dostali  $DTIME(T(n)) \subseteq \mathcal{U}_{BC} SIZE(T^2(n))$ .

Z vety 6.4.2 sme dostali  $\mathcal{U}_{BC} SIZE(T^2(n)) \subseteq DTIME(T(n))$ .

Spojením týchto výsledkov teda dostávame  $DTIME(Poly) = \mathcal{U}_{BC} SIZE(Poly)$ . To znamená, že vo výpočtovom modele  $BO$  vieme polynomiálny sekvenčný čas premieňať na polynomiálny paralelný priestor a opačne.  $\square$

**Veta 6.4.3.** *Ak  $L$  je jazyk akceptovaný  $NTS$ , pracujúcim s jednou vstupnou a jednou pracovnou páskou, v priestore  $S(n) \geq \log n$ , potom existuje  $BC$ -uniformná postupnosť  $BO \{C_n\}$  taká, že  $L(\{C_n\}) = L$  a  $DEPTH(C_n) = O(S^2(n))$ .*

<sup>3</sup>v každej úrovni by stačilo  $S(n)$  (veľkosť pracovného priestoru) obvodov, ale nemáme žiaden predpoklad na priestor  $DTS A$ , vieme však, že určite platí  $S(n) \leq T(n)$

<sup>4</sup>ohodnotiť hradlo znamená na páске označiť symboly príslušného hradla v kóde  $C_n$

**Dôkaz:** Uvažujme jazyk  $L$  a  $NTS$   $A$  zo znenia vety. Zostrojíme  $BO$   $C_n$  akceptujúci slová dĺžky  $n$  z jazyka  $L$ . Zoberme si nejaké slovo  $w \in L$ , kde  $|w| = n$ . Zamyslime sa nad tým v koľkých možných konfiguráciách môže byť  $A$  počas výpočtu na vstupnom slove  $w$ . Ak berieme do úvahy počty stavov, symbolov abecedy, políčok pracovnej pásy, možné pozície hlavy na vstupnej a pracovnej páske, dostaneme, že počet všetkých možných konfigurácií je  $k^{S(n)}$  pre vhodnú konštantu  $k$ .

Uvažujme ďalej reláciu krok výpočtu  $\vdash$  na týchto konfiguráciách. Za predpokladu, že binárne zakódujeme stavy a symboly abecedy, dokážeme reláciu  $\vdash$  reprezentovať booleovskou maticou rozmeru  $k^{S(n)} \times k^{S(n)}$ , čo je pre pevne stanovené  $n$  konečná matica. Predpokladajme, že  $A$  má jednoznačne danú akceptačnú konfiguráciu. Potom otázka, či vstupné slovo  $w$  je patrí do jazyka  $L$ , je vlastne otázka, či počiatočná konfigurácia  $A$  je v relácii  $\vdash$  s akceptačnou konfiguráciou. Relácia  $\vdash$  je reflexívnym a tranzitívnym uzáverom relácie  $\vdash$ . Keďže túto reláciu vieme reprezentovať konečnou booleovskou maticou, z príkladu 6.2.4 plynie, že aj reláciu  $\vdash$  vieme reprezentovať konečnou booleovskou maticou, ktorú dostaneme konečným násobením mocnín matice reprezentujúcej reláciu  $\vdash$ . Z uvedeného príkladu tiež vieme, že reflexívny a tranzitívny uzáver booleovskej matice  $M$  rozmeru  $k^{S(n)} \times k^{S(n)}$  vypočítame na  $BO$  hĺbky rádovo  $\log^2 k^{S(n)} = O(S^2(n))$ .

Teda náš  $BO$   $C_n$  so vstupom  $\langle w \rangle$  najskôr zostrojí maticu relácie  $\vdash$  na vstupnom slove  $w$  (to sa dá konečným  $BO$ , v ktorom je zakódovaná  $\delta$ -funkcia  $A$ ), a potom už spomínaným spôsobom urobí nad touto maticou jej reflexívny a tranzitívny uzáver.  $A$  akceptuje práve vtedy, keď vo výslednej matici je na  $i$ -tom riadku a  $j$ -tom stĺpci 1, kde  $i$ -ty riadok reprezentuje počiatočnú konfiguráciu a  $j$ -ty stĺpec reprezentuje akceptačnú konfiguráciu. Takto sme ukázali, že  $L \subseteq L(\{C_n\})$ , pričom  $DEPTH(C_n) = O(S^2(n))$ . Opačná inklúzia je z konštrukcie zrejmá.  $\square$

**Veta 6.4.4.** Ak  $L$  je jazyk akceptovaný  $BC$ -uniformnou postupnosťou  $BO$   $\{C_n\}$ , pričom  $DEPTH(C_n) = D(n)$ , potom existuje  $DTS$  akceptujúci jazyk  $L$  v priestore  $O(D(n))$ .

**Dôkaz:** Uvažujme jazyk  $L$  a postupnosť  $BO$   $\{C_n\}$  zo znenia vety. Chceme zostrojiť príslušný  $DTS$   $A$ . Nech  $w \in L$  dĺžky  $n$ , čiže kód  $\langle w \rangle$  je akceptovaný  $BO$   $C_n$  z postupnosti  $\{C_n\}$ . Vieme, že  $\{C_n\}$  je  $BC$ -uniformná, takže existuje generátor  $A'$  kódu  $\langle C_n \rangle$  pracujúci v priestore  $\log(SIZE(C_n))$ . Zrejme<sup>5</sup>  $\log(SIZE(C_n)) \leq D(n)$ , takže  $A$  má dostatok priestoru, aby mohol simulovať  $A'$ , ale nemá dosť priestoru na uloženie celého kódu  $\langle C_n \rangle$ . Nemôžeme teda použiť techniku ohodnocovania  $C_n$  ako v dôkaze vety 6.4.2.

$A$  bude postupne ohodnocovať vrcholy  $C_n$  postorder prehľadávaním  $C_n$  od výstupného vrcholu (vstupný vrchol ohodnotí podľa príslušnej časti vstupu  $\langle w \rangle$  a vnútorný podľa hodnôt jeho vstupov).  $A$  však nemá na páske ani toľko priestoru, aby si zapamätal kódy všetkých vrcholov na ceste od výstupného k práve prehľadávanému vrcholu<sup>6</sup>. Preto si bude na páske pamätať len navigačnú cestu od výstupného k práve prehľadávanému vrcholu (obr. XXX). Ak sa chce  $A$  posunúť pri prehľadávaní z jedného vrchola do druhého (t.j. z otca do syna alebo opačne), zakaždým musí spustiť simuláciu  $A'$  a v jeho výstupe nájsť kód príslušného vrchola. Vstupné slovo  $w$  bude  $A$  akceptovať práve vtedy, keď výstupný vrchol dostane hodnotu 1.

Takže  $A$  si bude na páske udržiavať kódy práve prehľadávaného vrcholu a niekoľko málo vrcholov v jeho okolí (aby mohol vrchol ohodnotiť), navigačnú cestu od koreňa k prehľadávanému vrcholu a už známe ohodnotenia vrcholov, ktoré má aktuálne na páske. Na toto potrebuje  $A$  priestor rádovo  $D(n)$ . Na simulovanie  $A'$  potrebuje tiež priestor rádovo  $D(n)$ , takže veľkosť pracovnej pásy  $A$  bude  $O(D(n))$ .

<sup>5</sup>uvažujeme  $BO$  v tvare binárneho stromu

<sup>6</sup>cesta môže byť dlhá maximálne  $D(n)$  a kód každého vrcholu je veľkosti rádovo  $D(n)$ , takže by sme potrebovali priestor rádovo  $D^2(n)$

Týmto sme ukázali, že  $L \subseteq L(A)$ , opačná inklúzia by opäť mala byť z konštrukcie zrejماً.  $\square$

**Dôsledok 6.4.2.**  $NSPACE(Poly) = U_{BC}DEPTH(Poly)$

**Dôkaz:** Z vety 6.4.3 sme dostali  $NSPACE(S(n)) \subseteq U_{BC}DEPTH(S^2(n))$ .

Z vety 6.4.4 sme dostali  $U_{BC}DEPTH(D(n)) \subseteq DSPACE(D(n))$ .

Vieme, že  $DSPACE(Poly) = NSPACE(Poly)$ .

Spojením týchto výsledkov teda dostávame  $NSPACE(Poly) = U_{BC}DEPTH(Poly)$ . To znamená, že vo výpočtovom modele  $BO$  vieme polynomiálny sekvenčný priestor premieňať na polynomiálny paralelný čas a opačne.  $\square$

**Dôsledok 6.4.3.** Výpočtový model  $BO$  patrí do druhej počítačovej triedy.

## 6.5 Iné uniformity pre booleovské obvody

$BC$ -uniformita nie je jedinou možnosťou ako uniformovať booleovské obvody. Niekedy sa nám táto uniformita na naše účely nehodí. Preto teraz ukážeme ďalšie dva spôsoby ako možno uniformovať booleovské obvody, a tie neskôr využijeme pri porovnaní s alternujúcimi Turingovými strojmi.

**Definícia 6.5.1.** *Nech  $\{C_n\}$  je postupnosť  $BO$ . Jej príslušným rozšíreným jazykom prepomení je jazyk  $L_e = \{\langle n, g, p, y \rangle \mid n \in \{1\}^+, g \in \{0, 1\}^+, p \in \{L, R\}^*, |p| \leq \log(SIZE(C_n)), y \in \{x, \wedge, \vee, \neg\} \cup \{0, 1\}^+, \text{ kde}$*

- $n$  udáva<sup>7</sup>, že slovo  $\langle n, g, p, y \rangle$  popisuje obvod  $C_n$
- $g$  je binárne kódované číslo vrchola obvodu  $C_n$
- $p$  je buď  $\varepsilon$  alebo navigačná cesta k vrcholu
- $y$  je typ<sup>8</sup> hradla alebo číslo vrchola

pričom platí:

1. ak  $p = \varepsilon$ , tak hradlo číslo  $g$  je typu  $y \in \{x, \wedge, \vee, \neg\}$
2. ak  $p \in \{L, R\}^+$ , tak  $p$ -predchodca hradla číslo  $g$  má číslo  $y$  t.j. hradlo, do ktorého sa dostaneme z hradla  $g$  po navigačnej ceste  $p$ , má číslo  $y \in \{0, 1\}^+$

}

**Príklad 6.5.1.** Na bližšie pochopenie predchádzajúcej komplikovanej definície si ukážeme časť jazyka  $L_e$  príslúchajúceho k obvodu  $C_4$  (obr.XXX) z nejakej postupnosti  $\{C_n\}$ .

$L_e \supseteq \{$   
 (vrcholy 1-4:)  $\langle 1111, 1, \varepsilon, x \rangle, \langle 1111, 10, \varepsilon, x \rangle, \langle 1111, 11, \varepsilon, x \rangle, \langle 1111, 100, \varepsilon, x \rangle,$   
 (vrchol 5:)  $\langle 1111, 101, \varepsilon, \wedge \rangle, \langle 1111, 101, L, 1 \rangle, \langle 1111, 101, R, 10 \rangle,$   
 (vrchol 6:)  $\langle 1111, 110, \varepsilon, \vee \rangle, \langle 1111, 110, L, 10 \rangle, \langle 1111, 110, R, 11 \rangle,$   
 (vrchol 7:)  $\langle 1111, 111, \varepsilon, \wedge \rangle, \langle 1111, 111, L, 101 \rangle, \langle 1111, 111, R, 110 \rangle,$   
 $\langle 1111, 111, LL, 1 \rangle, \langle 1111, 111, LR, 10 \rangle, \langle 1111, 111, RL, 10 \rangle, \langle 1111, 111, RR, 11 \rangle,$

<sup>7</sup>všimnime si, že  $n$  je kódované unárne

<sup>8</sup> $x$  je označenie pre vstupný vrchol

(vrchol 8):  $\langle 1111, 1000, \varepsilon, \vee \rangle, \langle 1111, 1000, L, 111 \rangle, \langle 1111, 1000, R, 100 \rangle,$   
 $\langle 1111, 1000, LL, 101 \rangle, \langle 1111, 1000, LR, 110 \rangle, \langle 1111, 1000, LLL, 1 \rangle,$   
 $\langle 1111, 1000, LLR, 10 \rangle, \langle 1111, 1000, LRL, 10 \rangle, \langle 1111, 1000, LRR, 11 \rangle$

**Definícia 6.5.2.** *Postupnosť booleovských obvodov  $\{C_n\}$  veľkosti  $S(n)$  a hĺbky  $D(n)$  je*

1.  $\mathcal{U}_E$  - *uniformná, ak existuje DTS  $A$  taký, že  $L(A) = L_e$  a slová  $\langle n, g, p, y \rangle$  akceptuje v čase  $\log(S(n))$*
2.  $\mathcal{U}_{E^*}$  - *uniformná, ak existuje ATS  $A$  taký, že  $L(A) = L_e$  a slová  $\langle n, g, p, y \rangle$  akceptuje v čase  $D(n)$  a priestore  $\log(S(n))$ .*

Všimnime si, že priestor potrebný na akceptovanie jazyka  $L_e$  nezávisí od dĺžky vstupného slova, ale od nejakého podslova vstupného slova. Preto nie je korektné na základe definície  $\mathcal{U}_E$  uniformity písať  $L_e \in \text{DSPACE}(\log(\text{SIZE}(C_n)))$ . Na vyjadrenie tejto situácie zavedieme špeciálne označenie:  $L_e \in \text{DSPACE}(\log(\text{SIZE}(C_n)))$ , a podobne pri  $\mathcal{U}_{E^*}$  uniformite:  $L_e \in \text{ASPACE}(\log(\text{SIZE}(C_n)))$  resp.  $L_e \in \text{ATIMESPACE}(D(n), \log(\text{SIZE}(C_n)))$ .

**Poznámka 6.5.1.** *V ďalšom budeme predpokladať, že v jazyku  $L_e$ , ktorý prislúcha postupnosti BO  $\{C_n\}$ , bude použité "tesné" ("bez dier") očíslovanie vrcholov t.j. ak má obvod  $m$  vrcholov, tak čísla vrcholov budú z množiny  $\{1, \dots, m\}$  resp.  $\{0, \dots, m-1\}$ .*

**Lema 6.5.1.** *Nech  $\{C_n\}$  je postupnosť BO,  $L_e$  je jej príslušný rozšírený jazyk prepojení, a nech  $f(n) \in \Omega(\log(\text{SIZE}(C_n)))$ . Potom štandardný kód  $\langle C_n \rangle$  sa dá vypočítať v  $\text{DSPACE}(f(n))$  práve vtedy, keď  $L_e \in \text{DSPACE}(f(n))$ .*

**Dôkaz:** Dokážeme obe inklúzie:

" $\Rightarrow$ " Máme DTS  $A$  generujúci kód  $\langle C_n \rangle$  v priestore  $f(n)$ . Chceme zostrojiť DTS  $A'$  akceptujúci jazyk  $L_e$  v rovnakom priestore.  $A'$  bude na vstupnom slove  $\langle n, g, p, y \rangle$  pracovať nasledovne:

$A'$  najskôr overí, či v obvode  $C_n$  existuje vrchol s číslom  $g$ . To urobí tak, že bude simulovať  $A$  na vstupe  $1^n$ , až kým nevygeneruje slovo  $\langle g, t, a, b \rangle$  v štandardom kóde. Ak  $p = \varepsilon$ , overí či  $y = t$  (ak nie, tak slovo neakceptuje). Ak  $p \in \{L, R\}^+$ , tak  $A'$  potrebuje overiť, či  $p$ -predchodca vrchola  $g$  má číslo  $y$ . To robí rekurzívne:

- ak  $p = Lp'$ , tak  $A'$  simuluje  $A$ , až kým nevygeneruje  $\langle a, t', a', b' \rangle$ . Ak  $p' = \varepsilon$ , overí typ t.j. či  $t' = y$  (ak nie, tak slovo neakceptuje). Inak opäť simuluje  $A'$ , až kým nevygeneruje  $p'$ -predchodcu vrchola  $a$  atď.
- ak  $p = Rp'$ , tak  $A'$  simuluje  $A$ , až kým nevygeneruje  $\langle b, t', a', b' \rangle$ , a pokračuje podobne ako v prvom prípade.

Na prácu  $A'$  nám stačí priestor  $f(n)$  (v zmysle  $\in$  notácie), lebo v tomto priestore dokážeme simulovať  $A$  a pamätať si jedno slovo štandardného kódu.

" $\Leftarrow$ " Máme DTS  $A'$  akceptujúci jazyk  $L_e$  v  $\in$  priestore  $f(n)$ . Chceme zostrojiť generátor  $A$ . Na vstupnom slove  $1^n$  potrebuje  $A$  pre každý vrchol číslo  $g$  v obvode  $C_n$  zistiť jeho typ  $t$  a jeho vstupy  $a, b$ , a potom zapísať na výstup štvoricu  $\langle g, t, a, b \rangle$ . To bude robiť nasledovne:

$A$  bude postupne pre  $g = 0, 1, 2, \dots$  a  $t = \wedge, \vee, \neq$  simulovať  $A'$  na vstupe  $\langle n, g, \varepsilon, t \rangle$ . Ak nejakú takúto štvoricu  $A'$  akceptuje, znamená to, že v obvode  $C_n$  sa nachádza vrchol s číslom  $g$  a typom  $t$ . Na zistenie vstupov vrchola  $g$  bude  $A$  postupne pre  $a = 0, 1, 2, \dots$  simulovať  $A'$  na vstupe  $\langle n, g, L, a \rangle$ . Ak  $A'$  pre nejaké  $a$  akceptuje, zistili sme číslo vrchola, ktorý je ľavým vstupom vrchola  $g$ . Podobne, simulovaním  $A'$  na vstupe  $\langle n, g, R, b \rangle$  pre

$b = 0, 1, 2, \dots$ , zistíme pravý vstup vrchola  $g$ . Teda  $A$  môže na výstup zapísať  $\langle g, t, a, b \rangle$ . Toto bude  $A$  vykonávať, až kým pre nejaké  $g$   $A'$  neakceptuje ani jednu zo štvoric  $\langle n, g, \varepsilon, t \rangle$  pre všetky  $t \in x, \wedge, \vee, \neq$ . To znamená, že vrchol s takýmto číslom v obvode  $C_n$  nie je a vďaka predpokladu o číslování vrcholov “bez dier” vieme, že  $A$  už vygeneroval kódy všetkých vrcholov v obvode.

Na túto prácu stačí  $A$  priestor  $f(n)$ , lebo v tomto priestore dokáže simulovať  $A'$  a pamätať si nejakú informáciu konštantnej dĺžky o práve generovanom vrchole.

□

Teraz si ukážeme aké vzťahy sú medzi troma uniformitami booleovských obvodov, ktoré sme doteraz spomenuli.

**Veta 6.5.1.** *Medzi uniformitami platia nasledujúce vzťahy:*

1.  $\mathcal{U}_E \text{DEPTH SIZE}(D(n), S(n)) \subseteq \mathcal{U}_{BC} \text{DEPTH SIZE}(D(n), S(n))$
2.  $\mathcal{U}_E \text{DEPTH SIZE}(D(n), S(n)) \subseteq \mathcal{U}_{E^*} \text{DEPTH SIZE}(D(n), S(n))$
3. *Nech  $D(n) \geq \log^2(S(n))$ , potom*  
 $\mathcal{U}_{BC} \text{DEPTH SIZE}(D(n), S(n)) \subseteq \mathcal{U}_{E^*} \text{DEPTH SIZE}(D(n), S(n))$

**Dôkaz:** Dokážeme všetky tri inklúzie:

1. Nech  $\{C_n\}$  je  $\mathcal{U}_E$ -uniformná postupnosť  $BO$ . Z definície vieme, že príslušný jazyk prepojení  $L_e \in \text{DTIME}(\log(S(n)))$ . Zrejme žiadny  $DTS$  nepoužije viac priestoru ako času, takže  $L_e \in \text{DSPACE}(\log(S(n)))$ . Potom z lemy 6.5.1 plynie, že štandardný kód  $\{C_n\}$  sa dá vypočítať v  $\text{DSPACE}(\log(S(n)))$ , teda  $\{C_n\}$  je  $\mathcal{U}_{BC}$ -uniformná.
2. Nech  $\{C_n\}$  je opäť  $\mathcal{U}_E$ -uniformná, teda vieme, že  $L_e \in \text{DTIME}(\log(S(n)))$ . Zrejme  $L_e \in \text{DTIMESPSPACE}(\log(S(n)), \log(S(n)))$ . Keďže  $DTS$  je špeciálnym prípadom  $ATS$ , tak  $L_e \in \text{ATIMESPSPACE}(\log(S(n)), \log(S(n)))$ . Zrejme  $D(n) \geq \log(S(n))$ , teda  $L_e \in \text{DTIMESPSPACE}(D(n), \log(S(n)))$ , čo znamená, že  $\{C_n\}$  je  $\mathcal{U}_{E^*}$ -uniformná.
3. Nech  $\{C_n\}$  je  $\mathcal{U}_{BC}$ -uniformná postupnosť booleovských obvodov, teda jej štandardný kód vieme vygenerovať v  $\text{DSPACE}(\log(S(n)))$ . Z lemy 6.5.1 vieme, že príslušný jazyk prepojení  $L_e \in \text{DSPACE}(\log(S(n)))$ . Zo simulácie  $DTS$  na  $ATS$  (veta 5.3.1) plynie  $L_e \in \text{ATIMESPSPACE}(\log^2(S(n)), \log(S(n)))$ . Z predpokladu  $D(n) \geq \log^2(S(n))$  dostávame  $L_e \in \text{ATIMESPSPACE}(D(n), \log(S(n)))$ , čo znamená, že  $\{C_n\}$  je  $\mathcal{U}_{E^*}$ -uniformná.

□

## 6.6 Porovnanie modelov $BO$ a $ATS$

**Veta 6.6.1.** *Nech  $S(n) \geq n$ , potom platí:*

$$\mathcal{U}_{E^*} \text{DEPTH SIZE}(D(n), S(n)) \subseteq \text{ATIMESPSPACE}(D(n), \log(S(n)))$$

**Dôkaz:** □

**Veta 6.6.2.** *Nech  $S(n) \geq \log n$ , potom platí:*

$$\text{ATIMESPSPACE}(T(n), S(n)) \subseteq \mathcal{U}_E \text{DEPTH SIZE}(T(n), 2^{O(S(n))})$$

**Dôkaz:** K danému *ATS*  $A$  chceme zostrojiť postupnosť  $BO \{C_n\}$  akceptujúcu jazyk  $L(A)$ . Skôr ako začneme konštruovať  $\{C_n\}$  zamyslime sa nad tým ako vyzerá výpočet na *ATS*.

Predstavme si úplný strom konfigurácií *ATS*  $A$  na nejakom vstupnom slove  $w$ . Na výpočet  $A$  sa môžeme pozeráť aj ako na vyhodnocovanie tohto stromu zdola. Každý vrchol reprezentujúci akceptačnú konfiguráciu na konci nejakej vetvy v strome konfigurácií ohodnotíme 1. Ostatné vrcholy na konci vetiev ako aj nekonečné vetvy ohodnotíme 0. Každý vrchol reprezentujúci univerzálnu konfiguráciu ohodnotíme 1 práve vtedy, keď všetci jeho synovia majú hodnotu 1. Vrchol reprezentujúci existenčnú konfiguráciu ohodnotíme 1 práve vtedy, keď aspoň jeden z jeho synov má hodnotu 1. Vstupné slovo  $w$  bude  $A$  akceptovať práve vtedy, keď koreň stromu (vrchol reprezentujúci počiatočnú konfiguráciu  $A$  na slove  $w$ ) ohodnotíme 1.

Uvažujme teraz *ATS*  $A$  v nasledovnom normálovom tvare:

$A$  pracuje s binárnou vstupnou aj pracovnou abecedou,  $A$  používa rýchly prístup na vstupnú pásku (t.j.  $A$  má špeciálnu pásku dĺžky  $\log n$ , na ktorú keď v binárnom kódovaní zapíše číslo  $j$ , dostane sa k  $j$ -temu symbolu na vstupe), úplný strom konfigurácií je binárny a v každej vetve tohto stromu môže  $A$  čítať vstup iba raz, a to na konci (t.j.  $A$  má špeciálne čítacie stavy  $q^0$  a  $q^1$ , do ktorých sa dostane na konci výpočtu vetvy, pričom v stave  $q^0$  očakáva na vstupe<sup>9</sup> 0 (ak je na vstupe naozaj 0, tak akceptuje, ak je tam 1, neakceptuje) a v  $q^1$  očakáva na vstupe 1).

Pozrime sa teraz na úplný strom konfigurácií  $A$ . Má  $T(n)$  úrovní, pričom v nulte úrovni bude jediný vrchol reprezentujúci počiatočnú konfiguráciu. Hľadaný  $BO C_n$  akceptujúci vstupné slovo  $w$  dĺžky  $n$  bude vyzeráť veľmi podobne. Každému vrcholu v úplnom strome konfigurácií zodpovedá jeden vrchol v obvode  $C_n$ .

Vrcholu na úrovni  $t$ , ktorý reprezentuje konfiguráciu<sup>10</sup>  $c$  v strome konfigurácií zodpovedá v obvode  $C_n$  vrchol s číslom  $f(t, c)$ , kde  $f : N \times N \rightarrow N$  je funkcia zachováajúca tesné očíslovanie vrcholov, pričom

- ak  $c$  je univerzálna konfigurácia, tak vrchol  $f(t, c)$  je typu  $\wedge$
- ak  $c$  je existenčná konfigurácia, tak vrchol  $f(t, c)$  je typu  $\vee$
- ak  $c$  je čítacia konfigurácia so stavom  $q^0$  a na špeciálnej páske je zapísané číslo  $j$ , tak vrchol  $f(t, c)$  je typu<sup>11</sup>  $\neg$  a jeho vstupom je  $j$ -ty bit vstupného slova  $w$
- ak  $c$  je čítacia konfigurácia so stavom  $q^1$  a na špeciálnej páske je zapísané číslo  $j$ , tak vrchol  $f(t, c)$  je typu  $I$  a jeho vstupom je  $j$ -ty bit vstupného slova  $w$
- ak  $c$  je akceptačná konfigurácia ale nie je čítacia, tak vrchol  $f(t, c)$  je typu "1"
- ak  $c$  je odmietacia konfigurácia ale nie je čítacia, tak vrchol  $f(t, c)$  je typu "0"

Vstupnými vrcholmi pre vrchol  $f(t, c)$  sú vrcholy s číslami  $f(t+1, c_1)$  a  $f(t+1, c_2)$  pričom platí  $c \vdash_A c_1$  a  $c \vdash_A c_2$  (obr.XXX).

Takto zostrojený obvod  $C_n$  pracuje presne tak ako výpočet *ATS*, ktorý sme opísali na začiatku dôkazu. Teda príslušná postupnosť  $BO \{C_n\}$  zrejme akceptuje jazyk  $L(A)$ .

<sup>9</sup>na jednej pozícii vstupu určenej číslom na špeciálnej páske

<sup>10</sup>Uvažujeme tu konfiguráciu bez vstupu, lebo každý FORKovaný výpočet používa len jeden symbol zo vstupu, takže vstup v konfigurácii de facto nepotrebujeme. Na druhej strane veľmi užitočná v konfigurácii je informácia o špeciálnej páske určujúcej ktorý symbol sa má čítať na konci výpočtu.

<sup>11</sup>Ak  $A$  očakáva na vstupe 0 a naozaj tam 0 je, tak hradlo  $\neg$  dostane na vstup 0 a na výstup pošle 1, čo signalizuje, že  $A$  očakával správne. Naopak ak je na vstupe 1, tak hradlo  $\neg$  pošle na výstup 0, čo signalizuje, že  $A$  sa mýlil. Analogicky to funguje pre stav  $q^1$  a hradlo  $I$ .

Pozrime sa ešte na miery zložitosti obvodu  $C_n$ . Obvod sme zostrojili “jedna k jednej” vzhľadom na úplný strom konfigurácií  $ATS$   $A$ . Z toho plynie, že  $DEPTH(C_n) = T(n)$ .  $SIZE(C_n)$  zodpovedá počtu vrcholov v strome konfigurácií. Keďže do konfigurácií nezahŕňame vstup, tak počet všetkých možných konfigurácií je  $2^{O(S(n))}$ . V zásade sa môže stať, že v úplnom strome konfigurácií máme v dvoch rôznych vetvách vrcholy reprezentujúce rovnakú konfiguráciu, čo by mohlo spôsobiť, že počet vrcholov, a teda aj veľkosť  $C_n$  by bola rádovo väčšia ako  $2^{O(S(n))}$ . Ak ale uvažujeme výpočet  $ATS$  ako FORKovanie procesov, tak nie je potrebné, aby boli spustené dva rovnaké procesy. Takže ak chce  $ATS$  spustiť proces, ktorého kópia už beží, tak tento duplicitný proces iba odkážeme na rovnaký už bežiaci proces. To v booleovskom obvode znamená jedno prepojenie medzi hradlami. Takže v konečnom dôsledku môžeme uvažovať opäť akýsi normálový tvar  $ATS$ , ktorý bude mať v úplnom strome konfigurácií všetky konfigurácie disjunktné, a teda  $SIZE(C_n) = 2^{O(S(n))}$ .  $\square$

## Kapitola 7

# Parallel Random Access Machine (*PRAM*)

Model, ktorý predstavujeme v tejto kapitole, je najviac podobný reálnym paralelným architektúram. Ide o sústavu veľa (teoreticky nekonečne) samostatných výpočtových jednotiek, ktoré nazývame procesory. Každý procesor má svoju privátnu pamäť a sadu inštrukcií podobných tým, ktoré poznáme z assembleru. Jednotlivé procesory spolu komunikujú cez spoločnú zdieľanú pamäť. Tento model je akousi automatovou analogiou k modelu *PCGS*.

Nebudeme sa zaoberať porovnávaním generatívnej sily *PRAMu* s modelmi Chomského hierarchie, pretože už jeden samostatný procesor (*RAM*) má generatívnu silu Turingovho stroja. Zameriame sa skôr na využitie sily tohto modelu na rýchle paralelné riešenie problémov. V závere porovnáme model *PRAM* s booleovskými obvodmi.

### 7.1 Definície a označenia

Skôr ako si zdefinujeme výpočtový model *PRAM*, s ktorým budeme ďalej pracovať, povieme si niečo o jeho základnej výpočtovej jednotke, ktorou je *RAM*.

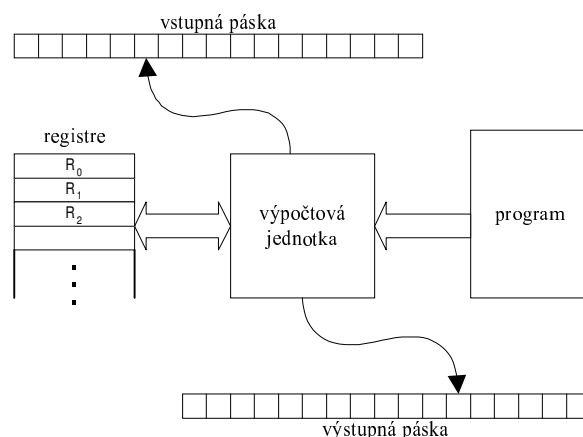
#### 7.1.1 *RAM*

**Definícia 7.1.1.** *RAM (Random Access Machine) je výpočtový model pozostávajúci z výpočtovej jednotky s pevne daným programom, jednej vstupnej a jednej výstupnej páske a neobmedzeného počtu registrov  $R_0, R_1, R_2, \dots$ , pričom v jednom registri môže uchovávať ľubovoľne veľké celé číslo (obr. 7.1). Program výpočtovej jednotky je postupnosť jednoduchých<sup>1</sup> inštrukcií, ktoré sú uvedené v tabuľke. Výpočet začína prvou inštrukciou a končí inštrukciou *HALT*.*

Model *RAM* sa dá zdefinovať viacerými spôsobmi. Uvedenú definíciu môžeme, bez zmeny výpočtovej sily a zložitosti, modifikovať tak, že vstup nebude zadaný na vstupnej páske, ale v špeciálnych registroch, pričom v jednom z nich bude zadaná veľkosť vstupu  $n$  a v ďalších  $n$  registroch bude bit po bite zadaný samotný vstup.

<sup>1</sup>V sade inštrukcií máme len jednoduché násobenie t.j. obsah registra krát nejaká konštanta. Nemáme tu násobenie medzi registrami, lebo to by dalo modelu *RAM* príliš veľkú silu





Obrázok 7.1: Model *RAM*

inštrukcia	popis
<i>READ</i>	prečítaj nasledujúci symbol zo vstupu a zapíš ho do registra $R_0$
<i>WRITE</i>	obsah registra $R_0$ zapíš na výstup
<i>STORE</i> $R_i$	obsah registra $R_0$ zapíš do registra $R_i$
<i>COPY</i> $R_i$	skopíruj obsah registra $R_i$ do registra $R_0$ ( $R_0 \leftarrow [R_i]$ )
<i>CONST</i> $c$	do registra $R_0$ zapíš hodnotu $c$
<i>ADD</i> $R_i$	$R_0 \leftarrow [R_0] + [R_i]$
<i>SUB</i> $R_i$	$R_0 \leftarrow [R_0] - [R_i]$
<i>MULT</i> $c$	$R_0 \leftarrow [R_0] \cdot c$
<i>DIV</i> $c$	$R_0 \leftarrow [R_0] / c$
<i>IFZERO</i> $i$	ak register $R_0$ obsahuje 0, tak pokračuj inštrukciou $i$
<i>GOTO</i> $i$	pokračuj inštrukciou $i$
<i>HALT</i> <sub>accept</sub>	ukonči výpočet a akceptuj
<i>HALT</i> <sub>reject</sub>	ukonči výpočet a neakceptuj

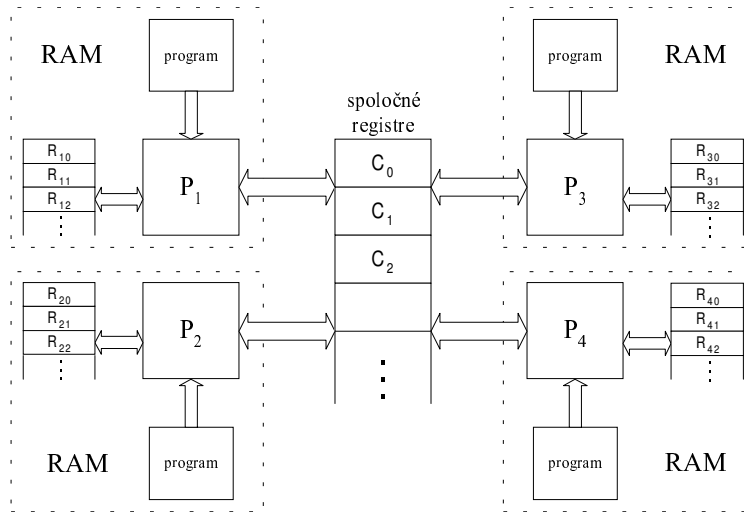
Tabuľka 7.1: Zoznam jednoduchých inštrukcií *RAM*<sub>*u*</sub>

Ďalšou modifikáciou môže byť rovnakým spôsobom upravený výstup, teda nie na výstupnej páske, ale opäť v (na to určených) registroch.

### 7.1.2 *PRAM*

Prirodzeným rozšírením modelu *RAM* v paralelných výpočtoch je model *PRAM*, ktorý v sebe integruje viacero *RAM*<sub>*ov*</sub> komunikujúcich prostredníctvom zdieľanej pamäte

**Definícia 7.1.2.** *PRAM (Parallel Random Access Machine)* je výpočtový model pozostávajúci z neobmedzeného počtu *RAM* procesorov označených  $P_0, P_1, P_2, \dots$  a neobmedzeného počtu spoločných (zdieľaných) registrov  $C_0, C_1, C_2, \dots$ . Každý procesor  $P_i$  má svoje identifikačné číslo (*index*), má svoju vlastnú pamäť t.j. neobmedzenú sadu registrov  $R_{i,0}, R_{i,1}, R_{i,2}, \dots$  a inštrukcie na priamy alebo nepriamy prístup (*read/write*) do spoločnej pamäte. Základná sada inštrukcií je zobrazená v tabuľke. Procesory sú zosynchronizované podľa globálnych hodín, teda inštrukcie



Obrázok 7.2: Model *PRAM*

v jednotlivých procesoroch sú vykonávané v taktach.  
 Vstup  $x \in \{0,1\}^n$  je zadaný nasledovne:

- v registri  $C_0$  je uložená dĺžka vstupu  $n$
- v registri  $C_i$  je uložený  $i$ -ty bit vstupu  $x_i$

Výstup  $y \in \{0,1\}^m$  je uložený v rovnakej forme.

inštrukcia	popis
$R_i \leftarrow [R_j]$	skopíruj obsah registra $R_j$ do registra $R_i$
<i>IDENT</i>	do registra $R_0$ zapíš číslo procesora
<i>CONST c</i>	do registra $R_0$ zapíš hodnotu $c$
<i>ADD</i> $R_i$	$R_0 \leftarrow [R_0] + [R_i]$
<i>SUB</i> $R_i$	$R_0 \leftarrow [R_0] - [R_i]$
<i>MULT</i> $c$	$R_0 \leftarrow [R_0].c$
<i>DIV</i> $c$	$R_0 \leftarrow [R_0]/c$
<i>IFZERO</i> $i$	ak register $R_0$ obsahuje 0, tak pokračuj inštrukciou $i$
<i>GOTO</i> $i$	pokračuj inštrukciou $i$
<i>HALT</i>	ukončí výpočet

Tabuľka 7.2: Zoznam jednoduchých inštrukcií *PRAMu*

Pri práve zadanom modeli sa ukazujú dva závažné problémy:

1. Nemôžeme predpokladať, že potenciálne nekonečne veľa procesorov sa bude podieľať na danom výpočte, a teda že budú všetky aktívne. Každý výpočet potrebuje isté množstvo procesorov, ktoré je závislé od vstupu resp. jeho dĺžky. Preto jeden z procesorov, označený ako  $P_0$ , má význačné postavenie, je to akýsi “generál”.  $P_0$  zapíše do špeciálneho registra

$C_{-1}$  maximálne číslo aktívneho procesora t.j. všetky procesory s menším indexom sú aktívne počas výpočtu. Teda najskôr je aktívny  $P_0$  a ostatné čakajú, kým zapíše hodnotu maximálneho indexu procesora.<sup>2</sup> Výpočet skončí, keď skončí procesor  $P_0$  t.j. vykoná HALT.

- Musíme vyriešiť konflikty pri viacnásobnom prístupe do spoločnej pamäte. Každá inštrukcia je vykonávaná v troch fázach. V prvej fáze je povolený prístup (ak treba) do spoločnej pamäte pre čítanie, potom sa vykoná príslušný výpočet pre danú inštrukciu, a nakoniec je povolený prístup (ak treba) do spoločnej pamäte pre zápis. Týmto sme oddelili prístup pre čítanie od prístupu pre zápis. Treba ešte vyriešiť situáciu, keď viac procesorov naraz pristupuje do spoločnej pamäte. Poznáme tri základné typy modelu *PRAM*:

- *CRCW – PRAM* (Concurrent Read Concurrent Write)

Dovolíme procesorom súčasné čítanie aj súčasný zápis do spoločnej pamäte (registra). Tento typ má tri verzie:

- *PRIORITY* : Ak chce do jedného registra zapisovať viacero procesorov, zápis vykoná len procesor s najmenším číslom z procesorov, ktoré žiadali o zápis.
- *COMMON* : Ak chce do jedného registra zapisovať viacero procesorov, zápis sa uskutoční len vtedy, ak všetky procesory chcú zapísať rovnakú hodnotu. V opačnom prípade sa výpočet zasekne.
- *ARBITRARY* : Ľubovoľný<sup>3</sup> z procesorov žiadajúcich o zápis zapíše svoju hodnotu do registra

- *CREW – PRAM* (Concurrent Read Exclusive Write)

Dovolíme<sup>4</sup> procesorom súčasné čítanie spoločného registra, ale zapisovať do spoločného registra môže vždy len jeden procesor.

- *EREW – PRAM* (Exclusive Read Exclusive Write)

Čítať a zapisovať do spoločného registra môže vždy len jeden procesor.

## 7.2 Miery zložitosti

Pri modele *RAM* uvažujeme tieto jednotkové miery zložitosti:

- *TIME*  $T(n)$  = počet vykonaných inštrukcií
- *SPACE*  $S(n)$  = počet použitých registrov

Z definície týchto mier je zrejmé, že neuvažujú veľkosť dát (čísel), s ktorými inštrukcie pracujú, teda pracovať s veľkými číslami je rovnako "drahé" ako pracovať s malými číslami. Z toho plynie, že použitie jednotkovej miery nie je vhodné napr. pri porovnávaní *RAM* s Turingovým strojom. V takýchto prípadoch uvažujeme tzv. logaritmickú mieru, ktorá zohľadňuje veľkosti čísel pri jednotlivých operáciach. Logaritmická miera sa však používa len zriedka, lebo v praxi máme aj tak obmedzenú veľkosť aj počet registrov.

<sup>2</sup>inou možnosťou riešenia tohoto problému by bolo zavedenie inštrukcie FORK, ktorou ľubovoľný aktívny procesor môže aktivovať ďalšie, pričom na začiatku bude aktívny len procesor  $P_0$

<sup>3</sup>je to jediný nedeterministický prístup v modeli *PRAM* t.j. môže sa stať, že pri opakovanom výpočte na rovnakom vstupe dostaneme rozdielne výstupy

<sup>4</sup>to znamená, že program (postupnosť inštrukcií) pre jednotlivé procesory musí spĺňať túto podmienku

Pri modele *PRAM* uvažujeme tieto jednotkové miery zložitosti:

- *TIME*  $T(n)$  = počet krokov (taktov) procesora  $P_0$  pri výpočte na vstupe dĺžky  $n$
- *PROCESSORS*  $P(n)$  = maximálny počet aktívnych procesorov počas výpočtu na vstupe dĺžky  $n$

Model *PRAM* nemôže generovať príliš veľké (superpolynomiálne) čísla. To znamená, že po  $T(n) \leq \log n$  taktoch nebude ani v jednom registri zapísané číslo s viac ako  $O(T(n))$  bitmi. Z toho plynie, že pri výpočte s časovou zložitou  $T(n)$  na  $P(n)$  procesoroch *PRAM* nespotrebuje na uloženie dát viac ako  $O(P(n) \cdot T^2(n))$  bitov. Máme teda adekvátnu mieru pre pamäťové nároky (*SPACE*).

### 7.3 Generatívna sila modelu *PRAM*

Najskôr si ukážeme niekoľko príkladov výpočtov na modele *PRAM*.

**Príklad 7.3.1.** Chceme vypočítať maximum z postupnosti  $n$  zadaných čísel  $x_1, \dots, x_n$  na modele *COMMON CRCW – PRAM*.

Vstup bude zadaný nasledovne:  $C_0 \leftarrow n, C_1 \leftarrow x_1, \dots, C_n \leftarrow x_n$ .

Chceme<sup>5</sup> výstup:  $[C_0] = \max\{x_1, \dots, x_n\}$ .

Na výpočet použijeme  $n^2$  procesorov ozn.  $P_{i,j}$ , kde  $i, j \in \{1, \dots, n\}$  plus procesor  $P_0$ :

1. Každý procesor  $P_{i,1}$  vykoná:  $C_{n+i} \leftarrow 0$
2. Každý procesor  $P_{i,j}$  vykoná<sup>6</sup>: *if*  $[C_i] < [C_j]$  *then*  $C_{n+i} \leftarrow 1$   
Uvedomme si, že tu nenastane žiadny konflikt pri súčasnom zápise viacerých procesorov do toho istého registra, pretože všetky procesory, ktoré budú chcieť zapisovať do registra  $C_{n+i}$ , budú chcieť zapísať 1.
3. Každý procesor  $P_{i,1}$  vykoná: *if*  $[C_{n+i}] = 0$  *then*  $C_0 \leftarrow [C_i]$   
Po kroku 2 je medzi registrami  $C_{n+1}, \dots, C_{2n}$  jediný s nulovou hodnotou a platí  $[C_{n+i}] = 0$  práve vtedy, keď  $x_i = \max\{x_1, \dots, x_n\}$

Ako vidieť, vďaka použitiu polynomiálneho počtu procesorov, sa nám podarilo nájsť maximum z číselnej postupnosti v konštantnom čase. Pri sekvenčných modeloch na to potrebujeme lineárny čas.

**Príklad 7.3.2.** Opäť chceme vypočítať maximum ako v predchádzajúcom príklade, ale tentoraz na modele *EREW – PRAM*.

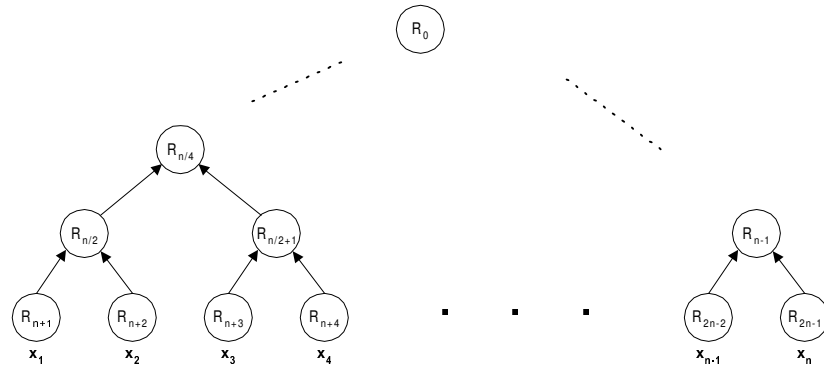
Vstup je zadaný tak isto ako v predchádzajúcom príklade. Budeme<sup>7</sup> porovnávať dvojice registrov, pričom výsledky zapíšeme do nových registrov, ktoré budeme opäť po dvojiciach porovnávať atď. takže dostaneme binárny porovnávací strom (obr.7.3). Keďže používame stále nové a nové registre a každý register má na starosti jeden procesor, tak procesory naraz nečítajú ani nezapisujú do toho istého registra.

Procesorovú zložitost' sme zmenšili  $P(n) = n$ , ale časová zložitost' vzrástla  $T(n) = \log n$ .

<sup>5</sup> $[C]$  označuje obsah registra  $C$ ,  $C \leftarrow a$  označuje zápis čísla  $a$  do registra  $C$

<sup>6</sup>procesor nemá k dispozícii takú silnú inštrukciu, ale iste si vieme predstaviť jej simuláciu pomocou konečného počtu *PRAM*-inštrukcií

<sup>7</sup>rovnakú myšlienku možno použiť pri úlohe sčítania  $n$  čísel



Obrázok 7.3: Výpočet maxima na *EREW* – *PRAM*

**Príklad 7.3.3.** Ukážeme si, ako efektívne vieme triediť postupnosť čísel  $x_1, \dots, x_n$  na modele *CREW* – *PRAM*.

Vo vstupných registroch  $C_1, \dots, C_n$  sú hodnoty  $x_1, \dots, x_n$ . Chceme, aby po skončení výpočtu boli v registroch  $C_1, \dots, C_n$  čísla zo vstupu v utriedenom poradí.

Na výpočet použijeme  $n^2$  procesorov ozn.  $P_{i,j}$ .

1. Každý procesor  $P_{i,j}$  vykoná:  $\text{if } [C_i] < [C_j] \text{ then } C_{i,j} \leftarrow 1 \text{ else } C_{i,j} \leftarrow 0$
2. Každých  $n$  procesorov  $P_{i,1}, \dots, P_{i,n}$  sa bude podieľať na výpočte<sup>8</sup>:  $C_{n+i} \leftarrow 1 + \sum_{j=1}^n [C_{i,j}]$
3. Každý procesor  $P_{i,1}$  vykoná:  $C_{C_{n+i}} \leftarrow [C_i]$

Časová zložitosť *TIME* je  $T(n) = O(\log n)$ , lebo prvý a tretí krok výpočtu trvá konštantný čas a na druhý potrebujeme logaritmický čas, počet procesorov je  $P(n) = n^2$ .

Teraz ľahko vidíme, že problém triedenia je v  $\mathcal{NC}$ , teda ho vieme efektívne paralelne riešiť.

Jednotlivé modely *PRAM* sú medzi sebou relatívne ekvivalentné, takže je v zásade jedno, ktorý používame. Tento poznatok využijeme pri porovnaní *PRAM* s booleovskými obvodmi. Teraz si ukážeme porovnanie medzi modelmi *EREW* a *CRCW*.

**Veta 7.3.1.**  $\mathcal{L}(\text{EREW} - \text{PRAM}) = \mathcal{L}(\text{CRCW} - \text{PRAM})$

**Dôkaz:** Inklúziu  $\subseteq$  netreba dokazovať, pretože z definícií jednotlivých modelov vyplýva, že *EREW* je špeciálnym prípadom *CRCW*. Dokážeme opačnú inklúziu.

Chceme simulovať *CRCW* na *EREW*. Treba vyriešiť situáciu, keď viacero procesorov chce naraz zapisovať do toho istého registra t.j. treba z nich vybrať jeden, ktorý svoju hodnotu do registra naozaj zapíše. Budeme z nich vyberať procesor s najmenším číslom (takto vyriešime naraz všetky tri verzie modelu *CRCW*).

Pod každým registrom  $C_i$  budeme mať binárny strom pozostávajúci z nových registrov obsahujúcich čísla procesorov. V listoch sú čísla všetkých aktívnych procesorov. Na začiatku každý procesor pozná svoju pozíciu v tomto strome. Nie každý aktívny procesor chce v danom okamihu zapisovať do  $C_i$ . Budeme postupne po dvojiciach porovnávať obsahy registrov v tomto strome. Ak je procesor ľavý (t.j. s menším číslom) a chce zapisovať do  $C_i$ , tak postúpi o úroveň vyššie

<sup>8</sup>sčítať  $n$  čísel je podobný problém ako nájsť maximum z  $n$  čísel (príklad 7.3.2)

(t.j. zapíše do príslušného registra svoje číslo).

Ak je procesor pravý, tak “skúsi”, či jeho ľavý sused postúpil.<sup>9</sup> Ak áno, tak v ďalšom už nechce zapisovať do  $C_i$ , ak nie, tak postúpi o úroveň vyššie. Toto sa vykonáva až ku koreňu stromu, teda ku koreňu sa dostane len jeden procesor (s najmenším číslom z aktívnych procesorov, ktoré chceli zapisovať do  $C_i$ ) a ten zapíše svoju hodnotu do registra  $C_i$ .

Čítanie bude fungovať podobne, iba s tým rozdielom, že výpočet na strome pod každým registrom sa bude vykonávať opačným smerom, aby každý z procesorov, ktorý mal záujem čítať, sa dostal k  $C_i$ .

Keďže binárne stromy pod registrami majú výšku  $\log P(n)$ , tak čítanie registra a zápis do registra sa z jedného kroku v modeli *CRCW* predĺži na  $O(\log P(n))$  krokov v modeli *EREW*, a teda čas *EREW* modelu sa oproti *CRCW* zhorší  $O(\log P(n))$  násobne. To je ale v celku dobrá simulácia.

□

## 7.4 Porovnanie modelov *BO* a *PRAM*

**Veta 7.4.1.** *Nech  $\{C_n\}$  je BC-uniformná postupnosť booleovských obvodov počítajúca funkciu  $f_n : \{0, 1\}^n \rightarrow \{0, 1\}^m$  taká, že  $DEPTH(C_n) = (\log n)^{O(1)}$  a  $SIZE(C_n) = n^{O(1)}$ . Potom existuje *CREW-PRAM*, ktorý počíta funkciu  $f_n$  v čase  $T(n) = (\log n)^{O(1)}$  na  $P(n) = n^{O(1)}$  procesoroch.*

**Dôkaz:** Základná idea dôkazu spočíva v tom, že každé hradlo booleovského obvodu (*BO*) bude simulované jedným procesorom a každé spojenie medzi hradlami (t.j. vstup resp. výstup) v *BO* bude reprezentované jedným spoločným registrom. Potom každý procesor v príslušnom takte, podľa hĺbky simulovaného hradla v *BO*, načíta obsahy registrov prislúchajúce k vstupom simulovaného hradla, vykoná daný výpočet podľa typu hradla, a následne zapíše výsledok do registra prislúchajúceho k výstupu hradla (pozri obrázok 7.4, kde kružnice predstavujú procesory a obdĺžniky registre). Tento register je zároveň vstupným registrom pre nejaký iný procesor, ktorý simuluje ďalšie hradlo v *BO*.

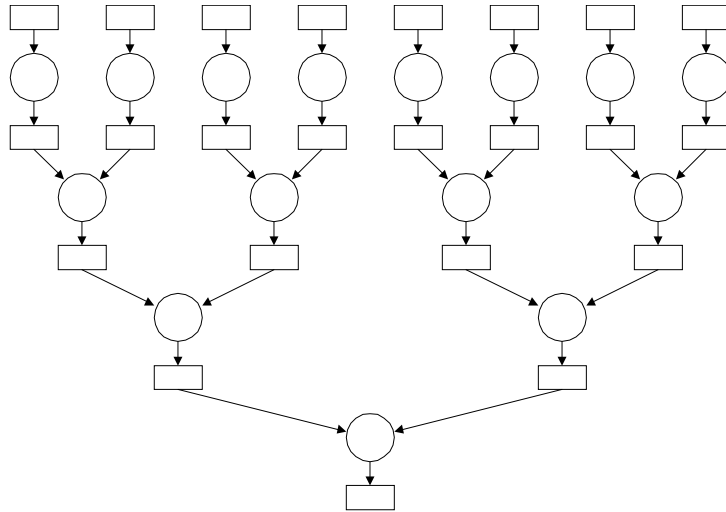
Každý procesor  $P_i$ , ktorý simuluje nejaké hradlo v *BO*, bude mať v spoločnej pamäti vyhradené štyri registre  $C_{i,IN1}, C_{i,IN2}, C_{i,OUT}, C_{i,TYP}$ , pričom v prvých dvoch budú zapísané adresy dvoch vstupných registrov, do tretieho procesor  $P_i$  zapíše výstup a vo štvrtom bude zapísaný typ simulovaného hradla. Procesor simulujúci vstupné hradlo samozrejme vystačí z jedným registrom pre adresu vstupu.

Obsahy registrov  $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$  závisia od daného *BO*, a pred samotnou simuláciou ich podľa neho musíme najskôr naplniť. Pripomeňme si, čo to znamená, že máme daný *BC-uniformný* booleovský obvod. Znamená to, že poznáme *DTS A* pracujúci s jednou vstupnou, jednou pracovnou a jednou výstupnou páskou, ktorý na vstupe  $1^n$  zapíše na výstupnú pásku kód *BO*  $C_n$ , pričom pracovnú pásku má obmedzenú na priestor  $\log SIZE(C_n)$ .

Chceme zostrojiť *PRAM* simulujúci postupnosť *BO*  $\{C_n\}$ . To znamená, že pre nejaký vstup dĺžky  $n$  potrebujeme najskôr odsimulovať výpočet *DTS A*, ktorý vygeneruje kód *BO*, potom ho dekódovať (t.j. správne naplniť registre  $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$ ), a nakoniec spustiť samotnú simuláciu výpočtu *BO*.

---

<sup>9</sup>To môžeme zabezpečiť tak, že každý postup o úroveň vyššie bude prebiehať v troch krokoch. V prvom ľaví z dvojice buď nechcú zapisovať a nepostupujú alebo postúpia, ak chcú zapisovať, zatiaľ čo praví čakajú t.j. vykonajú inštrukciu NOP. V druhom kroku praví, ak chcú zapisovať do  $C_i$ , tak čítajú obsah registra zodpovedajúceho vyššej úrovni. V treťom kroku praví, ak zistili, že ľaví nezapísali svoje číslo do registra vyššej úrovne, tak tam zapíšu svoje číslo.



Obrázok 7.4: Simulácia *BO* na modeli *PRAM*

- Simulácia výpočtu *DTS A*:

Čo vieme o *DTS A*? Ak uvažujeme, že *A* pracuje na vstupe dĺžky  $n$  v priestore  $S(n) = \log \text{SIZE}(C_n)$ , tak počet všetkých možných konfigurácií *A* je  $k^{S(n)} \cdot n \cdot S(n)$  pre nejakú konštantu  $k$ , čo je rádovo  $(\text{SIZE}(C_n))^{O(1)}$  čiže podľa predpokladu  $n^{O(1)}$ . Očíslujme si jednotlivé konfigurácie  $1, 2, \dots, r$ . Nech<sup>10</sup> počiatočná konfigurácia má číslo 1 a koncová má číslo  $r$ . Podľa predpokladu vety máme dostatok procesorov, aby sme každej konfigurácii priradili jeden procesor. Každému procesoru pridáme dva registre zo spoločnej pamäte, jeden<sup>11</sup> na výstup ( $C_{j,OUT}$ ) a druhý na číslo registra ( $C_{j,REG}$ ).

V prvom kroku každý procesor  $P_j$  prislúchajúci  $j$ -tej konfigurácii odsimuluje jeden krok výpočtu *DTS A* z tejto konfigurácie, pričom do prvého registra zapíše výstup zodpovedajúci tomuto kroku a do druhého zapíše číslo konfigurácie, do ktorej sa týmto krokom *A* dostal<sup>12</sup>. Keďže sme odsimulovali jeden krok *A* z každej konfigurácie, vyčerpali sme tým všetky možnosti  $\delta$ -funkcie, takže  $\delta$ -funkciu už simulovať nebudeme. V ďalšom chceme jednotlivé kroky výpočtu *A* vhodne “pospájať”, aby spolu tvorili celý výpočet *A*, čím by sme dostali výsledný kód  $\langle C_n \rangle$ .

V každom ďalšom kroku procesor  $P_j$  vykoná:

1.  $C_{j,OUT} \leftarrow [C_{j,OUT}] \cdot [C_{C_{j,REG},OUT}]$ , kde  $\cdot$  je zreťazenie<sup>13</sup> obsahov registrov
2.  $C_{j,REG} \leftarrow [C_{C_{j,REG},REG}]$

Každý procesor  $P$  načíta registre procesora prislúchajúceho ku konfigurácii, ktorej číslo má procesor  $P$  vo svojom druhom registri. Prvé (výstupné) registre zreťazí a toto zreťazenie zapíše ako novú hodnotu výstupného registra. Do druhého registra zapíše číslo konfigurácie, ktoré načítal (obr.7.5). Každý procesor  $P_j$  teraz reprezentuje dva kroky výpočtu

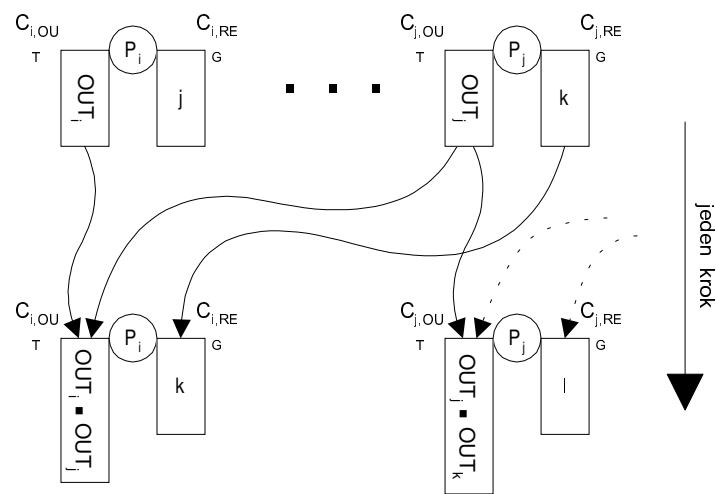
<sup>10</sup>takýto predpoklad môžeme urobiť, lebo pre daný vstup poznáme počiatočnú konfiguráciu a môžeme sa dohodnúť na jednej koncovkej konfigurácii pre všetky výpočty, z ktorej sa už nedá dostať do inej konfigurácie

<sup>11</sup>POZOR: Nemýliť si s vyššie uvedeným rovnako nazvaným registrom, použitým pri samotnej simulácii *BO*

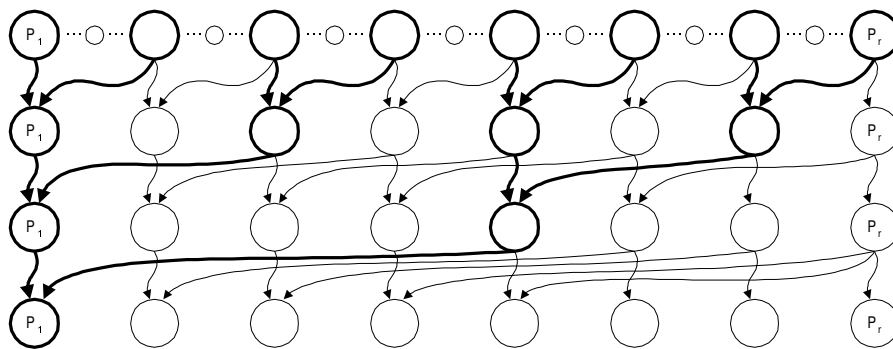
<sup>12</sup>tieto zápisy sú jednoznačné, keďže simulujeme deterministický stroj

<sup>13</sup>obsahmi registrov sú čísla v binárnom zápise, takže zreťazenie znamená nasledujúcu aritmetickú operáciu nad registrami:  $C_{j,OUT} \leftarrow C_{j,OUT} + 2^{|C_{j,REG},OUT|} \cdot C_{C_{j,REG},OUT}$

$A$  začínajúci konfiguráciou číslo  $j$ . Po ďalšom kroku  $PRAMu$  bude každý procesor  $P_j$  reprezentovať štyri kroky výpočtu  $A$  s výnimkou tých, ktoré už vo svojom druhom registri majú číslo koncovej konfigurácie, z ktorej sa už nedá dostať. Toto sa opakuje až kým procesor  $P_1$  nemá vo svojom druhom registri hodnotu  $r$ . Vtedy sa simulácia generovania kódu zastaví<sup>14</sup> a  $P_1$  bude mať vo svojom prvom registri celý výstup  $A$ , teda kód  $BO C_n$ . Na obrázku 7.6 sú vyznačené procesory prislúchajúce ku konfiguráciám, v ktorých sa nachádza  $DTS$  pri generovaní daného kódu. Hrube vyznačené procesory sú pre nás významné z pohľadu generovania kódu. Tenko vyznačené procesory nám už nepomôžu k vygenerovaniu kódu, napriek tomu stále vykonávajú svoj program, pretože dopredu nevieme povedať, ktoré procesory budú významné a ktoré nie. Podobne nevieme dopredu určiť, ktoré procesory (konfigurácie) sa budú podieľať na výpočte, a preto aj procesory, ktoré sa v konečnom dôsledku na výpočte podieľať nebudú (na obrázku znázornené malými kružnicami v prvom riadku), vykonávajú svoj program.



Obrázok 7.5: Jeden krok simulácie  $DTS$



Obrázok 7.6: Generovanie kódu  $BO$  na modeli  $PRAM$

<sup>14</sup>realizáciu si môžeme predstaviť nasledovne:  $P_1$  zapíše do nejakého špeciálneho booleanského registra, ktorý bol inicializovaný na TRUE, hodnotu FALSE a program pre procesory upravíme tak, že dané inštrukcie budú vykonávať len vtedy, keď v tomto registri je hodnota TRUE



Keďže po každom kroku *PRAMu* sa dĺžka odsimulovaného výpočtu  $A$  zdvojnásobí, tak na odsimulovanie celého výpočtu vystačíme s časom  $\log$ aritmus z počtu konfigurácií, čo je  $O(\log \text{SIZE}(C_n))$ .

- Dekódovanie:

Predpokladajme, že už máme v nejakom registri kód  $\langle C_n \rangle$ . Ten je v tvare:

(( číslo hradla, typ hradla, číslo ľavého vstupu, číslo pravého vstupu ))\*

Zrejme dĺžka kódu je  $|\langle C_n \rangle| = O(\text{SIZE}(C_n) \cdot \log \text{SIZE}(C_n))$ . Opäť máme dostatok procesorov, aby sme nimi “pokryli” každý symbol výstupu. Každý procesor  $P_j$  načíta register s kódom  $\langle C_n \rangle$  a akoby nastaví svoju čítaciu hlavu na  $j$ -ty symbol kódu  $\langle C_n \rangle$ . Procesory samozrejme žiadne čítacie hlavy nemajú, ale ľahko si vieme predstaviť, že *PRAM* by vedel na jeden krok “rozložiť” obsah registra na veľa registrov obsahujúcich po jednom symbole, a potom by už procesory pracovali nad registrami ako je to v modeli *PRAM* zvykom a nie s čítacou hlavou ako prezentujeme tu.

Po prvom kroku prestanú pracovať všetky procesory, ktoré neprečítali na vstupe symbol<sup>15</sup> “(”. Tie, ktoré “(” prečítali (t.j. boli nastavené na začiatok podslova v kóde  $\langle C_n \rangle$  reprezentujúce kód nejakého hradla), v každom ďalšom kroku prečítajú jeden symbol, až kým neprečítajú “)” (t.j. koniec kódu hradla). Každý takýto procesor “zistí” informácie o jednom hradle  $BO$ , teda číslo hradla, typ hradla a čísla vstupov, a zapíše ich do príslušných registrov  $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$ .

Keďže čísla hradiel sú v kóde zapísané v binárnom tvare, tak dĺžka kódu jedného hradla je  $O(\log \text{SIZE}(C_n))$ . Každý procesor potrebuje prečítať kód jedného hradla a zapísať získané informácie do registrov. Preto čas, ktorý na dekodovanie potrebujeme, je  $O(\log \text{SIZE}(C_n))$ .

- Simulácia výpočtu  $BO$ :

Teraz už máme všetko pripravené na simuláciu výpočtu  $BO$ , teda v registroch  $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$  sú zapísané správne hodnoty prislúchajúce simulovanému  $BO$ . Predpokladajme, že všetky registre  $C_{i,OUT}$  sú inicializované na nejakú  $NIL$ ovú hodnotu<sup>16</sup>. Procesor  $P_i$  bude vykonávať program:

1. načítaj obsahy registrov  $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$
2. čakaj kým  $[C_{C_{i,IN1}}] \neq NIL$  a  $[C_{C_{i,IN2}}] \neq NIL$
3. vykonaj operáciu podľa typu hradla  $[C_{i,TYP}]$  so vstupmi  $[C_{C_{i,IN1}}], [C_{C_{i,IN2}}]$
4. zapíš výsledok operácie do registra  $C_{i,OUT}$

Na vykonanie tohto programu potrebuje každý procesor čas najviac  $(\log n)^{O(1)}$ , pretože na druhom riadku bude procesor čakať toľko krokov koľko je hĺbka hradla v  $BO$  a z predpokladu vieme, že  $DEPTH(C_n) = (\log n)^{O(1)}$ , ostatné riadky programu sa vykonajú v konštantnom čase.

Z dôkazu ľahko vidieť, že sme splnili požiadavky kladené na časovú a procesorovú zložitosť, ako aj na model *CREW – PRAM*.  $\square$

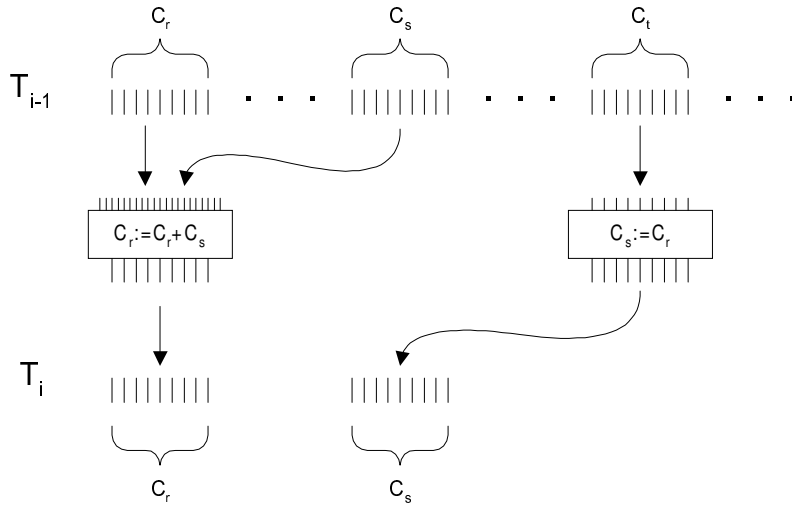
<sup>15</sup>všetky symboly sú zakódované binárne, takže procesory prestanú pracovať po tom ako rozpoznajú symbol “(”

<sup>16</sup>vzhľadom na to, že vo výstupných registroch budú len hodnoty 0 alebo 1, hodnotu  $NIL$  môže reprezentovať akákoľvek iná hodnota

**Veta 7.4.2.** Nech  $M$  je  $CREW - PRAM$ , ktorý v čase  $T(n) = (\log n)^{O(1)}$  a s počtom procesorov  $P(n) = n^{O(1)}$  počíta funkciu  $f$ . Potom existuje konštanta  $k$  a  $BC$ -uniformná postupnosť booleovských obvodov  $\{C_n\}$  taká, že  $C_n$  počíta na vstupe  $x_1 \dots x_n$  výstup  $y_{11}y_{12} \dots y_{ij} \dots$  kde  $y_{ij}$  je hodnota  $j$ -teho bitu spoločného registra  $C_i$  v čase  $T(n)$  pre  $1 \leq i \leq P(n).T(n)$  a  $1 \leq j \leq k.T(n)$ , pričom  $DEPTH(C_n) = (\log n)^{O(1)}$  a  $SIZE(C_n) = n^{O(1)}$ .

**Dôkaz:** Chceme zostrojiť  $BO$  simulujúci  $CREW - PRAM$ . Vstupom pre  $M$  je prvých  $n$  spoločných registrov  $C_1, \dots, C_n$ . Vstupom  $x_1 \dots x_n$  pre  $BO$   $C_n$  sú obsahy týchto registrov v čase  $T_0$ .

Výpočet  $M$  závisí od postupnosti inštrukcií jednotlivých procesorov. Pre každú inštrukciu zostrojíme elementárny obvod simulujúci túto inštrukciu. Vstupom preň bude obsah registra (registrov), s ktorým príslušná inštrukcia pracuje a výstupom bude nový obsah výstupného registra. Uvedomme si, že za nášho predpokladu existencie konštanty  $k$  takej, že  $1 \leq j \leq k.T(n)$  (t.j. vieme ohraničiť maximálnu veľkosť registra počas celého výpočtu), každú inštrukciu vieme simulovať elementárnym obvodom obsahujúcim konečný počet hradiel.



Obrázok 7.7: Simulácia výpočtu  $PRAM$  na  $BO$

$C_n$  bude mať  $T(n)$  úrovní, pričom v  $i$ -tej úrovni budeme mať obsahy všetkých registrov (bit po bite) v čase  $T_i$ . Dosiahneme to tak, že medzi  $(i-1)$ . úroveň a  $i$ -tu úroveň umiestnime (a vhodne pospájame) elementárne obvody príslušajúce k  $i$ -tým inštrukciám všetkých aktívnych procesorov (obr.7.7). Takže na poslednej úrovni budú obsahy registrov v čase  $T(n)$ , čo zodpovedá výstupu  $M$ .

Vďaka predpokladom o počte ( $1 \leq i \leq P(n).T(n)$ ), veľkosti ( $1 \leq j \leq k.T(n)$ ) registrov a z toho vyplývajúcej konečnej veľkosti elementárnych obvodov dostávame zložitosti pre  $C_n$ :  $DEPTH(C_n) = (\log n)^{O(1)}$  a  $SIZE(C_n) = n^{O(1)}$ .

Teda dokázali sme ekvivalentnosť mier zložitosti medzi týmito dvoma modelmi t.j. čas na modeli  $PRAM$  zodpovedá hĺbke na  $BO$  a počet procesorov na  $PRAM$  zodpovedá veľkosti  $BO$ .  $\square$

**Dôsledok 7.4.1.** Model  $PRAM$  je v druhej počítačovej triede.

**Dôsledok 7.4.2.** Triedu  $\mathcal{NC}$  môžeme pomocou modelu  $PRAM$  zdefinovať nasledovne:

$$\mathcal{NC} = TIMEPROCESSORS((\log n)^{O(1)}, n^{O(1)})$$

# Literatúra

- [1] John E. Hopcroft, Jeffrey D. Ullman: Formálne jazyky a automaty (Alfa SNTL, 1978)
- [2] Gabor T. Herman: Closure Properties of Some Families of Languages Associated with Biological Systems (Information and Control, 24, 101-121, 1974)
- [3] Peter Gvozdjak, Branislav Rován: Time-Bounded Parallel Rewriting
- [4] A. Salomaa: Formal Languages (Academic Press, New York, 1973)
- [5] Lila Santean, Jarkko Kari: The impact of the number of cooperating grammars on the generative power (Theoretical Computer Science 98, 249-262, 1992)
- [6] Juraj Hromkovič, Jarkko Kari, Lila Kari: Some hierarchies for the communication complexity measures of cooperating grammar systems (Theoretical Computer Science 127, 123-147, 1994)
- [7] Viliam Geffert: Problémy zložitosti generatívnych systémov, referát ku skúške z odborného minima (MFF UK, Bratislava, 1986)
- [8] Viliam Geffert: Problémy zložitosti generatívnych systémov, kandidátska dizertačná práca (MFF UK, Bratislava, 1987)
- [9] Christos H. Papadimitriou: Computational Complexity (Addison-Wesley Publishing Company, 1994)