Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

.....názov.....

Diplomová práca

201. **Tatiana Tóthová**

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

.....názov.....

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	.... Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.

Bratislava, 201. **Tatiana Tóthová**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Tatiana Tóthová
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: Moderné regulárne výrazy

Cieľ: Spraviť prehľad nových konštrukcií používaných v moderných knižniciach s regulárnymi výrazmi (ako napr. look-ahead a look-behind assertions). Analyzovať tieto rozšírenia z hľadiska formálnych jazykov a prípadne tiež z hľadiska algoritmickej výpočtovej zložitosti.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 23.10.2012

Dátum schválenia: 24.10.2012

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

.....tu bude poďakovanie.....

Tatiana Tóthová

Abstrakt

Tu bude abstrakt po slovensky.

Kľúčové slová: nejaké, kľúčové, slová

Abstract

Here will be abstract in english...

Key words: some, key, words

Obsah

Úvod

Pár slov na úvod...

1 Úvod do súčasneho stavu problematiky

Myšlienka regulárnych výrazov bola prvýkrát spomenutá vo formálnych jazykoch a automatoch ako iný spôsob popisu regulárnych jazykov. Vtedy pozostávali z operácií zjednotenia, zreťazenia a Kleeneho uzáver. Pre ich jednoduchosť boli implementované ako nástroj na vyhľadávanie slov zo špecifikovaného jazyka. Postupom času a s inšpiráciou zo strany užívateľov k nim pribúdali ďalšie operácie. Niektoré boli len skratkou k tomu, čo sa už dalo zapísať - umožnili zapísať to isté menej znakmi - ostatné otvárali dvere k popisu úplne nových jazykov.

Zmes týchto operácií nazývame moderné regulárne výrazy a zaujíma nás, kam sa až dostali v popisovaní jazykov v rámci Chomského hierarchie. Túto problematiku sme z väčšej časti rozobrali v bakalárskej práci [Tó13]. V tejto práci rozbor dokončíme a pozrieme sa na ne aj z hľadiska popisnej zložitosti.

1.1 Základné definície

Formálna definícia je uvedená v [Tó13], tu si len neformálne uvedieme, s ktorými operáciami pracujeme a ako fungujú. Každý regulárny výraz sa skladá zo znakov a metaznakov. Znaky sú symboly, ktoré charakterizujú samé seba, teda $L(a) = \{a\}$. Metaznaky popisujú operáciu nad regulárnymi výrazmi. Ak potrebujeme v slove zhodu s nejakým metaznakom, stačí pred neho dať \, teda $L(\backslash x) = \{x\}$. Ak metaznak vyžaduje vstup, je ním posledný znak/metaznak/uzátvorkovaný podvýraz pred ním. Metaznaky vyzerajú nasledovne:

- () - okrúhle zátvorky slúžia na oddeľovanie podvýrazov

- {} - kučeravé zátvorky - používané ako $\{n, m\}$ (opakuj aspoň n a najviac m-krát) a $\{n\} = \{n, n\}$ (opakuj n-krát)
- [ ] - hranaté zátvorky - znaky vnútri tvoria množinu, z ktorej si vyberáme. Vieme použiť aj intervaly, napr. a-z, A-Z, 0-9, ... a kombinovať. Všetky metaznaky vnútri [ ] sa považujú za normálne znaky.
- | - operácia zjednotenia
- \ - robí z metaznakov obyčajné znaky
- . - ľubovoľný znak
- $*$ - Kleeneho uzáver, opakuj ľubovoľný počet krát
- $+$ - opakuj 1 alebo viackrát
- ? - ak samostatne: opakuj 0 alebo 1-krát, ak za operáciou: namiesto greedy implementácie použi minimalistickú, t.j. zober čo najmenej znakov (platí pre $*, +, ?, \{n, m\}$)[1]
- ^ - začiatok slova; špeciálnym prípadom je výraz $[\hat{}\alpha]$ (kde α je nejaká množina znakov), ktorý špecifikuje ľubovoľný znak, ktorý sa v množine α nenachádza
- $ - koniec slova

Okrem operácií označených metaznakmi vznikli aj zložitejšie operácie, na ktorých popis treba dlhšie konštrukcie. V prvom rade sa zaviedlo číslovanie okrúhlych zátvoriek. Čísluje sa zľava doprava podľa poradia ľavej (otváracej) zátvorky. Toto číslovanie sa deje automaticky pri každom behu algoritmu, teda už pri písaní výrazu ho môžeme využívať. Popíšme si konečne spomínané zložitejšie operácie:

- **komentár** ozn. (?#*text*) - klasický komentár, určený čitateľom kódu; nemá vplyv na výraz, algoritmus ho ignoruje
- **spätné referencie** ozn. $\backslash k$ - môže sa nachádzať na ľubovoľnom mieste vo výraze ZA k-tou pravou (zatváracou) zátvorkou. Odkazuje sa na k-te zátvorky, presný význam sa určuje až pri hľadaní zhody na konkrétnom vstupe. Algoritmus si zapamätá aké podslovo zo vstupu matchoval výraz vnútri k-tych zátvoriek a presne toto podslovo čaká, keď vo výraze vidí $\backslash k$.

[1] všetky spomenuté operácie "žerú-znaky a na ich implementáciu sa použil greedy algoritmus

- **lookahead**
  - **pozitívny** ozn. $(? = \alpha)$, kde α je nejaký moderný regulárny výraz - tzv. nazeranie dopredu pracuje tak, že si zapamätáme miesto, na ktorom lookahead začíname vykonávať, potom ho vykonáme. Ak vyhlásil zhodu, vrátime sa naspäť na zapamätané miesto a odtiaľ hľadáme zhodu akokeby tam lookahead nikdy nebol. Inak povedané lookahead nevyžiera písmenká a robí akýsi prienik. Ak neobsahuje znak pre koniec slova, môže končiť kdekoľvek - hneď ako zistil zhodu.
  - **negatívny** ozn. $(?! \ \alpha)$, kde α je nejaký moderný regulárny výraz - hľadá slovo z komplementu jazyka popísaného α popísaným spôsobom
- **lookbehind**
  - **pozitívny** ozn. $(? <= \alpha)$, kde α je nejaký moderný regulárny výraz - tzv. nazeranie dozadu, hľadá slovo z $L(\alpha)$ naľavo od aktuálneho miesta v slove (musí končiť hneď vedľa aktuálneho miesta). Opäť nie je určená hranica slova, môže začínať kdekoľvek, ak nie je vynútený začiatok slova znakom ^.

    Ak by sme chceli deterministický algoritmus, vyzeral by nasledovne: symbol v slove, na ktorom stojíme, bude teraz pre nás znak pre koniec slova - endmarker. Najprv vyskúšame symbol naľavo, či patrí do jazyka. Ak nie skúsime čítať o jeden znak viac (2 symboly naľavo), keď neuspejeme, opäť posunieme pomyselný začiatok slova doľava. Ak akceptujeme, môže to byť len na endmarkeri. Ak sme neakceptovali a začiatok slova nejde viac posunúť, zamietneme.
  - **negatívny** ozn. $(? <! \ \alpha)$, kde α je nejaký moderný regulárny výraz - hľadá slovo z komplementu $L(\alpha)$ popísaným spôsobom

Pre lookahead a lookbehind používame spoločný názov lookaround, takisto s prívlastkom pozitívny/negatívny myslíme iba ich pozitívne/negatívne verzie.

Keďže sa týmito operáciami budeme zaoberať podrobnejšie, uvedieme pre upresnenie ich definície.

Definícia 1.1.1 (Pozitívny lookahead)**.**

$$L_1(? = L_2)L_3 = \{uvw \mid u \in L_1 \wedge v \in L_2 \wedge vw \in L_3\}$$

Operáciu $(? = \dots)$ *nazývame pozitívny lookahead alebo len lookahead.*

Definícia 1.1.2 (Negatívny lookahead)**.**

$$L_1(?!L_2)L_3 = \{uv \mid u \in L_1 \wedge v \in L_3 \wedge neexistuje\ také\ x, y,\ že\ v = xy\ a\ x \in L_2\}$$

Operáciu $(?! \dots)$ *nazývame negatívny lookahead.*

Definícia 1.1.3 (Pozitívny lookbehind)**.**

$$L_1(? <= L_2)L_3 = \{uvw \mid uv \in L_1 \wedge v \in L_2 \wedge w \in L_3\}$$

Operáciu $(? <= \dots)$ *nazývame pozitívny lookbehind alebo len lookbehind.*

Definícia 1.1.4 (Negatívny lookbehind)**.**

$$L_1(? <!L_2)L_3 = \{uv \mid u \in L_1 \wedge v \in L_3 \wedge neexituje\ také\ x, y,\ že\ u = xy\ a\ y \in L_2\}$$

Operáciu $(? <! \dots)$ *nazývame negatívny lookbehind.*

Moderné regulárne výrazy sa skladajú z **konečného počtu** znakov, metaznakov a zložitejších operácií.

Máme veľkú množinu operácií a budeme chcieť pracovať aj s jej podmnožinami, preto si zavedieme názvoslovie pre ich jednoznačnejšie a kratšie určenie.

regex - množina operácií, pomocou ktorých vieme popísať iba regulárne jazyky; presnejšie všetky znaky a metaznaky (bez zložitejších operácií)

e-regex - regexy so spätnými referenciami

le-regex - e-regexy s pozitívnym lookaroundom

nle-regex - le-regexy s negatívnym lookaroundom

Eregex - trieda jazykov nad e-regexami

LEregex - trieda jazykov nad le-regexami

nLEregex - trieda jazykov nad nle-regexami

1.2 Vlastnosti a sila moderných regulárnych výrazov

TODO!!!opraviť referencie na vety z bakalárky

Potrebné vety z bakalárskej práce:

Veta 1.2.1 (Veta 2.2.5.)**.** *Regulárne jazyky sú uzavreté na lookaround.*

Veta 1.2.2 (Veta 2.2.10.)**.** *Trieda nad regexami s pozitívnym lookaroundom je* $\mathcal{R}$*.*

Veta 1.2.3 (Veta 2.2.14.)**.** $LEregex \subseteq \mathscr{L}_{CS}$

1.3 Popisná zložitosť moderných regulárnych výrazov

V čase, keď sme hľadali články týkajúce sa popisnej zložitosti moderných regulárnych výrazov, nenašli sme nič týkajúce sa danej problematiky. Známe boli len výsledky pre regulárne výrazy s operáciami zjednotenia, zreťazenia a Kleeneho $*$ (RE), prípadne ešte prieniku a komplementu (GRE). Navyše mali ešte znak pre prázdny jazyk $\emptyset$ a prázdne slovo ε.

Spomeňme si najprv ako možno zadefinovať popisnú zložitosť [EKSwW13] [EZ75]:
Nech E je regulárny výraz, potom

- $|E|$ je jeho celková dĺžka
- $rpn(E)$ je jeho celková dĺžka v poľskej normálnej forme
- $|alpha(E)| = N(E)$ je počet alfabetických symbolov v E
- $H(E)$ je hĺbka vzhľadom na $*$, počet vnorení $*$
- $L(E)$ je dĺžka najdlhšej neopakujúcej sa cesty cez výraz
- $W(E)$ je maximum symbolov v zjednotení (duálne k L)

	Alphabetical Symbol	E ∪ F	E · F	E*
N	1	N(E)+N(F)	N(E)+N(F)	N(E)
H	0	max(N(E), N(F))	max(N(E),N(F))	N(E)+1
L	1	max(N(E),N(F))	N(E)+N(F)	N(E)
W	1	N(E)+N(F)	max(N(E),N(F))	N(E)

Ako pri mnohých iných modeloch, aj do regulárnych výrazov vieme zakomponovať časti, ktoré nič nerobia (okrem toho, že zaberajú miesto). V rámci skúmania najjednoduchších výrazov sa prišlo k nasledujúcim definíciám [EKSwW13]:

Definícia 1.3.1. *Nech E je regulárny výraz nad abecedou Σ a nech L(E) je jazyk špecifikovaný výrazom E. Hovoríme, že E je* ***zmenšiteľný****, ak platí nejaká z nasledujúcich podmienok:*

1. *E obsahuje $\emptyset$ a $|E| > 1$*
2. *E obsahuje podvýraz tvaru FG alebo GF, kde L(F)=ε*
3. *E obsahuje podvýraz tvaru $F|G$ alebo $G|F$, kde $L(F) = \{\varepsilon\}$ a $\varepsilon \in L(G)$*

Inak, ak žiadna z nich neplatí, E nazývame ***nezmenšiteľným****.*

V originále sa používajú výrazy *collapsible* a *uncollapsible*. Táto definícia neodhalí všetky zbytočne zopakované časti, napríklad $a|a$ je nezmenšiteľný, aj keď by sa dal zapísať jednoduchým a. Problém je, že identity výrazov nie sú konečne axiomatizovateľné (ani nad unárnou abecedou). Teda nie je reálne určiť také pravidlá, aby sme dosiahli konečné zjednodušenie. [EKSwW13]

Definícia 1.3.2. *Ak E je nezmenšiteľný regulárny výraz taký, že*

1. *E nemá nadbytočné (); a zároveň*
2. *E neobsahuje podvýraz tvaru $F**$*

potom vravíme, že E je ***nedeliteľný*** *(irreducible).*

Môžeme vyvodiť, že minimálny regulárny výraz pre daný jazyk bude nezmenšiteľný a nedeliteľný, naopak to však nemusí platiť. S takýmto základom možno dokázať napríklad tvrdenie: Ak E je nedeliteľný a $|alph(E)| \geq 1$, potom $|E| \leq 11|alph(E)| - 4$.

Iné spôsoby na ohraničenie regulárnych výrazov poskytujú nasledujúce pozorovanie a veta.

Veta 1.3.3 (Proposition 6 [EKSwW13])**.** *Nech L je neprázdny regulárny jazyk.*

(a) Ak dĺžka najkratšieho slova v L je n, potom $|alph(E)| \geq n$ pre ľubovoľný regulárny výraz E, kde L(E) = L.

(b) Ak naviac L je konečný a dĺžka najdlhšieho slova v L je n, potom $|alph(E)| \geq n$ pre ľubovoľný regulárny výraz, kde L(E) = L.

Definícia non-returning NKA hovorí, že sa nevracia do počiatočného stavu, t.j. žiadne prechody nevedú smerom do q_0.

Veta 1.3.4 (Theorem 10)**.** *Nech E je regulárny jazyk s* $|alph(E)| = n$*. Potom existuje non-returning NKA akceptujúci* $L(E)$ $s \leq n + 1$ *stavmi a DKA akceptujúci L(E) s* $\leq 2n + 1$ *stavmi.*

Rozbehnutých je viacero oblastí skúmania. Regulárne výrazy sú zaujímavé hlavne preto, že tvoria stručnejší popis jazyka a sú ekvivalentné automatom. S tým súvisí prvá oblasť. Skúma sa stavová zložitosť automatu ekvivalentného konkrétnemu výrazu a naopak tiež popisná zložitosť výrazu ekvivalentného konkrétnemu automatu. Dá sa to zhrnúť ako problémy konverzií medzi modelmi. Niektorí autori siahajú po väčšej abstrakcii a namiesto automatov uvažujú iba orientované grafy s hranami označenými symbolmi. Určia si parametre grafu a snažia sa napríklad čo najlepšie popísať cesty medzi vrcholmi.

Ďalšia oblasť zahŕňa operácie nad regulárnymi výrazmi. Jeden z problémov je napríklad vzťah popisnej zložitosti výrazu pre jazyk L a L^c. Pre automaty existuje konštrukcia pre vyrobenie komplementu jazyka. Avšak pre komplementárny regulárny výraz to nie je také jednoznačné.

Ako poslednú by sme spomenuli problém najkratšieho slova nešpecifikovaného regulárnym výrazom. Predpokladajme regulárny výraz E, kde $|alph(E)| = n$ nad konečnou abecedou Σ, pričom $L(E) \neq \Sigma^*$. Aké dlhé môže byť najkratšie slovo NEšpecifikované výrazom E? Najzaujímavejší výsledok je pre výraz s $|alph(E)| = 75n + 361$, kde najkratšie nešpecifikované slovo je dĺžky $3(2n - 1)(n + 1) + 3$.

2 Naše výsledky

2.1 Vlastnosti a sila moderných regulárnych výrazov

V bakaláskej práci sme zabudli ... ***negatívny lookaround*** TODO!!!

Lema 2.1.1. *Trieda $\mathcal{R}$ je uzavretá na negatívny lookahead.*

Dôkaz. Nech $L_1, L_2, L_3 \in \mathcal{R}$. Ukážeme, že $L = L_1(?!L_2)L_3 \in \mathcal{R}$.

Keďže L_1, L_2, L_3 sú regulárne, existujú DKA $A_i = (K_i, \Sigma_i, \delta_i, q_{0i}, F_i)$ také, že $L(A_i) = L_i, i \in \{1, 2, 3\}$. Zostrojíme NKA A pre L.

Konštrukcia bude veľmi podobná ako pre pozitívny lookahead, keď si správne predpripravíme A_2. Negatívny lookahead sa snaží za každú cenu nájsť slovo z L_2 (t.j. uspieť s A_2) a ak sa mu to nepodarí, akceptuje. Preto musíme zaručiť, že sa A_2 buď zasekne alebo prejde až do konca slova[1]. Ak A_2 dosiahne akceptačný stav, negatívny lookahead musí zamietnuť.

Ďalšie pozorovanie nám hovorí, že negatívny lookahead akceptuje vždy nejaké slovo z komplementu L_2. Ale komplement vzhľadom na akú abecedu? V skutočnosti nekonečnú - ľubovoľný znak, ktorý nepatrí do Σ_2, znamená zaseknutie A_2, t.j. akceptáciu. Ak sa na to pozrieme z väčšej ďiaľky, uvidíme L_3, s ktorým robíme prenik. A_3 musí dočítať slovo do konca a na úplne neznámom znaku sa zasekne, takže z pohľadu výslednej akceptácie slova nevadí, ak by kvôli tomu znaku zamietol už negatívny lookahead. Preto stačí urobiť komplement vzhľadom na $\Sigma_2 \cup \Sigma_3$.

Poďme upravovať A_2, začneme vytváraním komplementu. Ak $\Sigma_3 \setminus \Sigma_2 \neq \emptyset$[2], potom vytvoríme $A_2' = (K_2', \Sigma_2', \delta_2', q_0, F_2')$ tak, že pridáme nové znaky $\Sigma_2' = \Sigma_2 \cup \Sigma_3$, jeden

[1] Ak sa zasekne, slovo z L_2 tam určite nebude, lebo neexistuje akceptačný výpočet. Ak sa nezasekne, nevieme povedať o tom slove nič, pokiaľ ho neprejdeme celé.

[2] Inak $A_2' = A_2$.

nový stav[3] $K_2' = K_2 \cup \{q_{ZLE}\}$ a prechody na nové znaky z každého stavu:

$$\begin{aligned} \forall q \in K_2\ \forall a \in \Sigma_2 &: \delta_2'(q,a) = \delta_2(q,a) \\ \forall q \in K_2\ \forall a \in \Sigma_3 \setminus \Sigma_2 &: \delta_2'(q,a) = q_{ZLE} \\ \forall a \in \Sigma_2' &: \delta_2'(q_{ZLE},a) = q_{ZLE} \end{aligned}$$

$F_2' = F_2$. Do všetkých stavov sme pridali prechody na nové znaky a q_{ZLE} má 1 prechod na každý znak, takže A_2' je stále deterministický. Zároveň nové znaky vedú do stavu, z ktorého sa nedá dostať do žiadneho akceptačného, z čoho vyplýva $L(A_2') = L_2$.

Teraz A_2' zmeníme na A_2'' tak, aby akceptoval práve vtedy, keď $(?!L_2)$. Najprv si skonštruujeme množinu stavov, z ktorých sa vieme dostať do akceptačného stavu: $H = \{q \in K_2' \mid \exists w \in {\Sigma_2'}^*\ \exists q_A \in F_2' :\ (q,w) \vdash_{A_2'} (q_A, \varepsilon)\}$, nasledovným spôsobom:

$$H_0 = F_2'$$

$$H_{i+1} = \{q \in K_2' \mid \exists p \in H_i\ \exists a \in \Sigma_2' :\ \delta(q,a) = p\}$$

Zrejme $\exists i \in \mathbb{N} :\ H_{i+1} = H_i = H$. Nás zaujíma množina $K_2' \setminus H$, v ktorej sa nachádzajú všetky stavy, z ktorých sa na žiadne slovo nie je možné dostať do žiadneho akceptačného stavu.

$A_2'' = (K_2'', \Sigma_2'', \delta_2'', q_0, F_2'')$: $K_2'' = K_2' \cup \{q_A, q_Z\},\ \Sigma_2'' = \Sigma_2',\ F_2'' = (K_2' \setminus H) \cup \{q_A\}$

$$\begin{aligned} \forall q \in K_2' \setminus F_2'\ \forall a \in \Sigma_2' &: \delta_2''(q,a) = \delta_2'(q,a) \\ \forall q \in K_2' \setminus F_2'\ \forall a \in \Sigma_2' &: \delta_2''(q,\varepsilon) = q_A \\ \forall q \in F_2'\ \forall a \in \Sigma_2' &: \delta_2''(q,a) = q^4 \\ \forall a \in \Sigma_2' &: \delta_2''(q_A,a) = q_Z \end{aligned}$$

Je dobré poznamenať, že A_2'' je takmer deterministický. Chýbajú mu prechody z q_Z - môžeme ho nechať cykliť v tomto stave, ale nevadí nám ani keď sa zasekne. Nedeterministické rozhodnutie vykonáva iba jedno - pri prechode do stavu q_A. Vtedy háda, že už je na konci slova. Ak nie je, δ-funkcia ho pošle do stavu q_Z. Podľa toho je zrejmé, že ak existuje akceptačný výpočet a dočítal slovo do konca, je práve jeden[5] [6].

[3] Pre každý nový stav predpokladáme, že je jedinečný - t.j. žiaden taký v množine stavov ešte nie je.

[4] Tento riadok v podstate netreba, A_2'' sa môže rovno zaseknúť, lebo A_2' práve akceptoval.

[5] Okrem prípadu, kedy dočíta slovo a je v stave z $(K_2' \setminus H)$ - vieme ho predĺžiť o krok na ε a skončiť v q_A. Jadro výpočtu ale zostáva rovnaké - jednoznačné.

[6] Zaujímavé je, že aj zamietací výpočet je pre každé slovo jednoznačný

Tvrdíme $L_1(? = L(A_2''))L_3 = L_1(?!\ L_2)L_3 = L$. Keďže A_1 a A_3 sa pri oboch výpočtoch budú chovať rovnako (sú nezávislé od lookaheadov), nebudeme sa nimi pri dôkaze zaoberať.

$\subseteq$: Máme akceptačný výpočet pre A_2'' na nejakom vstupe. Akceptačný stav mohol byť buď z množiny $K_2' \setminus H$, to znamená, že pôvodný automat A_2' sa dostal do stavu, z ktorého už nemohol nijak akceptovať[7]. V takom prípade $(?!\ L_2)$ akceptuje. V druhom prípade mohol A_2'' akceptovať pomocou q_A, čo znamená, že dočítal slovo do konca (pretože z q_A sa na ľubovoľný znak dostaneme do q_Z, v ktorom sa A_2'' zasekne a nebude akceptovať) a ani raz sa nedostal do akceptačného stavu automatu A_2'. Teda aj $(?!\ L_2)$ akceptuje.

$\supseteq$: Nech $(?!\ L_2)$ akceptoval, t.j. na A_2 bol vykonaný nejaký neakceptujúci výpočet (A_2 je deterministický, takže existuje práve jeden pre každé slovo). Rovnakú postupnosť stavov bude mať aj výpočet na A_2'' (automat obsahuje všetky stavy aj δ-funkciu z A_2, počiatočný stav je ten istý). Ak sa A_2 zasekol na neznámom znaku, v A_2' na ten znak pridáme prechod do stavu q_{ZLE} a v ňom zostaneme až do konca slova. Keďže q_{ZLE} nie je v A_2' akceptačný a nedá sa z neho na žiadne slovo dostať do ľubovoľného akceptačného stavu, $q_{ZLE} \in K_2' \setminus H$ a teda $q_{ZLE} \in F_2''$. Ak sa A_2 nezasekol, tak dočítal slovo do konca a v postupnosti stavov jeho výpočtu sa nenachádza žiaden z množiny F_2. V tom prípade A_2'' z posledného stavu prejde na ε do q_A (2.riadok definície δ_2'') a akceptuje.

Z práve dokázaného tvrdenia a vety 1.2.1 už vyplýva aj platnosť našej lemy. □

Lema 2.1.2. *Trieda $\mathcal{R}$ je uzavretá na negatívny lookbehind.*

Dôkaz. Nech $L_1, L_2, L_3 \in \mathcal{R}$, existujú DKA $A_i = (K_i, \Sigma_i, \delta_i, q_{0i}, F_i)$ také, že $L(A_i) = L_i, i \in \{1, 2, 3\}$.. Ukážeme, že $L = L_1(? <!\ L_2)L_3 \in \mathcal{R}$.

Nemôžme skonštruovať A_2'' také, že bude akceptovať práve vtedy, keď $(? < !L_2)$, pretože nevieme čítať doľava. Tzn. zaručiť, že miesto, kde začína A_2'' výpočet je skutočný začiatok slova. Mechanizmus lookbehindu musí byť riadený zvonku, automatom pre L. Skonštruujeme ho. $C = (K, \Sigma, \delta, q_0, F)$:

$$K = K_3 \cup \{(q, A) \mid q \in K_1 \wedge A \subseteq K_2\},\ \Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3,\ q_0 = (q_{01}, \{q_{02}\}), F = F_3\ \delta :$$

[7]Sem spadá aj prípad, keď A_2 prečítal neznámy znak a zasekol sa - A_2' zostáva až do konca slova v stave q_{ZLE}.

$$
\begin{aligned}
\forall q \in K_3\ \forall a \in \Sigma_3 \quad &: \quad \delta(q,a) \ni \delta_3(q,a) \\
\forall q \in K_1\ \forall A \subseteq K_2\ \forall a \in \Sigma_1 \cap \Sigma_2 \quad &: \quad \delta((q,A),a) \ni (p, B \cup \{q_{02}\}),\ \text{kde } \delta_1(q,a) = p, \\
&\qquad B = \{r \mid \exists s \in A :\ \delta_2(s,a) = r\} \\
\forall q \in K_1\ \forall A \subseteq K_2\ \forall a \in \Sigma_1 \setminus \Sigma_2 \quad &: \quad \delta((q,A),a) \ni (p, \{q_{02}\}),\ \text{kde } \delta_1(q,a) = p \\
\forall q \in F_1\ \forall A :\ A \subseteq K_2 \wedge A \cap F_2 = \emptyset \quad &: \quad \delta((q,A),\varepsilon) \ni q_{03}
\end{aligned}
$$

Druhý riadok δ-funkcie hovorí, že A_1 a všetky rozbehnuté výpočty A_2 prejdú do ďalšieho stavu a zároveň sa rozbehne nový výpočet A_2. Ak by sme hľadali jeden akceptačný výpočet A_2 stačilo by tipnúť si začiatok a odtiaľ ho simulovať. Lenže simulujeme negatívny lookbehind, teda ak neexistuje akceptačný výpočet, tak my akceptujeme (prejdeme na simulovanie A_3, 4. riadok). A to vieme povedať len v prípade, že sme videli všetky možné výpočty. Keďže A_2 je deterministický, pre každé slovo existuje práve jeden výpočet a zároveň sa nikdy nezasekne (na svojej abecede), teda nám stačí skontrolovať množinu stavov vtedy, keď sa C rozhodne, že A_1 končí výpočet. Ak tam je, nedostane sa k simulovaniu A_3 a tým ani k akceptačným stavom.

$L(C) = L$.

$\subseteq$: Majme akceptačný výpočet C na w. Z definície F vyplýva, že A_3 akceptoval, teda $\exists u, v$ také, že $w = uv$ a $v \in L_3$. Do q_{03} sa dá dostať len z dvojice q, A takej, že $q \in F_1$, teda $u \in L_1$, a $a \cap F_2 = \emptyset$, teda $\nexists xy$ také, že $u = xy$ a $y \in L_2$. To vyplýva z toho, že v každom znaku u začal jeden výpočet na A_2 a žiaden z nich neakceptoval. Teda negatívny lookbehind akceptoval a $w \in L$.

$\supseteq$: Nech $w \in L$, teda $\exists u, v$ také, že $w = uv,\ u \in L_1, v \in L_3$ a $\forall x, y :\ u = xy$ platí $y \notin L_2$. Pre u, v teda existujú akceptačné výpočty na A_1, A_3 a pre všetky y neakceptačné na A_2. Z toho už vieme vyskladať akceptačný výpočet na C. □

Veta 2.1.3. *Trieda $\mathcal{R}$ je uzavretá na negatívny lookaround.*

Lema 2.1.4. *Nech $L_1, L_2, L_3, L_4 \in \mathcal{R}$, $\alpha = (L_1\,(?!\ L_2)\,L_3) * L_4$. Potom $L(\alpha) \in \mathcal{R}$.*

Dôkaz. Podobne ako v dôkaze vety 2.1.1 pretransformujeme $(?!\ L_2)$ na akýsi $(? = L(A_2''))$, kde A_2'' bude akceptovať práve vtedy, keď $(?!\ L_2)$. Potom $\beta = (L_1\,(? = L(A_2''))\,L_3)*$ $L_4,\ L(\beta) = L(\alpha) \in \mathcal{R}$ podľa vety 1.2.2. □

Lema 2.1.5. *Nech $L_1, L_2, L_3, L_4 \in \mathcal{R}$, $\alpha = L_4\,(L_1\,(? <!\ L_2)\,L_3)*$. Potom $L(\alpha) \in \mathcal{R}$.*

Dôkaz. TODO!!! □

Veta 2.1.6. *Trieda nad regexami s negatívnym lookaroundom je $\mathcal{R}$.*

Dôsledok 2.1.7. *Trieda nad regexami s pozitívnym a negatívnym lookaroundom je $\mathcal{R}$.*

Veta 2.1.8. *Trieda LEregex je uzavretá na zreťazenie.*

Dôkaz. Nech sú le-regexy α, β. Chceme ukázať, že jazyk $L(\alpha)L(\beta) \in LEregex$. Intuitívne nás to vedie k riešeniu $\alpha\beta$, čo ale nemusí byť vždy správne. Problémom sú operácie lookaround, presnejšie každý lookahead v α môže zasahovať do slova z $L(\beta)$ a takisto každý lookbehind z β môže zasahovať do slova z $L(\alpha)$. Navyše, ak lookahead obsahuje \$ a lookbehind ^, potom zasahujú do slova z iného jazyka určite. V takom prípade môže le-regex $\alpha\beta$ vynechať niektoré slová z L_1L_2 a tiež pridať nejaké nevhodné slová naviac. Preto treba nájsť spôsob, ako operácie lookaroundu vhodne 'skrotiť', predpokladajme, že α má k označených zátvoriek:

$$(?= \underset{1}{(}\ \alpha\ \underset{1}{)}\ \underset{k+2}{(}\ \beta\ \underset{k+2}{)}\ \$)\alpha'\backslash k+2(?<=\ \hat{}\backslash 1\beta') \tag{2.1}$$

V α, β treba vhodne prepísať označenie zátvoriek (po poradí). α' je α prepísaný tak, že pre každý lookahead:

- bez \$ - na koniec pridáme $.*\backslash k+2\$$
- s \$ - pred \$ pridáme $\backslash k+2$

β' je β prepísaný tak, že pre každý lookbehind:

- bez ^ - na začiatok pridáme $\hat{}\backslash 1.*$
- s ^ - pred ^ pridáme $\hat{}\backslash 1$

Čo teda robí le-regex 2.1? Nech w je vstupné slovo, ktoré chceme matchovať. Lookahead na začiatku ho nejak rozdelí na w_1 a w_2, pričom $w = w_1w_2$, tak, že do $L(\alpha)$ pridelí w_1 a do $L(\beta)$ podslovo w_2. Ešte musíme overiť, či α matchuje w_1 samostatne.

Preto sme v α prepísali všetky lookaheady. V α' každý z nich musí na konci slova w matchovať w_2 (toto zabezpečí spätná referencia $\backslash k+2$ a \$) a teda matchovanie le-regexu α zostane výlučne na podslove w_1. Analogicky to platí pre lookbehindy v β'. □

Veta 2.1.9. *LEregex je neporovnateľná s $\mathscr{L}_{CF}$.*

Dôkaz. Majme jazyk $L = \{ww^R \mid w \in \{a,b\}^*\} \in \mathscr{L}_{CF}$, nech $\alpha = ([ab]*)\backslash 1$. Zrejme $L(\alpha) = L$, teda $L \in LEregex$.

Ukážeme, že jazyk $L = \{a^n b^n \mid n \in \mathbb{N}\} \notin LEregex$. Sporom, nech $L \in LEregex$.

Vieme, že $L \notin Eregex$, preto musí obsahovať nejaký lookaround. Zároveň z $L \notin \mathcal{R}$ a 1.2.2 vyplýva, že musí obsahovať aj spätné referencie.

Kam sa môžu spätné referencie odkazovať a kam ich potom môžeme umiestniť? Nech výraz, na ktorý ukazujú, vyrobí nejaké:

- a^i, potom $\backslash k$ musí byť v prvej polovičke slova (medzi a, inak by pokazil štruktúru slova), takže nevplýva na časť s b a teda sa zaobídeme bez nich.
- $a^i b^j$, potom $\backslash k$ by mohol byť len medzi b, ale tam by pokazil štruktúru slova $a^n b^n$
- b^j, potom $\backslash k$ môže byť len medzi b a je tam zbytočný z rovnakých dôvodov, aké má prípad a^i

Vidíme, že so spätnými referenciami dosiahneme rovnaký výsledok ako bez nich, čo je spor s tým, že sa vo výraze musia nachádzať (bez nich vieme urobiť len regulárny jazyk). □

Dôsledok 2.1.10. $LEregex \subsetneq \mathscr{L}_{CS}$

Veta 2.1.11. $nLEregex \subseteq \mathscr{L}_{CS}$

Dôkaz. Vieme, že $LEregex \in \mathscr{L}_{CS}$ (veta 1.2.3), teda ľubovoľný le-regex vieme simulovať pomocou LBA. Ukážeme, že ak pridáme operáciu negatívny lookahead/lookbehind, vieme to simulovať tiež.

Nech α je nle-regex. Potom A je LBA pre α, ktorý ignoruje negatívny lookaround (t.j. vyrábame LBA pre le-regex). Teraz vytvoríme LBA B pre le-regex vnútri negatívneho lookaroundu. Z nich vytvoríme LBA C pre úplný nle-regex α tak, že bude simulovať A a keď príde na rad negatívny lookaround zaznačí si, v akom stave je A a na ktorom políčku skončil. Skopíruje slovo na ďalšiu stopu a na nej simuluje od/do toho miesta B (podľa toho, či je to lookahead alebo lookbehind).

Teraz je to s akceptáciou náročnejšie ako pri pozitívnom lookarounde. Pokiaľ B akceptoval, C sa zasekne. Ak sa B zasekol, C sa vráti naspäť k zastavenému výpočtu A

a pokračuje v ňom. Keďže slovo je konečné a B na ňom testuje le-regex, ktorý postupne vyjedá písmenká, určite raz príde na koniec slova. Môže sa stať, že bude skúšať viaceré možnosti - napr. skúsi pre $*$ zobrať menej znakov, teda príde na koniec slova viackrát. Ako určíme, že skončil výpočet? Bez újmy na všeobecnosti môžeme predpokladať, že sa B nezacyklí, ale zamietne slovo na konci výpočtu. To preto, že všetkých možností na rozdelenie znakov medzi operácie $*, +, ?, \{n, m\}$ je konečne veľa a keď ich systematicky skúša[8], raz sa mu musia minúť. To znamená, že ak nemá v δ-funkcii umelo vsunuté zacyklenie, nezacyklí sa. Teda ak slovo neakceptuje, tak ho určite zamietne a vtedy C môže prejsť na zastavený výpočet A a pokračovať v ňom.

Podobne ako pri lookarounde aj jeho negatívna verzia môže obsahovať nle-regex, t.j. vnorený (negatívny) lookaround. Tých však môže byť iba konečne veľa, keďže každý nle-regex musí mať konečný zápis. Čo znamená, že aj stôp bude konečne veľa a naznačeným postupom si vieme postupne vybudovať LBA, ktorý bude simulovať α. □

Veta 2.1.12. *$nLEregex$ je neporovnateľná s $\mathscr{L}_{CF}$.*

Dôkaz. Opäť použijeme jazyk $L = \{a^n b^n \mid n \in \mathbb{N}\}$ a budeme dokazovať sporom. Nech $L \in LEregex$.

Z vety **??** vieme, že výraz musí obsahovať negatívny lookaround. Z jej dôkazu vidíme, že spätné referencie nepomôžu nech sa ich pokúsime hocijako použiť. Z toho vyplýva, že ich nepoužijeme a z vety 2.1.6 zistíme, že nám zostal model schopný vyrobiť iba regulárne jazyky. Spor. □

Dôsledok 2.1.13. $nLEregex \subsetneq \mathscr{L}_{CS}$

2.2 Popisná zložitosť moderných regulárnych výrazov

V tejto kapitole rozoberieme moderné regulárne výrazy z dvoch hľadísk. Najprv nás bude zaujímať, ako pomohli nové konštrukcie pri popise regulárnych jazykov a potom prejdeme na analýzu dĺžky výrazov pre zložitejšie jazyky.

[8] Algoritmus pre $*$ je v praxi greedy. Ak slovo nesedí, tak sa vráti, odoberie posledný znak zožratý $*$ a opäť skúša zvyšok výrazu, či slovo sedí. Ak stále nie, algoritmus odoberá hviezdičke znaky dovtedy, dokým nenájde zhodu alebo odoberie všetky znaky - vtedy sa mu minuli všetky možnosti a môže slovo neakceptovať.

Lookaround môže výrazne pomôcť pri definovaní konečných jazykov, napríklad le-regex z [Tó13, Poznámka 1.] $\beta = ((? = (a^m) * \$)a^{m+1}) * a\{1, m-1\}\$$. Aké slová obsahuje?

- $a^{m+1+(m-1)}$
- $a^{2m+2+(m-2)}$
  $\vdots$
- $a^{(m-1)(m+1)+1}$

avšak $a^{m(m+1)} \notin L(\beta)$, lebo nám chýba zvyšok 0. Zaujímavé je, že le-regex využíva iteráciu $(m-1)$–krát a napriek tomu je konečný.

Záver

Zhrnutie na záver...

Literatúra

[EKSwW13] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics u (v) w, x–y*, 2013.

[EZ75] Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. 1975.

[Tó13] Tatiana Tóthová. Moderné regulárne výrazy. Bachelor's thesis, 2013. `https://github.com/Tatianka/bak/blob/master/tothova-bc.pdf?raw=true` [Online; accessed 22-Nov-2013].