Teória paralelných výpočtov

Poznámky z prednášok prof. RNDr. Branislav Rovan, PhD
spísané študentmi informatiky na FMFI UK

6.10.2022

Obsah

Kapitola 1

Paralelné gramatiky

Skúsme sa trochu zamyslieť nad tým, aký paralelizmus 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 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 chybne myslieť. Pán Lindenmeyer, podľa ktorého je tento model pomenovaný, skúmal správanie sa istého druhu fylamentóznych 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 taktoch 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, $proj_1(P) = V$ (teda musí existovať pravidlo pre každý symbol abecedy L-systému), $w \in V^+$ nazývame axiom, alebo počiatočné slovo*[1]

Definícia 1.1.2. *Krok odvodenia je relácia $\Rightarrow$ na V^* definovaná nasledovne: $u \Rightarrow v$ práve vtedy, ak $u = a_1 \dots a_n$, $\forall i \;\; a_i \in V$, $v = b_1 \dots b_n$, $\forall i \;\; b_i \in V^* \; a \; \forall i \;\; a_i \to b_i \in P$*

Definícia 1.1.3. *Jazyk*[2] *generovaný OL-systémom G je $L(G) = \{ \; u \in V^* \; | \; w \overset{*}{\Rightarrow} u \; \}$*

Definícia 1.1.4. *DOL-systém je taký OL-systém, kde $\forall a \in V \;\; \exists! \;\; a \to u \in P, u \in V^*$ (takémuto OL-systému hovoríme aj deterministický)*

[1] 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

[2] teda prakticky každá vetná forma, ktorú z počiatočného slova dostaneme (vrátane jeho samého), patrí do jazyka $L(G)$, ak G je L-systém

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

Príklad 1.1.1. $G_1 = (\{a\}, \{a \to a^2\}, a)$ 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:
$G_1' = (\{a\}, \{a \to a, a \to a^2\}, a)$ potom $L_{G_1'} = a^+$ a veľká sila je razom preč. Je tomu tak preto, lebo pridaním pravidla $a \to a$ sme umožnili akoby rozsynchronizovať odvodenie, a teda symboly sa teraz neprepisujú naraz, ale každý v inom čase

Príklad 1.1.2. $G_2 = (\{a, b\}, \{a \to b, b \to ab\}, a)$ 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. $G_3 = (\{a, b, c\}, \{a \to abcc, b \to bcc, c \to c\}, a)$ potom dĺžky slov jazyka $L(G_3)$ tvoria postupnosť štvorcov (druhých mocnín) prirodzených čísel

Definícia 1.1.6. *$\mathcal{AFL}$ (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-$\mathcal{AFL}$ (t.j. nie je uzavretá na žiadnu $\mathcal{AFL}$ operáciu)*

Dôkaz: Pre každú $\mathcal{AFL}$ 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 $a \to a^2$, 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 $a \to \varepsilon$, ale aj $a \to a$, 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 ε chtiac-nechtiac niekedy vyrobíme, čo je opäť v spore s tým, že $\varepsilon \notin L$

  Teda dostávame $L \notin \mathcal{L}_{OL}$ [3]

- $\cdot$: 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^+:

  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 $b \to a$ alebo $b \to a^2$, ale potom by sme vyrobili aj slovo tvaru $b^i a^j, \quad i, j \neq 0$, ktoré do jazyka L^+ nepatrí
  2. $\varepsilon \notin L^+ \rightsquigarrow G$ je POL-systém

[3] Dostávame sa teda k možno trošku 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

3. $a^4 \in L^+ \rightsquigarrow a \to a^2 \in P$ (nesmú tu byť pravidlá tvaru $a \to a, a \to a^3$, pretože by sme mohli vyrobiť aj slovo $ab^i \notin L^+$)
4. $a^2 \overset{*}{\Rightarrow} b^4 \rightsquigarrow a \to b^i \in P$ pre $1 \leq i \leq 3$. Ak teraz použijeme $a \to a^2$ a $a \to b^i$ dostaneme $a^2 \Rightarrow a^2 b^i \notin L^+$

- h_ε : 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 $h^{-1}(L) = \varnothing \notin \mathcal{L}_{OL}$
- $\cap\mathcal{R}$: Uvažujme jazyk $L_1 = \{\varepsilon, a, a^2\} \in \mathcal{L}_{OL}$ a regulárny jazyk $L_2 = \{a^3\}^*$, dostávame rovnosť $L_1 \cap L_2 = \{\varepsilon\} \notin \mathcal{L}_{OL}$ [4]

□

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 $a \to b_1 \dots b_n \in P$, potom $a \to b_n \dots b_1 \in P'$ 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 1.1.3. *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 $G' = (\{a\}, \{a \to w_1, \dots, a \to w_n, a \to \varepsilon\}, w_1)$
2. L je nekonečný: $\forall i \;\; 1 \leq i \leq m$ nech v_i je najkratšie slovo v L také, že $|v_i| \cong i \mod m$ ak existuje, inak ho nedefinujeme. Nech G' je OL-systém tvaru $G' = (\{a\}, P', a^m)$, kde $P' = \{a \to \varepsilon\} \cup \{a \to u \mid a^m \underset{G}{\Rightarrow} u$ alebo $\exists i \;\; v_i \underset{G}{\Rightarrow} u\}$. 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 ε (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 odvodenia nasledovne:

[4] $\{\varepsilon\} \notin \mathcal{L}_{OL}$, pretože každý OL-systém, ktorý by tento jazyk generoval, by musel mať ako axiom ε, čo je z definície axiomu nemožné

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

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

$$\overbrace{\underbrace{aa\ldots a}_{m}\underbrace{aa\ldots a}_{m}\ldots\underbrace{aa\ldots a}_{m}\underbrace{aa\ldots a}_{m}\ldots\underbrace{aa\ldots a}_{m}\underbrace{aa\ldots a}_{i}}^{x}$$
$$v_i$$

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

B) Nech $w \in L^*$: ak $w = \varepsilon, w \in L$, tak je to zrejmé z A). Nech $w = w_1 \ldots w_k$, $w_i \in L$, $\forall i \;\; w_i \neq \varepsilon, k \ge 2, \forall i \;\; a^m \underset{G'}{\overset{*}{\Rightarrow}} w_i$

$a^m \overset{*}{\Rightarrow} (a^m)^k x$ pre nejaké $x \in a^*$ teda $a^m \underset{G'}{\overset{*}{\Rightarrow}} a^{mk} \underset{G'}{\overset{*}{\Rightarrow}} w^1 \ldots w_k$ teda sme našli nejaké odvodenie slova w, no ešte musíme zabezpečiť, aby slová $w_1, \ldots, w_k$ vznikali synchrónne na rovnaký počet krokov (musíme akosi ntiahnuť odvodenie tých w_i, ktoré vznikajú skôr ako ostatné. Urobíme to nasledovne: ak $a \underset{G'}{\Rightarrow} y$, tak odvodenie natiahneme[5] $a^m \underset{G'}{\Rightarrow} a^m a^t \underset{G'}{\Rightarrow} y\varepsilon$. 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 $M(n, m_1, \ldots, m_s)$, ktorej tvar je $M(n, m_1, \ldots, m_s) = \{n\} \cup \{k_1 m_1 + \cdots$*

Veta 1.1.4. *(Charakterizácia nekonečných OL-jazykov obsahujúcich ε)*
Nech $L \subseteq a^*$ je nekonečný a obsahuje ε. 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:

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

⊆: 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[6] $a^n \underset{G}{\Rightarrow} a^{m_{i_1}} \ldots a^{m_{i_k}} (k \le n) \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} a^{k_1 m_1} a^{k_2 m_2} \ldots a^{k_s m_s}$, pričom niektoré k_i (možno aj všetky, vďaka $a \to \varepsilon$) sa môže rovnať 0, stačí zvoliť $m = k_1 m_1 + \cdots + k_s m_s$, pričom $\forall i$ je k_i rovnaké ako v odvodení

⊇: zoberme prvok $m \in M_L$ a k nemu hľadajme 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 $m = k_1 m_1 + \cdots + k_s m_s \in$

[5] toto máme existenciou neepsilonových pravidiel zabezpečené

[6] takto musí vyzerať každé odvodenie, lebo v každom kroku sa a prepíše na nejaké a^{m_i} alebo na ε

M_L. Odvodenie nájdeme nasledovným spôsobom: najskôr si vyrobíme dostatočne veľa symbolov a, konkrétne toľko, aby ich bolo viac ako $\Sigma_{i=1}^{s} k_i$ (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 $\Sigma_{i=1}^{s} k_i$ 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[7] $a \to a^{m_1}$, na ďalších k_2 symbolov pravidlo $a \to a^{m_2}$ a tak ďalej, až na posledných k_s symbolov aplikujeme pravidlo $a \to a^{m_s}$), na zvyšné symboly aplikujeme pravidlo $a \to \varepsilon$ 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*

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

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. $\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}_{CS} - \mathcal{L}_{CF}$

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

□

Poznámka 1.1.3. *Konečné jazyky v $\mathcal{L}_{OL}$ sú (v abecede $\{a\}$) 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 $m \geq 1$*

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

[7] nesmieme zabúdať na to, že pracujeme v OL-systéme, a teda v jednom kroku odvodenia musíme pravidlá aplikovať na všetky symboly vo vetnej forme

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 $M_L = \{n, m_1, \ldots, m_k\}$. Potom regulárna gramatika pre jazyk L bude: $G = (\{\sigma, A\}, \{a\}, \{\sigma \to a^n, \sigma \to A, A \to a^{m_1}A, \ldots, A \to a^{m_k}A, A \to \varepsilon\}, \sigma)$

□

Poznámka 1.1.4. *Každý jazyk generovaný OL-systémom, v ktorom $a \to a \in P$ pre každé $a \in V$, je bezkontextový*[8]

Poznámka 1.1.5. *Každý jazyk $L \in \mathcal{L}_{CF}$ je tvaru $L' \cap R$ pre $L' \in \mathcal{L}_{OL}, R \in \mathcal{R}$*

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: $G' = (V \cup \{x_0\}, P \cup \{x_0 \to w_0\}, x_0)$, $x_0 \notin V$, platí $L(G') = L(G) \cup \{x_0\}$. Teraz ku každému odvodeniu v G' uvažujme jeho strom[9]. Hovoríme, že odvodenie je redukované, keď jemu zodpovedajúci strom spĺňa nasledujúcu podmienku: neexistuje podstrom T taký, že súčasne platí:

1. všetky listy T sú ε
2. T obsahuje vetvu, v ktorej majú dva vrcholy rovnaké návestie

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 ε) originálneho stromu odvodenia.

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

$$k = 3m^n$$

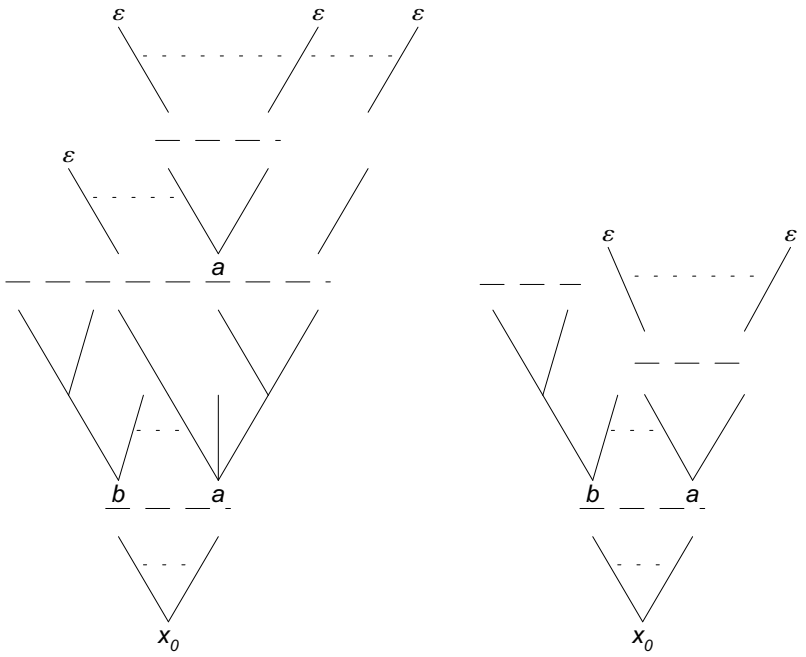
Pre $\forall w \neq \varepsilon, w \in L(G')$ s odvodením $x_0 \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n \equiv w$ bez újmy na všeobecnosti predpokladajme, že toto je redukované. Výskyt symbolu a v jednom zo slov w_i nazveme neproduktívny $\iff$ ak tento výskyt je návestím koreňa podstromu, ktorého všetky listy majú návestie ε. 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:

$$Q = Q_1Q_2Q_3, |Q_1| = |Q_2| = |Q_3| = m^n$$

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 návestie v strome T, ktorého koreň má návestie b. Ďalej, voľbou Q, m, n všetky listy v T majú návestia ε. To je spor, lebo potom existuje v T vetva s dvoma uzlami s rovnakým návestím □

[8] pravidlo $a \to a$, $\forall a \in V$ nám umožní si pri každom kroku vybrať jeden symbol a nahradiť ho príslušnou pravou stranou pravidla a na ostatné symboly použiť pravidlo $a \to a$

[9] ako pre bezkontextové gramatiky - koreň tvorí x_0, návestia vrcholov sú symboly $\in V \cup \{x_0\}$, hrany sú orientované a množina orientovaných hrán vychádzajúca z vrchola v má tvar $E(v) = \{v_1, \ldots, v_k\} \iff v \to v_1 \ldots v_k \in P \cup \{x_0 \to w_0\}$



Obrázok 1.1: *Vytváranie redukovaného odvodenia*

Veta 1.1.7. *(O lineárnom priestore)*
Nech A je Turingov stroj[10] *(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 $L(G) \in \mathcal{L}_{ECS}$ □

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 dovolia 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 odvodenia 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* $\forall i \;\; a_i \to b_i \in P$

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

[10] definície, popis modelu a iné v [1]

Príklad 1.1.4. $G_1 = (\{\sigma\}, \{a, b\}, \{\sigma \to a, \sigma \to b, a \to a^2, b \to b^2\}, \sigma)$, 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. $G_2 = (\{A\}, \{a, b\}, \{A \to A, A \to a, a \to a^2, b \to b\}, AbA)$ 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. $G_3 = (\{S, A, B, C, \bar{A}, \bar{B}, \bar{C}, F\}, \{a, b, c\}, P, S)$

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

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 odvodenia 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 odvodenia[11]

Dôkaz: Ukážeme konštrukciu synchronizovaného EOL-systému G'. Najskôr si vytvoríme nové neterminály, ktoré budeme potrebovať:

- $\forall a \in T$ vytvoríme ξ_a (ak $a, b \in T$, tak $\xi_a \neq \xi_b$)
- F, σ', budú predstavovať FALSE, resp. nový počiatočný symbol

Označme $N' = \{\xi_a \mid a \in T\}$. Pre jednoduchosť zaveďme niekoľko pojmov, ktoré neskôr využijeme:

1. Nech S je ľubovoľná množina reťazcov z $(N \cup T)^*$, pre ňu definujeme množinu $A(S)$ nasledovne:
   (a) reťazec $b_1 \dots b_n \in A(S) \overset{def}{\iff}$ ak existuje reťazec $a_1 \dots a_n \in S$ taký, že buď $b_i = a_i$ alebo $a_i \in T$ a $b_i = \xi_i$ (teda $b_i \in N'$), potom $A(S) \subset (N \cup N' \cup T)^*$
   (b) ak $S \neq \varnothing$, potom $A(S) \neq \varnothing$, lebo $S \subseteq A(S)$
2. $\delta_P(a)$ nazveme množinu všetkých pravých strán pravidiel pre symbol a z P

Definujme synchronizovaný EOL-systém G' nasledovne: $G' = (N \cup N' \cup \{F, \sigma'\}, T, P', \sigma')$, pričom

$$
\begin{array}{lll}
P': & \delta_{P'}(\sigma') = A(\{w\}) & \\
& \forall a \in (T \cup \{F\}) & (a \to F) \in P' \\
& \forall a \in N & \delta_{P'}(a) = A(\delta_P(a)) \\
& \forall \xi_a \in N' & \delta_{P'}(\xi_a) = A(\delta_P(a))
\end{array}
$$

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

Malo by byť zrejmé, že teminálne slová musia v G' vznikať naraz, pretože ak sa náhodou nejaký terminál do venej formy dostane skôr ako ostatné, tak sa v ďalšom kroku prepíše nutne na F a z F sa už nikdy terminál nestane. Rovnako zrejmé by malo byť $L(G') = L(G)$, pretože nové neterminály ξ_i vo vetnej forme akoby nahrádzali terminálne symboly, a tak simulovali odvodenie v pôvodnom systéme G □

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 $a \to a \ \ \forall a \in (N \cup T)$) aj hľadaným EOL-systémom □

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 $a \to a$*

Dôkaz: Zkonštruujeme EOL-systém G' nasledovne: pre každé $a \in T$ zavedieme nový neterminál ξ_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 $a \to a$
3. pre každé nové $\xi_a \in N$ pridáme pravidlo $\xi_a \to a$

□

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 $N_1' \cap N_2' = \varnothing$. Vytvoríme G_3 pre $L_1 \cup L_2$ nasledovne: zavedieme nový neterminál σ_3 tak, že $\sigma_3 \notin N_1'$ a $\sigma_3 \notin N_2'$. Ďalej $N_3 = N_1' \cup N_2' \cup \{\sigma_3\}$, $T_3 = T_1' \cup T_2'$, $P_3 = P_1' \cup P_2' \cup \{\sigma_3 \to \sigma_1' | \sigma_2'\}$ 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"[12] b, tak takéto slovo nedokážeme vyrobiť (nemáme na to dostatok času), čo je spor s tým, že G je EOL-systém pre $h^{-1}(L)$

□

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

Dôkaz: Každá bezkontextová gramatika vyhovuje definícii EOL-systému (po doplnení pravidiel $a \to a \ \ \forall a \in (N \cup T)$), 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 $L = \{a^n b^n c^n \mid n \geq 1\} \in \mathcal{L}_{CS} - \mathcal{L}_{CF}$ □

Poznámka 1.1.6. *1L a 2L-systémy sú nejakou 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*

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

1.1.4 TOL-systémy (table)

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

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

Poznámka 1.1.7. *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

1.2.1 Indické paralelné gramatiky

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

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. $G_1 = (\{\sigma\}, \{a\}, \{\sigma \to \sigma\sigma, \sigma \to a\}, \sigma)$ 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: definujme rekurentne podmnožiny L a uvedomme si, koľko najmenej neterminálov treba na generovanie slov z každej z nich:

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

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

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

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 pätica $G = (N, T, P_1, P_2, \sigma)$, 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* $(A_1, \ldots, A_n) \to (w_1, \ldots, w_n)$*, 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* $w = u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1}$ *a* $v = u_1 w_1 u_2 w_2 \ldots u_n w_n u_{n+1}$*, pričom* $u_1, \ldots u_{n+1} \in T^*$ *(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:
$G = (\{S\}, \{a, b, c\}, \{(S) \to (SSS), (S, S, S) \to (aS, bS, cS), (S, S, S) \to (a, b, c)\}, S)$ 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 $N = \{A_1, \ldots, A_n\}$. Definujme N' nasledovne: $N' = \{A_p\} \cup \{A_{i_1,\ldots,i_k} \mid \forall j \ A_{i_j} \in N, \ A_{i_1}, \ldots, A_{i_k}$ sa nachádzajú na pravej strane niektorého (rovnakého) z pravidiel P v tomto poradí$\}$. Bez újmy na všeobecnosti môžeme predpokladať, že všetky pravidlá z P sú tvaru $(A_{i_1}, \ldots, A_{i_m}) \to (a^{k_1}\pi_1, \ldots, a^{k_m}\pi_m)$, kde $\forall i \ \pi_i \in N^*$. Definujme $\varphi_i = i_1, \ldots, i_l \iff \pi_i = A_{i_1} \ldots A_{i_l}$ Vytvoríme množinu pravidiel P' nasledovne:

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

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

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*

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 trošku 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* $(N \cup T)^* \to 2^{(N \cup T)^*}$, σ *je počiatočný neterminál pričom* N, T *nie sú nutne disjunktné abecedy.*

Definícia 2.1.2. *Krok odvodenia abstraktného generatívneho sytému 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* $L(G) = \{w \in T^* \mid \sigma \overset{*}{\Rightarrow} w\}$.

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*[1] $H \subseteq K \times \Sigma_1 \times {\Sigma_2}^* \times K$.

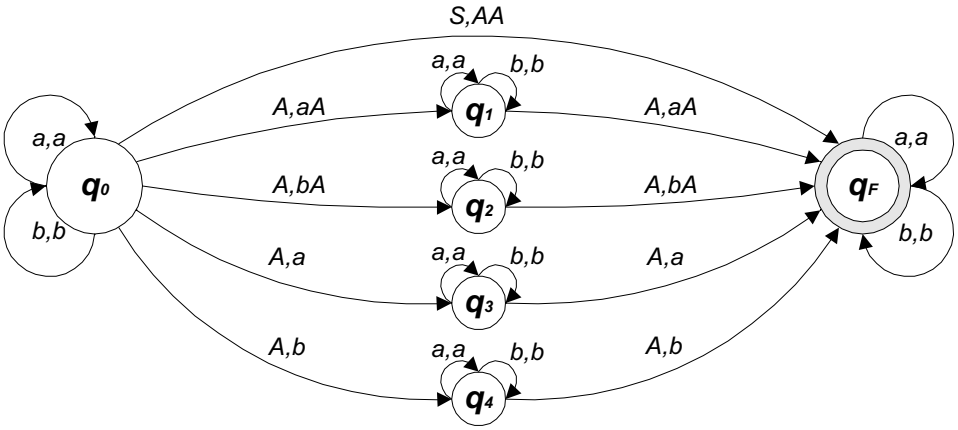
Definícia 2.1.5. *Generatívny systém (g-systém) je abstraktný generatívny systém, v ktorom* f *je zobrazenie 1-a-prekladačom. Zapisujeme* $G = (N, T, M, \sigma)$ *kde* M *je 1-a-prekladač.*

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

[1] 1-a-prekladač nemá pružnú čítaciu hlavu t.j. môže čítať práve jeden symbol

[2] pre vstupné slovo u 1-a-prekladač M vygeneruje výstup v

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-ε g-systémov, t.j. 1-a-prekladač nedáva na výstup ε.

2.2 Porovnanie generatívnej sily g-systémov s gramatikami Chomského hierarchie

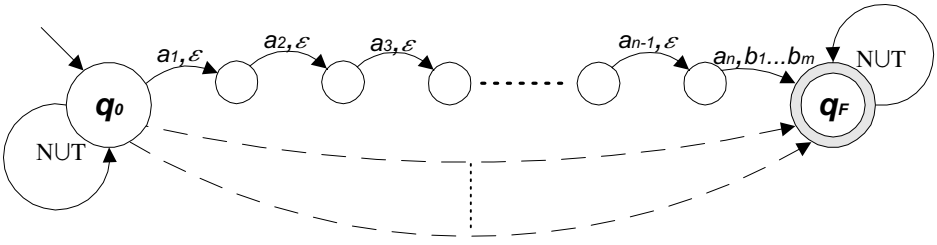
Veta 2.2.1. $\mathcal{L}_\mathcal{G} = \mathcal{L}_{RE}$

Dôkaz: Dokážeme obe inklúzie:

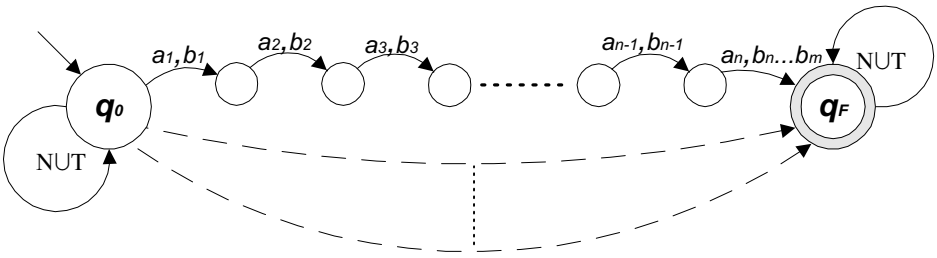
$\subseteq$: Táto inklúzia vyplýva z Turingovej tézy, ale iste by sme si vedeli predsaviť aj TS, ktorý dostane na pásku slovo a na druhej páske simuluje odvodenie tohto slova g-systémom od počiatočného neterminálu a porovnáva, či je slovo z druhej pásky zhodné so vstupom na prvej páske.

$\supseteq$: 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.

□



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



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

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

Dôkaz: Dokážeme obe inklúzie:

$\subseteq$: K bez-ε 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 ε, 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 ε 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.*

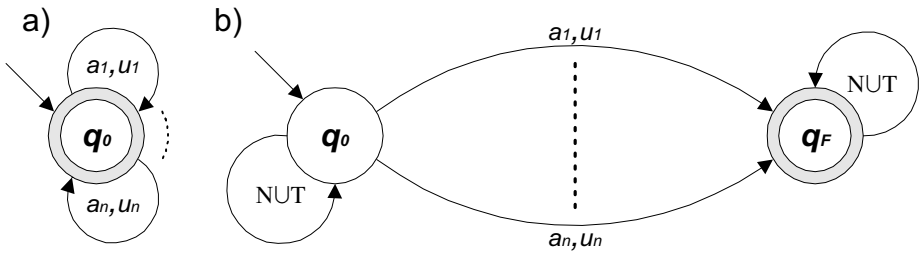
Označenie: $\mathcal{S}$ trieda všetkých sekvenčných g-systémov.
$\mathcal{S}_\varepsilon$ trieda všetkých bez-ε sekvenčných g-systémov.

Veta 2.2.3. $\mathcal{L}_{\mathcal{S}} = \mathcal{L}_{RE}, \mathcal{L}_{\mathcal{S}_\varepsilon} = \mathcal{L}_{CS}$

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

2.3 Modelovanie známych gramatík pomocou g-systémov

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



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

CF: Nech množina pravidiel *CF* je $P = \{a_1 \to u_1, a_2 \to u_2, \ldots, a_n \to u_n\}$ potom príslušný g-systém bude vyzerať 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):*

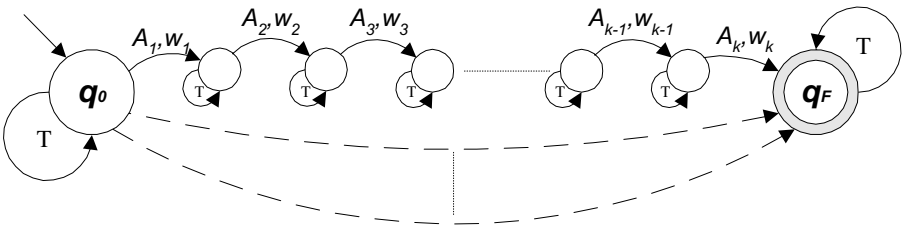
1. *každý cyklus je kopírovací cyklus pre terminál*
2. *v každom stave je kopírovací cyklus pre každé $a \in T$*
3. *terminálne symboly sú prepisované len v kopírovacích cykloch*

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 $(A_1, \ldots A_k) \to (w_1, \ldots w_k))$ 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: $(A_1, \ldots A_k) \to (w_1, \ldots w_k))$.

□



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
- $STATE_U(L) = \min\{STATE(G) \mid L = L(G), G \in U\}$
- $ARC_U(L) = \min\{ARC(G) \mid L = L(G), G \in U\}$
- $TIME(G,w) = \min\{k \mid \sigma \overset{k}{\underset{G}{\Rightarrow}} w\}; w \in L(G)$
- $TIME(G,n) = \max\{TIME(G,w) \mid w \in L(G), |w| \leq n\}$
- $TIME_U(f(n)) = \{L \mid \exists G \in U, L = L(G), TIME(G,n) = O(f(n))\}$
- $\overline{TIME}_U(f(n)) = \{L \mid \forall G \in U, L = L(G), TIME(G,n) = \Omega(f(n))\}$
- $SPACE(G,w) = \min\{SPACE(\alpha)\}$; kde $\alpha : \sigma \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n$ 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. $L \in \overline{TIME}_{\mathcal{S}}(n)$
2. $L \in \overline{TIME}_{\mathcal{G}}(\log n)$

Dôkaz:

1. Nech $G \in \mathcal{S}$. Zamyslime 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{ súčet dĺžok výstupov na jednej ceste v 1-a-prekladači - dĺžka tejto cesty }
   Majme odvodenie $\sigma = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_i \Rightarrow w_{i+1} \Rightarrow \ldots \Rightarrow w_n$. 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{ výstup v 1-a-prekladači }
   Potom platí $|w_{i+1}| \leq |w_i|.m$. 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. $\mathcal{L}_{\mathcal{G}} = \mathcal{L}_{\mathcal{S}}, \mathcal{L}_{\mathcal{G}_\varepsilon} = \mathcal{L}_{\mathcal{S}_\varepsilon}$

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, $L(G) \in \mathcal{L}_{\mathcal{G}}$, počiatočný stav M je q_0 a koncový q_F. Uvedomme si, čo potrebujeme na G prepracovať, aby sme ho "zosekvenčnili". V prvom rade potrebujeme mať v počiatočnom aj koncovom stave kopírovacie cykly pre všetky[3] 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 počiatočnom, resp. koncovom stave). Keď sa nám toto podarí, budeme s konštrukciou hotoví. G'' budeme konštruovať v dvoch krokoch:

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

   (a) zavedieme nový počiatočný stav q_0' a nový koncový stav q_F' a v týchto stavoch zkonštruujeme kopírovacie cykly[4] $\forall a \in (N' \cup T)$, 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 udržovať 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 $N' = N \cup \{\overline{a} \mid a \in (N \cup T)\} \cup \{\underline{a} \mid a \in (N \cup T)\} \cup \{\underline{\overline{\sigma}}\}$. Počiatočný neterminál G' bude $\underline{\overline{\sigma}}$. Vetná forma bude vyzerať nasledovne: $\overline{a}_1 a_2 a_3 \dots a_{n-1} \underline{a_n}$.

   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 $a, d, A, B \in (N \cup T), b, c \in (N \cup T)^*$):

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

   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 počiatočného neterminálu $\underline{\overline{\sigma}}$ a vynecháme aj prípady, kedy M "príliš veľa" vymazáva, teda ak sa v odvodení môže vyskytnúť vetná forma obsahujúca jediný symbol, prípadne ε. 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) G' ekvivalentný s G a splnené (a) aj (b). Ale

[3] ako sa neskôr ukáže, tak nie nutne pre úplne všetky, stačí pre niektoré

[4] 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

my uvažujeme $G \in \mathcal{G}$. Uvedomme si, aké nepríjemnosti nám môže spôsobiť skutočnosť, že 1-a-prekladač môže zapisovať na výstup ε. Môže sa stať, že v H existuje $(q_0, A, \varepsilon, q_1)$. Podľa doterajšej konštrukcie by sme do H' pridali $(q_0', \overline{A}, \varepsilon, q_1)$. 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, ε, 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 $u \neq \varepsilon$ (zámerne sme do tretej komponenty vždy písali reťazec $w \in (N' \cup T)^+$).

Pre ε-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 $\overline{q}$. V týchto stavoch si budeme pamätať, že sme si zmazali prvý symbol vetnej formy a keď budeme robiť v danom kroku odvodenia prvý ne-ε 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 $\overline{H}$prechodovú množinu, ktorú definujeme nasledovne ($a, b \in (N \cup T)$, $u \in (N \cup T)^*$):

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

(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 vracať, 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 ε, a potom len kontrolujeme, či sme hádali správne. Formálne zapísané v prechodovej množine $\underline{H}$ ($w \in (N \cup T)^*$):

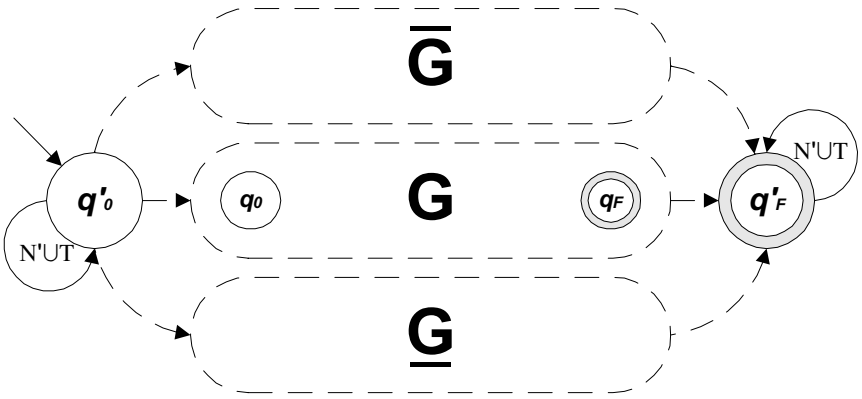
$$\begin{aligned}\underline{H} = {} & \{(q, a, w\underline{b}, \underline{p}) \mid (q, a, wb, p) \in H\} \text{ nedet. zapíše nový posledný symbol} \\ & \cup \{(\underline{q}, \underline{a}, \varepsilon, \underline{p}) \mid (q, a, \varepsilon, p) \in H\} \cup \quad \text{stále maže všetky symboly} \\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

Vytvorili sme akoby virtuálne g-systémy $G, \underline{G}, \overline{G}$ (obr.2.6), pričom v jednom kroku odvodenia sa môžeme medzi nimi pohybovať. Neexistujú však žiadne šípky $G \to \overline{G}$, $\underline{G} \to G$, $\underline{G} \to \overline{G}$.

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 definujme:

$$H_T = \{(q_0', \overline{a}, a, q_T), (q_T, a, a, q_T), (q_T, \underline{a}, a, q_F') \mid a \in T\}$$

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
$G' = (N', T, M', \overline{\sigma})$. Čo sa týka M' pre nás je zaujímavé, ako sa zmenila prechodová množina:



Obrázok 2.6: *1-a-prekladač g-systému G′*

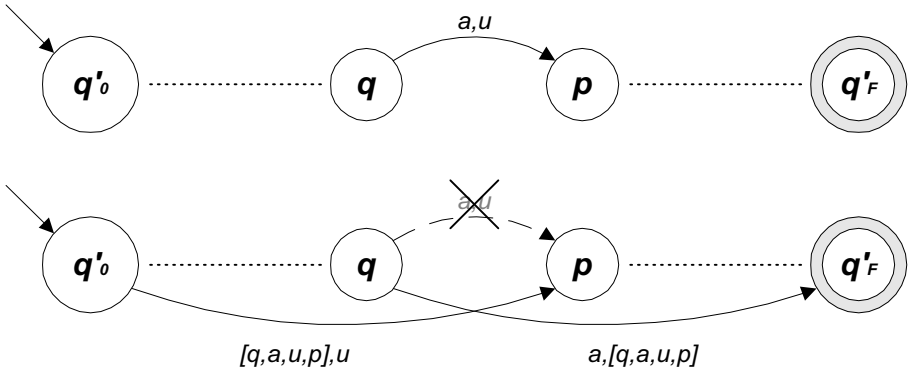
keď v M to bola H, tak v M' to bude $H_{M'} = H' \cup \overline{H} \cup \underline{H} \cup H_T$. Pokiaľ ide o stavy a abecedy, ich konštrukcia by mala byť z už uvedeného zrejmá.

2. Ku $G' = (N', T, M', \overline{\sigma})$ zostrojíme ekvivalentný g-systém G'' tak, že prerušíme všetky vnútorné cykly (teda tie, ktoré nie sú kopírovacie v počiatočnom alebo koncovom stave). Vonkajšou šípkou 1-a-prekladača nazveme takú, ktorá buď vychádza z počiatočného 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[5] (obr.2.7)



Obrázok 2.7: *Nahrádzanie vnútorných šípok v G″*

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 počiatočnom a koncovom stave, lebo teraz

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

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 "zosekvenčňovanie" g-systému G je na konci. □

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

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

- $\forall L \;\; STATE_{\mathcal{S}}(L) \leq 3 \cdot STATE_{\mathcal{G}}(L) + 2$
- $ARC_{\mathcal{S}}(L) \leq 32 \cdot STATE_{\mathcal{G}}(L) + 22 \cdot \#\Sigma_L$

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

Veta 2.5.3. $SPACE_{\mathcal{S}}(f(n)) = SPACE_{\mathcal{G}}(f(n))$ *pre* $\forall f(n)$

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

$$L \in TIME_{\mathcal{S}}(SPACE_{\mathcal{G}}(G,n).TIME_{\mathcal{G}}(G,n))$$

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. $\overline{TIME}_{\mathcal{S}_\varepsilon}(f(n)) \subseteq \overline{TIME}_{\mathcal{G}_\varepsilon}(\frac{f(n)}{n})$

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 $L = \{wcw \mid w \in \{a,b\}^*\} \in \overline{TIME}_{\mathcal{S}_\varepsilon}(n^2)$, teda podľa vety 2.5.5 $L \in TIME_{\mathcal{G}_\varepsilon}(n)$. 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 ε*

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 ξ_a
   - v q_0 nedeterministicky uhádneme, že vetná forma obsahuje už len neterminály typu ξ_a a celú ju zterminálnime. Do M' pridáme nový stav q' a do H' pridáme štvorice: pre $\forall\ a \in T$ (q_0, ξ_a, a, q') a (q', ξ_a, a, q'), pričom q' bude akceptačným stavom.
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 ε g-sytémami, tak nemusíme strojnásobovať 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. $STATE_{\mathcal{N}}(L) \leq STATE_{\mathcal{G}_\varepsilon}(L) + 2$
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*[6] $\forall n > 1$

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 ε prechody v 1-a-prekladači.

[6] 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. Geffertov normálny tvar:
K ľubovoľnému jazyku $L \in \mathcal{L}_{RE}$ existuje taká frázová gramatika, že všetky jej pravidlá sú tvaru:

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

Ilustračný príklad:
Majme nejaké pravidlo $\xi \to u$, označme si ho ako pravidlo 100. 1 budeme kódovať ako A z ľavej strany a B z pravej strany. 0 budeme kódovať ako C z ľavej strany a D z pravej strany. Dve susedné S musia uhádnuť to isté pravidlo a potom sa nenerminály medzi nimi vyrušia. S hrá rolu všetkých pôvodných neterminálov.

Otvorenými problémami zostávajú:

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

2.7 Charakterizácia triedy $TIME_{\mathcal{G}}(f(n))$ pomocou sekvenčného priestoru

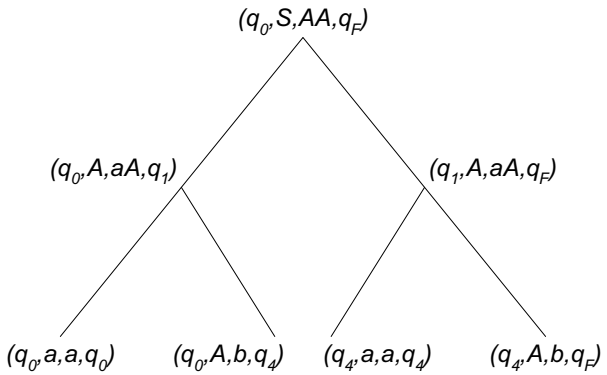
Definícia 2.7.1. *Nech D je odvodenie slova $w \in L(G)$ g-systémom $G = (N, T, M, \sigma)$, kde $M = (K, \Sigma, \Sigma, H, q_0, q_F)$. Strom odvodenia (ozn. T_D) nazývame strom, ktorého vrcholy sú označené ako štvorice z H, pričom označenie koreňa T_D je tvaru (q_0, σ, u, q_F), kde $u \in (N \cup T)^*$. 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 štvoríc na každej úrovni stromu je výpočet 1-a-prekladača. Zreťazením tretích komponent štvoríc na k-tej úrovni dostaneme k-tu vetnú formu výpočtu G. Teda zreťazením tretích komponent štvoríc na poslednej úrovni*[7] *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.

[7] 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 odvodenia 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.

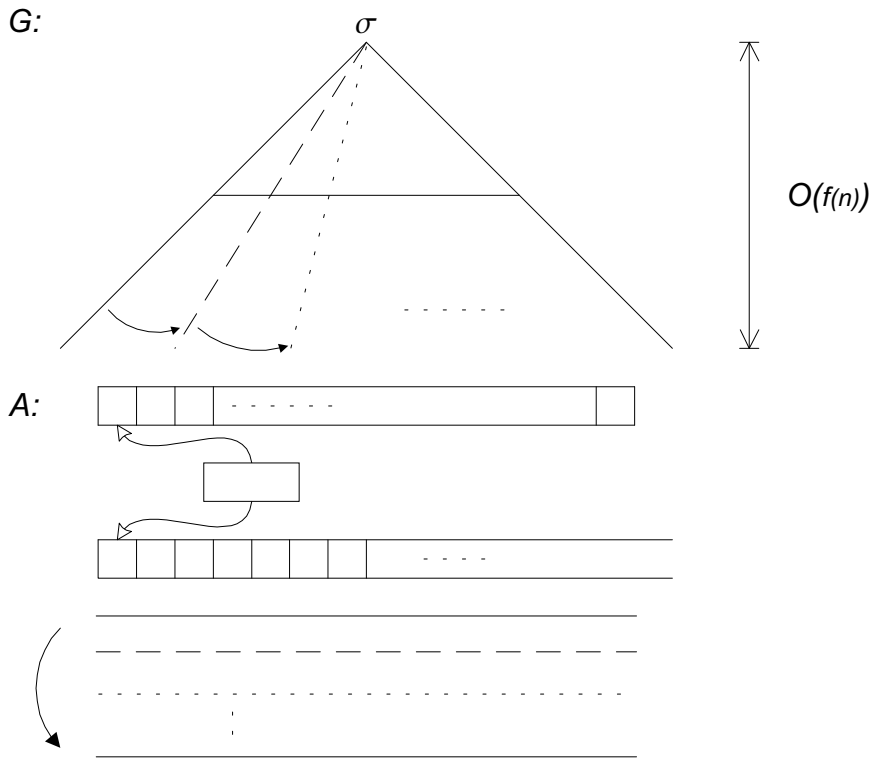
A najprv uhádne a zapíše na pracovnú pásku cestu z koreňa do najľavejšieho listu α, pričom musí skontrolovať či prvá štvorica je tvaru (q_0, σ, u, q_F) a ostatné štvorice musia začinať v stave q_0 a prepisovať prvý symbol výstupu predchádzajúcej štvorice. Naviac výstup listu α musí byť prefixom slova w. Ak nejaká z týchto podmienok neplatí, tak A neuhádol najľavejšiu cestu správne a zasekne sa. Ak A uhádol, tak nahradí štvoricu α jeho najľavejším bratom β (to samozrejme tiež uhádne (obr.2.10a) a zaznačí si, že β je druhým synom ich spoločného otca γ. A overí, či β je dobre uhádnutý t.j. počiatočný stav β je rovnaký ako koncový stav α, β prepisuje druhý symbol výstupu γ a výstup β je rovnaký ako ďalšia časť w (bez prefixu, ktorý bol vo výstupe α). Takto A pokračuje až kým neuhádne a neoverí posledného syna γ. Potom A nahradí γ jeho najľavejším bratom δ (podobne ako bol α nahradený β). 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 ε (obr.2.10b). Teda zvyšné cesty bude A hľadať s tým, že v listoch musí byť na výstupe ε. 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.

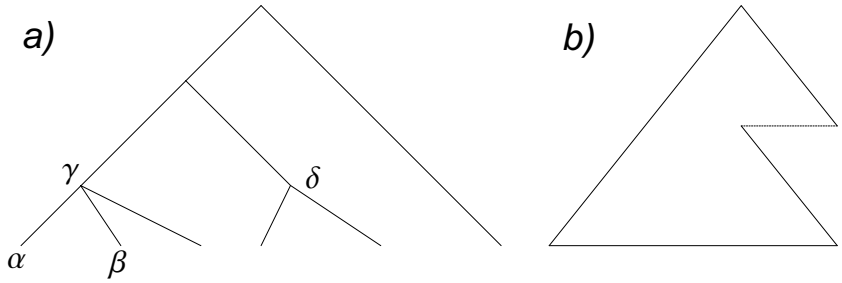
Z konštrukcie A je zrejmé, že A akceptuje w práve vtedy, keď $w \in L(G)$. Keďže $w \in L(G)$, tak hĺbka stromu odvodenia nie je väčšia ako $c.f(|w|)$ kde c je konštanta nezávislá od w, teda hĺbka stromu odvodenia je $O(f(n))$, a teda A na akceptovanie slova w nepotrebuje viac políčok na páske ako $O(f(n))$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

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…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 δ-funkcie A.

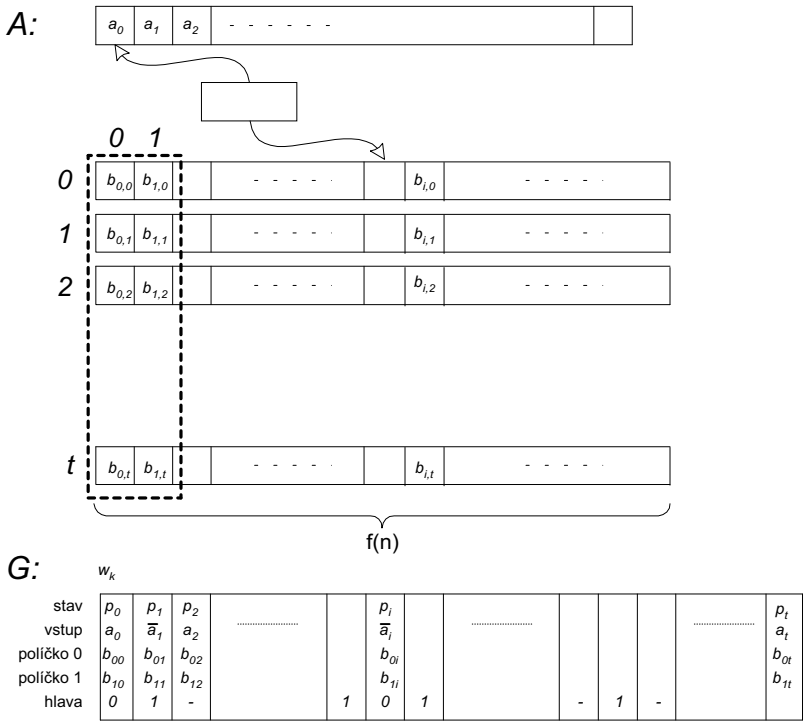
Uvažujme jeden výpočet A na vstupnom slove $w \in L(A)$. Nech s je priestor a t je čas potrebný na výpočet w. Odvodenie slova w g-systémom G je tvaru:

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

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 δ-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[8]) 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.



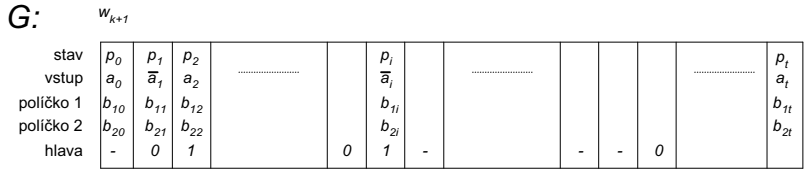
Obrázok 2.11: *Simulácia TS A g-systémom G*

[8] to sa dá na $O(f(|w|))$ krokov keďže A pracuje v priestore $O(f(|w|))$ a teda v čase $O(|w|.c^{f(|w|)})$ pre nejakú konštantu c t.j. počet všetkých možných konfigurácií A pri výpočte w. g-systém vie toľkoto symbolov vygenerovať v logaritmickom čase, čo je v tomto prípade $O(f|w|)$.

V kroku $k+1$ G overí či, to čo uhádol v predchádzajúcom kroku je v súlade s δ-funkciou A a vygeneruje[9] w'_k. Ak to bolo uhádnuté dobre, tak G vygeneruje w_{k+1} t.j. prvé a druhé poschodie 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[10] (obr.2.12). V ďalšom kroku G overí či to čo uhádol teraz je legálne vzhľadom na δ-funkciu A a vygeneruje $w'_{k+1}\ldots$

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 δ-funkcie, ktorá je schopná takéto zmeny spôsobiť. Ak takúto časť δ-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.



Obrázok 2.12: *Posun symbolov vo vetnej forme*

V poslednom kroku G prepíše vetnú formu tak, že každý päťposchodoví 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 ε.

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 odvodenia G $w_l \Rightarrow w'_l \Rightarrow w_{l+1}$ sa dajú nahradiť jedným, potom w je vygenerované v čase $O(f(n)) + O(f(n)) = O(f(n))$. Tento spôsob odvodenia sme zvolili iba pre lepšiu čitateľnosť dôkazu. □

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 $f(n) \geq n$.

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

[10] 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 $TIME_{\mathcal{G}}(f(n))$, pre ktoré $f(n) < n$.

Veta 2.8.1. *$TIME_{\mathcal{G}}(\log^p n)$ je $\mathcal{AFL}$ pre*[11] $p \geq 1$.

Veta 2.8.2. $TIME_{\mathcal{G}}(\log^p n) \subsetneq TIME_{\mathcal{G}}(\log^q n)$ *pre* $q > p > 1$.

Dôsledok 2.8.1. Hierarchia $TIME_{\mathcal{G}}(\log^p n)$ pre $p > 1$ je nekonečná.

Veta 2.8.3. *Pre každý jazyk $L \in \mathcal{L}_{RE}$ existuje $L' \in TIME_{\mathcal{G}}(\log n)$ 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 logaritmickom 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 logaritmickom čase. G upravíme na G' nasledovne:

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

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

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

Zamyslime sa teraz nad časovou zložitosťou 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 logaritmickom čase, a teda $L' \in TIME_{\mathcal{G}}(\log n)$. Homomorfizmus h zvolíme takto:

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

To znamená, že h vymaže všetky γ 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 odvodenia bude tento jazyk vygenerovaný v logaritmickom čase vzhľadom na túto zväčšenú dĺžku slova. □

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 navrhneme nový paralelný model, ktorý vieme "tesne" simulovať na g-systémoch, dostaneme priestorové ohraničenie $1NSPACE(f(n))$. Naviac pre každý jazyk $L \in 1NSPACE(f(n))$ existuje nejaký typ paralelnej gramatiky, ktorá L dokáže generovať v čase $f(n)$.

[11] rýchlejšie ako v logaritmickom čase g-systém nedokáže pracovať

2.10 Meranie nedeterminiznu v g-systémoch

Poznámka 2.10.1. *Viac v práci* http://www.dcs.fmph.uniba.sk/~rovan/Krch-dipl.ps.

2.10.1 Počet nedeterministických stavov

Veta 2.10.1. *Nech M je sekvenčný g-systém, potom existuje ekvivalentný M', v ktorom sú všetky stavy deterministicné, okrem q_0.*

Poznámka 2.10.2. *Na sekvenčnosti až tak nezáleží, len dôkaz je zložitejší. Premyslite si keby sme vedeli cez q_0 prejsť viac krát.*

2.10.2 Počet nedeterministických rozhodnutí

$$dec(q,a) = \forall\{h \in H \mid pr_1(h) = q \quad pr_2(h) = a\} - 1$$

$$= 0 \text{ ak nie je ziadna hrana}$$

$$dec(q) = \sum_{\forall a \in \Sigma} dec(q,a)$$

$$dec(M) = \sum_{q \in k} dec(q)$$

Veta 2.10.2. *K ľubovoľnému g-systému G s jediným nedeterministickým stavom q_0 existuje ekvivalentné G' také, že $dec(G') \leq |T| + 1$*

2.11 Deterministické g-systémy DGS

Poznámka 2.11.1. *Viac v práci* http://www.dcs.fmph.uniba.sk/~rovan/Kra-dipl.ps.

#F nemusí byť 1

DGS bez neterminálov $\mathcal{DTG}$ (fancy pismom)

Veta 2.11.1. *Existuje konečný jazyk L, $L \in \mathcal{L}(\mathcal{DTG})$.*

Dôkaz:

$$L = \{a, aa, b, bb, bba, bbb, \varepsilon\}$$

□

2.11.1 DGS s neterminálmi

Veta 2.11.2. $\mathcal{L}(\mathcal{DTG}) \subsetneq \mathcal{L}(\mathcal{DG})$

Veta 2.11.3. $\mathcal{L}_{FIN}(\mathcal{DTG}) \subseteq \mathcal{L}(\mathcal{DG})$

$\mathcal{R} \notin \mathcal{L}(\mathcal{DG})$

Kapitola 3

Kooperujúce distribuované systémy gramatík ($CDGS$)

V tejto kapitole ukážeme ďalšiu možnosť paralelizmu, kde viac gramatík pracuje na jednej spoločnej vetnej forme: gramatika dostane vetnú formu a pracuje na nej tak dlho, ako je jej určené...

Definícia 3.0.1. *$CDGS$ je $(n+2)$-tica $\Gamma = (T, G_1, G_2, ..., G_n, S)$, 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*

Poznámka 3.0.1. *Terminály jednej gramatiky môžu byť neterminály druhej.*

Definícia 3.0.2. *Nech $\Gamma = (T, G_1, G_2, ..., G_n, S)$. Krok odvodenia je relácia $\overset{\leq k}{\underset{\Gamma}{\Longrightarrow}}$, kde $\overset{\leq k}{\underset{\Gamma}{\Longrightarrow}} = \overset{\leq k}{\underset{G_i}{\Longrightarrow}}$ $i \in \{1, 2, ..., n\}$ definovaná nasledovne: $\overset{\leq k}{\underset{G_i}{\Longrightarrow}} = (\bigcup_{j=1}^{k} \overset{j}{\underset{G_i}{\Longrightarrow}})$, podobne definujeme aj: $\overset{\geq k}{\underset{\Gamma}{\Longrightarrow}}$, $\overset{=k}{\underset{\Gamma}{\Longrightarrow}}$. Definujeme $x \overset{\widetilde{\tau}}{\underset{\Gamma}{\Longrightarrow}} y$[1]: $x \overset{t}{\underset{\Gamma}{\Longrightarrow}} y = x \overset{t}{\underset{G_i}{\Longrightarrow}} y$ $i \in \{1, 2, ...n\}$ platí práve vtedy, ak: $x \overset{*}{\underset{G_i}{\Longrightarrow}} y$ a $\nexists z \neq y : y \underset{G_i}{\Rightarrow} z$*

Definícia 3.0.3. *Nech $f \in \{t, *, = 1, = 2, ..., \leq 1, \leq 2, ..., \geq 1, \geq 2, ...\}$ a nech Γ je $CDGS$. Potom jazyk definovaný systémom pri spôsobe prepisovania f je*

$$L_f(\Gamma) = \{w \in T^* \mid \exists r, i_1, i_2, ..., i_r \;\; S \overset{f}{\underset{G_{i_1}}{\Longrightarrow}} w_1 \overset{f}{\underset{G_{i_2}}{\Longrightarrow}} w_2 \overset{f}{\underset{G_{i_3}}{\Longrightarrow}} ... \overset{f}{\underset{G_{i_r}}{\Longrightarrow}} w_r \equiv w\}$$

Príklad 3.0.1. $\Gamma = (\{a, b, c\}, G_1, G_2, S)$
$G_1 = (\{A, B\}, \{A', B', a, b, c\}, \{A \to aA'b, B \to cB', A \to ab, B \to c\})$ a
$G_2 = (\{S, S', A', B'\}, \{A, B\}, \{S \to S', S' \to AB, A' \to A, B' \to B\})$, potom

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

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

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

[1] ďalej budeme písať len $x \overset{t}{\underset{\Gamma}{\Longrightarrow}} y$

Príklad 3.0.2. $\Gamma = (\{a\}, G_1, G_2, G_3, S)$
$G_1 = (\{S\}, \{A\}, \{S \to AA\})$
$G_2 = (\{A\}, \{S\}, \{A \to S\})$ a
$G_3 = (\{A\}, \{a\}, \{A \to a\})$. Potom $L_t(\Gamma) = \{a^{2^n} \mid n \geq 1\}$

Príklad 3.0.3. $\Gamma = (\{a, b, c\}, G_1, G_2, G_3, S)$
$G_1 = (\{S, A, A'\}, \{a, b, c\}, \{S \to S, S \to AcA, A' \to A\})$
$G_2 = (\{S, A, A'\}, \{a, b, c\}, \{A \to aA', a \to a\})$ a
$G_3 = (\{S, A, A'\}, \{a, b, c\}, \{A \to bA', A \to b\})$. Potom $L_{=2}(\Gamma) = L_{\geq 2}(\Gamma) = \{wcw \mid w \in \{a, b\}^+\}$

Skúmali sa viaceré možnosti voľby T:

Definícia 3.0.4. *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: $f \in \{*, t, =1, =2, ..., \leq 1, \leq 2, ..., \geq 1, \geq 2, ...\} = D$ $D' = \{*, =1, \geq 1, \leq 1, \leq 2, ...\}$[2] $A \in \{arb, ex, all, one\}$. (CD_nCF, 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.

Veta 3.0.1. $\mathcal{L}(CD_*CF, f, arb) = \mathcal{L}(CD_*CF, f, ex) = \mathcal{L}(CD_*CF, f, all) = \mathcal{L}(CD_*CF, f, one)$

Dôkaz:

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

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

(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. $P_i' = \{A' \to w' \mid A \to w \in P_i\}$, kde $w' = a_1', ...a_n'$ ak $w = a_1, ...a_n$. $G_{n+1} = (N_{n+1}, T_{n+1}, P_{n+1})$, kde $N_{n+1} = \{a' \mid a \in (\bigcup_i N_i \cup \bigcup_i T_i)\} \cup \{F\}$ $T_{n+1} = T$[3] $P_{n+1} = \{a' \to a \mid a \in T\} \cup \{a' \to F \mid a \notin T\} \cup \{F \to FF\}$

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

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

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

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

[2] v D' nie sú tie, ktoré nás nútia robiť viac ako 1 krok
[3] Týmto sme zabezpečili, že $(\bigcap_i T_i' \cap T_{n+1}) = T$

(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. $P'_i = \{A' \to w' \mid A \to w \in P_i\}$, kde $w' = a'_1, ...a'_n$ ak $w = a_1, ...a_n$.
$G_{n+1} = (N_{n+1}, T_{n+1}, P_{n+1})$, kde $N_{n+1} = \{a' \mid a \in (\bigcup_i N_i \cup \bigcup_i T_i)\} \cup \{F\}$ $T_{n+1} = T$ $P_{n+1} = \{a' \to a \mid a \in T\} \cup \{a' \to F \mid a \notin T\} \cup \{F \to FF\}$

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

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

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

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

$\supseteq$: 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 3.0.2. [4]

- $\mathcal{L}(CD_*CF, f) = \mathcal{L}_{CF}$ $\forall f \in D'$
- $\mathcal{L}_{CF} = \mathcal{L}(CD_1CF, f) \subsetneq \mathcal{L}(CD_2CF, f) \subseteq \mathcal{L}(CD_nCF, f) \subseteq \mathcal{L}(CD_*CF, f) \subseteq \mathcal{L}_{CFMatrix}$ [5] $\forall f \in D - D'$, $n \geq 3$
- $\mathcal{L}(CD_nCF, = k) \subseteq \mathcal{L}(CD_nCF, = s.k)$ $\forall k, n, s \geq 1$ [6]
- $\mathcal{L}(CD_nCF, \geq k) \subseteq \mathcal{L}(CD_nCF, \geq k+1)$ $\forall n, k \geq 1$
- $\mathcal{L}(CD_*CF, \geq) \subseteq \mathcal{L}(CD_*CF, =)$, *kde* $\mathcal{L}(CD_*CF, \geq) = \mathcal{L}(CD_*CF, \geq 1) \cup \mathcal{L}(CD_*CF, \geq 2) \cup ...$ *a* $\mathcal{L}(CD_*CF, =) = \mathcal{L}(CD_*CF, = 1) \cup \mathcal{L}(CD_*CF, = 2) \cup ...$
- $\mathcal{L}_{CF} = \mathcal{L}(CD_1CF, t) = \mathcal{L}(CD_2CF, t) \subsetneq \mathcal{L}(CD_3CF, t) = \mathcal{L}(CD_*CF, t) = \mathcal{L}(ETOL)$ [7]

Dôkaz: Za všetky len jeden príklad: $\mathcal{L}(CD_*CF, t) \subseteq \mathcal{L}(CD_3CF, t)$:
Nech $\Gamma \in \mathcal{L}(CD_*CF, t)$. Zostrojíme $\Gamma' \in \mathcal{L}(CD_3CF, t)$. V Γ' to bude vyzerať nasledovne: V prvej gramatike budú schované všetky gramatiky z Γ. Ďalšie dve gramatiky budú slúžiť na prepínanie v tej jednej[8].
$\Gamma = (T, G_1, G_2, ..., G_n, S)$[9], kde $G_i = (N_i, T_i, P_i)$
$\Gamma' = (T, G'_1, G'_2, G'_3, [S, 1])$
$G'_1 = (N'_1, T'_1, P'_1)$, kde $N'_1 = \{[A, i] \mid A \in N_i\}$, $T'_1 = \bigcup_{i=1}^{n} T_i$, $P'_1 = \{[A, i] \to [w', i] \mid A \to w \in P_i,\ 1 \leq i \leq n\}$, kde w' je vlastne w, ibaže všetky staré neterminály sú nahradené novými.

[4] Toto je: "Kilometrová veta s plno tvrdeniami na zamyslenie sa"

[5] 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

[6] toto tvrdenie sa nepodarilo doposiaľ dokázať všeobecnejšie, len pre násobky

[7] tabuľkové rozšírené 0L - systémy

[8] Musia byť dve, lebo keby sme mali iba jednu a keďže sa nachádzame v mode t, táto jedna gramatika by sa nám zacyklila

[9] Tu je nutné predpokladať, že n je párne. Ak by tomu tak nebolo, pridáme gramatiku, v ktorej bude $P = \emptyset$

$G_2' = (N_2', T_2', P_2')$, kde $N_2' = \{[A,i] \mid A \in \bigcup_{j=1}^{n} N_j,\ \ i = 1,...,n\}$, $T_2' = \varnothing$ a $P_2' = \{[A,i] \to [A,i+1] \mid i \equiv 1(mod\ \ 2)\}$
$G_3' = (N_3', T_3', P_3')$, kde $N_3' = N_2'$, $T_3' = \varnothing$ a $P_3' = \{[A,i] \to [A,i+1] \mid i \equiv 0(mod\ \ 2)\}$
$\cup \{[A,n] \to [A,1] \mid [A,n] \in N_3'\}$ □

Príklad 3.0.4.

G_1: $S \to aAB|...$ $\qquad [S,1] \to a[A,1][B,1]|...$
G_2:
G_3: $A \to bAS|...$ $\qquad [A,3] \to b[A,3][S,3]|...$

$S \underset{G_1}{\Rightarrow} aAB \underset{G_3}{\Rightarrow} abASB \Rightarrow ...$ $\qquad [S,1] \underset{G_1'}{\Rightarrow} a[A,1][B,1] \underset{G_2'}{\Rightarrow} a[A,2][B,1] \underset{G_2'}{\Rightarrow} a[A,2][B,2] \underset{G_3'}{\Rightarrow}$
$\underset{G_3'}{\Rightarrow} a[A,3][B,2] \underset{G_3'}{\Rightarrow} a[A,3][B,3] \underset{G_1'}{\Rightarrow} ab[A,3][S,3][B,3] \Rightarrow ...$

3.1 Niektoré otázky popisnej zložitosti

Definícia 3.1.1. *Definujeme miery:*

$Var(\Gamma) = \#(\bigcup_i N_i)$ *- počet neterminálov*

$Prod(\Gamma) = \sum_i \#P_i$ *- suma počtu pravidiel*

$Symb(\Gamma) = \sum_i (\sum_{A \to w \in P_i} (|\, w \,| + 2))$

Definícia 3.1.2. *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 3.1.3. *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:*

$Y \overset{M}{=} X \iff M_Y(L) = M_X(L)\ \ \forall L \in \mathcal{L}$

$Y \overset{M}{<_1} X \iff \exists L \in \mathcal{L}\ \ M_Y(L) < M_X(L)$

$Y \overset{M}{<_2} X \iff \forall n\ \ \exists L_n \in \mathcal{L}\ \ M_X(L_n) - M_Y(L_n) > n$

$Y \overset{M}{<_3} X \iff \exists L_n \in \mathcal{L},\ \ n \geq 1$ *také, že* $\lim_{n\to\infty} \frac{M_Y(L_n)}{M_X(L_n)} = 0$

$Y \overset{M}{<_4} X \iff \exists p$ *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 3.1.1. *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: Príklad: $(CD * CF, t) \overset{Var}{<}_4 CFG$ (VAR)
Uvažujme $L_n = \bigcup_{i=1}^{n} b(a^i b)^+$
Potom $Var_{CFG}(L_n) = n + 1$
$P_n = \{S_0 \to bS_i \mid 1 \leq i \leq n\} \cup \bigcup_{i=1}^{n} \{S_i \to a^i bS_i,\ \ S_i \to a^i b\}$
S menej neterminálmi to neide, lebo pomiešaním pravidiel by sme dostali zlé slová.
$Var_{CD_* CF,t}(L_n) \leq 3$
$\Gamma = (\{a, b\}, G_1, G_2, ..., G_{n+1}, S)$, kde
$G_i = (\{A\}, \{a, b\}, \{A \to a^i b\});\ 1 \leq i \leq n$
$G_{n+1} = (\{S, S', A\}, \{a, b\}, \{S \to bS', S' \to AS', S' \to A\})$ - táto gramatika pracuje ako prvá. □

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 taktoch. Označme si tieto gramatiky $G_1, G_2, \ldots, G_n$. Každá z týchto gramatík pracuje na svojej vetnej forme podľa svojich pravidiel. Naviac 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* $\Gamma = (N, K, T, G_1, \ldots, G_n)$ *kde* N *je množina neterminálov,* K *je množina komunikačných* ($query$) *symbolov štandardne označovaných* $K = \{Q_1, \ldots, Q_n\}$, T *je množina terminálov, pre každé* i $G_i = (N \cup K, T, P_i, S_i)$ *sú bez-ε gramatiky ľubovoľného typu*[1]*, 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* $(x_1, \ldots, x_n)$ *kde* $x_i \in V_\Gamma^*$.

Definícia 4.1.3. *Krok odvodenia je relácia* $\underset{\Gamma}{\Rightarrow}$ *na n-vetných formách definovaná nasledovne:* $(x_1, \ldots, x_n) \underset{\Gamma}{\Rightarrow} (y_1, \ldots, y_n)$ *práve vtedy keď nastane jeden z prípadov:*

1. *(prepisovací krok)* x_i *neobsahuje* $query$ *symbol a*
   - x_i *obsahuje neterminál, potom* $x_i \underset{G_i}{\Rightarrow} y_i$
   - x_i *je terminálne slovo, potom* $y_i = x_i$

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

2. *(komunikačný krok)* x_i *obsahuje* $query$ *symboly* $Q_{j_1}, \ldots, Q_{j_s}$ *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 Chomskeho 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* $\underset{\Gamma}{\Rightarrow}$ *na n-vetných formách definovaná nasledovne:* $(x_1, \ldots, x_n) \underset{\Gamma}{\Rightarrow} (y_1, \ldots, y_n)$ *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* $x_i \underset{G_i}{\Rightarrow} y_i$
   - $x_i = w_i$ *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 nejaký komponent n-vetnej formy je terminálne slovo, tak sa v ďalšom nemení až kým nejaká gramatika nepožiada o jej obsah. Ak sa v nejakej vetnej forme vyskytne neterminál, ktorý sa nedá prepisať, tak sa odvodenie 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.

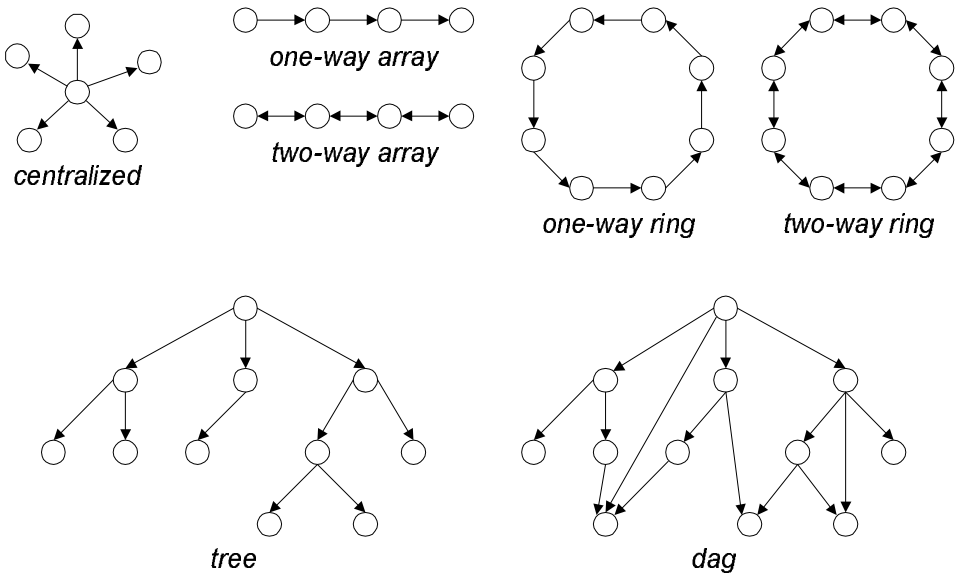
Definícia 4.1.5. *Jazyk generovaný PCGS systémom* Γ *je množina terminálnych slov vygenerovaných gramatikou* G_1. *Teda* $L(\Gamma) = \{x \in T^* \mid (S_1, \ldots, S_n) \overset{*}{\underset{\Gamma}{\Rightarrow}} (x, v_2, \ldots, v_n), v_i \in V_\Gamma^*\}$.

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, ...



Obrázok 4.1: *Príklady komunikačných štruktúr PCGS*

2. typ gramatík v komponentoch
3. počet komponentov (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[2], n je počet komponentov ($(n = *)$ označuje ľubolný počet komponentov) a $X \in \{REG, LIN, CF, \dots\}$ je typ komponentov[3].

4.3 Generatívna sila $PCGS$

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

Príklad 4.3.1. $\Gamma_1 = (\{S_1, S_2, S_3\}, K, \{a, b, c\}, G_1, G_2, G_3)$, kde množiny pravidiel príslušných gramatík sú:

- $P_1 = \{S_1 \to aS_1, S_1 \to aQ_2, S_2 \to bQ_3, S_3 \to c\}$
- $P_2 = \{S_2 \to bS_2\}$
- $P_3 = \{S_3 \to cS_3\}$

Skúsme si napísať pár krokov odvodenia:
$(S_1, S_2, S_3) \Rightarrow (aS_1, bS_2, cS_3) \Rightarrow \dots k - krokov \dots \Rightarrow (a^kS_1, b^kS_2, c^kS_3) \Rightarrow$

[2] ak nie je uvedený typ komunikačnej štruktúry, máme na mysli triedu všetkých $PCGS$-systémov bez ohľadu na štruktúru
[3] lineárne gramatiky (LIN) sú bezkontextové gramatiky, ktoré majú na pravej strane pravidiel najviac jeden neterminál

$(a^{k+1}Q_2, b^{k+1}S_2, c^{k+1}S_3) \Rightarrow (a^{k+1}b^{k+1}S_2, S_2, c^{k+1}S_3) \Rightarrow (a^{k+1}b^{k+2}Q_3, bS_2, c^{k+2}S_3) \Rightarrow$
$(a^{k+1}b^{k+2}c^{k+2}S_3, bS_2, S_3) \Rightarrow (a^{k+1}b^{k+2}c^{k+3}, b^2S_2, cS_3)$
Teraz už vidíme, že $L(\Gamma_1) = \{a^n b^{n+1} c^{n+2} \mid n \geq 1\}$, a teda centralizovaný $PCGS$ s troma regulárnymi komponentami vygeneroval jazyk, ktorý nie je regulárny, ba dokonca ani bezkontextový.

Príklad 4.3.2. $\Gamma_2 = (\{S_1, S_2, A\}, K, \{a, b, c\}, G_1, G_2)$, kde

- $P_1 = \{S_1 \to aS_1, S_1 \to Q_2, A \to aS_1, A \to c\}$
- $P_2 = \{S_2 \to A, A \to bA\}$

Napíšme si pár krokov odvodenia:
$(S_1, S_2) \Rightarrow (aS_1, A) \overset{*}{\Rightarrow} (a^k S_1, b^{k-1}A) \Rightarrow (a^k Q_2, b^k A) \Rightarrow (a^k b^k A, S_2) \Rightarrow (a^k b^k aS_1, A) \Rightarrow \ldots$
Ľahko vidno, že $L(\Gamma_2) = (\{a^n b^n \mid n \geq 1\})^+ c$, čo je opäť jazyk, ktorý nie je ani bezkontextový.

Príklad 4.3.3. $\Gamma_3 = (\{S_1, S_2\}, K, \{a, b, c, d\}, G_1, G_2)$

- $P_1 = \{S_1 \to cS_1 d, S_1 \to cQ_2 d\}$
- $P_2 = \{S_2 \to aS_2 b, S_2 \to ab\}$

Všimnime si niektoré možné odvodenia:

1. $(S_1, S_2) \overset{*}{\Rightarrow} (c^{k-1}S_1 d^{k-1}, a^{k-1}S_2 b^{k-1}) \Rightarrow (c^k Q_2 d^k, a^k S_2 b^k) \Rightarrow (c^k a^k S_2 b^k d^k, S_2)$ a tu sa Γ_3 zasekne
2. $(S_1, S_2) \overset{*}{\Rightarrow} (c^{k-1}S_1 d^{k-1}, a^{k-1}S_2 b^{k-1}) \Rightarrow (c^k Q_2 d^k, a^k b^k) \Rightarrow (c^k a^k b^k d^k, S_2)$
3. $(S_1, S_2) \overset{*}{\Rightarrow} (c^{k-1}S_1 d^{k-1}, a^{k-1}S_2 b^{k-1}) \Rightarrow (c^k S_1 d^k, a^k b^k) \overset{*}{\Rightarrow} (c^{k+i} Q_2 d^{k+i}, a^k b^k) \Rightarrow$
$(c^{k+i} a^k b^k d^{k+i}, a^k b^k)$

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

Príklad 4.3.4. $\Gamma_4 = (\{S_1, S_2\}, K, \{a, b, c\}, G_1, G_2)$

- $P_1 = \{S_1 \to S_1, S_1 \to Q_2 c Q_2\}$
- $P_2 = \{S_2 \to aS_2, S_2 \to bS_2, S_2 \to a, S_2 \to b\}$

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

Príklad 4.3.5. $\Gamma_5 = (\{S_1, S_2, S_3\}, K, \{a, b, c, d\}, G_1, G_2, G_3)$

- $P_1 = \{S_1 \to aS_1, S_2 \to aQ_2, S_3 \to d\}$
- $P_2 = \{S_2 \to bS_2, S_2 \to bQ_3\}$
- $P_3 = \{S_3 \to cS_3\}$

Všimnime si odvodenie, v ktorom nasleduje po sebe viac komunikačných krokov:
$(S_1, S_2, S_3) \overset{*}{\Rightarrow} (a^k S_1, b^k S_2, c^k S_3) \Rightarrow (a^{k+1}Q_2, b^{k+1}Q_3, c^{k+1}S_3) \Rightarrow (a^{k+1}Q_2, b^{k+1}c^{k+1}S_3, S_3) \Rightarrow (a^{k+1}b^{k+1}c^{k+1}S_3, S_2, S_3) \Rightarrow$
$(a^{k+1}b^{k+1}c^{k+1}d, bS_2, cS_3)$
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 $L(\Gamma) = \{w^{2^n}c \mid w \in \{a,b\}^*, n \geq 1\}$:
$\Gamma_6 = (\{S_1, S_2, S_3\}, K, \{a,b\}, G_1, G_2, G_3)$

- $P_1 = \{S_1 \to aS_1, S_1 \to bS_1, S_1 \to Q_2, \omega_1 \to Q_3, \omega_2 \to S_1', S_1' \to c\}$
- $P_2 = \{S_2 \to S_2, S_2 \to Q_1, S_1 \to \omega_1, S_1' \to \omega_1\}$
- $P_3 = \{S_3 \to S_3, S_3 \to Q_1, S_1 \to \omega_1, \omega_1 \to \omega_2, S_1' \to \omega_1\}$

Pozrime sa na to ako funguje odvodenie v Γ:
$(S_1, S_2, S_3) \overset{*}{\Rightarrow} (wS_1, S_2, S_3) \Rightarrow (wS_1, Q_1, Q_1) \Rightarrow (S_1, wS_1, wS_1) \Rightarrow (Q_2, w\omega_1, w\omega_1) \Rightarrow$
$(w\omega_1, S_2, w\omega_1) \Rightarrow (wQ_3, S_2, w\omega_2) \Rightarrow (ww\omega_2, S_2, S_3) \Rightarrow (wwS_1', Q_1, Q_1) \overset{*}{\Rightarrow} (w^4S_1', Q_1, Q_1) \overset{*}{\Rightarrow} (w^{2^n}S_1', S_2, S_3) \Rightarrow$
$(w^{2^n}c, \dots)$
Pravidlá v Γ 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 požiada o vetné formy G_2 a G_3, teda obsah vetnej formy G_1 sa zdvojnásobí. Ak by gramatiky takto "nespolupracovali", tak sa Γ zasekne.
Všimnime si ešte, že Γ vygeneruje slovo $w^{2^n}c$ v čase $O(n)$. G_1 na začiatku vygeneruje w a potom pri každom zdvojnásobení Γ 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. *Niekoľko porovnaní PCGS s triedami Chomského hierarchie:*

1. $\mathcal{L}(PCGS_nREG) - \mathcal{L}_{\mathcal{LIN}} \neq \emptyset$ *pre* $n \geq 2$
2. $\mathcal{L}(PCGS_nREG) - \mathcal{L}_{\mathcal{CF}} \neq \emptyset$ *pre* $n \geq 3$
3. $\mathcal{L}(PCGS_nLIN) - \mathcal{L}_{\mathcal{CF}} \neq \emptyset$ *pre* $n \geq 2$

Dôkaz: Pozri príklady 4.3.2, 4.3.1 a 4.3.3. □

Veta 4.3.2. $\mathcal{L}_{LIN} - \mathcal{L}(centrPCGS_*REG) \neq \emptyset$

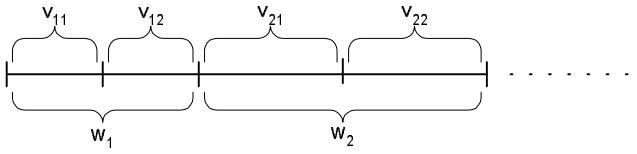
Dôkaz: Uvažujme jazyk $L = \{a^nb^mcb^ma^n \mid n, m \geq 1\}$.
Zrejme $L \in \mathcal{L}_{LIN}$. Ukážeme, že $L \notin \mathcal{L}(centrPCGS_*REG)$.
Bez ujmy na všeobecnosti môžme predpokladať, že G_1 negeneruje a-čka, lebo ak by generovala tak to isté dokáže aj iný komponent a G_1 môže požiadať o jej 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 (dosť 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[4] ako počet a-čiek, čo je spor s tým, že potrebujeme vygenerovať aj slová s ľubovoľne veľkým rozdielom medzi m a n. □

[4] m môže byť väčšie od n najviac lineárne v závislosti od pravidiel v G_2, G_3 a v G_4, G_5

Veta 4.3.3. $\mathcal{L}(centrPCGS_2REG) \subsetneq \mathcal{L}_{CF}$

Dôkaz: Podľa vety 4.3.2 stačí dokázať už len inklúziu $\subseteq$:
Majme $\Gamma = (N, K, T, G_1, G_2) \in centrPCGS_2REG$. Chceme zostrojiť bezkontextovú gramatiku $G = (N', T, P, S)$ takú, že $L(\Gamma) = L(G)$. Rozšírime množinu neterminálov takto:
$N' = N \cup \{[A, B] \mid A \in N, B \in N\} \cup \{\overline{A} \mid A \in N\}$, neskôr si ukážeme ako tieto nové neterminály budeme využívať. Najskôr sa zamyslime nad tým, ako bude vyzerať výsledné slovo $w \in L(\Gamma)$. w bude mať nasledujúce vlastnosti (obr.4.2):

1. w sa dá dekomponovať na $w = w_1 w_2 \dots w_s$ 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



Obrázok 4.2: *Tvar slova generovaného* $centrPCGS_2REG$

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:

$S \to [S_1, B]B\ \forall B \in N$
Teda máme špeciálnu sadu neterminálov $[S_1, B]$, ktoré určujú, že podslovo v_{1_1} sa začína generovať z počiatočného neterminálu S_1 a podslovo v_{1_2} bude mať posledný neterminál B. To v podstate znamená, že G sa nedeterministicky rozhodne pre neterminál B.

$[A, B] \to a[A', B']b$ ak $A \to aA' \in P_1$ a $B' \to bB \in P_2$
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.

$[Q_2, S_2] \to \varepsilon$
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 S_2, teda neterminál $[Q_2, S_2]$ sa už nebude ďalej rozvíjať, a teda ho vymažeme.

$A \to [A, B]B\ \forall A, B \in N$
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 Γ.
$(S_1, S_2) \Rightarrow (u_1A_1, v_1B_1) \overset{*}{\Rightarrow} (u_1u_2 \dots u_kQ_2, v_1v_2 \dots v_kB_k) \Rightarrow (u_1 \dots u_kv_1 \dots v_kB_k, S_2) \Rightarrow \dots$
Príslušná časť odvodenia v G bude vyzerať takto:
$S \Rightarrow [S_1, B_k]B_k \Rightarrow u_1[A_1, B_{k-1}]v_kB_k \Rightarrow u_1u_2[A_2, B_{k-2}]v_{k-1}v_kB_k \overset{*}{\Rightarrow}$
$u_1 \dots u_k[Q_2, S_2]v_1 \dots v_kB_k \Rightarrow u_1 \dots u_kv_1 \dots v_kB_k \Rightarrow \dots$
G najskôr nedeterministicky vybrala pravidlo $S \to [S_1, B_k]B_k$ 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 $[Q_2, S_2]$ z vetnej formy. Takto ostal vo vetnej forme len neterminál B_k, ktorým začne ďalšiu simuláciu G_1. Teraz G použije nejaké z pravidiel $B_k \to [B_k, B]B$ 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 nepožiada G_2 o jej vetnú formu (teda nevyprodukuje Q_2) a sama vyprodukuje terminálne slovo. Pre túto možnosť dodáme do P tieto pravidlá:

   - $B \to \overline{B}\ \forall B \in N$
   - $\overline{A} \to u\overline{B}$ ak $A \to uB \in P_1$

   Tým, že sme použili "pruhované" neterminály a nemáme pravidlo typu $\overline{A} \to A$ sme zmarili ďalšiu[5] komunikáciu, a teda simulujeme G_1 až kým nevygeneruje terminálne slovo.

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

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

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

□

Lema 4.3.1. *(Pumpovacia lema pre $centrPCGS_nREG$)*
Nech $L \in \mathcal{L}(centrPCGS_nREG)$. Potom existuje prirodzené číslo M, že $\forall w \in L$ také, že $|w| > M$ existuje m $1 \leq m \leq n$ a existujú x_i, y_i tak, že sú splnené nasledovné podmienky:

1. $w = x_1y_1x_2y_2 \dots x_my_mx_{m+1}$
2. $\forall i\ y_i \neq \varepsilon$
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 $\Gamma = (N, K, T, G_1, \dots, G_n) \in centrPCGS_nREG$. Každá gramatika má vo svojej vetnej forme najviac jeden neterminál a ten je posledným symbolom tejto vetnej formy. Teda konfigurácia Γ je tvaru $c = (x_1A_1, x_2A_2, \dots, x_nA_n)$. Budeme hovoriť, že dve konfigurácie $c_1 = (x_1A_1, x_2A_2, \dots, x_nA_n)$ a $c_2 = (y_1B_1, y_2B_2, \dots, y_nB_n)$, 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 M je počet všetkých možných rôznych

[5] k dispozícii máme samozrejme aj pravidlo $S \to \overline{S}$ teda žiadna komunikácia nemusí nastať

výskytov neterminálov vo vetných formách[6], tak za predpokladu, že $|w| > M$, 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:
$(S_1, S_2, \ldots, S_n) \overset{*}{\Rightarrow} (x_1A_1, x_2A_2, \ldots, x_nA_n) \overset{*}{\Rightarrow} (x_1z_1A_1, x_2z_2A_2, \ldots, x_nz_nA_n) \overset{*}{\Rightarrow} (w, \ldots)$
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, naviac $z_j \in T^+$ 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 ($c_1 \overset{*}{\Rightarrow} c_2$) 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 opakovať k krát pre ľubovoľné[7] k. Po týchto k iteráciach sa komponent j, pre ktorý z_j bolo neprázdne terminálne slovo, zmení na $x_j{z_j}^k A_j$. Ak po týchto k iteráciach dokončíme odvodenie časťou pôvodného odvodenia $c_2 \overset{*}{\Rightarrow} (w, \ldots)$, 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. □

Podobné pumpovacie lemy platia aj pre triedy $treePCGS_nREG$ a $dagPCGS_nREG$, avšak pre všeobecnú necentralizovanú komunikačnú štruktúru $PCGS$ sa pumpovať nedá.

Veta 4.3.4. *$\mathcal{L}(centrPCGS_{n-1}REG) \subsetneq \mathcal{L}(centrPCGS_nREG)$ pre $n \geq 2$*

Dôkaz: Inklúzia $\subseteq$ je zrejmá.
Treba ukázať, že existuje jazyk L taký, že $L \in \mathcal{L}(centrPCGS_nREG)$ a $L \notin \mathcal{L}(centrPCGS_{n-1}REG)$. Uvažujme jazyky $L_n = \{{a_1}^{k+1}{a_2}^{k+2}{a_3}^{k+3}\ldots{a_n}^{k+n} \mid k \geq 0\}$. Zostrojme $centrPCGS_nREG$ $\Gamma_n = (\{S_1, \ldots, S_n\}, K, \{a_1, \ldots, a_n\}, G_1, \ldots, G_n)$, kde

- $P_1 = \{S_1 \to a_1S_1, S_n \to a_n\} \cup \{S_i \to a_iQ_{i+1} \mid 1 \leq i \leq n-1\}$
- $P_j = \{S_j \to a_jS_j\}$ pre $2 \leq j \leq n$

Zrejme[8] $L(\Gamma_n) = L_n$. Chceme ukázať, že $L_n \notin \mathcal{L}(centrPCGS_{n-1}REG)$. Sporom, predpokladjme, že $L_n \in \mathcal{L}(centrPCGS_{n-1}REG)$. Podľa lemy 4.3.1 pre tento jazyk existuje prirodzené číslo M. Zoberme si slovo $w = {a_1}^{M+1}{a_2}^{M+2}{a_3}^{M+3}\ldots{a_n}^{M+n} \in L_n$. Zjavne $|w| \geq M$. Slovo w môžme 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. □

[6] tých je $M = |N \cup K|^n$
[7] ak túto postupnosť krokov úplne vynecháme, tiež dostaneme legálne odvodnie v Γ, teda pripúšťame aj $k = 0$
[8] čitateľovi to môže byť zrejmejšie, ak sa pozrie na príklad 4.3.1

Dôsledok 4.3.1. Hierarchia $\mathcal{L}(centrPCGS_nREG),\ n \geq 1$ je nekonečná.

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

Veta 4.3.5. *Hierarchie nasledujúcich tried sú nekonečné:*

- $\mathcal{L}(treePCGS_nREG),\ n \geq 1$
- $\mathcal{L}(dagPCGS_nREG),\ n \geq 1$

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

Veta 4.3.6. *Hierarchie nasledujúcich tried sú nekonečné:*

- $\mathcal{L}(two-way-arrayPCGS_nREG),\ n \geq 1$
- $\mathcal{L}(two-way-ringPCGS_nREG),\ n \geq 1$
- $\mathcal{L}(one-way-ringPCGS_nREG),\ n \geq 1$

Veta 4.3.7. $\mathcal{L}(PCGS_{n-1}REG) \subsetneq \mathcal{L}(PCGS_nREG)$ *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 $\Gamma \in PCGS_nREG$ tak, že $L(\Gamma) = L_n$. Potom zrejme $L_n \in \mathcal{L}(PCGS_nREG)$. Treba ukázať, že $L_n \notin \mathcal{L}(PCGS_{n-1}REG)$. Tento dôkaz je však dosť technický, a preto ho tu nebudeme uvádzať[9]. □

Dôsledok 4.3.2. Hierarchia $\mathcal{L}(PCGS_nREG),\ n \geq 1$ je nekonečná.

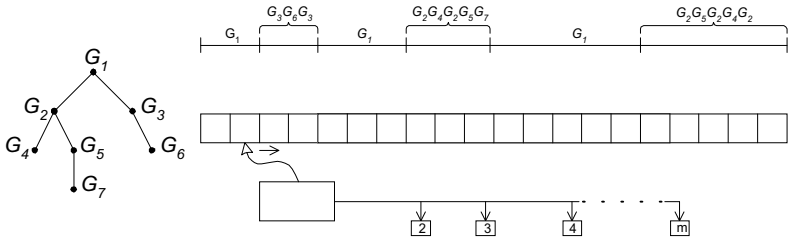
Poznámka 4.3.1. *Otvorenými problémami zostávajú nekonečnosť hierarchií* $\mathcal{L}(centrPCGS_nX), \mathcal{L}(PCGS_nX)$ *pre* $n \geq 1$ *a* $X \in \{CF, CS\}$.

4.4 Porovnanie $PCGS$ so sekvenčnými triedami

Veta 4.4.1. *Nech* Γ *je* $treePCGS_mREG(f(n))$*, kde* $f(n)$ *je počet komunikačných krokov potrebných na vygenerovanie slova dĺžky n. Potom existuje*[10] *nedeterministický* $(m-1)$ *počítadlový automat M pracujúci v lineárnom čase a akceptujúci jazyk* $L(\Gamma)$ *pričom vykoná* $2f(n)$ *obratov*[11] *a* $f(n)$ *testov na nulu*[12].

[9] podrobnosti možno nájsť v [5]
[10] n počítadlový automat je zásobníkový automat s n zásobníkmi, ktoré pracujú nad jednopísmenkovou abecedou
[11] zmena smeru hlavy v zásobníku resp. zmena inkrementácie counteru na dekrementáciu a opačne
[12] dosiahnutie dna zásobníka resp. counter dosiahol nulu

Dôkaz: Nech Γ má komponenty $G_1, \ldots, G_m$. Chceme zostrojiť nedeterministický $(m-1)$ počítadlový automat M simulujúci Γ. Celá simulácia Γ nám bude jasnejšia, ak sa zamyslíme nad tým, ako môže vyzerať slovo $w \in L(\Gamma)$ pri stromovej komunikačnej štruktúre. Slovo $w = w_{i_1} w_{i_2} \ldots w_{i_k}$, $\forall j \; i_j \in \{1, \ldots, m\}$ pozostáva z podslov w_{i_j} generovaných gramatikami G_i, pričom dĺžky týchto podslov závisia od konkrétnej komunikačnej štruktúry (obr.4.3).



Obrázok 4.3: *Viacpočítadlový automat a tvar slova generovaného treePCGS*

Vieme, že každá regulárna gramatika sa dá simulovať pomocou nedeterministického konečného automatu[13]. Podobne budú v pravidlách nášho automatu M simulované pravidlá regulárnych komponentov $G_1, \ldots, G_m$. M bude používať svoje počítadlá $C_2, \ldots, C_m$ na zabepečenie toho, aby jednotlivé gramatiky $G_1, \ldots, G_m$ neboli simulované dlhšie ako to vyžaduje daná situácia (konfigurácia). V každej konfigurácii M bude číslo $c(C_i)$ $\forall i \in \{2, \ldots, m\}$ 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 svojim otcom o vyprodukovanú vetnú formu, tak túto vetnú formu môže G_i generovať najviac $c(C_i)$ krokov).
M nedeterministicky striedavo simuluje gramatiky $G_1, \ldots, G_m$ 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 vegeneruje 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 poslená gramatika nevygeneruje terminálnu vetnú formu.
Z popisu práce M by malo by zrejmé, že počet obratov 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*[14], 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. □

[13] pozri [1]
[14] pravidlá typu $A \to B$ 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. $\mathcal{L}(treePCGS_mREG(f(n)) \subseteq NTIME(n) \cap NSPACE(\log n)$

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 $PCGS_mREG(f(n))$ priamočiaro pomocou m-páskového TS. Ten by na každej páske 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 $dagPCGS$.

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

Dôkaz: Nech $\Gamma = (N, K, T, G_1, \ldots, G_m) \in dagPCGS_mREG$. Chceme zostrojiť m-páskový[15] nedeterministický TS A pracujúci v lineárnom čase a akceptujúci jazyk $L(\Gamma)$.
V δ-funkcii si A uchováva pravidlá všetkých m gramatík a v stavoch si drží aktuálne neterminály[16] všetkých gramatík. Na páskach $T_1, \ldots, T_m$ sú vetné formy generované gramatikami $G_1, \ldots, G_m$ 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 Γ 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 páske T_i. Ak A rozhodne, že nebude, tak A zapíše na pásku T_i symbol B a ďalej nesimuluje odvodenie G_i na páske 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 páske T_1 neobjaví terminálne slovo, alebo až kým sa aspoň na jednej páske 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 páske 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[17] a výpočet sa zasekne.

[15] to znamená, že TS A má m hláv

[16] keďže gramatiky sú regulárne, vo vetnej forme danej gramatiky sa vyskytuje najviac jeden neterminál

[17] 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 $O(n)$. □

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. $\mathcal{L}(centrPCGS_2REG(n)) - \mathcal{L}(treePCGS_*REG(f(n))) \neq \varnothing$ *pre* $f(n) \notin \Omega(n)$

Dôkaz: Uvedieme[18] 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 $L \in \mathcal{L}(centrPCGS_2REG(n))$ a $L \notin \mathcal{L}(treePCGS_*REG(f(n)))$. Využíva sa pri tom simulácia $treePCGS_mREG(f(n))$ pomocou $m-1$ počítadlového automatu ako to bolo uvedené vo vete 4.4.1. □

Veta 4.4.5. *Pre* $f(n) \notin \Omega(n)$ *platí:*

- $\mathcal{L}(one-way-arrayPCGS_m(f(n))) \subsetneq \mathcal{L}(one-way-arrayPCGS_m(n))$ *pre* $m \geq 2$
- $\mathcal{L}(centrPCGS_m(f(n))) \subsetneq \mathcal{L}(centrPCGS_m(n))$ *pre* $m \geq 2$
- $\mathcal{L}(treePCGS_m(f(n))) \subsetneq \mathcal{L}(treePCGS_m(n))$ *pre* $m \geq 2$

Veta 4.4.6. $\mathcal{L}(centrPCGS_{k+1}(k)) - \mathcal{L}(treePCGS_*(k-1)) \neq \varnothing$ *pre ľubovoľnú konštantu* k.

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

- $\mathcal{L}(xPCGS_{k+1}(k-1)) \subsetneq \mathcal{L}(xPCGS_{k+1}(k))$
- $\mathcal{L}(xPCGS_*(k-1)) \subsetneq \mathcal{L}(xPCGS_*(k))$

4.5 Niektoré ďalšie vlastnosti $PCGS$

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

Veta 4.5.2. $centrPCGS_*CF <_4{}^{Var} CF$

Veta 4.5.3. $centrPCGS_*CF <_4{}^{Prod} CF$

[18] 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 zadefinovať 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 trošku v reálnejšej podobe známa z niektorých operačných systémov (UNIX), kde jeden proces vytvára nové a prideľuje 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, Σ je abeceda vstupných symbolov, Γ je pracovná (pásková) abeceda, q_0 je počiatočný stav a F je množina akceptačných stavov, δ je prechodová funkcia*

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

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[1]. 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 toľko vrcholov, koľko konfigurácii 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*

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

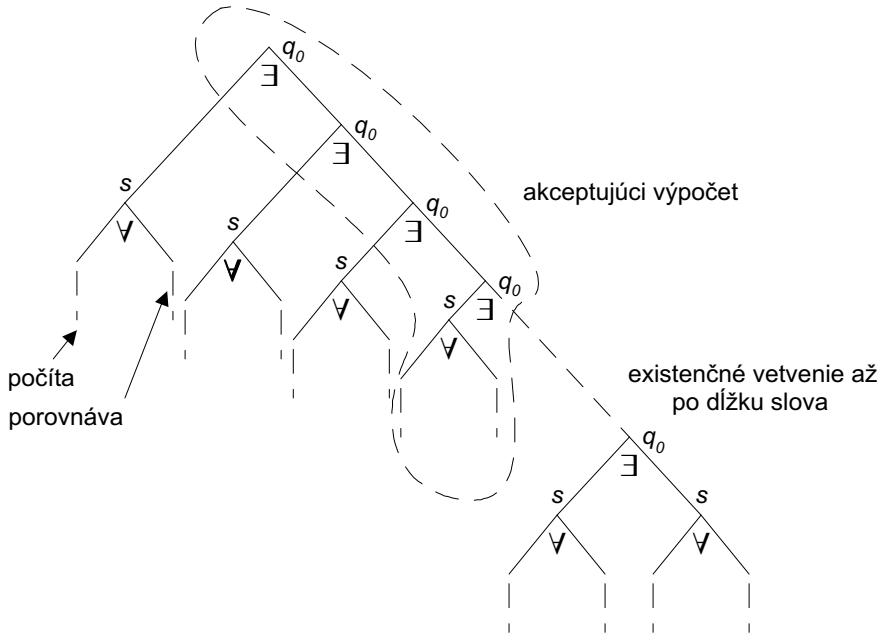
Príklad 5.1.1. Ukážeme si, ako alternovanie pomôže dvojhlavým konečným automatom[2] v generatívnej sile. Zoberme si jazyk $L = \{ww \mid w \in \{a, b\}^*\}$. Dalo by sa ukázať, že ho nie je možné akceptovať dvojhlavým konečným automatom, no zostrojíme preň alternujúci dvojhlavý konečný automat, ktorý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

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

[1] bol napr. dobrou pomôckou pri dôkaze ekvivalencie deteministických a nedeterministických Turingových strojov
[2] budeme uvažovať zariadenie s možnosťou pohybu hlavy iba jedným smerom



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

Dôkaz: Povieme si pár slov o oboch inklúziách:

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

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

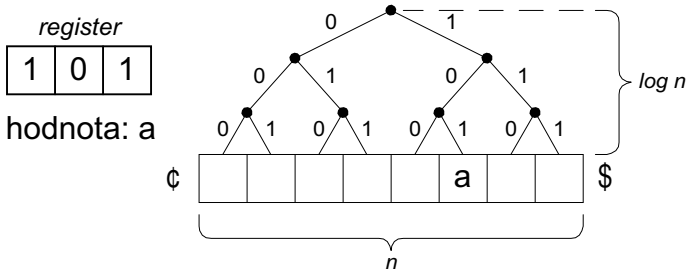
□

5.2 Miery zložitosti

Pre ATS si zadefinujeme 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

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 zadefinujeme.

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

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[3] c. Položme $m = c^{S(n)}$, $\forall i$ k_i je konfigurácia A, akceptujúci výpočet má tvar:

$$k_1 \vdash k_2 \vdash \cdots \vdash k_m$$

pričom k_1 je počiatočná konfigurácia, k_m je akceptačná konfigurácia (ak je náhodou akceptujúci výpočet kratší, dodefinujeme δ-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:

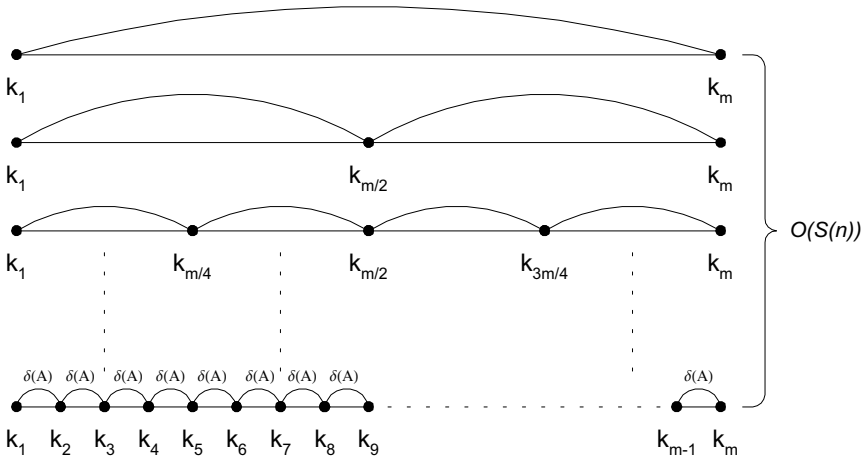
[3] ako už neraz, $c^{S(n)}$ je počet rôznych konfigurácií A na slove dĺžky n, keď máme k dispozícii priestor $S(n)$

(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 vnárať[4] dovtedy, kým sa nedostane na úroveň, kedy bude overovať, či sa dá z k_i dostať do k_{i+1} na jeden krok $\forall i = 1, \ldots, m-1$ (obr.5.3). Táto úroveň je z hľadiska rekurzie elementárna, na nej stačí overiť, či v δ-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 náveznosti konfigurácií v ATS A'*

Pozrime sa na proces hádania 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 $k_1 = q_0 w$, tento stav je existenčný. Teraz A' prejde do stavu q_H a háda k_m, pod $q_0 w$ 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 $S(n)$. Kvôli jednoduchosti si ho môžeme popísať nasledovne[5] (na čitateľa nechávame domyslenie ukončenia generovania):

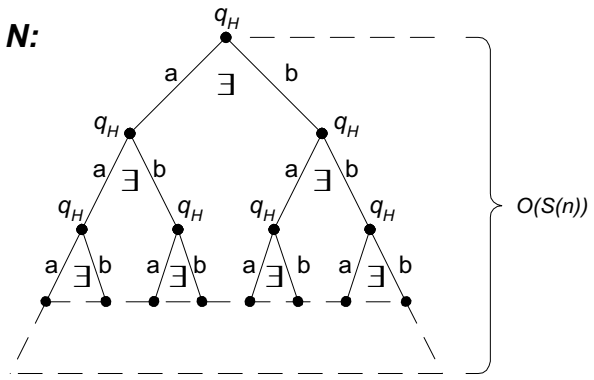
$$\delta(q_H, 1) = \{(q_H, a, 1), (q_H, b, 1)\}$$

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[6], ktoré pracujú paralelne a každý overuje polovicu toho, čo mal overiť materský proces, atď.

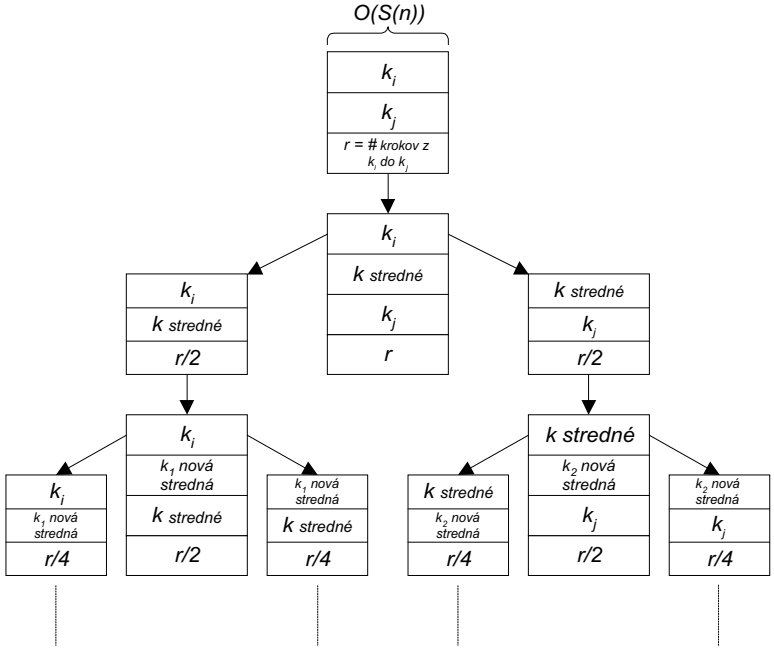
[4] každé rekurzívne volanie si môžme reprezentovať z hľadiska alternovania tak, že A' vytvorí nový proces, ktorý ako parametre dostane k_i, k_j, r a za úlohu má overiť, či sa z k_i dá dostať do k_j na r krokov (obr.5.5)

[5] konfigurácia ako ju popisujeme, neobsahuje stav, ani pozíciu hlavy, čitateľ si tieto veci iste rád domyslí sám

[6] to je práve jedno rekurzívne volanie



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



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

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 uhádnuť 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 $L(A') \in ATIME(S^2(n))$ □

Poznámka 5.3.1. *V predchádzajúcej vete sme použili "paralelnú verziu" konštrukcie, ktorá bola (v sevkvenčnej 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. Ako naznačila konštrukcia z vety 5.3.1, pri alternujúcich strojoch je situácia trochu iná. Pozornému čitateľovi iste neunikol fakt, že priestorové ohraničenie zostrojeného ATS zostalo nezmenené.

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

Lema 5.3.1. *K ľubovoľnému ATS A pracujúcemu v čase $T(n)$ s ľubovoľnou veľkosťou vetvenia*[7] *v úplnom strome konfigurácii existuje ATS A' pracujúci v čase $T(n)$, ktorého úplný strom konfigurácii 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 rekurzii skončíme až vtedy, keď bude vrchol stupňa ≤ 2. Na čitateľa nechávame premysieť, ako algoritmus aplikovať na celý strom konfigurácií[8]. 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 □

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

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

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 konfigurácií A nachádza podstrom akceptujúceho výpočtu.

A' bude prehľadávať strom konfigurácií A algoritmom PREORDER do hĺbky a priradzovať 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[9]

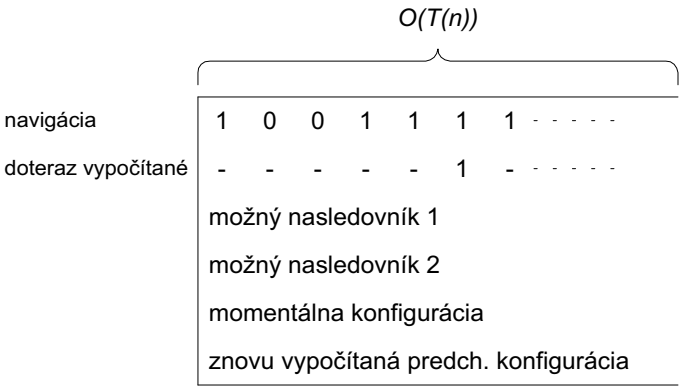
[7] veľkosťou vetvenia rozumieme maximálny počet konfigurácii dosiahnuteľný z ľubovoľnej konfigurácie na jeden krok

[8] pomôcka: δ-funkcia A má iba konečne veľa prvkov a vetvenie v úplnom strome konfigurácií je iba grafické znázornenie δ-funkcie, resp. výpočtu

[9] 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

- listom priradzujeme 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 konfigurácií 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 vracať 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 konfigurácii, teda máme priestorové ohraničenie $T(n)$ pre 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 □

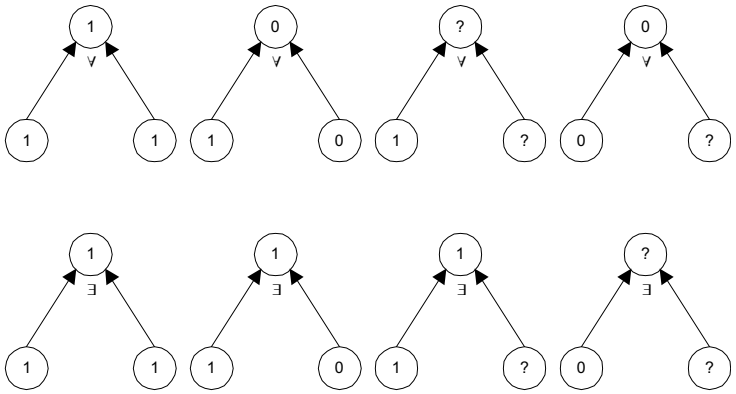
Veta 5.3.3. $ASPACE(S(n)) \subseteq \bigcup_{c>0} DTIME(c^{S(n)})$ *pre* $S(n) \geq \log n$

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 konfigurácii 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 $|\Gamma_A|^{S(n)}.S(n).|K_A| < r^{S(n)}$ konfigurácii A v lexikografickom usporiadaní, za každou konfiguráciou si nechá priestor (jedno políčko), ktorý ďalej využije[10], na to potrebuje čas $S(n).r^{S(n)}$
- do vyznačeného priestoru bude každej konfigurácii priraďovať hodnoty $0, 1, ?$ prechodom stromu konfigurácii A tak, že v každom prechode priradí hodnoty $0, 1$ rodičom tých detí, 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. $c = r^{10}$) a sme hotoví □

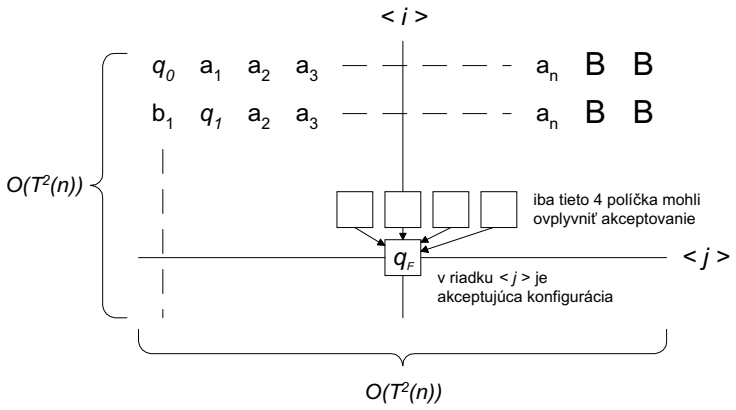


Obrázok 5.7: *Priraďovanie hodnôt rodičom v strome konfigurácii ATS A*

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 $< qw_k >_l$ konfiguráciu v k-tom riadku pre nejaký stav $q \in K_A$ a pozíciu stavu (hlavy) v tejto konfigurácii l, tak platí $< qw_k >_l \vdash < pw_{k+1} >_m$, 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 $< q_F w_i >_j$ je $q_F \in F_A$.

[10] priestor si na páske vyznačíme napr. #?#, pričom # slúži ako oddeľovač konfigurácii, nie je prvkom páskovej abecedy Γ, rovnako ani ?, ktorý hovorí, že o tomto políčku zatiaľ nič nevieme



Obrázok 5.8: *Tabuľka konfigurácií jednopáskového DTS A*

A' bude pracovať nasledovne:

- uhádne pozíciu <riadok, stĺpec> akceptačného stavu q_F v akceptujúcej konfigurácii tvaru $< q_F w_i >_j$, teda háda $< i, j >$, toto zaberie $\log T^2(n)$ políčok (môžeme zvoliť napr. binárnu reprezentáciu čísel i, j), to je $2.\log T(n) \approx \log T(n)$ políčok
- uhádne, ktorý akceptujúci stav sa na pozícii $< i, j >$ nachádza
- overí, či dobre hádal pozíciu $< i, j >$ a akceptačný stav nasledovne:

  1. uhádne (zmysluplne[11]) obsahy políčok $< i-1, j-1 >, < i-1, j >, < i-1, j+1 >,$ $< i-1, j+2 >$, pretože týmito obsahmi je obsah $< i, j >$ jednoznačne určený
  2. v univerzálnom vetvení overí, či hádal správne (každá vetva overí jedno políčko)

  Takto sa bude A' vracať v tabuľke až do prvého riadku (zrejme si bude dekrementovať i a pracovať s j podľa pohybu hlavy A), keď bude na pozícii $< 1, 1 >$ a bude v počiatočnom 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 informácii (dokonca veľmi málo). Ale o dĺžke sme si už povedali, že je $\log T(n)$, takže sme hotoví □

Dôsledok 5.3.2. $ASPACE(S(n)) = \bigcup_{c>0} DTIME(c^{S(n)})$ 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

[11] podľa δ-funkcie A

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 zprava doľava je triviálna a preto ju nebudeme ukazovať. Na dôkaz opačnej inklúzie 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 ε-prechody (konštrukcii sa nebudeme bližšie venovať, čitateľ ju nájde napr. v [1]). Zavedieme pár označení, ktoré sa nám neskôr budú hodiť:

- $all(q, a) = \{$množina všetkých stavov dosiahnuteľných v A_1 na jeden krok zo stavu q 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-ε ATS $A_1 = (K, \Sigma, \delta, q_0, F)$ zostrojíme NKA $A' = (K', \Sigma', \delta', q_0', F')$ nasledovne:

- Vezmime si úplný strom konfigurácii A_1 a pozrime sa naň po úrovniach. Keďže v A_1 nie sú ε-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

$$[q_{i_1}, \ldots, q_{i_k}] \in K' \overset{def}{\iff} q_{i_1}, \ldots, q_{i_k} \in K$$

- $\Sigma' = \Sigma$, teda vstupná abeceda sa nemení
- δ' definujeme nasledovne:
  - ak q je univerzálny a $\delta(q, a) = \{q_{i_1}, \ldots, q_{i_k}\}$, tak $\delta'([q], a) = \{[q_{i_1}, \ldots, q_{i_k}]\}$
  - ak q je existečný a $\delta(q, a) = \{q_{i_1}, \ldots, q_{i_k}\}$, tak $\delta'([q], a) = \{[q_{i_1}], \ldots, [q_{i_k}]\}$
  - rekurzívne definujeme $\delta'([q_{i_1}, \ldots, q_{i_j}, q_{i_{j+1}}, \ldots, q_{i_k}], a)$ ako $\{[all(q_{i_1}, a), \ldots, all(q_{i_j}, a), ex_1(q_{i_{j+1}}, a)], \ldots, [all(q_{i_1}, a), \ldots, all(q_{i_j}, a), ex_n(q_{i_k})]\}$ pričom $q_{i_1}, \ldots, q_{i_j}$ sú univerzálne stavy a $q_{i_{j+1}}, \ldots, q_{i_k}$ 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 druhý krát, ako stav si budeme pamätať iba $[r]$
- prirodzene definujeme $q_0' = [q_0]$
- $F' = \{[q_{i_1}, \ldots, q_{i_k}] \mid q_{i_1}, \ldots q_{i_k} \in F\}$

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

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 definovanm 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 $\{q_0, \ldots, q_4, p_1, p_2, r_1, \ldots, r_4\} \subseteq K$, $\Sigma = \{a, b\}$, $\{r_1, r_2, r_3\} \subseteq F$ a δ je zrejmá z obrázku[12], zostrojíme NKA $A' = (K', \Sigma', \delta', q_0', F')$ (jeho časť) podľa konštrukcie z vety 5.4 nasledovne:

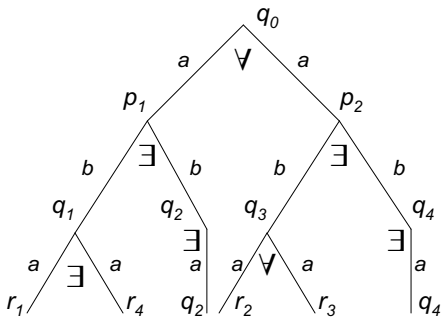
1. $\{[q_0], [p_1, p_2], [q_1, q_3], [q_1, q_4], [q_2, q_3], [q_2, q_4], [r_1, r_2, r_3], [r_2, r_3, r_4]\} \subset K'$
2. $\Sigma' = \Sigma$
3. $q_0' = [q_0]$
4. $[r_1, r_2, r_3] \subseteq F'$
5. δ' definujeme schématicky nasledovne:

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

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

[12] definovali sme iba časť AFSA A potrebnú pre výpočet na w

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

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

Na dôkaz $ASPACE(n) \subseteq \mathcal{L}_{APDA}$ potrebujeme k ATS A, ktorý pre vstup dĺžky n používa n políčok 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[13].

A' bude pracovať nasledovne[14]:

- najskôr v existenčnom vetvení uhádne počiatočnú konfiguráciu (bude hádať $n+1$ symbolov, háda aj stav) $q_0 w$, 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 $q_0 w$ vo vetvení, ktorého typ zodpovedá vetveniu 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 δ_A si usporiadame L,R)
  - A' musí po jednotlivých symboloch prejsť konfigurácie a testovať ich na rovnosť (obr.5.10b), resp. možnosť prechodu v δ_A, univerzálne sa rozvetví: v jednej vetve overí, či sa 1. symbol zhoduje s $(n+k)$-tym symbolom[15], resp.či existuje v δ_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 δ_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 □

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*

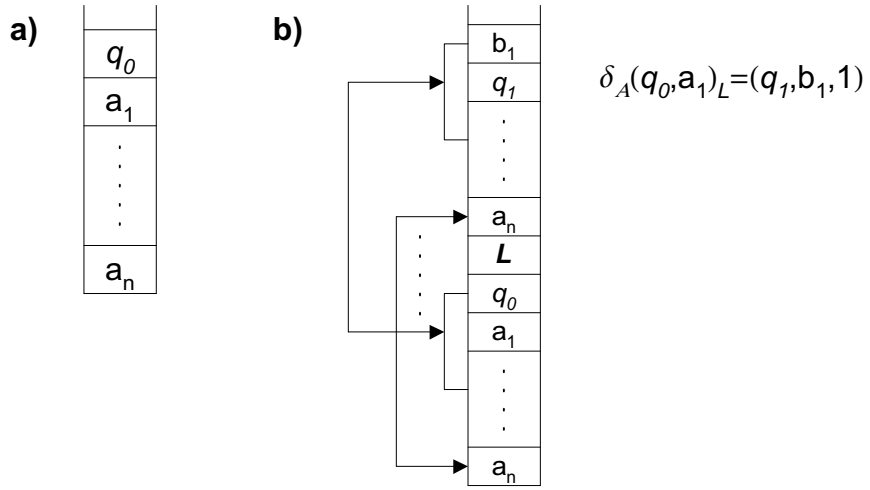
5.6 Synchronizované alternujúce stroje

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 univerzálne stavy, na:

[13] čitateľ si iste vie predstaviť podobnú konštrukciu ako v leme 5.3.2 pre APDA

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

[15] k je konštanta, jej určenie prenechávame na čitateľa

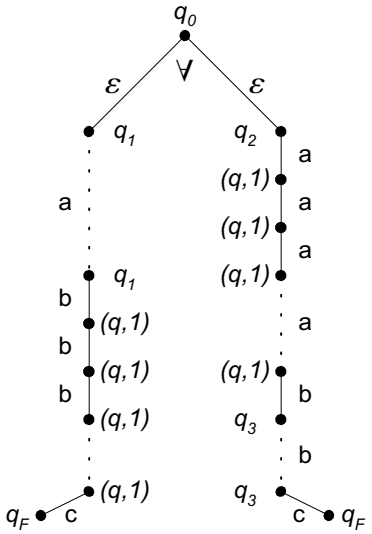


Obrázok 5.10: *Zásobník APDA A' a) po uhádnutí q_0w b) pri overovaní náväznosti konfigurácií*

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

Keď proces prejde do synchronizačného stavu, čaká, kým všetky ostatné prejdu 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* $\mathcal{L}_{CS}$

HKRS On power of synchronization on APDA

Simulácia pre LBA, proces pre políčko, hlavu,

Veta 5.6.1. $NLOG = \bigcup_k NFA(k) = SAFA(1)$

King

Poznámka 5.6.2. *Synchronizácia pri alternujúcich strojoch čas neovplyvní.*

Veta 5.6.2. *King*

$\mathcal{L}(SAFA) = \mathcal{L}_{CS}$

Veta 5.6.3.

- $PSPACE = AP = SALOGSPACE = \mathcal{L}(\bigcup_k 2SAFA(k))$
- $SASPACE(S(n)) = \bigcup_{c>0} DSPACE(c^{S(n)}) = \bigcup_{c>0} ATIME(c^{S(n)}) = \bigcup_{c>0} SATIME(c^{S(n)})$

$$DLOG \quad NLOG \qquad P \qquad\qquad NP \quad PSPACE$$

$$DFA \qquad NKA \quad AFA \quad \bigcup_k \mathcal{L}(SAFA(k, m^k)) \qquad SAFA$$

On comunication bounded SAFA acta inf. volume 31 (1994) 315-327

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 $\tau(v) \in \{B_{IN}\} \cup \{B_0\} \cup \{B_1\} \cup \{B_2\}$ a hodnotu $\mathcal{V}(v) \in \{0,1\}$. Vrchol v, pre ktorý typ $\tau(v) \in \{B_{IN}\}$, má vstupný stupeň 0 a nazývame ho vstupný vrchol. Vstupom pre booleovský obvod je n-tica rôznych vstupných vrcholov označených $\langle x_1, \ldots, x_n \rangle$. 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 "$\neg$" reprezentujúce booleovské funkcie identita a negácia a do B_2 patria hradlá "$\wedge$" a "$\vee$" reprezentujúce booleovské funkcie AND a OR. Vrcholy s výstupným stupňom 0 nazývame výstupné. Výstupom pre booleovský obvod je m-tica rôznych výstupných vrcholov označených $\langle y_1, \ldots, y_m \rangle$.*

Definícia 6.1.2. *Booleovský obvod so vstupom $\langle x_1, \ldots, x_n \rangle$ a výstupom $\langle y_1, \ldots, y_m \rangle$ počíta funkciu $f : \{0,1\}^n \to \{0,1\}^m$ 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 $\langle \mathcal{V}(y_1), \ldots, \mathcal{V}(y_m) \rangle$.*

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

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 $f_n : \{0,1\}^n \to \{0,1\}^{m(n)}$. Túto postupnosť nazývame trieda booleovských obvodov $\{C_n\}$ počítajúca funkciu $f : \{0,1\}^* \to \{0,1\}^*$ 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 $f : \{0,1\}^* \to \{0,1\}$, 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

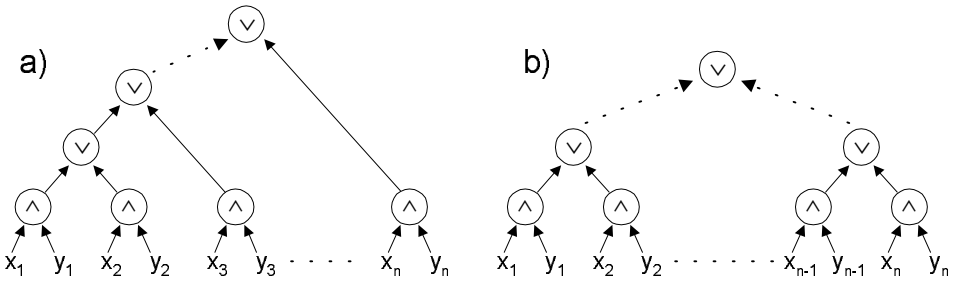
Pre booleovské obvody definujeme nasledujúce miery zložitosti:

- $DEPTH(C_n)$ = dĺžka najdlhšej cesty v obvode C_n
- $SIZE(C_n)$ = počet hradiel obvodu 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 $(x_1, \ldots, x_n)$ a $(y_1, \ldots, y_n)$. Vieme, že ich skalárny súčin vypočítame takto: $(x_1 \wedge y_1) \vee \cdots \vee (x_n \wedge y_n)$. Príslušný BO realizujúci tento výpočet je znázornený na obrázku 6.1a, 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 (obr. 6.1b). Pri zachovaní rovnakého počtu hradiel sme znížili hĺbku obvodu na $DEPTH(C_n) = O(\log n)$.



Obrázok 6.1: *Výpočet skalárneho súčinu na BO*

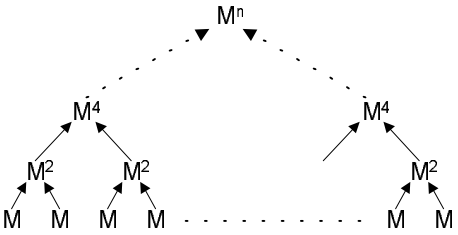
Príklad 6.2.3. Chceme vynásobiť dve booleovské matice A a B rozmeru $n \times n$. Výsledok je matica rovnakého rozmeru C, pričom jej prvky vypočítame nasledovne: $c_{i,j} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$. To je n^2 nezávislých skalárnych súčinov. Ak budeme každý takýto súčin počítať podľa príkadu 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:

$M^* = I \vee M \vee M^2 \vee \ldots M^n$, 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 (obr. 6.2) 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 zatial zadefinovali dokáže akceptovať ešte viac, pretože postupnosť BO môžeme určiť tak, že



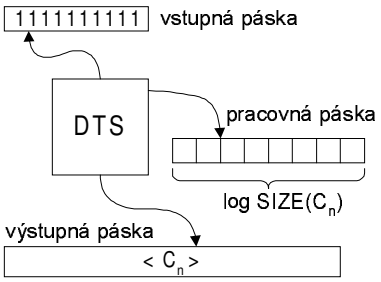
Obrázok 6.2: *Výpočet reflexívneho a tranzitívneho uzáveru matice*

pre rôzne dĺžky vstupu bude používať úplne iný mechanizmus spracovania vstupného slova, čo v iných modeloch v zásade nie je možné. 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 štvoríc* $\langle g, t, a, b \rangle$*, 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* $x_1, \ldots, x_n$ *štandardne priradíme čísla* $1, \ldots, n$ *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*[1]*), ak existuje DTS, ktorý pre každé n na vstupe* 1^n *vygeneruje kód booleovského obvodu* $\langle C_n \rangle$ *v priestore*[2] $\log(SIZE(C_n))$.



Obrázok 6.3: *DTS generujúci kód booleovského obvodu*

Čísla vrcholov v kóde obvodu C_n kódujeme binárne, takže pre veľkosť kódu dostávame $|\langle C_n \rangle| = O(SIZE(C_n).\log SIZE(C_n))$, čo nám určuje 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 6.3.1. *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.*

[1] *BC* - Borodin, Cook
[2] všimnime si, že pracovný priestor neurčujeme podľa veľkosti vstupu, ale na základe veľkosti vygenerovaného výstupu

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

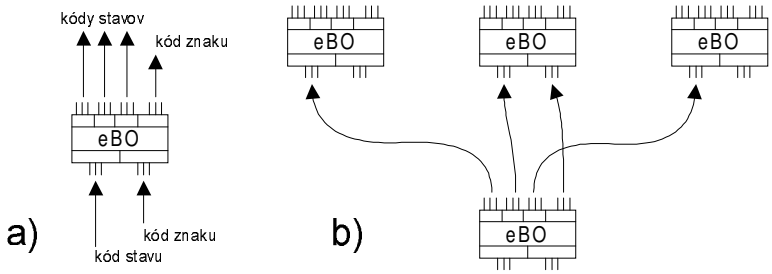
6.4 Porovnanie BO a TS

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 $SIZE(C_n) = O(T^2(n))$.*

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 vyzerať 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 (obr. 6.4a).

Z mechanizmu výpočtu Turingovho stroja vieme, že zmeny konfigurácií sú len lokálneho charakteru t.j. v jednom kroku výpočtu sa môže zmeniť len jednopísmenkové 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 6.4b.



Obrázok 6.4: *Elementárny booleovský obvod* (eBO)

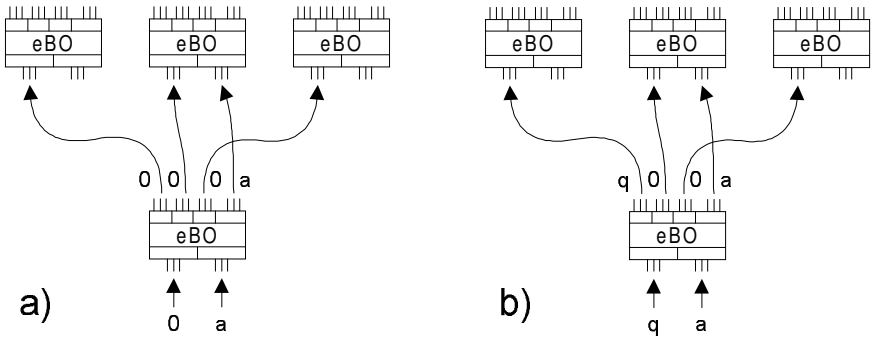
Každý elementárny obvod bude pracovať nasledovne:

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 (obr. 6.5a)

2. ak na vstup dostane kód znaku a a kód stavu q, tak podľa definície δ-funkcie A pošle na výstup kód nového znaku b, a podľa pohybu hlavy v δ-funkcii pošle príslušnému obvodu v nasledujúcej úrovni kód nového stavu, čím ho informuje o tom, že v nasledujúcom kroku bude hlava nad jeho políčkom a ostatným dvom pošle ako kód stavu nulový vektor (obr. 6.5b pre δ-funkciu DTS, ktorá pošle hlavu vľavo)

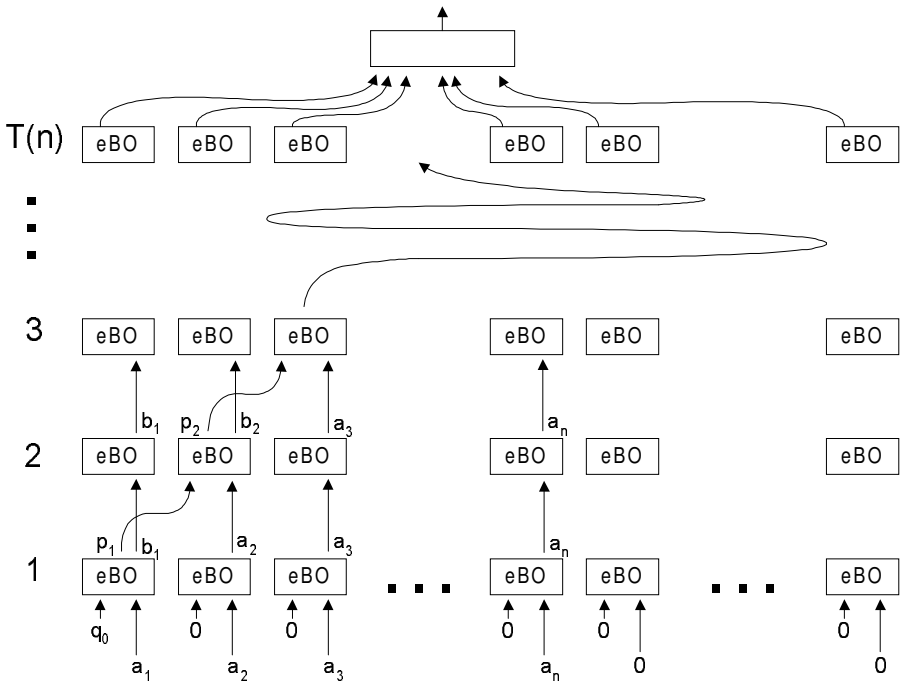
Takže celý obvod C_n bude mať $T^2(n)$ elementárnych obvodov ($T(n)$ úrovní pre každú konfiguráciu a v každej úrovni $T(n)$ obvodov pre každé políčko[3]). Vstupom pre C_n bude počiatočná konfigurácia A, teda prvý

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



Obrázok 6.5: *Komunikácia medzi úrovňami eBO*

obvod v prvej úrovni dostane na vstup kód počiatočného stavu a kód prvého písmenka slova $w = a_1 \dots a_n$, 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 (obr. 6.6). 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.



Obrázok 6.6: *Simulácia DTS booleovským obvodom*

Z uvedeného vyplýva, že $DEPTH(C_n) = T(n)$, a keďže uvažujeme konečný počet stavov, symbolov abecedy a konečný zápis δ-funkcie DTS A, tak dokážeme realizovať elementárny obvod z konečného počtu hradiel, 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 zrejmá.

To, že takto vytvorená postupnosť BO je naozaj BC-uniformná, nebudeme formálne dokazovať. Obmedzíme sa len na fakt, že každý obvod C_n tejto postupnosti bude vytváraný rovnakým postupom, čo v súlade s poznámnkou 6.3.1 zabezpečuje BC-unifomitu. □

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 zozstrojiť 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 w dĺžky n pracovať nasledovne:

1. A si na pásku napíše kód BO C_n simulovaním A' so vstupom[4] 1^n. To dokáže v čase $O(S(n).\log S(n))$, lebo kód $\langle C_n \rangle$ je takejto dĺžky.
2. A bude postupne ohodnocovať[5] vrcholy BO C_n tak, že vstupné vrcholy ohodnotí podľa príslušných hodnôt vstupu a hradlá ohodnotí podľa hodnôt vstupov hradla a jeho typu. 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 tretí krát na nájdenie a ohodnotenie samotného vrchola). Vrcholov je $S(n)$, veľkosť pásky je $O(S(n).\log S(n))$, teda na ohodnotenie všetkých vrcholov C_n potrebuje A čas $O(S^2(n).\log S(n))$.

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

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 $\mathcal{U}_{BC}SIZE(S(n)) \subseteq DTIME(S^3(n))$.

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

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 $DEPTH(C_n) = O(S^2(n))$.*

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 konfiguráciach môže byť A počas výpočtu na vstupnom slove w. Ak berieme do úvahy počty stavov, symbolov abecedy, políčok pracovnej pásky, 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 konfiguráciach. 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 patrí do jazyka L, je vlastne otázka, či počiatočná konfigurácia A je v relácii $\overset{*}{\vdash}$ s akceptačnou konfiguráciou. Relácia $\overset{*}{\vdash}$ je reflexívnym a tranzitívnym uzáverom relácie $\vdash$.

[4] Takýto vstup máme k dispozícii zo vstupného slova w tým, že všetky symboly budeme považovať za 1.
[5] ohodnotiť hradlo znamená na páske označiť symboly príslušného hradla v kóde C_n hodnotami 0 alebo 1

Keďže túto reláciu vieme reprezentovať konečnou booleovskou maticou, z príkladu 6.2.4 plynie, že aj reláciu $\overset{*}{\vdash}$ 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 $\log^2 k^{S(n)} = O(S^2(n))$.

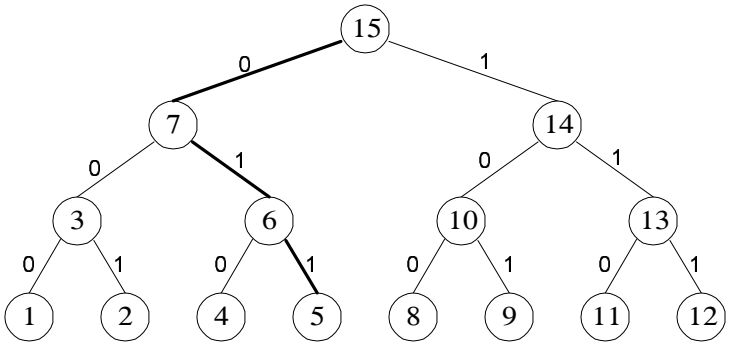
Teda náš BO C_n so vstupom $\langle w \rangle$ najskôr zostrojí maticu relácie $\vdash$ na vstupnom slove w (to sa dá konečným BO, v ktorom je zakódovaná δ-funkcia A), a potom už spomínaným spôsobom urobí nad touto maticou jej reflexívny a tranzitívny uzáver. A akceptuje práve vtedy, keď vo výslednej 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.

Tvrdenie o tom, že vytvorená postupnosť BO je BC-uniformná, nechávame opäť na poznámku 6.3.1. □

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 $w \in L$ je 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 $\log(SIZE(C_n)) \in O(D(n))$, 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[6]. Preto si bude na páske pamätať len navigačnú cestu od výstupného k práve prehľadávanému vrcholu (obr. 6.7). 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.



Obrázok 6.7: *Postorder prehľadávanie booleovského obvodu*

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 pásky A bude $O(D(n))$.

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

[6] Cesta môže byť dlhá maximálne $D(n)$ a kód každého vrcholu je veľkosti rádovo $\log SIZE(C_n)$, takže by sme potrebovali priestor rádovo $D^2(n)$.

Dôsledok 6.4.2. $NSPACE(Poly) = \mathcal{U}_{BC}DEPTH(Poly)$

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

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

Zo Savitchovej vety vieme, že $DSPACE(Poly) = NSPACE(Poly)$.

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

6.5 Druhá počítačová trieda a Nick Class

V tejto časti si povieme nakoľko sú paralelné výpočtové modely, ktoré sme doteraz spomenuli a ktoré ešte len spomenieme, vhodné na efektívne riešenie problémov, či je daný model vhodne zadefinovaný a ukážeme si akými mierami budeme posudzovať vhodnosť daného modelu.

Definícia 6.5.1. *Model počítača patrí do druhej počítačovej triedy, ak sekvenčný nedeterministický priestor je v polynomiálnom vzťahu s časom na danom modeli.*

Predchádzajúca definícia hovorí o tom aké kritérium sme zvolili na posudzovanie toho, či je nejaký výpočtový model pre nás zaujímavý (vhodný). Z doteraz spomenutých modelov patria do druhej počítačovej triedy modely Alternujúcich Turingových strojov (dôsledok 5.3.1) a BC-uniformných booleovských obvodov (dôsledok 6.4.2).

Takmer všetky paralelné modely patria do druhej počítačovej triedy. Ak uvažujeme nejaký nový model a chceme ho zaradiť do druhej počítačovej triedy, tak môžme urobiť simuláciu daného modelu s Turingovým strojom podobne, ako sme to urobili vo vetách 6.4.3 a 6.4.4, alebo urobíme simuláciu s modelom, ktorý tam už patrí.

Ďalšou otázkou, ktorá je pre nás zaujímavá, je aké problémy môžeme považovať za efektívne paralelne riešiteľné. Vieme, že pri sekvenčných modeloch tieto problémy tvoria triedu $\mathcal{P}$, teda sú to problémy, ktoré vieme riešiť v polynomiálnom čase. Za efektívne riešiteľné považujeme pri paralelných modeloch problémy patriace do triedy $\mathcal{NC}$ (Nick Class).

Definícia 6.5.2. *Nick Class*

- $\mathcal{NC}^i = \mathcal{U}_{BC} DEPTHSIZE(\log^i n, n^{O(1)})$
- $\mathcal{NC} = \bigcup_{i \geq 1} \mathcal{NC}^i$

Z doterajších poznatkov (pozri vety 6.4.2 a 6.4.4) vieme o práve zadefinovanej triede $\mathcal{NC}$ vysloviť pár tvrdení:

Veta 6.5.1. *Postavenie triedy $\mathcal{NC}$ medzi inými zložitostnými triedami.*

- $\mathcal{NC} \subseteq \mathcal{P}$
- $\mathcal{NC}^i \subseteq DSPACE(\log^i n)$

Doterajšie teoretické výsledky ukazujú, že medzi triedami platí nasledovná hierarchia:

$$\mathcal{NC} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE$$

Vieme, že $\bigcup_{i \geq 1} DSPACE(\log^i n) \subset PSPACE$. Z predchádzajúcej vety dostávame $\mathcal{NC} \neq PSPACE$. Otvoreným problémom zostáva, ktorá z inklúzií v spomenutej hierarchii tried je ostrá.

6.6 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 booleovké obvody, a tie neskôr využijeme pri porovnaní s alternujúcimi Turingovými strojmi.

Definícia 6.6.1. *Nech $\{C_n\}$ je postupnosť BO. Jej príslušným rozšíreným jazykom prepojení je jazyk $L_e = \{ \langle n, g, p, y \rangle \mid n \in \{1\}^+, g \in \{0,1\}^+, p \in \{L, R\}^*, |p| \leq \log SIZE(C_n), y \in \{x, \wedge, \vee, \neg\} \cup \{0,1\}^+$, kde*

- *n udáva*[7]*, že slovo $\langle n, g, p, y \rangle$ popisuje obvod C_n*
- *g je binárne kódované číslo vrchola obvodu C_n*
- *p je buď ε alebo navigačná cesta k vrcholu*
- *y je typ*[8] *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 6.6.1. Na bližšie pochopenie predchádzajúcej komplikovanej definície si ukážeme časť jazyka L_e prislúchajúceho k obvodu C_4 (obr. 6.8) z nejakej postupnosti $\{C_n\}$.

[7] všimnime si, že n je kódované unárne
[8] x je označenie pre vstupný vrchol

vrchol 1: $\langle 1111, 1, \varepsilon, x\rangle$

vrchol 2: $\langle 1111, 10, \varepsilon, x\rangle$

vrchol 3: $\langle 1111, 11, \varepsilon, x\rangle$

vrchol 4: $\langle 1111, 100, \varepsilon, x\rangle$

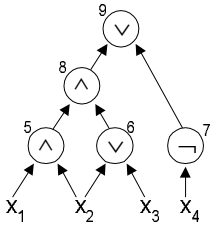
vrchol 5: $\langle 1111, 101, \varepsilon, \wedge\rangle, \langle 1111, 101, L, 1\rangle, \langle 1111, 101, R, 10\rangle$

vrchol 6: $\langle 1111, 110, \varepsilon, \vee\rangle, \langle 1111, 110, L, 10\rangle, \langle 1111, 110, R, 11\rangle$

vrchol 7: $\langle 1111, 111, \varepsilon, \neg\rangle, \langle 1111, 111, L, 100\rangle$

vrchol 8: $\langle 1111, 1000, \varepsilon, \wedge\rangle, \langle 1111, 1000, L, 101\rangle, \langle 1111, 1000, R, 110\rangle, \langle 1111, 1000, LL, 1\rangle,$
$\langle 1111, 1000, LR, 10\rangle, \langle 1111, 1000, RL, 10\rangle, \langle 1111, 1000, RR, 11\rangle$

vrchol 9: $\langle 1111, 1001, \varepsilon, \vee\rangle, \langle 1111, 1001, L, 1001\rangle, \langle 1111, 1001, LL, 101\rangle, \langle 1111, 1001, LR, 110\rangle,$
$\langle 1111, 1001, LLL, 1\rangle, \langle 1111, 1001, LLR, 10\rangle, \langle 1111, 1001, LRL, 10\rangle, \langle 1111, 1001, LRR, 11\rangle,$
$\langle 1111, 1001, R, 111\rangle, \langle 1111, 1001, RR, 100\rangle$



Obrázok 6.8: *Booleovský obvod C_4 z príkladu 6.6.1*

Definícia 6.6.2. *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 $L_e \in DTIME(\log SIZE(C_n))$. Na vyjadrenie tejto situácie zavedieme špeciálne označenie: $L_e \Subset DTIME(\log SIZE(C_n))$, a podobne pri $\mathcal{U}_{E^*}$ uniformite:
$L_e \Subset ATIMESPACE(DEPTH(n), logSIZE(C_n))$.

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 $\{1, \ldots, m\}$ resp. $\{0, \ldots, m-1\}$.

Lema 6.6.1. *Nech $\{C_n\}$ je postupnosť BO, L_e je jej príslušný rozšírený jazyk prepojení, a nech $f(n) \in \Omega(\log SIZE(C_n))$. Potom štandardný kód $\langle C_n\rangle$ sa dá vypočítať v $DSPACE(f(n))$ práve vtedy, keď $L_e \Subset DSPACE(f(n))$.*

Dôkaz: Dokážeme obe inklúzie:

"⇒" 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 $\langle g, t, a, b$ 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 $\Subset$ notácie), lebo v tomto priestore dokážeme simulovať A a pamätať si jedno slovo štandardného kódu.

"⇐" Máme DTS A' akceptujúci jazyk L_e v $\Subset$ 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 $g = 0, 1, 2, \ldots$ a $t = x, 0, 1, \wedge, \vee, I, \neg$ 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 $a = 0, 1, 2, \ldots$ 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 $b = 0, 1, 2, \ldots$, 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 štvoríc $\langle n, g, \varepsilon, t \rangle$ pre všetky $t \in \{x, 0, 1, \wedge, \vee, I, \neg\}$. To znamená, že vrchol s takýmto číslom v obvode C_n nie je, a vďaka predpokladu o číslovaní vrcholov "bez dier" vieme, že A už vygeneroval kódy všetkých vrcholov v obvode.

Na túto prácu stačí A priestor $f(n)$, lebo v tomto prietore 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 6.6.1. *Medzi uniformitami platia nasledujúce vzťahy:*

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

Dôkaz: Dokážeme všetky tri inklúzie:

1. Nech $\{C_n\}$ je $\mathcal{U}_E$-uniformná postupnosť BO. Z definície vieme, že príslušný jazyk prepojení $L_e \Subset DTIME(\log S(n))$. Zjavne žiadny DTS nepoužije viac priestoru ako času, takže $L_e \Subset DSPACE(\log S(n))$. Potom z lemy 6.6.1 plynie, že štandardný kód $\{C_n\}$ sa dá vypočítať v $DSPACE(\log S(n))$, teda $\{C_n\}$ je $\mathcal{U}_{BC}$-uniformná.

2. Nech $\{C_n\}$ je opäť $\mathcal{U}_E$-uniformná, teda vieme, že $L_e \in DTIME(\log S(n))$. Zrejme $L_e \in DTIMESPACE(\log S(n), \log S(n))$. Keďže DTS je špeciálnym prípadom ATS, tak $L_e \in ATIMESPACE(\log S(n), \log S(n))$. Zrejme $D(n) \geq \log S(n)$, teda $L_e \in DTIMESPACE(D(n), \log S(n))$, 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 $DSPACE(\log S(n))$. Z lemy 6.6.1 vieme, že príslušný jazyk prepojení $L_e \in DSPACE(\log S(n))$. Zo simulácie DTS na ATS (veta 5.3.1) plynie $L_e \in ATIMESPACE(\log^2 S(n), \log S(n))$. Z predpokladu $D(n) \geq \log^2 S(n)$ dostávame $L_e \in ATIMESPACE(D(n), \log S(n))$, čo znamená, že $\{C_n\}$ je $\mathcal{U}_{E^*}$-uniformná.

□

6.7 Porovnanie modelov BO a ATS

Prv než prejdeme k samotnému porovnaniu modelov, uvedieme si jeden normálový tvar booleovských obvodov, ktoré neobsahujú hradlo negácie. Tento normálový tvar následne využijeme v ďalšej vete.

Lema 6.7.1. *Pre každý BO C_n existuje ekvivalentný BO C'_n obsahujúci len vrcholy typu $\wedge$, $\vee$, "0", "1", "x", "$\overline{x}$", kde "$\overline{x}$" označuje negáciu vstupného vrchola "x".*

Dôkaz: Majme daný BO C_n. Chceme k nemu skonštruovať BO C'_n, ktorý nebude obsahovať hradlo $\neg$. Jediným miestom v obvode C'_n, kde sa môže negácia prejaviť, je na vstupe nahradením vstupného vrchola "x" jeho negáciou "$\overline{x}$".

Odstránenie hradiel $\neg$ z obvodu bude prebiehať nasledovne. Obvod C_n budeme prehľadávať (do šírky) od výstupného vrchola k vstupným. Ak narazíme na hradlo $\neg$, budeme rozlišovať šesť možných prípadov v závislosti na tom, aký vrchol je vstupom pre nájdené hradlo:

$\wedge$: Vstupom pre $\wedge$ sú nejaké hodnoty A, B. Vieme, že platí: $\neg(A \wedge B) = \neg A \vee \neg B$, teda vrcholy $\neg, \wedge$ nahradíme vrcholmi $\vee, \neg, \neg$ podľa obrázka 6.9a.

$\vee$: Vstupom pre $\vee$ sú nejaké hodnoty A, B. Vieme, že platí: $\neg(A \vee B) = \neg A \wedge \neg B$, teda vrcholy $\neg, \vee$ nahradíme vrcholmi $\wedge, \neg, \neg$ podľa obrázka 6.9b.

$\neg$: Vstupom pre $\neg$ je nejaká hodnota A. Platí $\neg(\neg A) = A$, teda vrcholy $\neg, \neg$ z obvodu vynecháme (obr. 6.9c).

"0" : Vrcholy $\neg$,"0" nahradíme vrcholom "1" (obr. 6.9d).

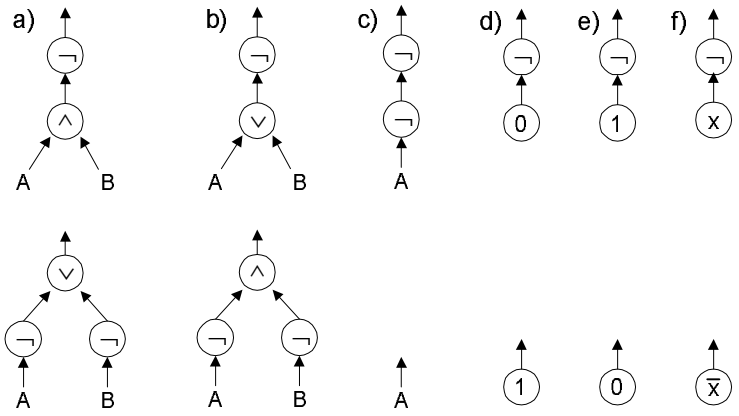
"1" : Vrcholy $\neg$,"1" nahradíme vrcholom "0" (obr. 6.9e).

"x" : Vrchol $\neg$ a vstupný vrchol "x" nahradíme vstupným vrcholom "$\overline{x}$" s opačnou hodnotou (obr. 6.9f).

V prvých dvoch prípadoch pri modifikácii obvodu opäť využívame hradlá $\neg$. Treba si však uvedomiť, že tieto hradlá sú v obvode o úroveň nižšie ako pôvodné nahradzované hradlo. Po príslušnej úprave obvodu pokračujeme rekurzívne v prehľadávaní obvodu v ďalšej úrovni až sa dostaneme k vstupným vrcholom.

Takto modifikovaný obvod C'_n zrejme akceptuje rovnaký jazyk ako C_n, lebo každá elementárna modifikácia bola logicky ekvivalentná. □

Veta 6.7.1. *Nech $S(n) \geq n$, potom platí:*
$\mathcal{U}_{E^*} DEPTHSIZE(D(n), S(n)) \subseteq ATIMESPACE(D(n), \log S(n))$.



Obrázok 6.9: *Modifikácia BO do normálového tvaru*

Dôkaz: Nech $\{C_n\}$ je E^*-uniformná postupnosť BO hĺbky $D(n)$ a veľkosti $S(n)$ v normálnom tvare z predchádzajúcej lemy, a nech A' je ATS akceptujúci príslušný jazyk prepojení L_e v čase $D(n)$ a priestore $\log S(n)$. Chceme zostrojiť ATS A simulujúci $\{C_n\}$. Uvažujme nejaké slovo $w = a_1 \dots a_n \in L(\{C_n\})$ dĺžky n (t.j. w je akceptované BO C_n). Ukážeme, ako pracuje A na vstupnom slove w.

A uhádne číslo a typ výstupného vrchola g_{out}, t_{out}. To urobí tak, že sa existenčne rozvetví na veľa vetiev a v každej vetve na pracovnú pásku zapíše binárne číslo dĺžky najviac $\log S(n)$ a nejaký typ vrchola[9]. Každá takáto vetva overí svoje hádanie t.j. univerzálne spustí proces, v ktorom sa bude simulovať A' na vstupe[10] $\langle n, g_{out}, \varepsilon, t_{out}\rangle$, teda či vrchol s číslom g_{out} a typom t_{out} v obvode C_n existuje. Ak taký vrchol existuje, potrebujeme overiť, či je naozaj výstupný, teda opäť vo veľkom univerzálnom vetvení simuláciou A' overujeme, či pre všetky $h \in \{0,1\}^*$ také, že $|h| \leq \log S(n)$ platí $\langle n, h, L, g_{out}\rangle \notin L_e$, a zároveň $\langle n, h, R, g_{out}\rangle \notin L_e$ to znamená, že vrchol g_{out} nemá nasledovníkov.

Uvažujme ďalej výpočet na vetve, ktorá úspešne uhádla výstupný vrchol. Na pracovnej páske máme zapísané $\langle g_{out}, t_{out}, \varepsilon\rangle$. V nasledujúcom bude A hádať typy vrcholov, ktoré sú vstupmi g_{out}. A sa podľa typu t_{out} ($\wedge$ univerzálne, $\vee$ existenčne) rozvetví pričom v jednej vetve bude mať na páske zapísané $\langle g_{out}, L\rangle$ a v druhej $\langle g_{out}, R\rangle$.

Vo všeobecnosti bude mať A na páske zapísané $\langle g, p\rangle$, kde g je číslo nejakého vrchola a $p \in \{L, R\}^*$ je nejaká navigačná cesta dĺžky najviac $\log S(n)$. A v tomto prípade uhádne typ vrchola $g(p)$ (p-predchodca vrchola g) a univerzálne sa rozvetví na dve vetvy:

- V jednej overí, že hádal správne typ t.j. uhádne číslo p-predchodcu vrchola g (teda v existenčnom vetvení si A zapíše na pásku binárne číslo $g(p)$) a overí, že hádal správne, teda simuluje A' na vstupe $\langle n, g, p, g(p)\rangle$. Následne overí, že pre p-predchodcu vrchola g hádal správny typ, teda simuluje A' na vstupe $\langle n, g(p), \varepsilon, t_{g(p)}\rangle$.
- V druhej vetve pokračuje vo výpočte. To znamená, že podľa hádaného typu vrchola $g(p)$ sa ($\wedge$ univerzálne, $\vee$ existenčne) rozvetví, pričom v jednej vetve si k navigačnej ceste p na pásku pripíše L a v druhej si pripíše R, teda A bude v situácii kedy má na páske zapísané $\langle g, p'\rangle$, kde p' je v jednom prípade pL, v druhom pR. V oboch vetvách A postupuje rovnako ako sme naznačili vyššie.

[9] Naozaj to bude tak, že pod počiatočnou konfiguráciou sa rozvetví binárny strom hĺbky $\log S(n)$, pričom pri prechode na nižšiu úroveň A pripíše k už vygenerovanému binárnemu číslu jeden bit podľa cesty v binárnom strome od koreňa.

[10] A má v každej vetve na pracovnej páske zapísané iba $\langle g_{out}, t_{out}\rangle$, preto treba ešte vhodne dopísať ε. n na pracovnú pásku zapísať nemôžeme, lebo jej priestor $\log S(n)$ je na to malý. Nie je však problém upraviť A' tak, aby n, keďže je kódované unárne, prečítal zo vstupnej pásky A, kde ja zapísané vstupné slovo w dĺžky n.

V prípade, že typ vrchola $g(p)$ je vstup, A uhádne, ktorý i-ty vstup to je a či je typu "x_i" alebo "$\overline{x}_i$". Hádanie v univerzálnom vetvení overí, teda v jednej vetve simuluje A' na vstupe $\langle n, g(p), \varepsilon, x_i \rangle$ resp. $\langle n, g(p), \varepsilon, \overline{x}_i \rangle$ a v druhej vetve výpočet končí s tým, že

- ak je vstup typu "x_i", tak A akceptuje, ak $a_i = 1$ (neakceptuje, ak $a_i = 0$)
- ak je vstup typu "$\overline{x}_i$", tak A akceptuje, ak $a_i = 0$ (neakceptuje, ak $a_i = 1$)

Uvažujme teraz prípad, že dĺžka navigačnej cesty p na páske dosiahne hranicu $\log S(n)$. To je problém, pretože priestor pracovnej pásky A je ohraničený na $O(\log S(n))$ a naviac pri väčšej dĺžke p by sme už ani nemohli jednoducho overovať hádanie simuláciou A', lebo slová jazyka L_e sú definované s navigačnou cestou dĺžky najviac $\log S(n)$. V tomto prípade postupojeme nasledovne:
A má na páske zapísané $\langle g, p \rangle$, pričom $|p| = \log S(n)$. V existenčnom vetvení uhádne číslo h a typ t_h vrchola $g(p)$ a univerzálne

- overí či $\langle n, g, p, h \rangle \in L_e$ a $\langle n, h, \varepsilon, t_h \rangle \in L_e$
- pokračuje vo výpočte, pričom na páske má zapísané $\langle h, t_h, \varepsilon \rangle$, teda sme v podobnej situácii, ako sme boli na začiatku po uhádnutí výstupného vrchola.

Z konštrukcie by malo byť vidieť, že skonštruovaný ATS A naozaj akceptuje práve slová z jazyka $L(\{C_n\})$.
Zamyslime sa teraz nad časovou a priestorovou zložitosťou A:

- Uhádnutie výstupného vrchola trvá čas $\log S(n) \leq D(n)$.
- Ak má A na páske zapísanú cestu p dĺžky menšej ako $\log S(n)$, tak A sa v hlavnom výpočte (hádanie ďalšej úrovne C_n) posunie v konštantnom čase vďaka tomu, že pri rozvetvovaní stačí na páske predĺžiť cestu p o L resp. R. Všetky dlhé hádania a overovania sa robia v bočných vetvách výpočtu, a teda ich netreba do celkového času zarátať. Musíme si však uvedomiť, že tieto bočné vetvy sú dlhé najviac $D(n)$, lebo v nich A buď háda, teda zapisuje na pásku (ale sú to vždy informácie dĺžky najviac $D(n)$), alebo simuluje A', ten však z definície pracuje v čase $O(D(n))$.
- Každú $(\log S(n))$-tú úroveň výpočtu dosiahne dĺžka p hranicu $\log S(n)$. A vtedy musí znova hádať v hlavnom výpočte číslo vrchola - na to spotrebuje čas $O(\log S(n))$. Keďže obvod má hĺbku $D(n)$, takýchto situácií sa počas výpočtu vyskytne $\frac{D(n)}{\log S(n)}$ krát. Na riešenie všetkých týchto situácií spotrebuje A čas $O(\frac{D(n)}{\log S(n)} \log S(n)) = O(D(n))$.
- Na koncoch jednotlivých vetiev výpočtu pri zisťovaní hodnoty i-teho vstupu musí A presunúť hlavu nad tento symbol. Pri dĺžke vstupu n by sme na túto operáciu potrebovali čas $O(n)$, čo je priveľa. Preto budeme uvažovať, že ATS A má rýchly prístup na vstupnú pásku, teda nám bude stačiť čas $\log n$, čo je menej ako $D(n)$ vďaka predpokladu $S(n) \leq n$ zo znenia vety.
- Počas celého výpočtu bolo na páske zapísané číslo nejakého vrchola dĺžky najviac $\log S(n)$, navigačná cesta dĺžky najviac $\log S(n)$ a typ vrchola konštantnej dĺžky, čo celkovo dáva nároky na priestor $O(\log S(n))$.

Z uvedeného vyplýva, že sme konštrukciou ATS A splnili požiadavky na korektnosť akceptácie ako aj na časovú a priestorovú zložitosť. □

Veta 6.7.2. *Nech $S(n) \geq \log n$, potom pre vhodnú konštantu k platí:*
$ATIMESPACE(T(n), S(n)) \subseteq \mathcal{U}_E DEPTHSIZE(T(n), k^{S(n)})$

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 môžme pozerať 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[11] 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 nultej úrovni bude jediný vrchol reprezentujúci počiatočnú konfiguráciu. Hľadaný BO C_n akceptujúci vstupné slovo w dĺžky n bude vyzerať veľmi podobne. Každému vrcholu v úplnom strome konfigurácií zodpovedá jeden vrchol v obvode C_n.

Vrcholu na úrovni t, ktorý reprezentuje konfiguráciu[12] c v strome konfigurácií zodpovedá v obvode C_n vrchol s číslom $f(t,c)$, kde $f : N \times N \to N$ je funkcia zachovávajúca 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[13] $\neg$ 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 $c \underset{A}{\vdash} c_1$ a $c \underset{A}{\vdash} c_2$ (obr. 6.10).

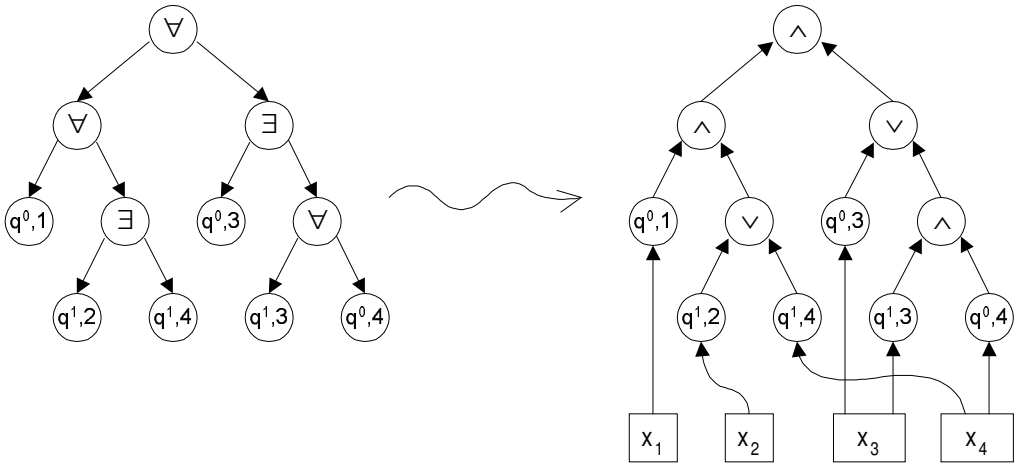
Takto zostrojený obvod C_n pracuje presne tak ako výpočet ATS, ktorý sme opísali na začiatku dôkazu. Teda príslušná postupnosť BO $\{C_n\}$ zrejme akceptuje jazyk $L(A)$.

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

[11] na jednej pozícii vstupu určenej číslom na špeciálnej páske

[12] 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.

[13] 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.



Obrázok 6.10: *Konštrukcia BO k ATS*

v strome konfigurácií. Keďže do konfigurácií nezahŕňame vstup, tak počet všetkých možných konfigurácií je $k^{S(n)}$ pre vhodné k. 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 $k^{S(n)}$. 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 môžme uvažovať opäť akýsi normálový tvar ATS, ktorý bude mať v úplnom strome konfigurácií všetky konfigurácie disjunktné, a teda $SIZE(C_n) = O(k^{S(n)})$. □

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 asembleru. Jednotlivé procesory spolu komunikujú cez spoločnú zdielanú 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

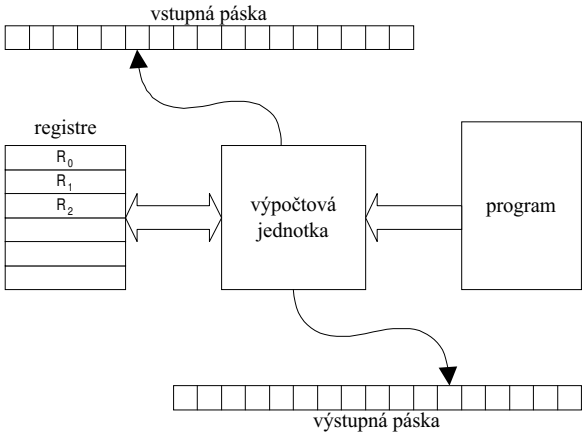
Skôr ako si zadefinujeme výpočtový model $PRAM$, s ktorým budeme ďalej pracovať, povieme si niečo o jeho základnej základnej výpočtovej jednotke, ktorou je RAM.

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 pásky a neobmedzeného počtu registrov $R_0, R_1, R_2, \ldots$, pričom v jednom registri môže uchovávať ľubovoľne veľké celé číslo (obr.7.1). Program výpočtovej jednotky je postupnosť jednoduchých[1] inštrukcií, ktoré sú uvedené v tabuľke ??. Výpočet začína prvou inštrukciou a končí inštrukciou HALT.*

Model RAM sa dá zadefinovať viacerými spôsobmi. Uvedenú definíciu môžme, 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.

[1] 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
READ	prečítaj nasledujúci symbol zo vstupu a zapíš ho do registra R_0
WRITE	obsah registra R_0 zapíš na výstup
STORE R_i	obsah registra R_0 zapíš do registra R_i
COPY R_i	skopíruj obsah registra R_i do registra R_0 ($R_0 \leftarrow [R_i]$)
CONST c	do registra R_0 zapíš hodnotu c
ADD R_i	$R_0 \leftarrow [R_0] + [R_i]$
SUB R_i	$R_0 \leftarrow [R_0] - [R_i]$
MULT c	$R_0 \leftarrow [R_0].c$
DIV c	$R_0 \leftarrow [R_0]/c$
IFZERO i	ak register R_0 obsahuje 0, tak pokračuj inštrukciou i
GOTO i	pokračuj inštrukciou i
HALTaccept	ukonči výpočet a akceptuj
HALTreject	ukonči výpočet a neakceptuj

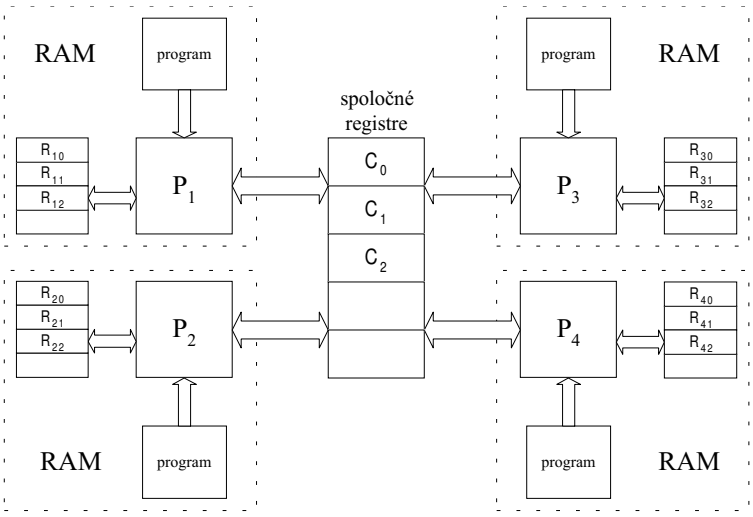
Tabuľka 7.1: Zoznam jednoduchých inštrukcií *RAMu*

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 *RAMov* komunikujúcich prostredníctvom zdieľanej pamäte.

Definícia 7.1.2. *PRAM (Parallel Random Access Machine) je výpočtový model pozostávajúci z neobmedzeného počtu RAM procesorov označených $P_0, P_1, P_2, \ldots$ a neobmedzeného počtu spoločných (zdieľaných) registrov $C_0, C_1, C_2, \ldots$ Každý procesor P_i má svoje identifikačné číslo (index), má svoju vlastnú pamäť t.j. neobmedzenú sadu registrov $R_{i,0}, R_{i,1}, R_{i,2}, \ldots$ a inštrukcie na priamy alebo nepriamy prístup (read/write) do spoločnej pamäte. Základná sada inštrukcii je zobrazená v tabuľke ??. Procesory sú zosynchronizované podľa globálnych hodín, teda inštrukcie*



Obrázok 7.2: *Model PRAM*

v jednotlivých procesoroch sú vykonávané v taktoch.
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
$IDENT$	do registra R_0 zapíš číslo procesora
$CONST\ c$	do registra R_0 zapíš hodnotu c
$ADD\ R_i$	$R_0 \leftarrow [R_0] + [R_i]$
$SUB\ R_i$	$R_0 \leftarrow [R_0] - [R_i]$
$MULT\ c$	$R_0 \leftarrow [R_0].c$
$DIV\ c$	$R_0 \leftarrow [R_0]/c$
$IFZERO\ i$	ak register R_0 obsahuje 0, tak pokračuj inštrukciou i
$GOTO\ i$	pokračuj inštrukciou i
$HALT$	ukonči výpočet

Tabuľka 7.2: Zoznam jednoduchých inštrukcií *PRAMu*

Pri práve zadefinovanom 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.[2] Výpočet skončí, keď skončí procesor P_0 t.j. vykoná HALT.

2. 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[3] z procesorov žiadajúcich o zápis zapíše svoju hodnotu do registra
   - $CREW - PRAM$ (Concurrent Read Exlusive Write)
     Dovolíme[4] procesorom súčasné čítanie spoločného registra, ale zapisovať do spoločného registra môže vždy len jeden procesor.
   - $EREW - PRAM$ (Exlusive Read Exlusive Write)
     Čítať a zapisovať do spoločného registra môže vždy len jeden procesor.

7.2 Miery zložitosti

Pri modeli *RAM* uvažujeme tieto jenotkové 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.

[2] inou možnosťou riešenia tohoto problému by bolo zavedenie inštrukcie FORK, ktorou ľuboľný aktívny procesor môže aktivovať ďalšie, pričom na začiatku bude aktívny len procesor P_0

[3] je to jediný nedeterministický prístup v modeli *PRAM* t.j. môže sa stať, že pri opakovanom výpočte na rovnakom vstupe dostaneme rozdielne výstupy

[4] to znamená, že program (postupnosť inštrukcií) pre jednotlivé procesory musí spĺňať túto podmienku

Pri modeli $PRAM$ uvažujeme tieto jenotkové 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 zložitosťou $T(n)$ na $P(n)$ procesoroch $PRAM$ nespotrebuje na uloženie dát viac ako $O(P(n).T^2(n))$ bitov. Máme teda adekvátnu mieru pre pamäťové nároky ($SPACE$).

7.3 Výpočtová sila modelu $PRAM$

Najskôr si ukážeme niekoľko príkladov výpočtov na modeli $PRAM$.

Príklad 7.3.1. Chceme vypočítať maximum z postupnosti n zadaných čísel $x_1, \ldots, x_n$ na modeli $COMMON\ CRCW - PRAM$.
Vstup bude zadaný nasledovne: $C_0 \leftarrow n$, $C_1 \leftarrow x_1, \ldots, C_n \leftarrow x_n$.
Chceme[5] výstup: $[C_0] = \max\{x_1, \ldots, x_n\}$.
Na výpočet použijeme n^2 procesorov ozn. $P_{i,j}$, kde $i, j \in \{1, \ldots, n\}$ plus procesor P_0:

1. Každý procesor $P_{i,1}$ vykoná: $C_{n+i} \leftarrow 0$
2. Každý procesor $P_{i,j}$ vykoná[6]: $if\ [C_i] < [C_j]\ then\ C_{n+i} \leftarrow 1$
   Uvedomme si, že tu nenastane žiadny konflikt pri súčasnom zápise viacerých procesorov do toho istého registra, pretože všetky procesory, ktoré budú chcieť zapísovať 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 $C_{n+1}, \ldots, C_{2n}$ jediný s nulovou hodnotou a platí $[C_{n+i}] = 0$ práve vtedy, keď $x_i = \max\{x_1, \ldots, x_n\}$

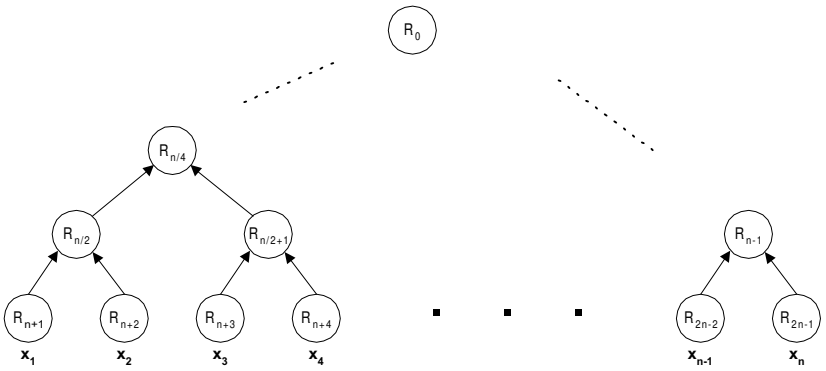
Ako vidieť, vďaka použitiu polynomiálneho počtu procesorov, sa nám podarilo nájsť maximum z číselnej postupnosti v konštantnom čase. Pri sekvenčných modeloch na to potrebujeme lineárny čas.

Príklad 7.3.2. Opäť chceme vypočítať maximum ako v predchádzajúcom príklade, ale tentoraz na modeli $EREW - PRAM$.
Vstup je zadaný tak isto ako v predchádzajúcom príklade. Budeme[7] 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 zložitosť sme zmenšili $P(n) = n$, ale časová zložitosť vzrástla $T(n) = \log n$.

Príklad 7.3.3. Ukážeme si, ako efektívne vieme triediť postupnosť čísel $x_1, \ldots, x_n$ na modeli $CREW - PRAM$.
Vo vstupných registroch $C_1, \ldots, C_n$ sú hodnoty $x_1, \ldots, x_n$. Chceme, aby po skončení výpočtu boli v registroch $C_1, \ldots, C_n$ čísla zo vstupu v utriedenom poradí.
Na výpočet použijeme n^2 procesorov ozn. $P_{i,j}$.

[5] $[C]$ označuje obsah registra C, $C \leftarrow a$ označuje zápis čísla a do regitra C
[6] procesor nemá k dispozícii takú silnú inštrukciu, ale iste si vieme predstaviť jej simuláciu pomocou konečného počtu $PRAM$-inštrukcií
[7] rovnakú myšlienku možno použiť pri úlohe sčítať n čísel



Obrázok 7.3: *Výpočet maxima na EREW − PRAM*

1. Každý procesor $P_{i,j}$ vykoná: $if\ [C_i] < [C_j]\ then\ C_{i,j} \leftarrow 1\ else\ C_{i,j} \leftarrow 0$
2. Každých n procesorov $P_{i,1}, \ldots, P_{i,n}$ sa bude podieľať na výpočte[8]: $C_{n+i} \leftarrow 1 + \sum_{j=1}^{n} [C_{i,j}]$
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. $\mathcal{L}(EREW - PRAM) = \mathcal{L}(CRCW - PRAM)$

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 (t.j. zapíše do príslušného registra svoje číslo).
Ak je procesor pravý, tak "skúsi", či jeho ľavý sused postúpil.[9] 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.

[8] sčítať n čísel je podobný problém ako nájsť maximum z n čísel (príklad 7.3.2)

[9] 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 zodpovedajúcemu vyššej úrovni. V treťom kroku praví, ak zistili, že ľaví nezapísali svoje číslo do registra vyššej úrovne, tak tam zapíšu svoje číslo.

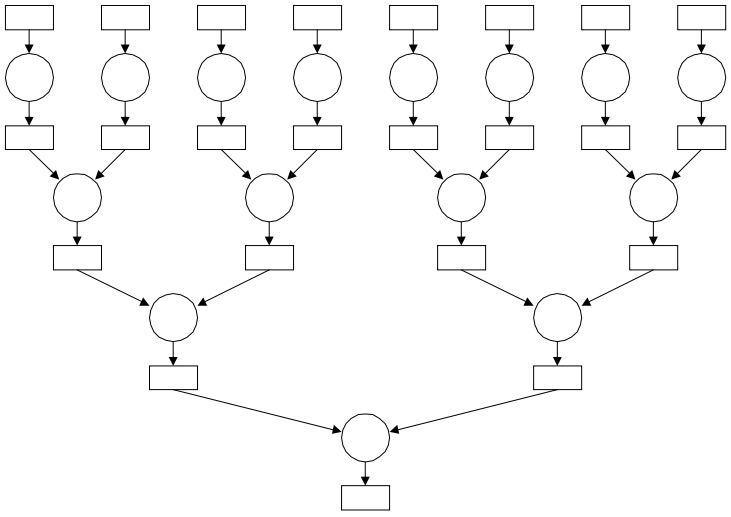
Čítanie bude fungovať podobne, iba s tým rozdielom, že výpočet na strome pod každým registrom sa bude vykonávať opačným smerom, aby každý z procesorov, ktorý mal záujem čítať, sa dostal k C_i.

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 $f_n : \{0,1\}^n \to \{0,1\}^m$ 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.



Obrázok 7.4: *Simulácia BO na modeli PRAM*

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.

- Simulácia výpočtu $DTS\ A$:

  Čo vieme o $DTS\ \ A$? Ak uvažujeme, že A pracuje na vstupe dĺžky n v priestore $S(n) = \log SIZE(C_n)$, tak počet všetkých možných konfigurácií A je $k^{S(n)}.n.S(n)$ pre nejakú konštantu k, čo je rádovo $(SIZE(C_n))^{O(1)}$ čiže podľa predpokladu $n^{O(1)}$. Očíslujme si jednotlivé konfigurácie $1, 2, \ldots r$. Nech[10] 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 pridelíme dva registre zo spoločnej pamäte, jeden[11] 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[12]. Keďže sme odsimulovali jeden krok A z každej konfigurácie, vyčerpali sme tým všetky možnosti δ-funkcie, takže δ-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. $C_{j,OUT} \leftarrow [C_{j,OUT}].[C_{C_{j,REG},OUT}]$, kde $.$ je zreťazenie[13] 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 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[14] 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. Hrubo 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 podielať na výpočte, a preto aj procesory, ktoré sa v konečnom dôsledku na výpočte podielať nebudú (na obrázku znázornené malými kružnicami v prvom riadku), vykonávajú svoj program.

  Keďže po každom kroku $PRAMu$ sa dĺžka odsimulovaného výpočtu A zdvojnásobí, tak na odsimulovanie celého výpočtu vystačíme s časom logaritmus z počtu konfigurácií, čo je $O(\log SIZE(C_n))$.

- Dekódovanie:

  Predpokladajme, že už máme v nejakom registri kód $\langle C_n \rangle$. Ten je v tvare:

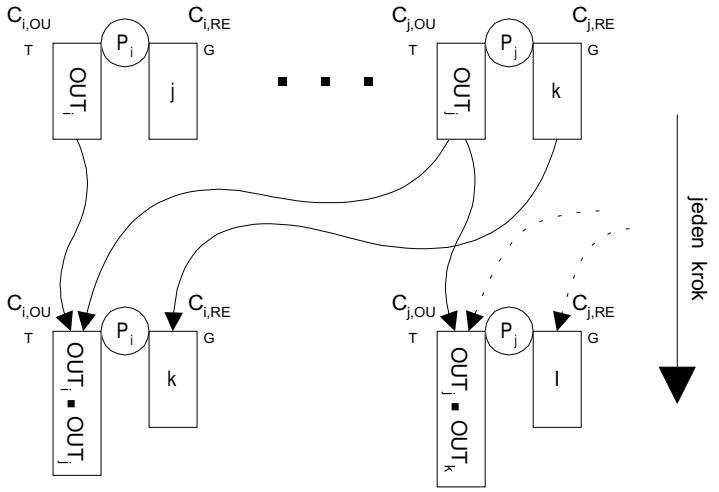
[10] takýto predpoklad môžeme urobiť, lebo pre daný vstup poznáme počiatočnú konfiguráciu a môžeme sa dohodnúť na jednej koncovej konfigurácií pre všetky výpočty, z ktorej sa už nedá dostať do inej konfigurácie

[11] POZOR: Nemýliť si s vyššie uvedeným rovnako nazvaným registrom, použitým pri samotnej simulácii BO

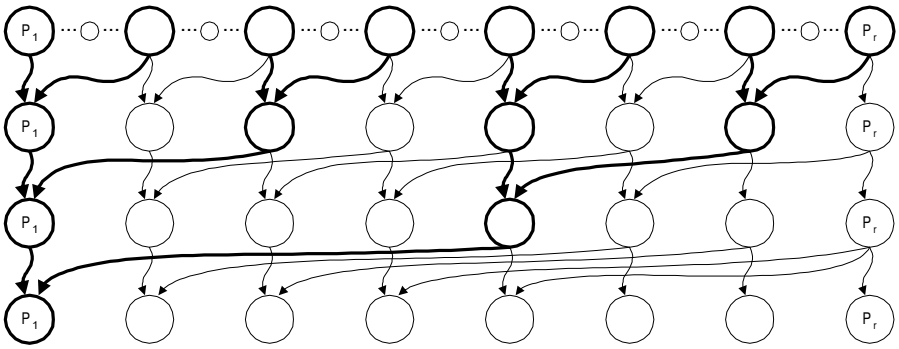
[12] tieto zápisy sú jednoznačné, keďže simulujeme deterministický stroj

[13] obsahmi registrov sú čísla v binárnom zápise, takže zreťazenie znamená nasledujúcu aritmetickú operáciu nad registrami: $C_{j,OUT} \leftarrow C_{j,OUT} + 2^{|C_{C_{j,REG},OUT}|}.C_{C_{j,REG},OUT}$

[14] realizáciu si môžeme predstaviť nasledovne: P_1 zapíše do nejakého špeciálneho booleovského registra, ktorý bol inicializovaný na TRUE, hodnnotu FALSE a program pre procesory upravíme tak, že dané inštrukcie budú vykonávať len vtedy, keď v tomto registri je hodnota TRUE



Obrázok 7.5: *Jeden krok simulácie DTS*



Obrázok 7.6: *Generovanie kódu BO na modeli PRAM*

(⟨ číslo hradla,typ hradla,číslo ľavého vstupu, číslo pravého vstupu ⟩)*

Zrejme dĺžka kódu je $|\langle C_n \rangle| = O(SIZE(C_n).\log SIZE(C_n))$. 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[15] "(". 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 $O(\log SIZE(C_n))$. Každý procesor potrebuje prečítať kód jedného hradla a zapísať získané informácie do registrov. Preto čas, ktorý na dekódovanie potrebujeme, je $O(\log SIZE(C_n))$.

[15] všetky symboly sú zakódované binárne, takže procesory prestanú pracovať po tom ako rozpoznajú symbol "("

- 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 NILovú hodnotu[16]. Procesor P_i bude vykonávať program:

  1. načítaj obsahy registrov $C_{i,IN1}, C_{i,IN2}, C_{i,TYP}$
  2. čakaj kým platí: $[C_{C_{i,IN1}}] = NIL$ a $[C_{C_{i,IN2}}] = NIL$
  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 $CREW - PRAM$. □

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 $1 \le i \le P(n).T(n)$ a $1 \le j \le k.T(n)$, 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 jenotlivých 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 $1 \le j \le k.T(n)$ (t.j. vieme ohraničiť maximálnu veľkosť registra počas celého výpočtu), každú inštrukciu vieme simulovať elementárnym obvodom obsahujúcim konečný počet hradiel.

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 prislúchajúce k i-tym 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 $(1 \le i \le P(n).T(n))$, veľkosti $(1 \le j \le k.T(n))$ registrov a z toho vyplývajúcej konečnej veľkosti elementárnych obvodov dostávame zložitosti pre C_n: $DEPTH(C_n) = (\log n)^{O(1)}$ a $SIZE(C_n) = n^{O(1)}$.

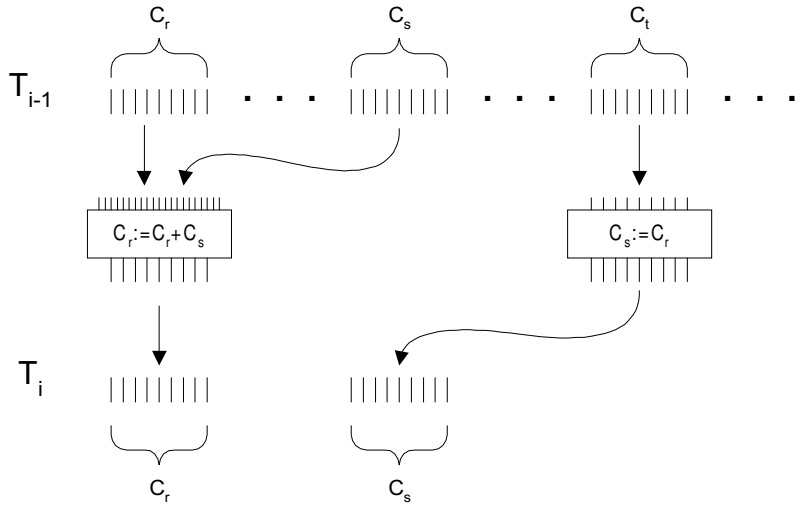
Teda dokázali sme ekvivalentnosť mier zložitosti medzi týmito dvoma modelmi t.j. čas na modeli $PRAM$ zodpovedá hĺbke na BO a počet procesorov na $PRAM$ zodpovedá veľkosti BO. □

Dôsledok 7.4.1. Model $PRAM$ je v druhej počítačovej triede.

Dôsledok 7.4.2. Triedu $\mathcal{NC}$ môžeme pomocou modelu $PRAM$ zadefinovať nasledovne:

$$\mathcal{NC} = TIMEPROCESSORS((\log n)^{O(1)}, n^{O(1)})$$

[16] vzhľadom na to, že vo výstupných registroch budú len hodnoty 0 alebo 1, hodnotu NIL môže reprezentovať akákoľvek iná hodnota



Obrázok 7.7: *Simulácia výpočtu PRAM na BO*

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 Rovan: 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)