Seminár k záverečnej práci

Matúš Piroh

4Ib, 2016 - 2017

Abstrakt. Cieľom tejto práce je navrhnúť a implementovať vlastné riešenie na porovnávanie regulárnych výrazov na základe prevodov na automaty, následnou determinizáciou, minimalizáciou a porovnaním dvoch minimálnych automatov. Súčasťou práce je aj preskúmanie a zhodnotenie už existujúcich knižníc a programov určených na prácu s automatmi.

Kľúčové slová: regulárny výraz, automat, determinizácia, minimalizácia, Java

1 Úvod

Z teórie formálnych jazykov a automatov je známe, že každý regulárny výraz generuje nejaký jazyk. To je nejaká množina slov. Ak vezmeme dva regulárne výrazy, ktoré hoci nie sú rovnaké, ale generujú rovnaký jazyk, hovoríme, že dané regulárne výrazy sú navzájom ekvivalentné. Pozrime sa na nasledujúcu dvojicu regulárnych výrazov:

$$a(a + b)$$
$$aa + ab$$

Obidva regulárne výrazy predstavujú jazyk $\{aa, ab\}$, teda sú navzájom ekvivalentné. Je to zrejmé aj tým, že druhý výraz dostaneme „roznásobením“ zátvorky v prvom výraze. Vezmime si ďalšie dva výrazy:

$$(a + ab)^*aba^*$$
$$a((ba)^*a^*)^*ba^*$$

Ak sa pozrieme na tieto výrazy a chceli by sme zistiť, či sú aj tieto dva ekvivalentné, vidíme, že tieto výrazy sú komplikovanejšie než predchádzajúce dva a nevieme to zistiť jednoduchým spôsobom „pozriem a vidím“. V skutočnosti aj tieto reťazce sú ekvivalentné. Ako to zistiť?
Porovnanie dvoch regulárnych výrazov nie je až taká jednoduchá záležitosť. Štandardný postup ako spraviť toto porovnanie pozostáva zo štyroch krokov:

1. Prevod daných výrazov na automaty
2. Determinizácia automatov
3. Minimalizácia automatov
4. Porovnanie dvoch minimálnych automatov

Aj keď je tento algoritmus univerzálny, nie je veľmi praktický. Jeden z problémov môže spôsobovať aj samotná determinizácia. Je známe, že ak má regulárny výraz n znakov, nedeterministický automat bude mať $2n$ stavov, no po determinizácii môže počet stavov narásť až na hodnotu 2^{2n}. Teda ak by sme zobrali aj relatívne malý regulárny výraz, povedzme 20 znakový, nedeterministický automat by obsahoval 40 stavov a tak po determinizácii by automat mohol dosiahnúť až 2^{40} stavov a určite pracovať ručne s takýmto automatom nie je jednoduchá záležitosť, práve naopak, je to veľmi nepraktické. Preto by bolo rozumné mať program, ktorý by dokázal riešiť tieto veci namiesto nás.

2 Návrh a implementácia riešenia

Definícia 2.1 *Regulárny výraz* môžeme rekurzívne definovať nasledovne:

- $\emptyset$ je regulárny výraz predstavujúci jazyk $\emptyset$.
- ε je regulárny výraz predstavujúci jazyk $\{\varepsilon\}$.
- Nech $a \in \Sigma$, potom a je regulárny výraz predstavujúci jazyk $\{a\}$.
- Nech α,β sú regulárne výrazy predstavujúce jazyky K,L. Potom $\alpha+\beta$ je regulárny výraz predstavujúci jazyk $K \cup L$.
- Nech α,β sú regulárne výrazy predstavujúce jazyky K,L. Potom $\alpha.\beta$ je regulárny výraz predstavujúci jazyk $K \,.\, L$.
- Nech α je regulárny výraz predstavujúci jazyk K. Potom α^* je regulárny výraz predstavujúci jazyk K^*.
- Nič iné nie je regulárny výraz a žiaden regulárny výraz nepredstavuje žiaden iný jazyk.

Rozdeľme si celý problém porovnania regulárnych výrazov na menšie podproblémy, ktorými sa budeme zaoberať a ktoré budúimplementované.
Hlavnými požiadavkami na náš program je samozrejme spávnosť výsledku a zároveň rýchla časová zložitosť. Ako už bolo spomenuté, tento algoritmus je veľmi nepraktický, preto budeme od nášho riešenia požadovať, aby pracoval v čo najrýchlejšom čase.
Ako programovací jazyk sme si zvolili Javu, pretože je nazávislá od platformy, patrí medzi moderné objektovo orientované jazyky a zároveň je štandardne vyučovaná na našej univerzite a máme s ňou najvačšie skúsenosti.

2.1 Prevod regulárneho výrazu na automat

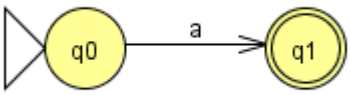
Prvou operáciou je prevod regulárneho výrazu na nedeterministický automat.

Definícia 1. *Nedeterministický automat* je pätica $(Q, \Sigma, \delta, q_0, F)$, kde Q je konečná množina stavov, Σ je konečná vstupná abeceda, $q_0 \in Q$ je počiatočný stav, $F \subseteq Q$ je

množina akceptačných (koncových) stavov a $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ je prechodová funkcia.

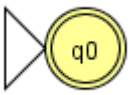
Predpokladáme, že máme na vstupe regulárny výraz *R*. Podľa Definínie 2.1 vieme, že regulárny výraz môže mať celkom šesť rôznych tvarov. Rozlíšime tak šesť rôznych prípadov:

1. $R = a$ pre nejaký symbol $a \in \Sigma$. Automat, ktorý by prijímal takýto výraz by vyzeral takto:



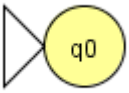
Obrázok 1. Automat prijímajúci výraz *a*.

2. $R = \varepsilon$ prázdny symbol. Automat, ktorý by prijímal prázdny symbol by vyzeral takto:



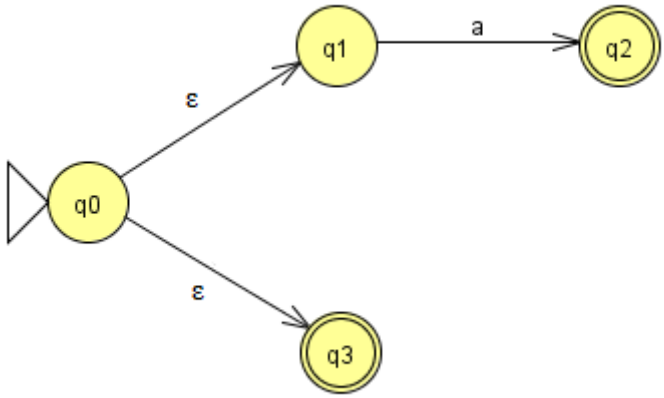
Obrázok 2. Automat prijímajúci prázdny symbol.

3. $R = \emptyset$, regulárny výraz, ktorý generuje prázdny jazyk. Automat, ktorý by prijímal prázdny jazyk, je tento:



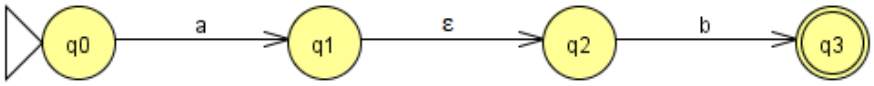
Obrázok 3. Automat prijímajúci prázdny jazyk.

4. $R = \alpha+\beta$, kde α,β sú regulárne výrazy. Tento výraz generuje zjednotenie jazykov, ktoré predstavujú výrazy α,β. Automat, ktorý by prijímal takýto jazyk, napríklad pre $\alpha = a$ a $\beta = \varepsilon$, je tento:



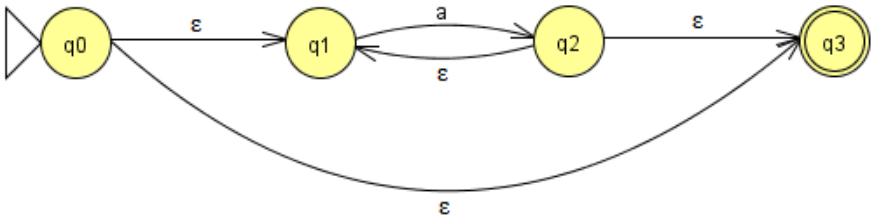
Obrázok 4. Automat prijímajúci prázdny jazyk $a + \varepsilon$.

5. $R = \alpha.\beta$, kde α,β sú regulárne výrazy. Tento výraz generuje zreťazenie jazykov, ktoré predstavujú výrazy α,β. Automat, ktorý by prijímal takýto jazyk, napríklad pre $\alpha = a$ a $\beta = b$, je tento:



Obrázok 5. Automat prijímajúci prázdny jazyk $a.b$.

6. $R = \alpha^*$, kde α je regulárny výraz. Tento výraz generuje uzáver jazyka, ktorý predstavuje výraz α. Automat, ktorý by prijímal takýto jazyk, napríklad pre $\alpha = a^*$, je tento:



Obrázok 6. Automat prijímajúci prázdny jazyk *a**.

2.1.1 Implementácia prevodu

Implementovať prevod regulárneho výrazu na nedeterministický konečný automat nie je vôbec zložité. Stačí si len dobre navrhnúť štruktúru programu.

Ako abecedu sme zvolili všetky malé písmená, teda od *a* po *z*.

Kedže niektorí používatelia majú vo zvyku písať zreťazenie bodkou (napríklad *a*.*b*) a niektorí bez bodky, preto potrebujeme najprv tieto zápisy zjednotiť. To dosiahneme tak, že jednoducho pred spustením samotného prevodu odstránime všetky bodky z výrazu.

Celý automat si neskôr budeme pamätať v samostatnej triede, ktorá obsahuje zoznam stavov a referencie na počiatočný stav a na koncové stavy. Jednotlivé stavy budú obsahovať poradové číslo a prechody, teda zoznam stavov, kam sa dostanú na jednotlivé znaky.

Postupným čítaním daného výrazu sa rozhodujeme, o akú operáciu ide. Ak sú za sebou dva znaky, vieme že ide o zreťazenie a teda vo výslednom automate stavy ukladáme za sebou, tak ako na Obrázku 5. Ak nájdeme znak +, vykonáme zjednotenie, teda v automate vzniknú dve nové vety, viď Obrázok 4. Ak sa vo výraze nachádza znak *, ide o uzáver, preto do automatu pridáme cyklus, podľa Obrázka 6.

Problémom ešte ostávajú zátvorky, ale kedže vieme, že výraz v zátvorke je tiež regulárnym výrazom, vyriešime ho osobitne a jeho počiatočný stav spojíme s koncovým stavom automatu pred zátvorkou.

Takýmto spôsobom po prečítaní celého výrazu vznikne nedeterministický automat, ktorý prijíma jazyk generovaný regulárnym výrazom, ktorý sme dostali na vstupe.

2.2 Determinizácia automatu

Definícia 2. *Deterministický automat* je pätica $(Q, \Sigma, \delta, q_0, F)$, kde Q je konečná množina stavov, Σ je konečná vstupná abeceda, $q_0 \in Q$ je počiatočný stav, $F \subseteq Q$ je množina akceptačných (koncových) stavov a $\delta : Q \times \Sigma \rightarrow Q$ je prechodová funkcia.

Veta 1. *K ľubovoľnému nedeterministickému automatu M existuje deterministický konečný automat M' taký, že L(M) = L(M').*

Pozrime sa ako previesť nedeterministický automat na deterministický. Majme nedeterministický automat $N = (Q_N, \Sigma_N, \delta_N, q_0, F_N)$. Deterministický automat $D = (Q_D, \Sigma_D, \delta_D, p_0, F_D)$ zostavíme tak, že:

- $Q_D \subseteq 2^{Q_N}$
- $\Sigma_D = \Sigma_N$
- $p_0 = q_0$
- $F_D = \{Q \in Q_D \mid Q \cap F_N \neq \emptyset\}$

Prechodovú funkciu definujeme nasledovne:

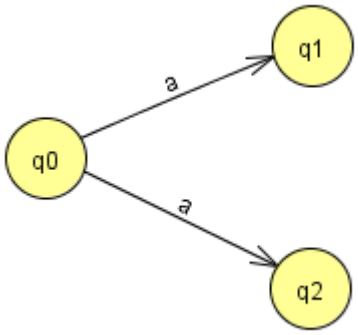
$$\forall q \in Q_D, a \in \Sigma_D : \delta_D(q,a) = \bigcup_{q \in Q_N} \delta_N(q,a)$$

To znamená, že nedeterministický automat sa môže nachádzať zárovaň vo viacerých stavoch natraz. Napríklad, môžeme sa dostať do situácie, kedy je náš nedeterministický automat v stavoch $\{q_0, q_1\}$. Prečítaním ďalšieho symbolu, napríklad a, sa môžeme dostať do množiny stavov $\{q_0, q_2, q_3\}$. V deterministickej verzii tohoto automatu sa to prejaví tak, že vytvoríme dva stavy, "pomenujeme" ich $\{q_0, q_1\}$ a $\{q_0, q_2, q_3\}$ a zavedieme medzi nimi prechod pre symbol a. Formálne sa tak dostaneme z jedného stavu do druhého, ale vďaka pomenovaniu budeme vedieť, že v nedeterministickom automate by sme sa ocitli v dvoch, respektíve troch stavoch.

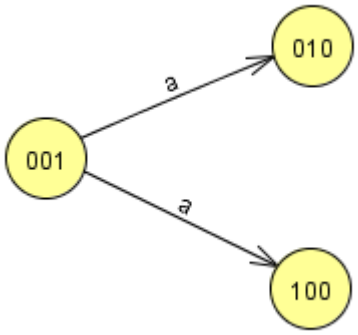
2.2.1 Implementácia determinizácie

Implementácia determinizácie je trochu zložitejšia než samotný prevod na automat. Je potrebné držať sa algoritmu popísaného vyššie. Celý princíp spočíva v tom, že začneme v počiatočnom stave, pozrieme sa na prechody a v novom automate vytvárame stavy, ktoré získame zjednotením stavov v starom automate. To opakujeme pre každý novovytvorený stav.

Ako už bolo spomenuté, potrebujeme si stavy nejako „pomenovať", aby sme vedeli z ktorých stavov v nedeterministickom automate vznikli. Na to využijeme tzv. bitkódy. To znamená, že každý stav v pôvodnom automate si označíme bitovým číslom, ktoré obsahuje práve jednu jednotku, a pozícia tejto jednotky v bitkóde nám určuje poradové číslo daného stavu. Napríklad:



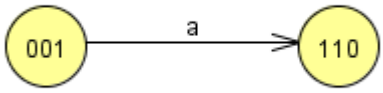
Obrázok 7. Automat pred bitovým označením



Obrázok 8. Automat po bitovom označení

Na Obrázku 7. máme daný automat. Ak označíme stavy pomocou spomínaných bitových kódov, získame automat na Obrázku 8.

Toto označenie nám značne zjednoduší proces determinizácie a to tak, že nové stavy budeme vytvárať bitovým súčtom. Použijme štandardný krok determinizácie na automat na obrázku. Vidíme, že zo stavu *001* vedú dva prechody pre symbol *a* do stavov *010* a *100*, preto v novom automate sa dostaneme zo stavu *001* na znak *a* do stavu, ktorý vznikne bitovým súčtom týchto dvoch stavov, teda *110*. Tento bitkód obsahuje jednotky na práve tých pozíciach, kde boli jednotky pôvodnych stavov, a tom nám zaručuje jednoduché ďalšie aplikovanie štandardného determinizačného kroku. Ak už nepribudne žiadny nový stav a každý stav má vytvorený prechod na každý symbol abecedy, nad ktorou pracujeme, potom je determinizácia na konci a získame výsledný deterministický automat.



Obrázok 9. Automat po determinizácii.

2.3 Minimalizácia automatu

Minimalizácia automatu sa skladá z dvoch krokov:

1. odstránenie nedosiahnuteľných stavov,
2. odstránenie ekvivalentných stavov.

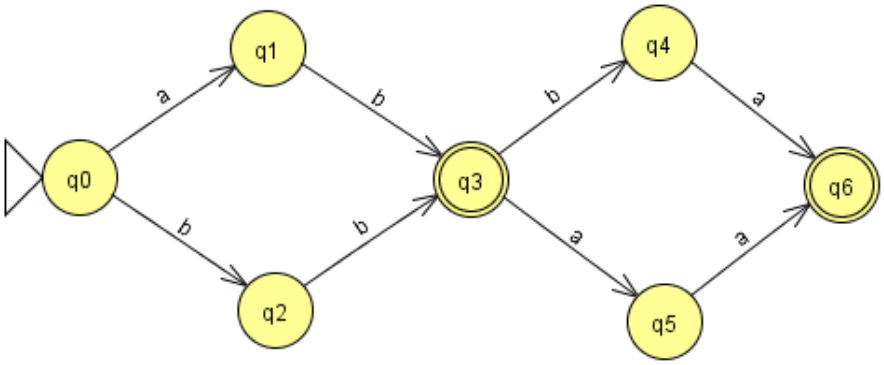
Automat, ktorý vzniká postupným prevodom z regulárneho výrazu nikdy nebude obsahovať nedosiahnuteľné stavy, pretože stavy sa vždy „lepia“ za sebou. Preto sa prvým krokom nemusíme zaoberať. Ako teda odstrániť ekvivalentné stavy?

Majme automat $M = (Q, \Sigma, \delta, q_0, F)$. Definujme jazyk stavu $q \in Q$, označíme $L(q)$ ako

$$L(q) = L((Q, \Sigma, \delta, q, F))$$

Takže zmeníme počiatočný stav z q_0 na q a spočítame jazyk prijímaný týmto automatom. Inými slovami v jazyku $L(q)$ sú všetky slová w také, že sa v automate M dostaneme zo stavu q na slovo w do koncového stavu. To znamená, že ak sú dva stavy ekvivalentné, tak ak z nich spravíme počiatočné stavy automatu, tak budú generovať rovaný jazyk. Pri minimalizácii automatu ide o to, nájsť ekvivalentné stavy a tieto stavy odstrániť tak, že ich zjednotíme do jedného nového stavu.

Vezmime si nasledujúci automat:



Obrázok 10. Automat pred minimalizáciou.

Začneme tým, že vytvoríme dve množiny stavov:

$$S_1 = F,$$
$$S_2 = Q \setminus F$$

Tieto skupiny stavov určite nebudú navzájom ekvivalentné, pretože koncové stavy prijímajú prázdne slovo ε a nekoncové stavy nie. Následne vytvoríme tabuľku, ktorá bude uchovávať informácie o skupinách S_1 a S_2:

Tabuľka 1.

skupina	stav	*a*	*b*
S_1	q_3	q_5	q_4
	q_6	–	–
S_2	q_0	q_1	q_2
	q_1	–	q_3
	q_2	–	q_3
	q_4	q_6	–
	q_5	q_6	–

Následne tabuľku upravíme tak, že pre každý stav si budeme pamätať do akej skupiny sa dostaneme pre daný symbol:

Tabuľka 2.

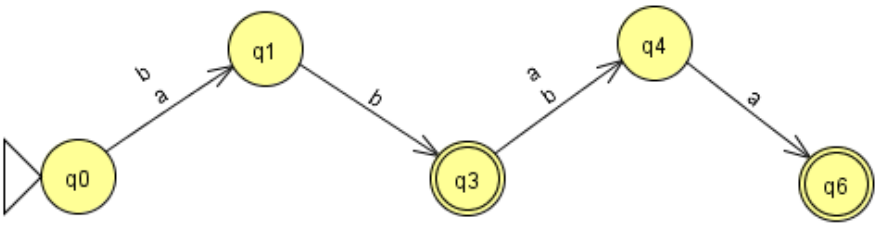
skupina	stav	*a*	*b*
S_1	q_3	S_2	S_2
	q_6	–	–
S_2	q_0	S_2	S_2
	q_1	–	S_1
	q_2	–	S_1
	q_4	S_1	–
	q_5	S_1	–

Ďalej rozdelíme skupiny. Do rovnakých skupín dáme vždy stavy, ktoré prechádzajú do rovnakých skupín pre všetky symboly. Vytvoríme tak nové skupiny:

Tabuľka 3.

skupina	stav	*a*	*b*
S_1	q_3	S_5	S_5
S_2	q_6	–	–
S_3	q_0	S_4	S_4
S_4	q_1	–	S_1
	q_2	–	S_1
S_5	q_4	S_2	–
	q_5	S_2	–

Takéto delenie opakujem, dokým nepribudnú žiadne nové skupiny. Nakoniec z každej skupiny nám stačí ybrať jeden zástupný stav. Môžeme si vybrať, že náš nový automat bude mať stavy q_3, q_6, q_0, q_1, q_4 a zvyšné stavy odstránime:



Obrázok 11. Automat po minimalizácii

2.3.1 Implementácia minimalizácie

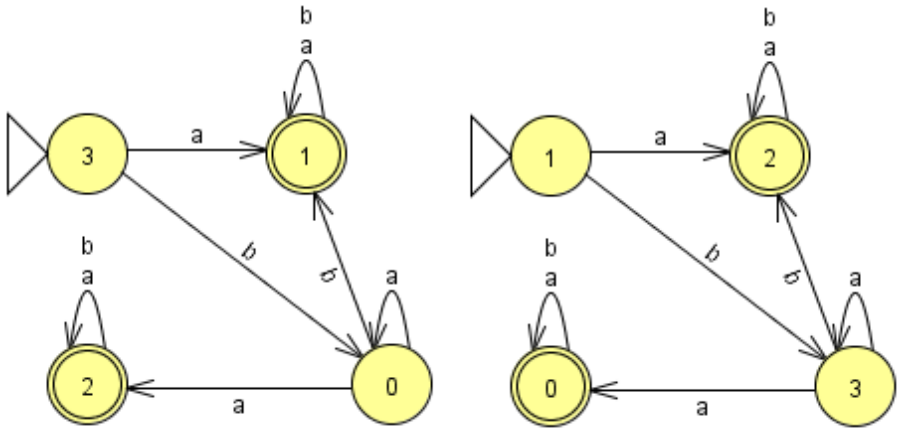
Ako už bolo spomenuté, podstatou minimalizácie je násť ekvivalentné stavy. To dosiahneme pomocou tabuľky. Danú tabuľku môžeme chápať ako mapu, kde kľuče budú skupiny a hodnoty predstavujú zoznam stavov, ktorý je dlhý ako abeceda, nad ktorou pracujeme, teda každý prvok znamená prechod na príslušný symbol. Začneme tým, že si stavy označíme podľa skupín, nie podľa poradových čísel. Postupnou iteráciou cez všetky skupiny pridávame každý stav do mapy podľa príslušnej skupiny a následne začneme mapu deliť, tak ako v Tabuľke 3. Potom nám stačí už len skontrolovať veľkosť mapy, a ak sa zmenila, znamená to, že pribudli nové skupiny. Preoznačíme všetky stavy podľa nových skupín, a opäť naplňujeme mapu. Tento cyklus opakujeme, kým veľkosť mapy neostane nezmenená.

Na koniec z výslednej mapy vytvoríme nový automat, a dodáme referencie na počiatočný a koncový stav z pôvodného automatu.

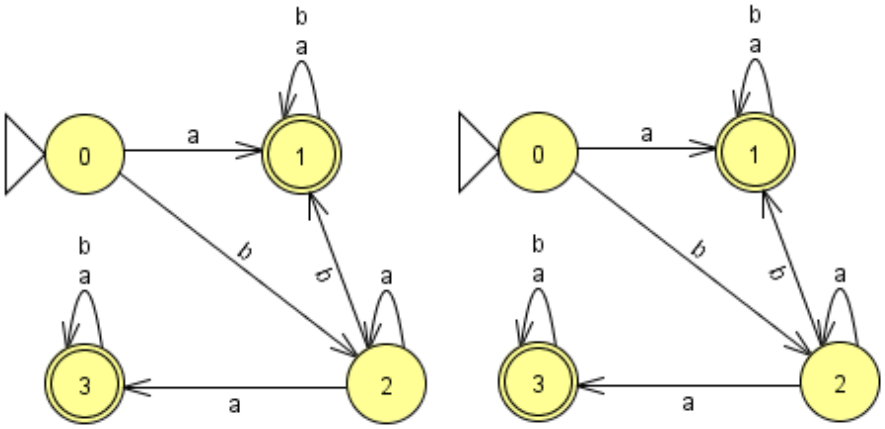
2.4 Porovnanie dvoch automatov

Posledným krokom alogitmu je porovnanie dvoch minimálnych automatov, ktoré sme získali zo zadaných regulárnych výrazov. Porovnanie automatov vykonáme tak, že oba minimálne automaty prevedieme na kánonický tvar, teda vykonáme normalizáciu. To znamená, že počiatočný stav označíme číslom *0*, stav do ktorého sa z neho dostaneme na prvý znak abecedy označíme číslom *1*, druhým symbolom *2*, atď. Takto očíslujeme celé automaty a platí, že automaty sú navzájom ekvivalentné, ak sú ich kánonické tvary rovnaké.

Napríklad, nasledujúce dva automaty sú na prvý pohľad rôzne, ale po vykonaní normalizácie vidíme, že sú navzájom ekvivalentné:



Obrázok 12. Automaty pred normalizáciou



Obrázek 13. Automaty po normalizácii

2.4.1 Implementácia porovnania

Pred samotným porovnaním je potrebné implementovať spomínaný proces normalizácie. Ten sa správa podobne ako prehľadávanie grafu. Na začiatku vynulujeme všetky označenia stavov v celom automate. Vojdeme do automatu počiatočným stavom, postupne prechádzame po hranách a zároveň si pamätáme, aké označenie sme použili naposledy. Ak sa dostaneme do stavu, v ktorom sme ešte neboli, označíme ho nasledujúcim číslom, ktoré následne inkrementujeme. Ak sú už všetky stavy označené, normalizácia je na konci.

Nakoniec je potrebné porovnať automaty a to tak, že pre každý stav v prvom automate skontrolujeme, či majú stavy, do ktorých sa dostane na každý symbol rovnaké označenia ako stavy v druhom automate. Ak áno, môžeme prehlásiť, že automaty sú ekvivalentné a tým pádom sú ekvivalentné aj zadané regulárne výrazy, z ktorých sme dostali tieto automaty.

3 Záver

Momentálne je dokončený celý algoritmus na porovnanie regulárnych výrazov, zároveň sme vytvorili pomocou knižnice Swing jednoduché používateľské prostredie. Ostáva ešte testovanie a dolaďovanie programu.

Zdroje

1. J.E. Hopcroft: Formálne jazyky a automaty (2001)
2. http://www.matematika.cz/regular-na-automat
3. https://www.cambridge.org/core/journals/rairo-theoretical-informatics-and-applications/article/conversion-of-regular-expressions-into-realtime-automata/6A5F5E28738249735739DA807628BFD6
4. www.brics.dk/automaton/