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

Enumerácia jednoducho popísateľných regulárnych jazykov

(Bakalárska práca)

Zuzana Petruchová

Odbor: Informatika
Vedúci: RNDr. Michal Forišek

Bratislava, 2009

Čestne prehlasujem, že som túto bakalársku prácu vypracovala samostatne s použitím citovaných zdrojov.

.....

Abstrakt

V práci sa zaoberáme zisťovaním počtu rôznych jazykov akceptovaných deterministickými konečnými automatmi s najviac n stavmi. Implementujeme rôzne algoritmy na zistenie tohoto počtu. Výpočet pozostáva z dvoch fáz, pričom každej z nich sa venujeme osobitne. Nakoniec robíme porovnanie algoritmov vzhľadom na čas potrebný k zisteniu výsledku a na efektivitu generovania automatov.

Kľúčové slová: konečný automat, hashovanie, minimálny automat, Hopcroftov algoritmus

Obsah

Kapitola 1

Úvod

Skúmanie regulárnych jazykov a konečných automatov patrí medzi základy teórie formálnych jazykov. Napriek tomu sú v tejto oblasti ešte stále otvorené problémy.

Jedným z nich je počet rôznych jazykov, akceptovaných deterministickým konečným automatom s najviac n stavmi. Presné riešenie tohoto problému zatiaľ nie je známe, známe sú iba horné a dolné odhady, resp. hodnoty pre dostatočne malé n.

Cieľom tejto práce je implementovať a porovnať rôzne metódy na zisťovanie počtu takýchto jazykov. V práci budeme skúmať efektivitu jednotlivých metód najmä z hľadiska času potrebného na výpočet. Všetky skúmané metódy generujú automaty, a každý vygenerovaný automat kontrolujú na unikátnosť jazyka, ktorý akceptuje. Venujeme sa teda osobitne spôsobom na generovanie automatov, a spôsobom na spracovanie vygenerovaného automatu.

Kapitola 2

Prehľad problematiky

V tejto kapitole na úvod uvedieme základné definície a označenia, s ktorými budeme ďalej pracovať – pojem konečného automatu, jazyka akceptovaného konečným automatom, minimálneho automatu.

Ďalej uvedieme niektoré dôležité vety a z nich vychádajúce algoritmy.

V závere kapitoly sa budeme venovať už dosiahnutým výsledkom v tejto oblasti.

2.1 Definície

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

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

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

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

Definícia 2.1.5 Symetrický rozdiel *dvoch jazykov L_1, L_2 je jazyk $L_1 \oplus L_2$ definovaný nasledovne: $L_1 \oplus L_2 = \{w \mid (w \in L_1 \wedge w \notin L_2) \vee (w \notin L_1 \wedge w \in L_2)\}$*

Majme dané automaty $A = (K_A, \{a, b\}, \delta_A, 0, F_A), B = (K_B, \{a, b\}, \delta_B, 0, F_B)$. Potom automat $C = (K_C, \{a, b\}, \delta_C, q_0, F_C)$ akceptujúci jazyk $L(A) \oplus L(B)$ zostrojíme nasledovne:

$$K_C = K_A \times K_B$$

Stavy K_C budú $[q_A, q_B] \in K_C \iff q_A \in A \wedge q_B \in B$.

$$[q_A, q_B] \in F_C \iff (q_A \in F_A \wedge q_B \notin F_B) \vee (q_A \notin F_A \wedge q_B \in F_B)$$

$$(\forall a \in \Sigma^*)([q_A, q_B] = \delta_C([q_1, q_2], a) \iff (q_A = \delta_A(q_1, a) \wedge q_B = \delta_B(q_2, a)))$$

Definícia 2.1.6 Minimálny automat. *Deterministický konečný automat $A = (K_A, \{a, b\}, \delta_A, 0, F_A)$ je minimálny, pokiaľ*

$$\forall B = (K_B, \{a, b\}, \delta_B, 0, F_B) : \ |K_A| \leq |K_B|$$

Veta 2.1.1 *(Myhill-Nerode) Majme jazyk $L \subset \Sigma^*$. Nasledujúce tvrdenia sú ekvivalentné:*

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

Dôkaz tejto vety je možné nájsť napríklad v [FR09]

Dôsledok 2.1.2 *Ku každému regulárnemu jazyku L existuje práve jeden minimálny deterministický konečný automat.*

Majme regulárny jazyk L a reláciu R_L definovanú v predchádzajúcej vete. Potom minimálny automat A akceptujúci jazyk L je jednoznačne určený reláciou R_L nasledovne:

Nech $[v] = \{v \mid uRv\}$. $A = (K, \Sigma, \delta, [\epsilon], F)$, pričom

$$K = \{[v] \mid v \in \Sigma^*\}$$

$$\delta([v], a) = [va]$$

$$F = \{[v] \mid v \in L\}$$

2.1.1 Hopcroftov algoritmus

Tento algoritmus rieši problém minimalizácie automatu - pre daný DKA $A = (K, \Sigma, \delta, q_0, F)$ nájde minimálny DKA akceptujúci ten istý jazyk.

Definícia 2.1.7 *Množiny $K_1, K_2, \ldots, K_n$ tvoria* **rozklad** *K, ak sú po dvojiciach disjunktné, a pokiaľ je K ich zjednotením. K_i je potom triedou rozkladu.*

Definícia 2.1.8 Kongruencia automatu *A je rozklad, pre ktorý platí, že ak $p, q \in K$ patria do tej istej triedy rozkladu, potom $\delta(p, a)$ a $\delta(q, a)$ ležia tiež v tej istej triede rozkladu pre všetky $a \in \Sigma$*

Definícia 2.1.9 *Rozklad $\{K_1, K_2, \ldots, K_n\}$ je hrubší ako rozklad $\{K'_1, K'_2, \ldots, K'_m\}$, ak pre každé K_i platí, že je zjednotením niekoľkých tried K'_j*

Algoritmus začína rozkladom K na triedy F, F^C a postupne zjemňuje rozklad, kým nedostane najhrubšiu kongruenciu automatu A.

Pracujeme s rozkladom $\mathcal{P}$, a množinou dvojíc $\mathcal{S} = (C, a)$, kde $a \in \Sigma$ a C je trieda rozkladu $\mathcal{P}$. Množina $\mathcal{S}$ obsahuje dvojice (C, a) ktoré budeme spracovávať. Na začiatku $\mathcal{P} = \{F, F^C\}$ a $\mathcal{S}$ obsahuje (C, a) $\forall(a \in \Sigma)$ a C je menšia z tried F, F^C. Algoritmus pracuje nasledovne:

Vyberie jednu dvojicu (C, a) z $\mathcal{S}$, a pre každú triedu B rozkladu $\mathcal{P}$ vykoná nasledovné:

- Zistí, či (C, a) rozdeľuje B na dve nové triedy rozkladu. To nastáva vtedy, keď existujú stavy $p, q \in B$ také, že $\delta(p, a) \in C$ a $\delta(q, a) \notin C$. Ak (C, a) nerozdeľuje B, prejdeme na spracovanie dalšej triedy B. Ak (C, a) rozdeľuje B, potom vytvoríme dve nové triedy $B_1 = \{q|\delta(q, a) \in C\}$ a $B_2 = \{q|\delta(q, a) \notin C\}$.

- Triedu B nahradíme v $\mathcal{P}$ triedami B_1, B_2. Pre každé $b \in \Sigma$, ak $(B, b) \in \mathcal{S}$ nahradíme ho pármi $(B_1, b), (B_2, b)$. Ak sa v $\mathcal{S}$ nenachádza, vyberieme menšiu z tried B_1, B_2 (B') a do $\mathcal{S}$ pridáme dvojicu (B', b).

Algoritmus skončí, keď je množina $\mathcal{S}$ prázdna.

Tento algoritmus sa dá implementovať s časovou zložitosťou $O(n \log n)$. Dôkaz správnosti algoritmu a časovej zložitosti je uvedený v [Hud07]

2.2 Podobný výskum

Domaratzki, Kisman a Shallit [DKS02] sa zaoberali odhadmi počtu rôznych jazykov akceptovaných n-stavovými automatmi. V tejto časti uvádzame výsledky týkajúce sa deterministických konečných automatov.

Na úvod zadefinujeme funkcie:

$f_k(n)$ = počet navzájom neizomorfných minimálnych deterministických konečných automatov s n stavmi nad k-písmenovou abecedou

$g_k(n)$ = počet jazykov akceptovaných n-stavovými automatmi nad k-písmenkovou abecedou

Veta 2.2.1 $g_k(n) = f_k(1) + f_k(2) + \ldots + f_k(n)$

Nech jazyk L je akceptovaný nejakým n-stavovým automatom. Potom existuje minimálny automat A s $k \leq n$ stavmi, akceptujúci L, a teda $g_k(n) \leq f_k(1)+f_k(2)+\ldots+f_k(n)$. Pokiaľ je jazyk L akceptovaný k-stavovým minimálnym automatom ($k \leq n$), potom vieme doplniť automat nedosiahnuteľnými stavmi na n-stavový, a teda je L akceptovaný n-stavovým automatom, platí teda $f_k(1) + f_k(2) + \ldots + f_k(n) \leq g_k(n)$

2.2.1 Jednoznaková abeceda

Ako prvé sa venovali prípadu, kedy $k = 1$ (jednoznaková abeceda)

V [Nic99] nájdeme nasledovnú vetu:

Veta 2.2.2 *N-stavový DKA $M = (K, \{a\}, \delta, q_0, F)$, kde $K = \{q_0, q_1, \ldots, q_{n-1}\}$ je minimálny práve vtedy, keď sú splnené nasledujúce tri podmienky:*

- *M je súvislý, čiže neobsahuje žiadne nedosiahnuteľné stavy. Potom jeho prechodový graf pozostáva z cyklu a "chvosta", tzn. $\delta(q_i, a) = q_{i+1}$ pre $0 \leq i \leq n-2$ a $\delta(q_{n-1}, a) = q_j$ pre nejaké $j, 0 \leq j \leq n-1$.*
- *Cyklus je minimálny. To znamená, že nemôže byť nahradený ekvivalentným kratším cyklom.*
- *Ak $j \neq 0$, potom pre q_{j-1} a q_{n-1} platí, že jeden z nich je akceptačný a druhý nie.*

Definícia 2.2.1 *Neprázdne slovo je primitívne vtedy, keď sa nedá napísať v tvare $w = x^k$ pre nejaké slovo x a celé číslo $k \geq 2$.*

Cyklus $q_j, q_{j+1}, \ldots, q_{n-1}$ je minimálny vtedy a len vtedy, keď je slovo $a_j a_{j+1} \ldots a_{n-1}$, definované

$$a_i = \begin{cases} 1 & \text{ak } q_i \in F \\ 0 & \text{ak } q_i \notin F \end{cases} \tag{2.1}$$

je primitívne.

Označme $\psi_k(n)$ počet primitívnych slov dĺžky n nad k-prvkovou abecedou. Platí

$$\psi_k(n) = \sum_{d|n} \mu(d) k^{n/d}$$

kde μ je Moebiova funkcia, definovaná:

$$\mu(n) = \begin{cases} 0 & \text{ak } n \text{ je deliteľné nejakým štvorcom väčším ako 1} \\ (-1)^s & \text{ak } n = p_1 p_2 \ldots p_s, \text{ kde } p_i \text{ sú navzájom rôzne prvočísla} \end{cases} \tag{2.2}$$

Z horeuvedených viet vyplýva nasledujúca:

Veta 2.2.3

$$f_1(n) = \psi_2(n) + \sum_{1 \le j \le n-1} \psi_2(n-j) 2^{j-1}$$

A z vety 2.2.1 vyplýva vzorec pre výpočet $g_1(n)$:

Veta 2.2.4 $g_1(n) = \sum_{1 \le t \le n} \psi_2(t) 2^{n-t}$

Ďalej sa v článku uvádza aj asymptotický odhad $g_1(n)$:

$$g_1(n) = 2^n (n - \alpha + O(n 2^{-n/2}))$$

kde $\alpha = \sum_{d \ge 2} \frac{\mu(d)}{1 - 2^{d-1}}$

2.2.2 Viacznaková abeceda

Ako prvý je uvedený konštrukčný dolný odhad $f_k(n)$.

Majme automat $M = (K, \Sigma, \delta, q_0, F)$ a podmnožinu vstupnej abecedy $\Delta \subset \Sigma$. Uvažujeme obmedzenie M na Δ, označujeme ho M_Δ, čo je automat $(K, \Delta, \delta_\Delta, q_0, F)$, kde δ_Δ je obmedzenie prechodovej funkcie na množinu $K \times \Delta$.

Nech $k \geq 2$ a $n \geq 1$. Nech $S_{k,n}$ je množina det. konečných automatov M nad k-prvkovou abecedou $\{0, 1, \ldots, k-1\}$, definovaná:

- $M_{\{0\}}$ je jeden z $f_1(n)$ rôznych minimálnych DKA nad jednoznakovou abecedou
- zvolíme $k-1$ funkcií $h_i : K \to K$ pre $1 \leq i < k$ a definujeme $\delta(q, i) = h_i(q)$ pre $1 \leq i < k$ a $q \in K$

Potom $S_{k,n}$ obsahuje $f_1(n)n^{(k-1)n}$ rôznych DKA, každý z nich minimálny a teda každý akceptujúci iný jazyk.

Z tohto tvrdenia vyplýva nasledovná veta:

Veta 2.2.5

$$f_k(n) \geq f_1(n)n^{(k-1)n} \sim n2^{n-1}n^{(k-1)n}$$

Jednoduchý horný odhad $g_k(n) \leq 2^n n^{kn}/(n-1)!$ dostaneme nasledovne:

Akceptačné stavy vieme vybrať 2^n spôsobmi. Prechodová funkcia zobrazuje $K \times \Sigma \to K$, a preto je n^{kn} takýchto funkcií. Toto číslo ešte vieme vydeliť $(n-1)!$, pretože po odstránení n-stavových automatov s nedosiahnuteľnými stavmi nezáleží na pomenovaní žiadneho stavu okrem počiatočného.

Kapitola 3

Prehľad použitých algoritmov

V tejto kapitole popíšeme algoritmy použité pri rátaní počtu rôznych jayzkov. Program ráta počet jazykov, akceptovaných DKA s najviac n stavmi. Každý testovaný spôsob generuje n a menej stavové automaty, ktoré následne spracuje. Samotným porovnaním efektivity jednotlivých riešení sa budeme zaoberať v kapitole 4

V prvej časti kapitoly popíšeme použité algoritmy na vygenerovanie potrebných automatov.

V druhej časti sú popísané algoritmy ktoré zisťujú či spracúvaný automat akceptuje nový jazyk. Prvé 2 spôsoby automaty porovnávajú, a posledný na základe Myhill-Nerodovej vety zisťuje, či je spracúvaný automat minimálny.

3.1 Popis problému

V práci sa zaoberáme jazykmi nad 2-písmenkovou abecedou $\{0, 1\}$. Množina stavov $K = \{0, 1, \ldots, n-1\}$. Automat reprezentujem v poli – $A[a][b]$ pre $a \in \{0, 1, \ldots, n-1\},\ b \in \{0, 1\}$ je stav, do ktorého sa automat dostane z a prechodom na znak b - $A[a][b] = \delta(a, b)$, a $A[a][2] = 1$ ak stav a je akceptačný, inak $A[a][2] = 0$.

Vo všetkých prípadoch je počiatočným stavom stav 0.

3.2 Počet skúmaných automatov

V tejto časti popíšeme použité spôsoby generovania automatov.

3.2.1 Triválne riešenie

Prvý spôsob spočíva vo vygenerovaní všetkých orientovaných grafov na n vrcholoch, pre ktoré platí, že z vrcholu vychádzajú práve dve hrany, označené 0 a 1. Takéto grafy predstavujú možné prechodové funkcie n-stavového automatu.

Pre každý takýto graf vygenerujeme všetky kombinácie akceptačných stavov (všetky podmnožiny F množiny K)

Počet všetkých takýchto grafov je n^{2n} – v každom stave na každý znak je možný prechod do n ďalších stavov. Počet kombinácií akceptačných stavov je 2^n. Celkovo teda vygenerujeme a potrebujeme overiť $n^{2n}2^n$ automatov.

3.2.2 Obmedzenie akceptačných stavov

Pre každý deterministický konečný automat A existuje ekvivalentný (akceptujúci ten istý jazyk) automat B, pre ktorý platí: ak je stav akceptačný, potom neexistuje stav s nižším číslom, ktorý nie je akceptačný (s výnimkou počiatočného stavu 0). Takýto automat vieme dostať z pôvodného prečíslovaním stavov tak, že akceptačné budú označené menšími číslami ako neakceptačné.

Pri tomto spôsobe teda vygenerujeme všetky grafy predstavujúce δ-funkciu automatu. Keďže akceptačné stavy sú vždy menšie ako neakceptačné, ak počet akceptačných stavov je k, potom množina týchto stavov vyzerá nasledovne: $F = \{0, 1, \ldots, k-1\}$ pokiaľ $0 \in F$, alebo $F = \{1, 2, \ldots, k\}$ pokiaľ

$0 \notin F$.

Ďalej vieme, že všetky automaty s 0 akceptačnými stavmi akceptujú ten istý – prázdny jazyk, a automaty s n akceptačnými stavmi akceptujú jazyk Σ^*. Preto miesto overovania každého automatu s 0 alebo n akceptačnými stavmi na začiatku vygenerujeme dva automaty: prechodová funkcia v oboch v každom stave na každý znak ide do počiatočného stavu, pričom v prvom je počiatočný stav neakceptačný a v druhom akceptačný. Vďaka tomuto stačí generovať automaty s $1 \leq k \leq n-1$ akceptačnými stavmi.

Vygenerovaných grafov je rovnako ako v predchádzajúcom prípade n^{2n}. Stav 0 môže byť buď akceptačný, alebo neakceptačný. Pre oba prípady môže byť zvyšných akceptačných stavov od 1 po $n-2$, a ešte máme 2 prípady, jeden kedy je stav 0 akceptačný a ostatné neakceptačné, a druhý keď je stav 0 neakceptačný a ostatné stavy sú akceptačné. Celkový počet kombinácií akceptačných stavov je teda $2(n-1)$. Celkový počet vygenerovaných automatov bude $n^{2n}2(n-1)$

3.2.3 Generovať iba súvislé automaty

Spracovávať budeme iba také automaty, pre ktoré platí nasledovné: neexistuje dosiahnuteľný stav s väčším číslom, ako je číslo nejakého nedosiahnuteľného stavu. Zároveň pre nedosiahnuteľné stavy už negenerujeme δ-funkciu.

Ku každému automatu existuje ekvivalentný v takomto tvare. Dostaneme ho prečíslovaním stavov.

Grafy budeme generovať nasledovne: δ-funkciu generujeme postupne po stavoch, počnúc stavom 0. Počas celého generovania si pamätáme najvyššie číslo stavu doteraz použité v prechodovej funkcii a počet doteraz použitých stavov. Stav je použitý vtedy, keď v δ-funkcii existuje prechod do tohto stavu z nejakého iného. S generovaním končíme, keď by sme mali v najbližšom kroku generovať prechody pre stav s vyšším číslom ako najvyšší doteraz po-

užitý – do takéhoto stavu už neexistuje prechod, a teda by nebol dosiahnuteľný. Potom overíme, či počet použitých stavov je rovnaký ako najvyššie číslo použitého stavu – ak nie, znamená to, že automat obsahuje nedosiahnuteľný stav, a teda ho nemusíme spracovať.

Pre stavy aj v tomto prípade platí, že akceptačné stavy majú menšie čísla ako neakceptačné (s výnimkou stavu 0). A teda počet testovaných kombinácií pre vygenerovanú prechodovú funkciu s najväčším použitým stavom k bude $2(k-1)$.

3.2.4 Pridávanie nových stavov

Pri tomto spôsobe najprv zvolíme počet akceptačných a počet neakceptačných stavov, ktoré bude najbližší spracovaný automat obsahovať, a až potom sa vygeneruje δ-funkcia.

Algoritmus funguje nasledovne:

Na začiatku sa určí f – počet akceptačných stavov, a g – počet neakceptačných stavov. Následne sa vygenerujú všetky súvislé (neobsahujúce nedosiahnuteľné stavy) automaty s f akceptačnými a g neakceptačnými stavmi – medzi tieto dve čísla nezahŕňame stav 0, to, či bude akceptačný alebo nie sa zvolí až po vygenerovaní zvyšku automatu.

Generovať začíname v stave 0 pre znak 0. V množine S budú stavy, ktoré už boli použité v prechodovej funkcii, ale ešte neboli spracované. p_f a p_g je počet doteraz použitých akceptačných resp. neakceptačných stavov. Na začiatku $p_f = p_g = 0$.

Rekurzívne spracúvame jednotlivé prechody δ-funkcie (q je aktuálne spracovávaný stav a a aktuálne spracovávaný znak) nasledovne:

- Postupne vyskúšame každú z nasledujúcich možností:
  - Do prechodovej funkcie v stave q na znak a pridáme už použitý stav, alebo

  - do prechodovej funkcie pridáme nový akceptačný stav (ak $p_f < f$), tento stav pridáme do množiny S a zvýšime p_f, alebo
  - do prechodovej funkcie pridáme nový neakceptačný stav (ak $p_g < g$), tento stav pridáme do S a zvýšime p_g.

- Ak $a = 0$ pokračujeme v generovaní pre stav q a znak $a = 1$, inak pokračujeme novým stavom z S a znakom $a = 0$.

Koniec nastáva, keď $|S| = 0$. Vtedy skontrolujeme, či $p_f = f$ a $p_g = g$. Ak nie, automat nespracúvame.

V poslednom kroku zvolíme to, či bude stav 0 akceptačný, alebo nie. Ak $f = 0$, potom má zmysel spracovať automat len ak 0 je akceptačný (inak by automat bez ohľadu na prechodovú funkciu akceptoval prázdny jazyk). Rovnako ak $g = 0$, spracujeme len automat s neakceptačným stavom 0. V ostatných prípadoch spracujeme automat pre stav 0 akceptačný aj neakceptačný.

V porovnaní s predchádzajúcim spôsobom, pri tomto sme zabránili vygenerovaniu dvoch automatov, ktoré sa dajú jeden z druhého dostať prečíslovaním stavov.

3.3 Spracovanie automatu

V tejto časti sa budeme venovať spracovaniu vygenerovaných automatov a samotnému určeniu hľadaného výsledku.

Prvé dva spôsoby využívaju priame porovnávanie automatov. Potrebujeme zistiť, či dva automaty A, B akceptujú rovnaký jazyk. Na to využijeme symetrický rozdiel jazykov $L(A), L(B)$. Vieme, že $L(A) = L(B) \iff L(A) \oplus L(B) = \emptyset$. Zostrojíme teda automat pre jazyk $L(A) \oplus L(B)$. Prehľadaním automatu z počiatočného stavu zistíme, či existuje cesta do akceptačného stavu. $\iff L(A) \oplus L(B) \neq \emptyset$ a teda $L(A) \neq L(B)$.

Posledný spôsob testuje automat na minimalitu, na čo používa Hopcroftov algoritmus.

3.3.1 Triviálne riešenie

Budeme si v poli pamätať všetky doteraz unikátne automaty (akceptujúce unikátne jazyky). Po vygenerovaní nového tento automat porovnáme so všetkými automatmi, ktoré sa už schválili, a ak akceptuje iný jazyk, než ktorýkoľvek z nich, zapamätáme si ho a začneme spracúvať ďalší.

Takéto riešenie potrebuje pamäť na zapamätanie každého automatu akceptujúceho nový jazyk.

Čas potrebný na porovnanie dvoch n-stavových automatov je $O(N^2)$ – čas prehľadania automatu pre symetrický rozdiel porovnávaných automatov. Nech f je hľadaný počet jazykov, potom celková zložitosť takéhoto riešenia bude $O(f^2n^2)$ – porovnaní dvojíc automatov bude f^2, keďže každý automat porovnávame so všetkými už akceptovanými.

3.3.2 Hashovanie

Nebudeme porovnávať nový automat s každým už uznaným, ale automat najprv zahashujeme a porovnávať budeme iba automaty s rovnakým hashom.

Hashovanie v tomto prípade vyzerá nasledovne:

Nech $s = a_1a_2 \dots a_k$ je nejaké slovo, $s \in \Sigma^*$, a A je automat. $b_0, b_1, b_2, \dots, b_k$ je postupnosť stavov, v ktorých sa automat nachádzal pri spracovávaní slova s, pričom $b_0 = 0$. Majme teraz binárny zápis čísla $C = c_1c_2 \dots c_k$, pričom $c_i = 1$ ak je stav b_i akceptačný, inak $c_i = 0$. Potom C je hash automatu A reťazcom s.

Miesto hashovania jedným reťazcom môžeme použiť niekoľko kratších – v tom prípade $C = C_1C_2 \dots C_m$. Hash C je zreťazením hashov $C_1, C_2, \dots, C_m$ automatu A reťazcami $s_1, s_2, \dots, s_m$.

Pamäťové nároky tohoto riešenia ostávajú rovnaké – pamätáme si každý uznaný automat.

Čas závisí od distribúcie hashov; tou sa budeme pre rôzne reťazce podrobnejšie zaoberať v 4. kapitole.

3.3.3 Testovanie minimálnosti

Vieme, že pre každý jazyk existuje práve jeden minimálny deterministický konečný automat (až na izomorfizmus automatov). Keďže generujeme všetky automaty, pokiaľ práve spracúvaný automat nie je minimálny, potom existuje ešte jeden vygenerovaný automat (minimálny), ktorý akceptuje ten istý jazyk. Automaty teda budeme testovať na minimalitu.

Použijeme v druhej kapitole popísaný Hopcroftov algoritmus. Ten pre daný automat rozloží stavy na triedy podľa relácie R_L. Pokiaľ je každý stav v novej triede, je automat minimálny a teda akceptuje nový jazyk.

Tento spôsob vieme použiť len v prípade, kedy negenerujeme automaty, ktoré sú izomorfné. Inak by sme započítali ten istý minimálny automat viackrát. Preto pri tomto spôsobe používame iba posledný spôsob generovania automatov.

Kapitola 4

Výsledky testov

V tejto kapitole sa budeme zaoberať samotnými výsledkami výpočtov, efektivitou jednotlivých riešení a ich porovnávaním.

Na úvod uvedieme vypočítané výsledky pre hľadaný počet automatov. V ďalšej sekcii sa budeme zaoberať efektivitou jednotlivých spôsobov generovania automatov. Ďalej sa uvedieme porovnanie jednotlivých spôsobov výpočtu.

Jednu sekciu budeme venovat hashovaniu a výberu najlepšej funkcie.

Na záver uvedieme použité optimalizácie na urýchlenie behu programu.

4.1 Výsledky výpočtov

Domaratzki, Kisman a Shallit [DKS02] uvádzajú výsledky hľadaného počtu jazykov pre $n \leq 6$. Tieto hodnoty sme počas práce overili a našli výsledok aj pre $n = 7$. Tieto hodnoty uvádzame v tabuľke 4.1

Vo vyššie spomínanom článku **??** je uvedený nasledovný odhad pre $f_k(n)$: $\ln f_k(n) \sim (k-1)n \ln n + \beta_k n$, kde $\beta_2 \doteq 1.5$.

Z vety 2.2.1 dostávame, že $f_k(n) = g_k(n) - g_{k-1}(n)$, a teda $f_2(7) = 25\ 493\ 886\ 752$. Potom $\ln f_k = 23.96$ a $7 \ln 7 + \beta_2 7 = 24.12$, a teda tento

Tabuľka 4.1: Počty rôznych jazykov

Počet stavov	1	2	3	4	5	6	7
Počet jazykov	2	26	1 054	57 068	3 762 374	290 480 270	25 784 367 022

odhad je pomerne presný.

4.2 Počty generovaných automatov

V tejto sekcii sa budeme zaoberať efektivitou jednotlivých spôsobov generovania automatov.

V tabuľke 4.2 uvádzame počty vygenerovaných automatov pre jednotlivé spôsoby, a pre porovnanie aj hľadaný počet jazykov. Prvý riadok predstavuje triviálny spôsob generovania automatov, a nasledovné sú ďalšie, v poradí, v akom boli uvedené v kapitole 3. V poslednom riadku sú pre porovnanie uvedené hodnoty hľadaného počtu jazykov.

Tabuľka 4.2: Počty vygenerovaných automatov

sp. generovania	2	3	4	5	6	7
1.	64	5 832	1 048 576	312 500 000	139 314 069 504	86 812 553 342 672
2.	32	2 916	393 216	78 125 000	21 767 823 360	8 138 676 874 188
3.	24	1 452	146 646	23 245 566	7 799 455 992	2 455 466 908 728
4.	24	1 320	74 792	4 895 042	372 650 522	32 651 619 062
výsledky	26	1 054	57 068	3 762 374	290 480 270	25 784 367 022

V prvých troch spôsoboch sa generovali aj automaty, ktoré boli navzájom izomorfné. Takýchto automatov sa pri n-stavovom automate vygeneruje v prvom prípade až $n!$, v druhom a treťom $f!g!$, kde f je počet akceptačných a g počet neakceptačných stavov. V poslednom spôsobe už nevygenerujeme žiadnu dvojicu izomorfných automatov.

Rozdiel v počte automatov medzi druhou a treťou metódou vznikol tým, že pre automaty s menej ako n stavmi negenerujeme všetky kombinácie δ-

funkcie pre nepoužité stavy, a negenerujeme všetky rozmiestnenia nedosiahnuteľných stavov medzi dosiahnuteľnými.

S výnimkou 2-stavových automatov je počet automatov vygenerovaných posledným spôsobom vždy okolo 4/3 počtu jazykov.

4.3 Porovnanie použitých metód

V tejto kapitole sa budeme venovať porovnaniu efektivity celého programu pre všetky použité spôsoby generovania a spracovania automatov. Označme jednotlivé spôsoby generovania a spracovania automatov ako v tabuľke 4.3.

Tabuľka 4.3: Označenia

Spôsob generovania	označenie	Spôsob spracovania	označenie
Triviálny spôsob	01	Triviálny spôsob	01
Rozmiestnenie akc. stavov	02	Hashovanie	02
Generovanie súvislých automatov	03	Hopcroftov algoritmus	03
Pridávanie nových stavov	04		

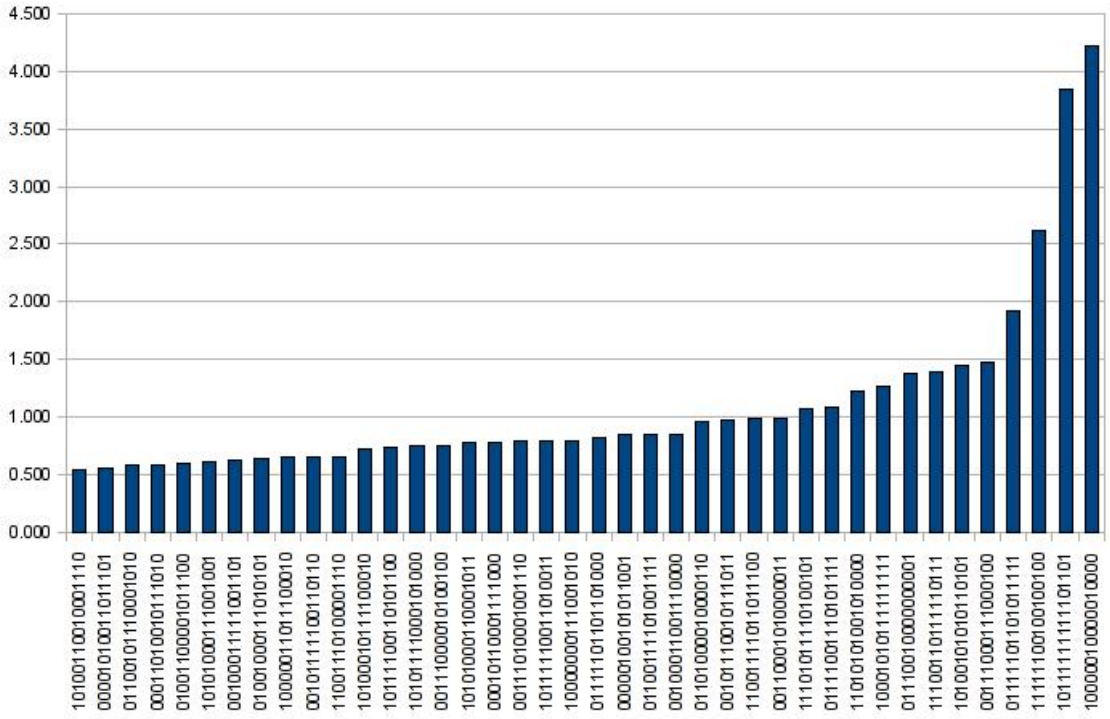
V tabuľke 4.4 sú uvedené časy behu jednotlivých programov pre rôzne hodnoty n. Označenie je v tvare XY, kde X je označenie spôsobu generovania a Y je označenie spôsobu spracovania automatov.

Uvedené sú iba tie hodnoty, ktoré sa dali dopočítať v rozumnom čase – vzhľadom na superexponenciálny rast počtu spracúvaných automatov pre vyššie n. Hashovanie bolo vo všetkých prípadoch robené dvomi reťazcami – 11001101 a 01001010.

Spôsob overovania automatov má najväčší vplyv na čas behu programu. Triviálny spôsob overovania je pomalší ako hashovanie, bez ohľadu na použitú metódu generovania.

4.4 Hashovanie

Hashovanie neurýchľuje beh programu pre všetky reťazce rovnako. V rámci práce sme skúmali efektivitu použitia jedného, alebo dvoch reťazcov. Testovali sme na verzii 0402 z predchádzajúcej kapitoly – čiže najefektívnejší spôsob generovania a hashovaním, a rátali sme výsledok pre $n = 4$. Pre väčšie n už trvá výpočet príliš dlho. Program sme spustili s 40 náhodne zvolenými 17 znakovými reťazcami. V grafe 4.1 môžeme vidieť usporiadané časy behu programu pre tieto reťazce.

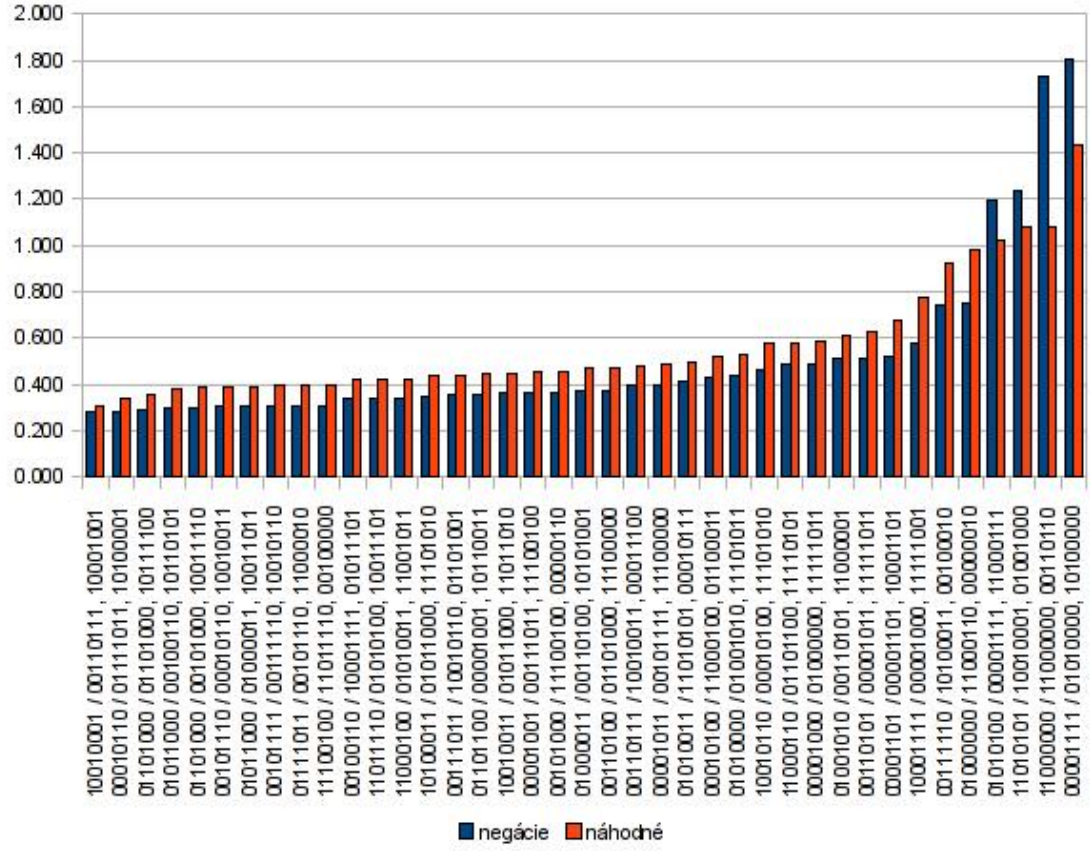


Obr. 4.1: Časy behu pre jeden reťazec

Ďalej sme použili na hashovanie dva reťazce. Skúmali sme, aké výsledky dosiahneme pri náhodne vybratých reťazcoch, a aké pri reťazcoch ktoré sú

jeden negáciou druhého. Reťazce sme však vždy volili tak, aby sa prvé znaky aj u náhodného výberu druhého reťazca líšili.

Hashovacie reťazce boli 8 znakové. Pre 40 náhodne zvolených prvých reťazcov sme spustili program vždy aj s náhodne zvoleným druhým reťazcom, aj s druhým reťazcom, ktorý je negáciou prvého. V grafe 4.2 sú uvedené dĺžky behu.



Obr. 4.2: Časy behu pre dva reťazce

V prvom grafe – s jedným reťazcom je najkratší čas behu nad 0.5 sekundy, vo väčšine prípadov je čas behu okolo 0.8 sekundy. Oproti tomu s dvomi

reťazcami je najčastejší čas behu okolo 0.5 sekundy, pričom najkratší je len 0.3 sekundy. Príčinou môže byť fakt, že rôzne prvé znaky v automatoch, v ktorých z počiatočného stavu vedie prechodová funkcia na rôzne znaky do rôznych stavov, zabezpečí prechod inej časti automatu a tým rovnomernejšiu distribúciu hashovacej funkcie.

V grafe 4.1 červené stĺpce prislúchajú reťazcom, ktoré sú navzájom negáciou, a modré náhodnému výberu druhého reťazca. Kým pri voľbe nesprávneho reťazca (všetky znaky také isté alebo pod.) je pri druhom reťazci negácii čas väčší, lebo sú oba reťazce rovnako „zlé“, zatiaľ čo náhodný druhý reťazec toto vyrovnáva. V priemernom prípade sa však o kúsok lepšie výsledky dosahovali, keď bol druhý reťazec negáciou prvého.

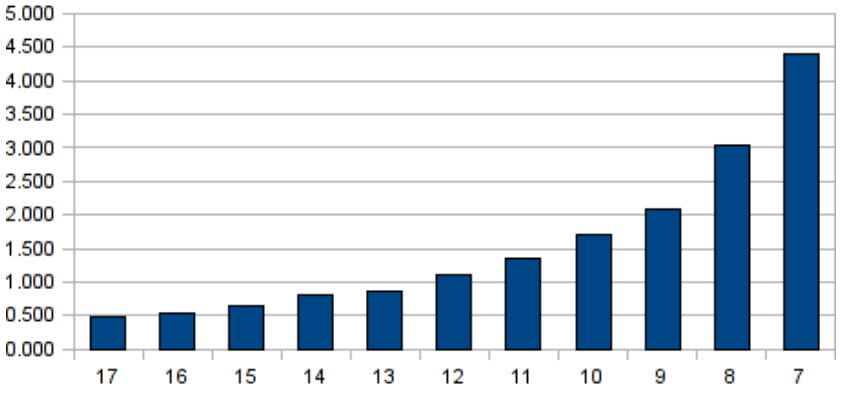
Z dôvodu pamäťových nárokov sme nepoužívali hashovacie reťazce, ktoré by mali v súčte dĺžku väčšiu ako 18.

Zároveň sme nepoužívali reťazce kratšie ako počet stavov automatu, lebo by pri n-stavovom automate nemal možnosť prejsť všetky stavy, čím by nastala menej rovnomerná distribúcia hashov.

Definícia 4.4.1 *Hovoríme, že dva automaty sú v rovnakých* **kopách***, ak je ich hash rovnaký.*

Čas behu najviac ovplyvňuje distribúcia hashovacej funkcie - čím viac automatov sa zahashuje na to isté číslo, tým viac ich spolu porovnávame a teda narastá čas behu, kým pri automate, ktorý má unikátny hash, netreba porovnávať nič. Pre porovnanie uvádzame v tabuľke 4.5 distribúciu funkcie pre najrýchlejšiu a najpomalšiu dvojicu reťazcov z grafu 4.2.

Dĺžka behu závisí od dĺžky hashovacieho reťazca. Program sme spustili pre dĺžky reťazcov od 7 po 17. Pre každú dĺžku sme použili čé náhodne vygenerovaných reťazcov. V grafe 4.3 uvádzame najrýchlejší čas behu pre každú dĺžku reťazca.



Obr. 4.3: Časy behu pre rôzne dĺžky reťazca

4.5 Optimalizácie

Pre urýchlenie behu sme použili rôzne low-level optimalizácie. Prvá verzia programu používala pokročilé dátové štruktúry, ktoré ju výrazne spomaľovali. Algoritmus využívajúci 2. spôsob generovania automatov a triviálne overovanie tak bežal 8.5 hodiny pre $n = 4$. Nahradením za jednoduchšie štruktúry sme urýchlili program pri použití toho istého algoritmu na 32 minút pre $n = 4$.

Niektoré dynamické štruktúry sme nahradili statickými, čím sme ušetrili pamäť, keďže už neboli potrebné pointre, použité v dynamických štruktúrach. Pri volaniach funkcií sme odovzdávali premenné referenciou miesto hodnotou, čím sa predišlo alokovaniu ďalšej pamäte na stacku.

Ďalšie urýchlenie sa dá docieliť tým, že sa celá prechodová funkcia pre automat s najviac 8 stavmi dá reprezentovať v jednej premennej typu integer. V takom prípade operácie, ktoré robíme s automatmi, bežia v konštantnom čase, na rozdiel od automatov reprezentovaných poľom.

Tabuľka 4.4: Procesorový čas

Označenie	3	4	5	6	7
0101	0,292 s	98 m 32 s			
0201	0,180 s	36 m 48 s			
0301	0,164 s	27 m 47 s			
0401	0,116 s	10 m 41 s			
0102	0,012 s	3,234 s	18 h 18 m		
0202	0,012 s	1,283 s	4 h 36 m		
0302	0,008 s	1,012 s	3 h 09 m		
0402	0,012 s	0,352 s	24 m 8 s		
0403	0,004 s	0,193 s	14,5 s	22 m 27 s	38 h 11 m

Tabuľka 4.5: Distribúcia hashovacej funkcie

veľkosť kopy	# DKA v kope	počet kôp	# DKA v kope	počet kôp
1	4142	4142	1040	1040
2 - 3	5796	2644	482	228
4 - 7	14648	2734	10594	2190
8 - 15	7670	684	2828	264
16 - 31	13236	624	8902	448
32 - 63	6846	160	9402	222
64 - 127	4472	54	8382	92
128 - 255	258	2	5080	30
256 - 511			7210	22
512 - 1023			3148	4

Kapitola 5

Záver

V práci sme sa zaoberali skúmaním algoritmov, zisťujúcich počet rôznych jazykov, akceptovaných deterministickými konečnými automatmi. V kapitole 3 sme popísali spôsoby generovania a spracovania automatov.

V kapitole 4 sme sa venovali porovnaniu použitých metód. Zvláštnu pozornosť sme venovali hashovaniu a vplyvu výberu reťazca na dĺžku výpočtu. Najefektívnejšou metódou bolo overovanie minimality automatov. Zistenie minimality automatu nie je časovo náročnejšie ako porovnanie dvoch automatov, ale zatiaľ čo porovnávanie automatov sa pre každý automat opakovalo viackrát, test na minimalitu sa robí iba raz. Táto metóda mala zároveň aj najmenšie požiadavky na pamäť.

Pri poslednej z uvedených metód generovania bol overovaný počet automatov blízky hľadanému výsledku. Vzhľadom na rýchly rast počtu jazykov pre väčšie n, sa už nebude dať výpočet veľmi zrýchliť zredukovaním počtu overovaných automatov. Preto pre zistenie výsledku pre vyššie hodnoty n bude potrebné postupovať inak ako kontrolou každého automatu.

Literatúra

[DKS02] Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. *J. Autom. Lang. Comb.*, 7(4):469–486, 2002.

[FR09] Michal Forišek and Branislav Rovan. *Formálne jazyky a automaty (skriptá).* 2009. `http://foja.dcs.fmph.uniba.sk/materialy/skripta.pdf`.

[Hud07] Ivana Hudáková. Zložitostné aspekty konečných automatov. 2007. Bakalárska práca, Comenius University, Bratislava.

[Nic99] Cyril Nicaud. Average State Complexity of Operations on Unary Automata. In *MFCS '99: Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, pages 231–240, London, UK, 1999. Springer-Verlag.