Univerzita Komenského, Bratislava
Fakulta matematiky, fyziky a informatiky

Stavová zložitosť deterministických a nedeterministických konečných automatov

Bakalárska práca

2014

Rafael Korbaš

Univerzita Komenského, Bratislava
Fakulta matematiky, fyziky a informatiky

Stavová zložitosť deterministických a nedeterministických konečných automatov

Bakalárska práca

Študijný program:	Informatika
Odbor:	2508 Informatika
Katedra:	Katedra informatiky
Vedúci:	Mgr. Marek Zeman

Bratislava, 2014

Rafael Korbaš

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Rafael Korbaš
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Stavová zložitosť deterministických a nedeterministických konečných automatov

Cieľ: Cieľom práce je skúmať zložitosť operácie determinizácie nedeterministických konečných automatov. Sú známe jazyky, pri ktorých je tento nárast exponenciálny, ale aj jazyky, kedy k nárastu neprichádza vôbec. V tejto práci budeme štatistickou metódou zisťovať zložitosť tejto operácie v priemernom prípade.

Vedúci: Mgr. Marek Zeman
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 11.10.2013

Dátum schválenia: 24.10.2013

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Týmto by som sa chcel poďakovať svojmu vedúcemu Mgr. Marekovi Zemanovi za rady, nápady a odbornú spoluprácu, Mgr. Jaroslavovi Budišovi za poskytnutie prostriedkov na vykonanie potrebných výpočtov a rodine a priateľom za podporu.

Abstrakt

Témou tejto práce je analyzovať rozdiel v počte stavov medzi malými nedeterministickými a deterministickými konečnými automatmi, predovšetkým na binárnej abecede. Jedným z prínosov je tiež enumerácia jazykov akceptovaných nedeterministickými konečnými automatmi do 4 stavov. Dôležitou súčasťou práce je kapitola 4, kde sú prezentované naše výsledky.

Kľúčové slová: konečný automat, minimálny NKA, minimálny DKA, stavová zložitosť, enumerácia regulárnych jazykov

Abstract

The topic of this bachelor thesis is to analyze the deterministic and nondeterministic state complexity of regular languages, focusing on the automata over binary alphabet. An important part of this thesis is the chapter 4, where we present our results.

Keywords: finite automata, minimal NFA, minimal DFA, state complexity, enumeration of regular languages

Obsah

Úvod **1**

1 Základné pojmy a definície **2**
1.1 Deterministický konečný automat 2
1.2 Nedeterministický konečný automat 3
1.3 Determinizácia NKA 4
1.4 Stavová zložitosť regulárnych jazykov 4

2 Minimalizácia DKA a NKA **5**
2.1 Minimalizácia DKA 5
2.1.1 Brzozowského algoritmus 5
2.1.2 Hopcroftov algoritmus 6
2.2 Minimalizácia NKA 7

3 Analýza problému **9**
3.1 Použitý programovací jazyk 9
3.2 Generovanie NKA 9
3.2.1 Fixovanie počiatočného stavu 10
3.2.2 Akceptačné stavy 10
3.2.3 Symetria v δ-funkcii. 11
3.2.4 Zahadzovanie nesúvislých δ-funkcii. 12
3.2.5 Počet NKA po aplikácii optimalizacií 12
3.3 Testovanie ekvivalencie NKA 12
3.3.1 Testovanie ekvivalencie DKA s využitím operácie komplementu a kartézskeho súčinu 13
3.3.2 Testovanie ekvivalencie DKA s využitím jednoznačnosti minimálneho DKA 13
3.4 Hashovanie 14
3.4.1 Hashovanie podľa slov do fixnej dĺžky 14
3.4.2 Hashovanie podľa vybraných slov 15

3.4.3 Hashovanie pomocou „odtlačku" slova 15
3.5 Jednoznačné mapovanie automatov na čísla 15

4 Výsledky a ich náväznosť na predošlý výskum 17
4.1 Hlavný výskum 17
4.2 Odhad deterministickej zložitosti jazykov s nedeterministickou zložitosťou 5 20
4.3 Ďalší výskum 21
4.3.1 Distribúcia jazykov vzhľadom na počet rôznych minimálnych NKA, ktoré ich akceptujú 22
4.3.2 Dĺžka slov, ktoré jednoznačne odlíšia dva NKA 24

5 Dokumentácia k programu 26
5.1 Variables.java 26
5.2 Automaton.java 26
5.3 Tuple.java 28
5.4 MinimalAutomatonHashMap.java 29
5.5 AutomatonIterator.java 29
5.6 Experiments.java 30
5.7 Bakalarka.java 31

A DVD médium 33

Zoznam tabuliek

3.1 Počet všetkých možných NKA s n stavmi. 10
3.2 Počet vygenerovaných NKA (namerané z programu). 12

4.1 $G_1(n)$ pre $1 \leq n \leq 5$. 17
4.2 Distribúcia minimálnych DKA s j stavmi pre nájdené n-stavové minimálne NKA nad unárnou abecedou. 18
4.3 $G_2(n)$ pre $1 \leq n \leq 4$. 18
4.4 Distribúcia veľkosti minimálnych DKA pre jazyky s n-stavovým minimálnym NKA nad binárnou abecedou. 18
4.5 Priemerné deterministické zložitosti pre jazyky s danými nedeterministickými zložitosťami. 19
4.6 Časy výpočtu $G_2(n)$ pre $1 \leq n \leq 4$. 19
4.7 Distribúcia stavových zložitostí pre nájdené jazyky akceptované 5-stavovými NKA. 21
4.8 Distribúcia jazykov vzhľadom na minimálne NKA s 1 stavom, ktoré ich akceptujú. 23
4.9 Distribúcia jazykov vzhľadom na minimálne NKA s 2 stavmi, ktoré ich akceptujú. 23
4.10 Distribúcia jazykov vzhľadom na minimálne NKA s 3 stavmi, ktoré ich akceptujú. Kompletné výsledky nájdete v priloženom DVD. 23
4.11 Počet jazykov s jednoznačným minimálnym NKA vzhľadom na ich nedeterministickú zložitosť. 23
4.12 δ_n pre n-stavové NKA nad 2-znakovou abecedou do 3 stavov. 24
4.13 Počet kolízií pre jednotlivé dĺžky slov pre NKA do 2 stavov. 25
4.14 Počet kolízii pre jednotlivé dĺžky slov pre NKA do 3 stavov. 25

Úvod

Konečné automaty majú uplatnenie vo viacerých oblastiach informatiky, napríklad pri lexikálnej analýze, hľadaní výskytov reťazcov v texte, vytváraní jednoduchých stratégii a inde.

Je známe, že hoci výpočtová sila deterministických konečných automatov (DKA) je ekvivalentná výpočtovej sile nedeterministických konečných automatov (NKA), minimálny DKA môže mať vzhľadom na ekvivalentný minimálny NKA potenciálne až 2^n stavov, kde n je počet stavov pôvodného NKA. Existuje ale veľa prípadov, v ktorých je tento nárast značne menší.

Výhodou NKA je menší počet stavov a jednoduchší návrh, keďže tam nie sú také striktné požiadavky. Naproti tomu, DKA sa jednoduchšie implementujú, keďže to lepšie zodpovedá spôsobu, akým fungujú súčasné počítače a v konečnom dôsledku aj väčšina programovacích jazykov. Aj preto by nás mohlo zaujímať, aké je to v skutočnosti „zlé" alebo „dobré" s počtom stavov pri DKA.

Naším cieľom bude tento nárast počtu stavov v priemernom prípade experimentálne odmerať pre NKA do 4 stavov nad binárnou abecedou a pokúsiť sa nahliadnuť, ako to vyzerá pri väčšom počte stavov a iných abecedách.

V práci sa zaoberáme aj otázkou, koľko jedinečných regulárnych jazykov sú schopné akceptovať nedeterministické automaty ohraničené malým počtom stavov nad malou abecedou. Bude to jeden z výsledkov, ktorý vyplynie zo spôsobu, akým budeme náš experiment vykonávať.

V minulosti sa touto otázkou zaoberali napr. v článku [DKS02]. Ich výsledky pokryli NKA do 3 stavov. V našej práci posúvame túto hranicu a snažíme sa zistiť situáciu pre NKA s viac stavmi, čo by nám mohlo poskytnúť lepší odhad všeobecného vzťahu. Výpočtová úloha, ktorú potrebujeme implementovať pre tieto potreby je vzhľadom na možnosti súčasného hardvéru pomerne náročná.

Kapitola 1

Základné pojmy a definície

V tejto práci skúmame konečné automaty. Najskôr uvedieme definície a označenie základných pojmov, uvedieme označenie a dokážeme potrebné vety, aby sme mohli ďalej na nich postaviť naše úvahy. Budeme sa držať notácie z [HMU01], resp. bakalárska práca [Pet09], ktorá sa tiež zaoberala konečnými automatmi. Najzákladnejšie pojmy, akými sú napr. abeceda, gramatika či jazyk možno nájsť v skriptách [FR13].

1.1 Deterministický konečný automat

Definícia 1.1.1. ***Deterministický konečný automat (skrátene DKA)*** *A je pätica* $(K, \Sigma, \delta, q_0, F)$*, kde* K *je konečná množina stavov,* Σ *je konečná vstupná abeceda,* $q_0 \in K$ *je počiatočný stav,* $F \subseteq K$ *je množina (konečných) akceptačných stavov a* $\delta : K \times \Sigma \to K$ *je prechodová funkcia (*δ*-funkcia).*

Definícia 1.1.2. ***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 1.1.3. ***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 1.1.4. ***Jazyk*** *akceptovaný deterministickým konečným automatom* A *je množina* $L(A) = \{w \mid \exists q_F \in F; (q_0, w) \vdash_A^* (q_F, \epsilon)\}$*.*

Definícia 1.1.5. ***Minimálny DKA****. Deterministický konečný automat* $A = (K_A, \Sigma, \delta_A, q_{0A}, F_A)$ *je minimálny, pokiaľ pre všetky DKA* $B = (K_B, \Sigma, \delta_B, q_{0B}, F_B)$ *platí:*

$$L(A) = L(B) \Rightarrow |K_A| \leq |K_B|$$

Veta 1.1.6. *(Myhill-Nerode) Majme jazyk* $L \subseteq \Sigma^*$*. 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.*

Táto veta nám hovorí, že každý regulárny jazyk možno „rozbiť" na konečný počet tried ekvivalencie, pričom jeden z dôsledkov je, že minimálny DKA má práve toľko stavov, koľko tried ekvivalencie má relácia R_L. Ďalší dôsledok je, že minimálny DKA k jazyku L je jednoznačný až na izomorfizmus, t.j. až na pomenovanie stavov musia byť dva ekvivalentné minimálne DKA totožné. Dôkaz Myhill-Nerodovej vety nájdete napr. v [FR13, Veta 2.9.1] a jednoznačnosť minimálneho DKA v [HMU01, Veta 4.26].

V tejto práci budeme uvažovať len deterministické automaty s úplnou δ-funkciou, t.j. nemôže sa stať, že by v niektorom stave nebol definovaný prechod na niektorý znak. Samozrejme, je ľahké prerobiť automat s neúplnou δ-funkciou. Stačí pre nedefinované prechody pridať „odpadový" stav. Teda automaty s neúplnou δ-funkciou neprinášajú podstatné zmenšenie počtu stavov v porovnaní s automatmi s úplnou δ-funkciou. Pri návrhu deterministických automatov sa na túto formalitu často zabúda a odpadový stav sa zamlčiava. Pre nás je takéto zanedbanie neprípustné, keďže chceme korektne zmerať nárast počtu stavov oproti nedeterministickým konečným automatom, ktoré teraz ideme definovať.

1.2 Nedeterministický konečný automat

Definícia 1.2.1. ***Nedeterministický konečný automat*** *A je pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je konečná vstupná abeceda, $q_0 \in K$ je počiatočný stav, $F \subseteq K$ je množina akceptačných (koncových) stavov a $\delta : K \times (\Sigma \cup \{\epsilon\}) \to 2^K$ je prechodová funkcia*

Definícia 1.2.2. ***Krok výpočtu*** *deterministického konečného automatu A je relácia $\vdash_A$ na konfiguráciách definovaná $(q, av) \vdash_A (p, v) \iff p \in \delta(q, a)$.*

Konfigurácia a jazyk akceptovaný NKA sú definované rovnako ako pre DKA. Narozdiel od DKA, minimálny NKA nemusí byť jednoznačný, teda môže existovať viacero možných minimálnych ekvivaletných NKA, ktoré nie sú navzájom izomorfné.

1.3 Determinizácia NKA

Definícia 1.3.1. ***Determinizácia NKA*** *$A = (K_A, \Sigma, \delta_A, q_0, F_A)$ je DKA $D(A) = (K_{D(A)}, \Sigma, \delta_{D(A)}, q'_O, F_{D_A})$, pre ktorý platí, že $L(A) = L(D(A))$.*

Je známy pomerne jednoduchý algoritmus na determinizáciu NKA ([HMU01, str. 61]). K NKA $A = (K_A, \Sigma, \delta_A, q_0, F_A)$ zostrojíme DKA $D = (K_D, \Sigma, \delta_D, \{q_0\}, F_D)$ tak, že jeho stavy budú reprezentovať jednotlivé podmnožiny 2^{K_A}. Za počiatočný stav zvolíme množinu prislúchajúcu počiatočnému stavu a následne budeme skúmať podobne, ako pri prehľadávaní do šírky, na ktoré stavy v tom pôvodnom NKA A sa môžeme dostať cez daný znak. Konečné stavy potom budú tie, ktoré reprezentujú množiny obsahujúce niektorý z konečných stavov v pôvodnom NKA A. Formálne:

$$K_D = 2^{K_A}$$
$$\forall a \in \Sigma, S \subseteq K_A : \delta_D(S, a) = \bigcup\nolimits_{p \in S} \delta_A(p, a)$$
$$F_D = \{S : (S \in K_D) \wedge (S \cap F_A \neq \emptyset)\}$$

Pri samotnej implementácii môžeme zanedbať nedosiahnuteľné stavy, ktorých môže byť veľa.

1.4 Stavová zložitosť regulárnych jazykov

Existuje mnoho mier zložitostí konečných automatov. V našej práci budeme uvažovať počet stavov. Okrem toho si zavedieme nasledujúce pojmy, ktoré nam zjednodušia vyjadrovanie vo zvyšnej časti práce.

Pod stavovou zložitosťou automatu rozumieme počet jeho stavov. S tým úzko súvisia nasledujúce pojmy, ktoré si definujeme a zjednodušia nám vyjadrovanie v nasledujúcich kapitolách.

Definícia 1.4.1. ***Deterministická stavová zložitosť*** *regulárneho jazyka L je počet stavov minimálneho DKA, ktorý tento jazyk akceptuje.*

Definícia 1.4.2. ***Nedeterministická stavová zložitosť*** *regulárneho jazyka L je počet stavov minimálneho NKA, ktorý tento jazyk akceptuje.*

Aby sme sa nemuseli vždy vyjadrovať takto zdĺhavo, tak nedeterministickú a deterministickú stavovú zložitosť budeme skrátene označovať „nedeterministická", resp. „deterministická zložitosť jazyka".

Kapitola 2

Minimalizácia DKA a NKA

V tejto časti sa budeme podrobnejšie venovať problému hľadania minimálneho DKA, resp. NKA.

2.1 Minimalizácia DKA

2.1.1 Brzozowského algoritmus

Princíp tohto algoritmu je veľmi jednoduchý. Najprv definujme, resp. popíšme základné operácie, ktoré budeme na jeho realizáciu využívať.

V nasledujúcej definícii budeme uvažovať variant NKA, ktorý pripúšťa viacero počiatočných stavov.

Definícia 2.1.1. ***Reverz konečného automatu*** $A = (K_A, \Sigma, \delta_A, q_0, F_A)$ *je konečný automat* $R(A) = (K_A, \Sigma, \delta_{R(A)}, F_A, q_O)$*, pre ktorý platí:*

$$\forall a \in \Sigma, \forall p, q \in K_A : (p, a) \to q \in \delta_A \Rightarrow (q, a) \to (p)$$

Je dôležité si všimnúť, že sme zamenili aj počiatočný stav s množinou akceptačných stavov, čo má za následok, že výsledný automat môže mať viac ako jeden počiatočný stav, prípadne žiaden.

Samotný algoritmus vyzerá nasledovne:
Na vstupe máme DKA $A = (K_A, \Sigma, \delta_A, q_{0A}, F_A)$. Nech operácia $R(A)$ označuje reverz automatu A a operácia $D(A)$ determinizáciu automatu A, tak ako boli vyššie definované. Potom minimálny DKA B akceptujúci rovnaký jazyk ako NKA A skonštruujeme nasledovne:

$$B = D(R(D(R(A))))$$

Operácia determinizácie môže mať až exponenciálnu časovú zložitosť vzhľadom na počet stavov pôvodného NKA, keďže toľko stavov môže teoreticky mať DKA ekvivalentný k danému NKA, avšak, v priemernom prípade sa to nejaví byť až také zlé. Takúto časovú zložitosť v konečnom dôsledku dosahuje aj Brzozowského algoritmus, keďže využíva túto operáciu. Existujú aj asymptoticky rýchlejšie algoritmy na minimalizáciu NKA, prekvapivo sa ale ukázalo, že aj takýto jednoduchý, i keď pomalší algoritmus bude postačovať pre naše potreby

V ďalšej časti uvedieme pre úplnosť ďalší, efektívnejší algoritmus na minimalizáciu DKA, i keď sme ho napokon neimplementovali.

2.1.2 Hopcroftov algoritmus

Tento algoritmus funguje na odlišnom princípe ako ten Brzozowského. Kľúčová je operácia zlučovania tzv. „nerozlíšiteľných" stavov. Uvedieme ho podobne, ako v [Pet09].

Definícia 2.1.2. ***Rozklad*** *množiny K tvoria po dvojiciach disjunktné množiny* $K_1, K_2, ..., K_N$ *(tzv. triedy rozkladu) také, že ich zjednotenie tvorí práve množinu K.*

Definícia 2.1.3. ***Kongruencia automatu*** *A je je rozklad množiny stavov K taká, že pre každú triedu rozkladu* K_i *platí:*

$$\forall p, q \in K, a \in \Sigma : (p, q \in K_i) \Rightarrow (\delta(p, a), \delta(q, a) \in K_j)$$

Definícia 2.1.4. *Rozklad* $\{K_1, K_2, ..., K_n\}$ *pokladáme za hrubší ako rozklad* $\{K'_1, K'_2, ..., K'_m\}$*, pokiaľ každá z tried* $K_1, ..., K_m$ *je zjednotením niekoľkých tried z* $K'_1, ..., K'_m$

Keď sme si už definovali potrebné pojmy, prejdime na samotný algoritmus. Na začiatku behu algoritmu budeme uvažovať dve základné triedy rozkladu množiny stavov automatu A, a to množinu akceptačných a tých ostatných, teda neakceptačných stavov, teda množiny F a $K - F$. Tento rozklad budeme postupne zjemňovať pokiaľ to len bude možné a takto dostaneme najhrubšiu kongruenciu automatu A, ktorá definuje stavy hľadaného minimálneho DKA.

Máme rozklad P a množinu dvojíc S = (C,a), kde $a \in \Sigma$ a C je trieda rozkladu P. Budeme pracovať s dvojicami v množine S. Na začiatku $P = \{F, K - F\}$ a $S = (C, a), \forall a \in \Sigma$, kde C je menšia z tried $F, K - F$. Algoritmus je potom nasledovný:

Vyberieme jednu dvojicu (C,a) z S a pre každú triedu B rozkladu P vykonáme nasledujúce:

- Overíme, či (C, a) rozdeľuje B na nové triedy rozkladu, t.j. či platí, že:
$$\exists p, q \in B : \delta(p, a) \in C \wedge \delta(q, a) \notin C$$
Ak (C,a) nerozdeľuje B, tak prejdeme na ďalšiu triedu B. Ak (C,a) rozdeľuje B, vytvoríme nové triedy $B_1 = \{q \mid \delta(q, a) \in C\}$ a $B_2 = \{q \mid \delta(q, a) \notin C\}$.

- Triedu B v P zrušíme a nahradíme ju triedami B_1 a B_2. Teda pre každé $b \in \Sigma$ ak $(B, b) \in S$, tak nahradíme ho pármi (B_1, b) a (B_2, b) Ak sa (B, b) v S nenachádza, vyberieme menšiu z množín B_1, B_2, nech je to B', a pridáme do S dvojicu (B', b).

Tieto kroky cyklicky opakujeme, až kým sa množina S vyprázdni. Rozklad P bude reprezentovať stavy výsledného minimálneho DKA a jeho δ-funkcia bude zasa množina S. Akceptačné stavy budú tie triedy, ktoré sú podmnožinami triedy F. Časová zložitosť algoritmu je $O(nslogn)$ kde n je počet stavov DKA na v stupe a s je veľkosť abecedy, ktorú pre naše potreby môžeme pokladať za konštantnú, čiže reálne máme časovú zložitosť $O(nlogn)$. Dôkaz správnosti tohto algoritmu je uvedený v [Hud07].

2.2 Minimalizácia NKA

Našou úlohou je pre dané NKA A nájsť NKA s najmenším možným počtom stavov, ktorý akceptuje rovnaký jazyk ako A.

Narozdiel od minimálneho DKA, môže existovať viacero rôznych neizomorfných ekvivalentných NKA. Navyše, problém minimalizácie NKA, presnejšie jeho rozhodovacia verzia (t.j. pre daný NKA rozhodnúť, či existuje k nemu menší ekvivalentný NKA) je PSPACE-úplný ([JR93]), t.j. ľubovoľný problém riešiteľný s polynomiálne veľkou pamäťou sa dá naň v polynomiálnom čase transformovať. Problém nájdenia minimálneho NKA, ktorý zjavne nemôže byť menej zložitý ako rozhodovací problém. Uveďme si exponenciálny algoritmus, ktorý rieši tento problém.

Na vstupe máme NKA $A = (K_A, \Sigma, \delta_A, q_{0A}, F_A)$ a ideme rozhodnúť, či ide o minimálny NKA pre L(A).

1. K NKA A zostrojíme ekvivalentný minimálny DKA D(A).
2. Vygenerujeme všetky NKA B s menej ako $|K_A|$ stavmi nad abecedou Σ - tých je exponenciálne veľa v závislosti od $|K_A|$ a Σ.
3. Zostrojíme minimálny DKA D(B) ekvivalentný s B a overíme, či je ekvivalentný s D(A). Keďže D(A) aj D(B) sú minimálne DKA, stačí nájsť izomorfizmus medzi stavmi.

4. Ak sú D(A) a D(B) ekvivalentné, t.j. existuje medzi množinami stavov izomorfizmus tak prehlásime, že A nie je minimálny NKA, inak sa vrátime ku kroku 2 a vygenerujeme ďalší NKA v poradí.

5. Pokiaľ sme vygenerovali všetky možné NKA B, a nenašli sme ekvivalentný k NKA A, tak vieme, že NKA A je minimálny.

Podobný algoritmus až na niektoré optimalizácie budeme využívať aj pri našom experimente.

Kapitola 3

Analýza problému

Jednou z našich hlavných úloh je zmerať nárast počtu stavov po prevedení minimálneho NKA na minimálny DKA. Tento nárast budeme merať vzhľadom na jazyky, t.j. pre každý regulárny jazyk práve raz. Podobným problémom, len pre konečné jazyky, sa zaoberali v [GH07], pričom stavovú zložitosť odhadovali vzhľadom na maximálnu dĺžku slova v jazyku.

Ďalším cieľom je zistiť počet rôznych regulárnych jazykov na binárnej abecede akceptované „malými" NKA. Touto otázkou sa zaoberali napríklad v [DKS02] a podarilo sa im enumerovať jazyky nad binárnou abecedou akceptované NKA do 3 stavov.

Tieto dve úlohy spolu veľmi úzko súvisia, nakoľko nárast počtu stavov pri determinizácii budeme merať experimentálne vygenerovaním všetkých jazykov akceptovaných NKA do daného počtu stavov. Rozoberme si podrobnejšie, čo na to budeme potrebovať.

3.1 Použitý programovací jazyk

Na implementáciu všetkých experimentov bola zvolená Java. Stručnú dokumentáciu k programu nájde čitateľ v kapitole 5.

3.2 Generovanie NKA

NKA budeme generovať vzostupne podľa počtu stavov. Akonáhle vygenerujeme NKA, overíme, či sme už predtým nevygenerovali NKA, ktoré by akceptovalo rovnaký jazyk.

Najprv vypočítame, koľko NKA pre daný počet stavov n. Presnejšie, zaujíma nás, koľko existuje všetkých možných NKA s n-stavmi pre binárnu abecedu. Ako počiatočný stav si možeme zvoliť ľubovoľný z n stavov, máme teda n možností voľby počiatočného stavu. Za konečné stavy si môžeme zvoliť ľubovoľnú podmnožinu množiny stavov, t.j.

máme 2^n možností. Ak si δ-funkciu predstavíme ako graf, resp. jeho maticu susednosti, tak medzi n stavmi môže pre jeden znak existovať n^2 prechodov, keďže pripúšťame aj slučky. Dokopy máme 2^{n^2} možností, ako zvoliť prechody medzi jednotlivými stavmi pre jeden znak. Naša abeceda má však dva znaky, teda možností bude 2^{2n^2}. Keď to všetko zosumarizujeme, dostávame $n2^n2^{2n^2} = n2^{2n^2+n}$ možných NKA.

n	# NKA
1	8
2	2 048
3	6 291 456
4	274 877 906 944
5	$\sim 1.8 \times 10^{17}$

Tabuľka 3.1: Počet všetkých možných NKA s n stavmi.

Mohli by sme sa ešte pýtať, či má zmysel generovať aj NKA s prechodmi na ϵ. Odpoveď je, že to zmysel nemá, nakoľko každý automat s prechodmi na ϵ dokážeme odepsilonovať bez zmeny počtu stavov. Nemôže sa stať, že by sme prechodmi na ϵ pridali automaty akceptujúce iné regulárne jazyky ako tie, čo prechody na ϵ nemajú.

Z tabuľky vidno, že počet automatov prudko rastie a pre $n = 4$ je počet NKA príliš veľký, aby sme si mohli dovoliť vygenerovať všetky (o $n = 5$ nehovoriac). Mohli by sme sa pýtať, či potrebujeme generovať skutočne všetky. Nemôžeme niektoré vynechať, lebo si budeme istí, že vygenerujeme iný ekvivalentný NKA s rovnakým počtom stavov? Odpoveď je, že môžeme a dokonca nebude ich málo. Bude to séria optimalizácii, ktoré teraz uvedieme.

3.2.1 Fixovanie počiatočného stavu

Ako prvé si môžeme všimnúť, že nemá zmysel skúšať všetky možné počiatočné stavy, lebo ku každému NKA A existuje ekvivalentný izomorfný NKA B (taký, že má len prečíslované stavy), kde počiatočný stav má čislo 0. Tým zredukujeme počet generovaných automatov n-násobne, kde n je počet stavov generovaných NKA.

3.2.2 Akceptačné stavy

Ďalšia vec, ktorá sa dá obmedziť, je množina akceptačných stavov. Zaujíma nás reálne len ich počet a na poradí až tak nezáleží. Presnejšie, je dôležité iba to, či sa medzi akceptačnými stavmi nachádza aj počiatočný alebo nie. Argument je podobný ako v

predchádzajúcom prípade. K danému NKA A existuje vždy ekvivalentný izomorfný NKA B, ktorého akceptačné stavy odlišné od počiatočného sú očíslované od 1 po k, resp. $k-1$ (ak 0 je akceptačný), kde k je počet akceptačných stavov NKA A. Stačí na to zobrať NKA A a v ňom priradiť počiatočnému stavu číslo 0. Ďalej, pokiaľ je 0 akceptačný stav, tak ostatným akceptačným stavom priradíme zaradom čísla 1 až $k-1$, inak od 1 po k. Takto namiesto 2^n podmnožín stavov stačí generovať $(2n-1)$ podmnožín, keďže prázdnu zjavne testovať nemá zmysel pre $n > 1$, nakoľko automat pre prázdny jazyk ma zrejme jeden stav.

3.2.3 Symetria v δ-funkcii.

Definícia 3.2.1. ***Substitúcia NKA podľa permutácie*** π ***abecedy*** Σ**.**
Nech $A = (K, \Sigma, \delta, q_0, F)$ *je NKA a* π *je permutácia na abecede* Σ*. Potom NKA* $S(A, \pi) = (K, \Sigma, \delta', q_0, F)$*, kde pre* δ' *platí:*

$$\forall a \in \Sigma, p, q \in K : \delta'(p, \pi(a)) = q \iff \delta(p, a) = q$$

nazývame substitúcia NKA A podľa permutácie π *abecedy* Σ*.*

Neformálne – $S(A, \pi)$ je automat, kde sú v každom prechode vymenené znaky tak ako určuje permutácia π. Zjavne jazyk akceptovaný $S(A, \pi)$ má oproti jazyku akceptovanému automatom A iba zamenené znaky podľa permutácie π, inak má rovnakú štruktúru.

Všimnime si, že pre každý NKA platí, že je minimálny pre jazyk, ktorý akceptuje práve vtedy, keď aj substitúcia tohto NKA podľa ľubovoľnej permutácie π abecedy Σ je minimálny NKA pre jazyk so zamenenými znakmi podľa tejto permutácie. Ak si reprezentujeme δ-funkciu ako n-ticu matíc susednosti, kde $n = |\Sigma|$ (prechody pre každý znak abecedy majú vlastnú maticu susednosti), tak stačí minimálnosť overovať len pre jednu n-ticu, napr. tú, v ktorej sú matice susednosti zoradené lexikograficky vzostupne. Ak sa ukáže, že tento automat minimálny nie je, alebo už predtým sme generovali ekvivalentný, tak vieme, že ďalej nemá zmysel testovať ostatné permutácie matíc susednosti. Ak zistíme, že sme našli NKA pre nový jazyk, tak otestujeme aj permutácie. Pre dvojznakovú abecedu sme týmto práve eliminovali potrebu generovať a testovať skoro polovicu automatov, keďže nad dvomi znakmi sú len dve permutácie. Samozrejme pritom predpokladáme, že regulárnych jazykov je rádovo menej oproti NKA, ktoré ich akceptujú, čo sa ukazuje byť pravdivý predpoklad aspoň pre NKA do 4 stavov nad binárnou abecedou.

3.2.4 Zahadzovanie nesúvislých δ-funkcii.

NKA s nesúvislou δ-funkciou sú pre nás nezaujímavé, lebo odstránením nedosiahnuteľných stavov zjavne dostaneme menší NKA akceptujúci ten istý jazyk. To znamená, že nemôžu byť minimálne. Ak vidíme, že vygenerovaná matica susednosti pre δ-funkciu (resp. ich bitový or, v prípade, že každý znak má vlastnú maticu susednosti) nereprezentuje súvislý graf, tak ju môžeme rovno ignorovať a ušetriť si ďalšie testy.

3.2.5 Počet NKA po aplikácii optimalizacií

V tabuľke 3.2 sú uvedené približné počty potrebných vygenerovaných NKA po uplatnení spomínaných optimalizácii:

n	# testovaných NKA
1	8
2	377
3	557 812
4	14 088 285 129

Tabuľka 3.2: Počet vygenerovaných NKA (namerané z programu).

Z tabuľky vidno, že pre $n = 4$ ide už o pomerne prijateľné počty. Pre $n = 5$ sa nám presné počty namerať nepodarilo nakoľko stále je ich príliš veľa, aby sa dali enumerovať. Ak zanedbáme zahodenie automatov s nesúvislou δ-funkciou, malo by ich byť približne 5.06×10^{15}.

3.3 Testovanie ekvivalencie NKA

V predošlej časti sme riešili generovanie NKA, resp. minimalizáciu počtu vygenerovaných NKA. V tejto časti podrobnejše rozoberieme, ako efektívne zistiť, či sme už predtým nevygenerovali ekvivalentný NKA.

Testovať ekvivalenciu dvoch NKA priamo nie je veľmi praktické. Jednoduchšie je tento problém previesť na testovanie ekvivalencie DKA. Tam si môžeme vybrať z viacerých možností.

3.3.1 Testovanie ekvivalencie DKA s využitím operácie komplementu a kartézskeho súčinu

Z teórie množín vieme, že $A = B \iff A' \cap B' = \emptyset$, kde A, resp. B sú množiny. Rovnaká myšlienka sa dá aplikovať aj na regulárne jazyky. Výhoda tohto prístupu je, že pre dané DKA dokážeme jednoducho skonštruovať automat pre komplement (vymeníme akceptačné a neakceptačné stavy) a pre prienik jazykov akceptovaných dvomi DKA (konštrukcia pomocou kartézskeho súčinu). Následne na otestovanie, či výsledný automat akceptuje prázdny jazyk, stačí spraviť napríklad prehľadávanie do šírky z počiatočného stavu, aby sme zistili, či existuje cesta z počiatočného stavu do niektorého akceptačného. Ak áno, tak vieme, že existuje slovo, ktoré tento automat akceptuje a jazyk ním akceptovaný nemôže byť prázdny. Táto situácia nastane práve vtedy, keď automaty na vstupe nie sú ekvivalentné.

3.3.2 Testovanie ekvivalencie DKA s využitím jednoznačnosti minimálneho DKA

Vieme, že ku každému regulárnemu jazyku existuje jednoznačný minimálny DKA až na izomorfizmus, t.j. premenovanie stavov. Podľa tohto stačí oba NKA, ktoré porovnávame, previesť na minimálne DKA a overiť, či sú izomorfné. Ešte efektívnejšie by bolo, keby sme dokázali nájsť normálny, resp. jednoznačne určený tvar pre minimálne DKA. Ukazuje sa, že taký tvar existuje, nazvime ho kanonický minimálny DKA:

Definícia 3.3.1. ***Kanonický minimálny DKA*** *Nech $A = (K, \Sigma, \delta, q_0, F)$ je minimálny DKA pre jazyk L, definujme zobrazenie C, ktoré minimálnemu DKA A priradí minimálny DKA $C(A)$ tak, že spustíme prehľadávanie do šírky z počiatočného stavu s tým, že budeme dávať prednosť pri návšteve hranám δ-funkcie v abecednom poradí vzhľadom na Σ. Následne budeme číslovať stavy podľa poradia, v ktorom ich navštívime od 0, ktorú dostane počiatočný stav, keďže z neho začíname. Minimálnemu DKA $C(A)$ potom hovoríme kanonický minimálny DKA.*

Veta 3.3.2. ***Kanonický minimálny DKA*** *Nech A_1, A_2 sú dva rôzne ekvivalentné minimálne DKA, potom $C(A_1) = C(A_2)$.*

Dôkaz. Majme dva ľubovoľné ekvivalentné minimálne DKA. Vieme, že sú izomorfné. Spustime na oboch prehľadávanie do šírky tak, ako sme popísali vo vete, ktorú dokazujeme. Indukciou dokážeme, že algoritmus uvedený vo vete bude mať rovnaký priebeh na oboch automatoch.

Bázu indukcie urobíme pre počiatočný stav. V oboch DKA im algoritmus priradí index 0, keďže je to prvý stav, ktorý navštívi. Aj susedov pridajú oba behy do fronty v rovnakom poradí, keďže oba automaty majú k hranám (resp. prechodom) priradené rovnaké písmená, z každého stavu vedie práve jeden prechod na každý znak a algoritmus pridáva susedov v abecednom poradí.

Predpokladajme, že algoritmus má rovnaký priebeh na oboch automatoch po očíslovaní i stavov, t.j. má rovnaké stavy vo fronte a doteraz priradil navštíveným vrcholom v oboch automatoch tie isté indexy.

Môžu nastať dva prípady. Buď je fronta prázdna, čo znamená, že sme navštívili a očíslovali už všetky vrcholy - vtedy je zjavné, že výstupom oboch behov je rovnaké očíslovanie stavov. Pokiaľ fronta nie je prázdna, tak obe inštancie algoritmu vyberú z nej ten istý $i + 1$-vý stav a pozrú sa na jeho susedov. Týchto susedov ale oba behy pridajú do fronty v rovnakom poradí - argument je rovnaký ako pri báze indukcie. □

Na testovanie ekvivalencie stačí pre oba NKA určiť ich kanonické minimálne DKA a otestovať, či sú totožné.

3.4 Hashovanie

Testovanie ekvivalencie priamym porovnávaním so všetkými predošlými nájdenými minimálnymi NKA, resp. reprezentantmi jazykov, ktoré akceptujú, sa ukázalo neefektívne pre naše potreby. Jedno z možných riešení je hashovanie, ktoré si rozoberieme v tejto časti.

Cieľom hashovania je obmedziť počet testovaní ekvivalencie s predošlými nájdenými minimálnymi NKA. Hashovacia funkcia h by mala každému automatu A, resp. B priradiť jednoznačne číslo tak, aby platilo:

$$L(A) = L(B) \Rightarrow h(A) = h(B)$$

To znamená, že sa stačí obmedziť na automaty s rovnakým odtlačkom hashovacej funkcie. Tých bude rádovo menej než všetkých automatov dokopy za predpokladu, že hashovacia funkcia je vhodne zvolená.

3.4.1 Hashovanie podľa slov do fixnej dĺžky

Spočíva v tom, že ako hash používame množiny slov, resp. ich reprezentáciu pomocou bitsetu (každý prvok má v reťazci priradenú 0 alebo 1 podľa toho, či tam patrí alebo

nie). Zvolíme si, napr., že chceme hashovať podľa slov do dĺžky 5, teda otestujeme, ktoré slová do tejto dĺžky akceptuje náš automat a ktoré nie a podľa toho vyplníme reťazec nulami a jednotkami. Pre dĺžku 5 potrebujeme reťazec dĺžky 63, keďže toľko slov do dĺžky 5 existuje nad binárnou abecedou. Pre väčšie dĺžky to už nie je veľmi efektívne nakoľko počet testovaných slov rastie exponenciálne a už pre dĺžku napríklad 10 by sme pre každý vygenerovaný automat museli testovať, či akceptuje alebo nie 1023 slov, čo je neprijateľné, nehovoriac o dĺžke reťazca, ktorý na hashovanie použijeme. Dĺžka 5 sa ukazuje byť prijateľná pre NKA do 3 stavov. Pre vačšie NKA už vznikalo príliš veľa kolízií a generovanie sa tým výrazne začalo spomaľovať, keďže každý vygenerovaný automat sa porovnával s desiatkami predošlých.

3.4.2 Hashovanie podľa vybraných slov

Ďalšou možnosťou je zamerať sa na všetky slová do nejakej fixnej dĺžky, ale zvolíme si, aké slova budeme testovať. Výhoda za predpokladu, že nájdeme dostatočne „dobré" slová by bola, že zmenšíme počet kombinácii, ktoré nemôžu nastať. Tento prístup je tu len načrtnutý, nakoľko nakoniec nebol reálne využitý, lebo hľadať vhodnú množinu slov sa ukázalo byť netriviálne a dosiahnuté výsledky pri experimentovaní aj tak nepredčili predošlý prístup.

3.4.3 Hashovanie pomocou „odtlačku" slova

Myšlienku sme čerpali z [Pet09]. Budeme mať množinu slov, ktorú vyskúšame na každom automate. Budeme si ale pamätať nielen to, či slovo bolo akceptované, ale aj výpočet, resp. pre každý znak slova si zapamätáme, či bol automat v danej chvíli v akceptačnom stave alebo nie. Ak testujeme viacero slov, tak tieto stopy môžeme napríklad navzájom zreťaziť alebo zoxorovať. Pri testovaní tento prístup nepriniesol významné zlepšenie a pre NKA do 4 stavov by sme sa aj tak s najvyššou pravdepodobnosťou nevyhli neúnosnému počtu kolízii.

Hashovanie v tejto forme sa neukázalo byť najvhodnejším prístupom na dosiahnutie požadovaných výsledkov.

3.5 Jednoznačné mapovanie automatov na čísla

Našim cieľom navrhnúť funkciu, ktorá každému NKA priradí jednoznačne číslo alebo k-ticu čísel tak, že dva automaty majú priradené rovnaké číslo práve vtedy, keď sú

ekvivalentné. Výhody takejto funkcie by boli veľké - čísla sa porovnávajú v počítači veľmi rýchlo a ľahšie sa ukladajú a hashujú.

Na prvý pohľad sa to môže zdať ako ťažká úloha, ale už sme spomínali, že ku každému NKA existuje kanonický DKA(3.3.2). Od tohto poznatku nás delí už len malý krok k tomu, aby sme dostali požadovaný výsledok. Myšlienka je nasledovná:

K danému NKA zostrojíme minimálny kanonický DKA. Počiatočný stav má číslo 0 pre každý takýto DKA, čiže si ho nepotrebujeme pamätať. Množinu akceptačných stavov teraz reprezentujeme ako bitset, t.j. v bitovej reprezentácii integeru si poznačíme na k-tu pozíciu jednotku, ak stav k je akceptačný, inak si tam poznačíme nulu. Ak NKA na vstupe má 4 stavy, tak minimálny kanonický DKA môže mať najviac 16 stavov, čiže bude nám stačiť 16-bitový integer.

Matice susednosti pre jednotlivé znaky tiež môžeme previesť na integery. Uvažujme maticu susednosti pre znak c takú, že na pozícii $[i, j]$ je jednotka, ak na znak c vedie prechod zo stavu i na stav j, inak je tam nula. Všimnime si, že pre každé DKA platí, že v každom riadku matice susednosti pre každý zo znakov je práve jedna jednotka, samozrejme, za predpokladu, že má úplnú δ-funkciu; v úvodnej kapitole spomínali, že uvažujeme len o takých. Ak náš minimálny kanonický DKA má maximálne 16 stavov, tak na zakódovanie jedného riadku nám stačia 4 bity aby sme si zapamätali pozíciu jednotky v riadku. Na zakódovanie matice typu 16x16 teda potrebujeme 64 bitov a tieto sa nám presne zmestia do 64-bitového integeru, ktorý máme tiež k dispozícii.

Napokon dostávame zakódovanie každého minimálneho kanonického DKA do trojice celých čísel - jedno 16-bitové a dve 64-bitové, keďže matica susednosti je pre každý znak jedna a uvažujeme binárnu abecedu. Takže spotrebujeme dovedna 144 bitov na jeden minimálny kanonický DKA, čo je oveľa úspornejšie ako pri predošlom prístupe.

Keď dokážeme každý NKA jednoznačne zakódovať do trojice integerov, tak nám stačí pamätať si len kód a keď vygenerujeme nový NKA, len ho zakódujeme a skontrolujeme, či jeho kód sa nenachádza v hashSete kódov predtým vygenerovaných automatov. Tým si ušetríme mimoriadne veľa procesorového času oproti tomu, keď sme pracne znova a znova testovali DKA na ekvivalenciu „štandardným" spôsobom. Navyše, výhoda je, že rýchlosť sa nebude v priebehu generovania počtom vygenerovaných NKA znižovať tak drasticky ako vtedy, keď sme hashovali priamo NKA.

Kapitola 4

Výsledky a ich náväznosť na predošlý výskum

V tejto kapitole zhrnieme výsledky našej práce a ich náväznosť na predchádzajúci výskum v tejto oblasti.

4.1 Hlavný výskum

Nadviažeme na výskum v článku Domaratzkého, Kismana a Shallita [DKS02], ktorí sa zaoberali okrem iného enumeráciou regulárnych jazykov akceptovanými n-stavovými NKA, z čoho, podobne ako my, dostali aj výsledky o deterministickej a nedeterministickej stavovej zložitosti týchto regulárnych jazykov. Definujme nasledovnú funkciu:

$G_k(n)$ = počet rôznych regulárnych jazykov nad práve k-znakovou abecedou akceptovaných NKA s n stavmi.

My sme skúmali predovšetkým $G_2(n)$, ale po menšej úprave programu by sa dali rátať i $G_1(n)$, prípadne $G_3(n)$. Vo všetkých prípadoch sa nám podarilo zreprodukovať výsledky z [DKS02] a navyše sa nám podarilo dorátať $G_2(4)$. Získané výsledky uvádzame v nasledujúcich tabuľkách:

n	1	2	3	4	5
$G_1(n)$	3	9	29	88	269

Tabuľka 4.1: $G_1(n)$ pre $1 \leq n \leq 5$.

$n\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	2	1																
2	2	4	3															
3	2	4	12	7	3	1												
4	2	4	12	30	16	11	8	2	1	1	1							
5	2	4	12	30	78	33	27	29	23	9	6	6	2	3	2	1	1	1

Tabuľka 4.2: Distribúcia minimálnych DKA s j stavmi pre nájdené n-stavové minimálne NKA nad unárnou abecedou.

V tabuľke 4.3 uvádzame výsledky pre $G_2(n)$, ktoré boli jedným z hlavných cieľov tejto práce.

n	1	2	3	4
$G_2(n)$	5	213	45 113	32 191 450

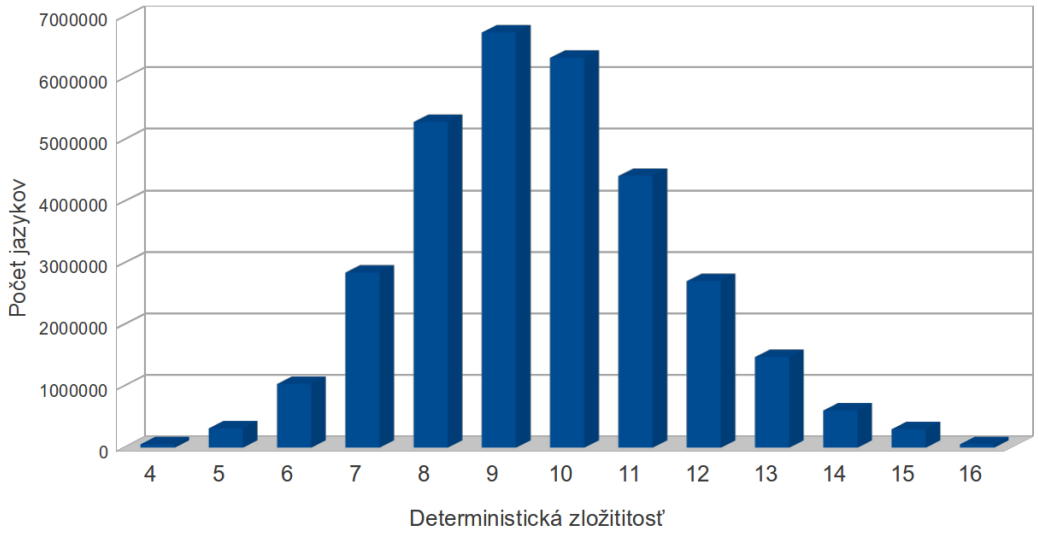
Tabuľka 4.3: $G_2(n)$ pre $1 \leq n \leq 4$.

Priemerná deterministická zložitosť jazykov s nedeterministickou zložitosťou 4 vychádza približne 9.588. Prikladáme aj distribúciu veľkosti minimálnych DKA a NKA pre nájdené jazyky (tabuľka 4.4, otočená o 90 ° oproti predošlým, aby sa zmestila).

$j\backslash n$	1	2	3	4
1	2	2	2	2
2	3	24	24	24
3		117	1 028	1 028
4		70	5 595	56 014
5			11 211	316 858
6			14 537	1 037 248
7			10 580	2 846 095
8			2 136	5 293 858
9				6 744 831
10				6 334 902
11				4 414 937
12				2 707 073
13				1 472 277
14				606 946
15				301 041
16				58 316
Σ	5	213	45 113	32 191 450

Tabuľka 4.4: Distribúcia veľkosti minimálnych DKA pre jazyky s n-stavovým minimálnym NKA nad binárnou abecedou.

Pre lepšiu predstavu výsledky z tabuľky 4.4 pre jazyky s nedeterministickou zložitosťou 4 vizualizujeme v obrázku 4.1.



Obr. 4.1: Distribúcia deterministickej zložitosti pre jazyky s nedeterministickou zložitosťou 4.

Prikladáme tiež tabuľku 4.5, kde na základe predošlých výsledkov sú k nedeterministickým zložitostiam jazykov vypočítané ich priemerné deterministické zložitosti.

nedeterministická zložitosť	1	2	3	4
priemerná det. zl. (približne)	1.6	3.2355	5.7741	9.5885

Tabuľka 4.5: Priemerné deterministické zložitosti pre jazyky s danými nedeterministickými zložitosťami.

Ešte uvedieme časy - koľko nám jednotlivé výsledky pre $G_2(n)$ trvali vyrátať na stroji, ktorý sme mali k dispozícii (Intel Core i7 3GHz, 4 jadrá, 24GB RAM).

n	1	2	3	4
čas	$<$1s	$<$1s	29s	168h 30m ($\sim$ 7 dní)

Tabuľka 4.6: Časy výpočtu $G_2(n)$ pre $1 \leq n \leq 4$.

Treba poznamenať, že nakoľko sme úlohu neparalelizovali, tak sa celý čas využíval výkon iba jedného jadra CPU.

4.2 Odhad deterministickej zložitosti jazykov s nedeterministickou zložitosťou 5

Keď už máme zistené jazyky akceptované NKA do 4 stavov, mohlo by nás zaujímať, ako je to pri NKA s 5 stavmi. Aj keby sme uvažovali optimalizácie, ktoré nám pomohli pri NKA do 4 stavov, zrejme by výpočty stále trvali neúnosne dlho. Môžeme si však vziať všetky nájdené jazyky akceptované NKA do 4 stavov a budeme veľakrát generovať náhodné 5-stavové NKA. Ak nájdeme nejaký nový jazyk, zistíme jeho deterministickú zložitosť. Takto dostaneme vzorku jazykov s nedeterministickou zložitosťou 5, pri ktorých môžeme analyzovať ich deterministickú stavovú zložitosť. Aby sme zabezpečili rovnomerný výber naprieč priestorom všetkých 5-stavových NKA, tak sme volili náhodný počiatočný stav, náhodnú množinu akceptačných stavov a dve náhodné matice susednosti reprezentujúce δ-funkciu.

Program našiel po vygenerovaní miliardy náhodných 5-stavových NKA 5 212 695 nových jazykov a ich priemerná deterministická zložitosť bola približne 9.2427. Keď sme experiment opakovali s 5 miliardami automatov, našlo sa 19 941 523 jazykov a priemerná deterministická zložitosť vyšla približne 9.6558. Výslednú distibúciu pre experiment s 5 miliardami automatov vidno v tabuľke 4.7.

Aby sme získali predstavu, nakoľko dobré sú získané výsledky, vyskúšali sme rovnakú metódu na odhad priemernej deterministickej zložitosti jazykov s nedeterministickou zložitosťou 4. Skúšali sme najprv generovať 5 000 náhodných 4-stavových NKA a dostali sme priemernú deterministickú zložitosť 6.42, pri vygenerovaní 5 miliónov náhodných 4-stavových NKA sme dostali priemernú deterministickú zložitosť 7.26. Vidno, že oproti priemernej deterministickej zložitosti, ktorú sme vyrátali exaktne (9.58), to je oveľa menej, z čoho môžeme usúdiť, že pre 5-stavové NKA bude situácia podobná. Počet pokusov, 5 000 a 5 miliónov, sme volili tak, aby to približne zodpovedalo pomeru, v akom je 5 miliárd k počtu všetkých 5-stavových NKA (cca. $1,8 \times 10^{17}$).

V priemere vyšla takmer rovnaká deterministická zložitosť, ako pri 4-stavových NKA. Za predpokladu, že sme korektne naimplementovali test by to mohlo nasvedčovať, že jazyky s nedeterministickou zložitosťou 5 s vyššou deterministickou zložitosťou majú tendenciu byť „jednoznačnejšie" v zmysle, že k ním existuje menej neizomorfných minimálnych NKA. Z toho by vyplývalo, že generovaním náhodných automatov je ťažšie trafiť automat s vysokou deterministickou zložitosťou.

Deterministická zložitosť	počet	%
5	17 040	0.326 894
6	210 762	4.043 244
7	682 295	13.089 102
8	1 176 965	22.578 819
9	1 171 386	22.471 792
10	823 659	15.801 020
11	499 051	9.573 761
12	281 152	5.393 601
13	155 863	2.990 065
14	86 740	1.664 014
15	47 259	0.906 613
16	27 164	0.521 112
17	14 671	0.281 447
18	8 114	0.155 658
19	4 551	0.087 306
20	2 494	0.047 844
21	1 445	0.027 720
22	809	0.015 519
23	466	0.008 939
24	311	0.005 966
25	131	0.002 513
26	185	0.003 549
27	57	0.001 093
28	27	0.000 517
29	14	0.000 268
30	29	0.000 556
31	51	0.000 978
32	4	0.000 076

Tabuľka 4.7: Distribúcia stavových zložitostí pre nájdené jazyky akceptované 5-stavovými NKA.

4.3 Ďalší výskum

V tejto podkapitole sa budeme zaoberať ďalšími otázkami, ktoré síce nie sú hlavným predmetom tejto práce, ale odpovede na ne sme dostali ako vedľajší produkt nášho výskumu, resp. zaujali nás v jeho priebehu.

4.3.1 Distribúcia jazykov vzhľadom na počet rôznych minimálnych NKA, ktoré ich akceptujú

Popis problému

Jeden z problémov, ktorý nás zaujal počas výskumu, bola aj distribúcia jazykov vzhľadom na počet minimálnych neizomorfných NKA, ktoré ich akceptujú. Zvlášť by nás mohli zaujímať jazyky, pre ktoré existuje jednoznačný minimálny NKA.

Realizácia

Experiment vykonáme tak, že budeme generovať zaradom všetky možné NKA. Popri tom budeme mať HashMapu, kde si budeme pamätať ku kódom jednotlivých jazykov počet vygenerovaných NKA, ktorý tento jazyk akceptujú. Po skončení generovania ešte tieto výsledky utriedime pre lepšiu analýzu.

Na rozdiel od toho, keď nás zaujímali samotné jazyky, pri zisťovaní distribúcie automatov musíme byť oveľa opatrnejší pri redukovaní počtu generovaných NKA, aby sme priestor všetkých možných NKA zmenšili rovnomerne pre všetky jazyky. Keby sme „bezhlavo" použili tie isté metódy ako pri experimente, kde sme zisťovali počet jazykov akceptovaných NKA tak by napríklad bolo ťažké predvídať, čo by presne urobila s priestorom NKA optimalizácia spojená s množinou akceptačných stavov, keďže táto by osekala viac množinu NKA, ktoré majú viacero akceptačných stavov oproti NKA s jedným, resp. menej akceptačnými stavmi a teda by sme nemuseli dostať očakávané výsledky.

Rozhodli sme sa preto povoliť len optimalizáciu s fixovaním počiatočného stavu na 0, keďže pri nej sa pre každý jazyk počet NKA, ktoré ho akceptujú, zredukuje rovnomerne. Nie je ťažké vidieť, že to bude n-násobne, kde n je počet stavov NKA, ktoré uvažujeme

Bolo by možné uvažovať aj ďalšie z predošlých optimalizácii, ktoré by osekali priestor možných NKA rovnomerne vzhľadom na jazyky. Keďže sme sa však rozhodli tento experiment vykonať len pre NKA do 3 stavov, neboli potrebné.

Výsledky

Ako sme už spomenuli, budeme uvažovať len navzájom neizomorfné NKA, t.j., nedá sa dostať jeden NKA z druhého pomocou prečíslovania stavov. Pre 1-stavové je situácia pomerne jednoduchá - všetky jazyky, okrem prázdneho, majú jednoznačný NKA, na prázdny jazyk pripadajú 4 NKA (pozn. „# NKA" je skratka pre „počet neizomorfných minimálnych NKA", to isté bude platiť v ostatných tabuľkách).

# NKA	1	4
# jazykov	4	1

Tabuľka 4.8: Distribúcia jazykov vzhľadom na minimálne NKA s 1 stavom, ktoré ich akceptujú.

S jazykmi s nedeterministickou zložitosťou 2 je to trochu rozmanitejšie. Môžeme si všimnúť, že jazykov s jednoznačným minimálnym NKA je stále veľká prevaha (181 z 208).

# NKA	1	2	4	6	10	12	14	28
# jazykov	181	6	6	4	6	2	2	1

Tabuľka 4.9: Distribúcia jazykov vzhľadom na minimálne NKA s 2 stavmi, ktoré ich akceptujú.

Pozrime sa na jazyky s nedeterministickou zložitosťou 3. Výsledky sú príliš rozsiahle na to, aby sme ich mohli uviesť v plnej miere, preto uvádzame len ich časť, aby si čitateľ mohol utvoriť predstavu. Opäť sa ukazuje, že jazyky s jednoznačným minimálnym NKA majú veľkú prevahu (29 208 z 44 900).

# NKA	1	2	3	4	5	...	1088	1096	1157	2368	6465
# jazykov	29 208	5 492	1 498	2 122	864	...	4	2	2	1	2

Tabuľka 4.10: Distribúcia jazykov vzhľadom na minimálne NKA s 3 stavmi, ktoré ich akceptujú. Kompletné výsledky nájdete v priloženom DVD.

Prikladáme ešte zhrňujúcu tabuľku, kde pre jednotlivé nedeterministické zložitosti jazykov nad binárnou abecedou je uvedené, koľko existuje takých, ktoré majú jednoznačný minimálny NKA.

nedeterministická zložitosť	1	2	3
# jazykov s jednoznačným NKA	4	181	29 208
# všetkých jazykov	5	208	44 900

Tabuľka 4.11: Počet jazykov s jednoznačným minimálnym NKA vzhľadom na ich nedeterministickú zložitosť.

Bolo by zaujímavé zistiť, koľko je jazykov s jednoznačným minimálnym NKA aj pre vyššie počty stavov, keďže z doterajších výsledkov sa zdá, že to nemusí byť nezanedbateľný počet. Mohli by sme navyše tieto jazyky oddeliť do samostatnej podtriedy regulárnych jazykov a analyzovať ich vlastnosti.

4.3.2 Dĺžka slov, ktoré jednoznačne odlíšia dva NKA

Ďalšia vec, ktorú sme skúmali, je problém rozlíšenia dvoch NKA na základe slov, ktoré akceptujú. Inými slovami zaujíma nás horná hranica dĺžky slova, ktoré rozlíši dva neekvivalentné NKA.

Popis problému

V prípade DKA sa dá horná hranica stanoviť ako $mn - 1$, kde m je počet stavov DKA M_1 a n počet stavov DKA M_2. Dané ohraničenie dostávame na základe konštrukcie DKA pre $L(M_1) \cap L^C(M_2)$, kde C značí operáciu komplementu jazyka. Toto horné ohraničenie následne môžeme preniesť aj na NKA, keďže počet stavov automatu pre $L(M_1) \cap L^C(M_2)$ môžeme zhora ohraničiť $2^{(m+n)}$ a teda dĺžku rozlišujúceho slova na $2^{(m+n)} - 1$. Pre malé automaty sa pokúsime experimentálne zistiť, aká je táto hranica v skutočnosti. Budeme ju označovať δ_n, kde n je horná hranica počtu stavov porovnávaných NKA. Ak by bola „rozumne" malá, mohol by to byť napríklad pomerne jednoduchý a rýchly test ekvivalencie dvoch NKA spočívajúci v overení všetkých slov do dĺžky δ_n.

Použitá metóda

Ako sme spomínali pri metódach hashovania, konkrétne pri hashovaní podľa slov do fixnej dĺžky (3.4), túto metódu môžeme využiť práve na vyrátanie danej hodnoty. Jediné, čo budeme robiť je, že zahashujeme každý vygenerovaný NKA týmto hashom a následne overíme pre všetky automaty s rovnakým hashom, či niektorý z nich je alebo nie je ekvivalentný tomuto NKA. Postupne budeme hashovacej funkcii navyšovať jej parameter (t.j. dĺžku, po ktorú overuje všetky slová, či ich daný NKA akceptuje) a zastavíme, keď v každom „chlieviku" bude nanajvýš jeden automat. Veľkosť tohto parametra bude hľadaná dĺžka slova, ktorá jednoznačne odlíši dva NKA.

Výsledky

Veľkosť δ_n sme skúmali pre n-stavové NKA nad dvojznakovou abecedou do 3 stavov, pre väčšie NKA sa už uvedená metóda javí byť neefektívna.

n	1	2	3
δ_n	1	4	11

Tabuľka 4.12: δ_n pre n-stavové NKA nad 2-znakovou abecedou do 3 stavov.

Ako vidno, nie je to veľmi efektívne. Hodnota δ_n pre väčšie n bude rásť zrejme dosť prudko, veď už len pre 3-stavové NKA by sme potreboval overovať $2^{11} - 1$, čiže

2047 slov, čo je síce ešte teoreticky prijateľné, ale zrejme pre väčšie NKA to budú už milióny až miliardy. To je zjavne menej efektívne ako známe algoritmy na testovanie ekvivalencie dvoch NKA.

Pozrime sa ešte, koľko kolízii vzniklo pri jednotlivých nastaveniach dĺžky slova, ktorú sme overovali – budeme ju označovať δ.

Pozrime sa najprv na NKA do 2 stavov.

δ	0	1	2	3	4
# kolízii	124	118	70	7	0

Tabuľka 4.13: Počet kolízií pre jednotlivé dĺžky slov pre NKA do 2 stavov.

Pre NKA do 3 stavov situácia vyzerá nasledovne:

δ	0	1	2	3	4	5	6	7	8	9	10	11
# kolízii	28 674	28 668	28 554	24 422	12 655	3 602	1 057	232	45	6	2	0

Tabuľka 4.14: Počet kolízii pre jednotlivé dĺžky slov pre NKA do 3 stavov.

Kapitola 5

Dokumentácia k programu

Program má čitateľ nájde v prílohe. Ide o Netbeans project v jazyku Java, ktorý stačí naimportovať a mal by ísť priamo spustiť. Užívateľ má možnosť ovplyvniť beh testov vpisovaním priamo do zdrojového kódu. Samotný program je došt komplexný a úplná dokumentácia by zabrala mnoho strán, preto sa pozrieme aspoň na tie najzákladnejšie triedy a funkcie pre lepšie pochopenie, ako to celé funguje a aby sa dali prípadne dorobiť testy podľa vlastnej potreby.

5.1 Variables.java

Táto trieda obsahuje globálne premenné nášho programu, medzi nimi je napr. abeceda (Variables.alphabet) definovaná ako pole Characterov, cesta k výstupnému súboru pre nájdené automaty (Variables.outputFileForAutomata), pre vypisovanie dvojíc "počet stavov v minimálnom NKA vs počet stavov v ekvivalentnom minimálnom DKA"(Variables.output) a podstatná je ešte hashmapa obsahujúca kódy všetkých nájdených jazykov, resp. minimálnych DKA, ktoré ich reprezentujú (Variables.allMinimalNFAs). Veľkosť abecedy, ktorou sa zaoberáme sa dá jednoducho zmeniť pridaním/odobratím znakov z Variables.alphabet. Sú tam aj ďalšie premenné, ku ktorým si čitateľ môže prečítať komentáre priamo v zdrojovom kóde.

5.2 Automaton.java

V tejto triede je definovaný konečný automat. Môžu sa doň pridávať stavy, prechody, simulovať jeho činnosť, determinizovať, minimalizovať ho, atď. K dispozícii je viacero metód, uvedieme tie najpodstatnejšie.

Automaton()

Ide o prázdny konštruktor, ktorý vráti prázdny automat, t.j. bez stavov a prechodov. K dispozícii je viacero metód, uvedieme tie najpodstatnejšie.

void addState(int stateId)

Pomocou tejto metódy môžeme do automatu pridať stav. Stav má svoj číselný identifikátor (stateId), ktorým ho dokážeme rozlíšiť od ostatných stavov. Rovnako existujú metódy addInitialState(int stateId) a addFinalState(stateId), ktoré umožňujú pridanie počiatočného, resp. akceptačného stavu. Za povšimnutie stojí, že pripúšťa sa viacero počiatočných stavov, čo síce nie je formálne korektné, ale zjednodušuje to prácu, okrem iného, pri rátaní minimálneho DKA pomocou Brzozowského algoritmu, kde sa robí reverz automatu, čím môže vzniknúť viacero počiatočných stavov.

void addTransition(int idFrom,int idTo, char c)

Týmto spôsobom môžeme pridať prechod medzi dvomi existujúcimi stavmi (z idFrom na idTo) na znak c.

Automaton determinize()

Determinizuje náš automat a vráti ho na výstup. Použije na to štandardnú konštrukciu cez podmnožiny stavov.

Automaton minimalDFA()

Vráti minimálny DKA k nášmu automatu. Použije na to Brzozowského algoritmus, ktorý sme definovali v 2.

Tuple myHashCode()

Na základe minimálneho DKA zozstrojí dvojicu čísel tak, ako sme popísali v (3.4.3). Vrátený výsledok je typu Tuple, t.j. ide o dvojicu pozostávajúcu z BigIntegeru a Integeru. Pre úsporu pamäte pri generovaní NKA len do 4 stavov sa pôvodne používali dva longy, t.j. 64-bitové integery a jeden short, t.j. 16bitový integer. Finálna verzia programu ale pre väčšiu flexibilitu a aby bolo možné rátať aj s 5-stavovými NKA, používa BigIntegery, pričom matice sa zreťazia za seba do jedného BigIntegeru - dekódovanie je potom jednoznačné, keďže rovnica $kx^2 = m$ má pre kladné k, m práve jedno kladné riešenie, čo su veľkosti našich matíc, pričom k je počet matíc, m je súčet počtov prvkov v

nich a x je hľadaná veľkosť matice. Časová zložitosť je potom daná časovou zložitosťou vyrátania minimálneho DKA k danému automatu, keďže je to časovo najnáročnejšia operácia. Prevedenie minimálneho DKA na kanonický tvar prebieha v lineárnom čase - implementované podľa 3.3.2.

HashSet<String> allWordsOfLength(int n)

Táto metóda má za úlohu nájsť a vrátiť všetky slová akceptované naším automatom. Časová zložitosť je exponenciálna od n, keďže to implementujeme ako prehľadávanie s návratom cez stavy nášho automatu. Táto metóda funguje aj pre NKA - na miestach, kde prechod nie je jednoznačne určený, sa rozvetvíme.

String toString()

Vráti string, kde je automat vypísaný vo formáte čitateľnom pre človeka.

void print(FastPrint out, long counter)

Vypíše automat pomocou vypisovača typu FastPrint (trieda vytvorená špeciálne pre vypisovanie do súboru), pričom k nemu pridá aj počitadlo, aby v súbore bolo jasne vidno, koľký automat v poradí to je. Vypisuje v nasledujúcom tvare:
/counter
počet stavov NKA
číslo počiatočného stavu
počet akceptačných stavov
akceptačné stavy (oddelené znakom nového riadku)
počet prechodov
jednotlivé prechody v tvare "stav od stav do znak", pričom každý prechod je na novom riadku

5.3 Tuple.java

Táto trieda reprezentuje dvojicu, konktrétne dvojicu pozostávajúcu z BigIntegeru reprezentujúceho zreťazenie jednotlivých matíc prechodu (každú pre jeden znak abecedy) a ešte jeden Integer, v ktorom je zakódovaná množina akceptačných stavov automatu. Predpokladá sa, že zakódovaný automat je minimálny DKA v kanonickom tvare, a teda jeho počiatočný stav je 0.

5.4 MinimalAutomatonHashMap.java

Zabezpečuje úložisko pre nájdené regulárne jazyky, resp. pre minimálne DKA, ktoré ich reprezentujú. Interne pre to používa hashMap trojíc čísel, v ktorých sú zakódované jednotlivé minimálne DKA. Uvedieme jej základné metódy:

boolean tryToInsert(Automaton a)

Pomocou tejto metódy sme schopní pridať automat. Jej cieľom je overiť, či sa v štruktúre už nenachádza ekvivalentný automat, ak áno, tak vráti false, keďže pokus vložiť automat bol neúspešný, inak tam ten automat (resp. minimálny DKA k nemu zakódovaný do Tuple) pridá a vráti true. Časová zložitosť závisí od časovej zložitosti nájdenia minimálneho DKA, čo je najnáročnejšia operácia. Zakódovanie minimálneho DKA do trojice čísel prebieha v lineárnom čase od počtu stavov minimálneho DKA a vloženie do hashMapy je potom v konštantnom čase.

void insertValue(Tuple hash)

Umožňuje pridanie Tuple priamo do štruktúry - využíva sa napr. pri načítaní kódov minimálnych DKA zo súboru. Na jej vykonanie stačí konštantný čas.

public boolean containsEquivalent(Automaton a)

Overí, či sa v štruktúre nenachádza ekvivalentný automat. Ak sa tam nachádza, vráti true, inak vráti false. Implementované to je tak, že zakódujeme automat do Tuple pomocou metódy myHashCode() a následne sa pozrieme do hashMapy kde máme uložené kódy ostatných jazykov, resp. minimálnych DKA, ktoré ich akceptujú. Časová zložitosť je teda rovnaká ako časová zložitosť vyrátania minimálneho DKA.

public long size()

Vráti veľkosť štruktúry, t.j. počet uložených kódov minimálnych DKA, ktoré akceptujú navzájom rôzne jazyky.

5.5 AutomatonIterator.java

Táto trieda slúži ako iterátor cez NKA s daným počtom stavov tak, aby bola garancia, že sa preiteruje cez všetky možné jazyky akceptované NKA do daného pcčtu stavov.

public AutomatonIterator(int n)

Konštruktor iterátora cez automaty. Vstupný parameter n určuje, koľko stavov budú mať NKA, ktoré sa budú generovať. Aby sme preiterovali postupne cez 1,2,atď - stavové NKA, treba opätovne inicializovať nový iterátor pre NKA s príslušným počtom stavov.

Automaton next()

Vráti nasledujúci automat v poradí.

Automaton random()

Vráti náhodný NKA s príslušným počtom stavov.

5.6 Experiments.java

V tejto triede sú definované experimenty s automatmi, z ktorých sme aj vyťažili výsledky v tejto práci.

static void generateAllNFAsOfSize(int limit)

Táto metóda sa postará o vygenerovanie NKA do n stavov tak, aby sme našli všetky regulárne jazyky akceptované NKA do n stavov, využijúc optimalizácie spomínané v kapitole 2. Výsledok vypíše do súboru automata.txt, výstup z programu (časový priebeh a i.) sa pošle do out.txt.

static void safeWordLengthExperiment(int n)

Slúži na zistenie dĺžky slova, ktorá jednoznačne rozlíši dva NKA do n stavov tak, ako sme si definovali v 4.3.1. Po vykonaní experimentu sa na výstup vypíše príslušná zistená dĺžka.

static void automataFileToHashes()

Prevedie súbor automata.txt s vypísanými automatmi do Variables.allMinNFAs, čo umožňuje ďalej rátať so všetkými jazykmi, resp minimálnymi DKA, ktoré ich akceptujú, ktoré sme našli pri niektorom predošlom behu nášho programu - dobré napríklad pri náhodnom generovaní NKA do 5 stavov, aby sme si načítali všetky jazyky akceptované NKA do 4 stavov.

void fiveStateNFAs(long numberOfSamples)

Tento experiment predpokladá, že máme k dispozícii súbor automata.txt obsahujúci všetky jazyky akceptované NKA do 4 stavov. Načíta ich do pamäte a generuje numberOfSamples-krát náhodné 5-stavové NKA a overuje, či akceptujú nejaký nový jazyk alebo nie. Pre každý nájdený jazyk potom spočíta jeho deterministickú zložitosť a vypíše ju do súboru 5StateNFAvsDFA"timestamp".txt. Automaty pre tieto jazyky vypíše do súboru 5StateAutomata"timestamp".txt.

automataDistributionExperiment(int n)

Cieľom tejto metódy je zistiť, koľko minimálnych nedeterministických automatov pripadá na jednotlivé jazyky s nedeterministickou zložitosťou n.

5.7 Bakalarka.java

Hlavná trieda, kam sa píše, čo sa má vykonať počas behu programu - môžeme si napr. vybrať niektorý z hore uvedených experimentov, alebo vytvoriť si vlastný.

Záver

Podarilo sa nám zistiť priemernú deterministickú zložitosť regulárnych jazykov s nedeterministickou zložitosťou 4. Ukázalo sa, že nárast počtu stavov bol v priemere len o niečo viac ako 2-násobný. Pre vzorku jazykov s nedeterministickou zložitosťou 5 vyšiel nárast dokonca menej ako 2-násobný. Na základe výsledkov získaných rovnakou metódou pre jazyky s nedeterministickou zložitosťou 4 však usudzujeme, že v priemere pre všetky jazyky s nedeterministickou zložitosťou 5 to bude zrejme väčšie číslo. Skreslenie výsledkov bolo pravdepodobne spôsobené tým, že sme generovali rovnomerne vzhľadom na priestor automatov, nie jazykov a že s vyššou pravdepodobnosťou sa generovali NKA s nižšou deterministickou zložitosťou.

Ďalším prínosom bolo, že sme zistili počet jazykov s nedeterministickou zložitosťou 4, čím sme rozšírili výsledky dosiahnuté v [DKS02].

Táto práca zároveň nastoľuje ďalšie otázky. Okrem zistenia deterministickej zložitosti jazykov s vyššími nedeterministickými zložitosťami vyvstáva otázka, aké vlastnosti by mohli mať regulárne jazyky, ktoré majú jednoznačný minimálny NKA. Podarilo sa nám určiť počet týchto jazykov do nedeterministickej zložitosti 3. Ďalej by sme sa mohli pýtať, na aké operácie sú uzavreté, koľko ich je v pomere so všetkými regulárnymi jazykmi, či existuje pre ne polynomiálny algoritmus, ktorý by našiel minimálny NKA a podobne.

Dodatok A

DVD médium

Súčasťou bakalárskej práce je aj priložené DVD médium obsahujúce program vo forme Netbeans projektu v jazyku Java, ktorý si čitateľ môže skompilovať a upravovať v ňom metódy, tak aby dostal požadované výsledky. Detaily sú uvedené v kapitole 5, kde sa nachádza stručná dokumentácia k programu.

Takisto sú na DVD médiu priložené aj získané výsledky vo forme .txt súborov. Podrobnosti nájdete v súbore readme.txt.

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.

[FR13] Michal Forišek and Branislav Rovan. Formálne jazyky a automaty (skriptá), 2013. URL: foja.dcs.fmph.uniba.sk/materialy/skripta.pdf.

[GH07] Hermann Gruber and Markus Holzer. On the average state and transition complexity of finite languages. *Theor. Comput. Sci.*, 387(2):155–166, November 2007. URL: http://dx.doi.org/10.1016/j.tcs.2007.07.035, http://dx.doi.org/10.1016/j.tcs.2007.07.035 doi:10.1016/j.tcs.2007.07.035.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Longman Publishing, second edition, 2001.

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

[JR93] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, December 1993.

[Pet09] Zuzana Petruchová. Enumerácia jednoducho popísateľných regulárnych jazykov (bakalárska práca), 2009. Comenius University, Bratislava.