Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Moderné regulárne výrazy

Bakalárska práca

2013
Tatiana Tóthová

UNIVERZITA KOMENSKÉHO, BRATISLAVA
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MODERNÉ REGULÁRNE VÝRAZY

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.

Bratislava, 2013
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

Poďakovanie

Tatiana Tóthová

Abstrakt

Abstrakt po slovensky

Kľúčové slová: napíšme, nejaké, kľúčové, slová

Abstract

Abstract in english

Key words: some, key, words

Obsah

Úvod

Bla bla úvodné info o regulárnych výrazoch: - vzniklo ako teória - niekto implementoval do textových editorov, neskôr Thomson do Unixu - odtiaľ sa rozšírili do programovacích jazykov - programovacie jazyky rozširujú svoju funkcionalitu, to platí aj pre časť regulárnych výrazov. Rozšírením o nové konštrukcie už nemusia patriť do triedy regulárnych jazykov. Mnohé konštrukcie sú len kozmetické úpravy a pomôcky, ktoré nezosilnia daný model. Zaujímavé sú tie, ktoré už pomôžu vytvoriť (akceptovať) jazyky z vyšších tried Chomského hierarchie. - Zaradením triedy jazykov aktuálneho modelu do Chomského hierarchie môže dopomôcť pri implementácii jednotlivých operácií. Trieda regulárnych jazykov vystačí s ľahko naprogramovateľnými konečnými automatmi, avšak vyššie triedy vyžadujú backtracking, ktorý samozrejme znamená väčšiu časovú zložitosť.

0.1 Základná forma regulárnych výrazov

Keďže implementované regulárne výrazy sa už natoľko líšia od počiatočného teoretického modelu, zaužíval sa pre ne názov **regexy**. Budem ho používať aj ja a prípadnými predponami budem rozlišovať, ktorú množinu operácií práve myslím.

Pojem **regex** bude slúžiť na pomenovanie regulárnych výrazov, ktoré pokrývajú triedu regulárnych jazykov. Pre ozrejmenie uvediem základnú definíciu regexu z článku [CSY03]. Niektoré konštrukcie sú oproti teoretickému modelu nové, ale dôkaz toho, že pokrýva stále rovnakú triedu jazykov, je triviálny.

Základná forma regexov

(1) Pre každé $a \in \Sigma$, a je regex a $L(a) = \{a\}$. Poznamenajme, že pre každé $x \in \{(,),\{,\},[,],\$,|,\backslash,.,?,*,+\}$, $\backslash x \in \Sigma$ a je regexom a $L(\backslash x) = \{x\}$. Naviac aj $\backslash n$ a $\backslash t$ patria do Σ a oba sú regexami. $L(\backslash n)$ a $L(\backslash t)$ popisujú jazyky skladajúce sa z nového riadku a tabulátora.

(2) Pre regexy e_1 a e_2

$(e_1)(e_2)$ (zreťazenie),

$(e_1)|(e_2)$ (alternácia), a

$(e_1)*$ (Kleeneho uzáver)

sú regexy, kde $L((e_1)(e_2)) = L(e_1)L(e_2)$, $L((e_1)|(e_2)) = L(e_1) \cup L(e_2)$ a $L((e_1)*) = (L(e_1))^*$. Okrúhle zátvorky môžu byť vynechané. Ak sú vynechané, alternácia, zreťazenie a Kleeneho uzáver majú vyššiu prioritu.

(3) Regex je tvorený konečným počtom prvkov z (1) a (2).

<u>Skrátnená forma</u>

(1) Pre každý regex e: $(e)+$ je regex a $(e)+ \equiv e(e)*$.

(2) Znak ' . ' znamená ľubovolný znak okrem $\backslash n$.

<u>Triedy znakov</u>

(1) Pre $a_{i_1}, a_{i_2}, \ldots, a_{i_t} \in \Sigma,\ t \geq 1,\ [a_{i_1} a_{i_2} \ldots a_{i_t}] \equiv a_{i_1} | a_{i_2} | \ldots | a_{i_t}$.

(2) Pre $a_i, a_j \in \Sigma$ také, že $a_i \leq a_j,\ [a_i - a_j]$ je regex a $[a_i - a_j] \equiv a_i | a_{i+1} | \ldots | a_j$.

(3) Pre $a_{i_1}, a_{i_2}, \ldots, a_{i_t} \in \Sigma,\ t \geq 1,\ [\hat{}\, a_{i_1} a_{i_2} \ldots a_{i_t}] \equiv b_{i_1} | b_{i_2} | \ldots | b_{i_s}$, kde $\{b_{i_1} | b_{i_2} | \ldots | b_{i_s}\} = \Sigma - \{a_{i_1}, a_{i_2}, \ldots, a_{i_t}\}$.

(4) Pre $a_i, a_j \in \Sigma$ také, že $a_i \leq a_j,\ [a_i - a_j]$ je regex a $[\hat{}\, a_i - a_j] \equiv b_{i_1} | b_{i_2} | \ldots | b_{i_s}$, kde $\{b_{i_1} | b_{i_2} | \ldots | b_{i_s}\} = \Sigma - \{a_i | a_{i+1} | \ldots | a_j\}$.

(5) Zmes (1) a (2) alebo (3) a (4).

<u>Ukotvenie</u>

(1) Znak pre začiatok riadku ˆ.

(2) Znak pre koniec riadku $.

Kapitola 1

Názov kapitoly 1

V tejto kapitole formálne definujem operácie z uvedenej dokumentácie jazyka Python [doc12] a ukážem ich silu. Budem používať nasledovné zápisy:

L_1L_2 – zreťazenie jazykov L_1 a L_2

L^* – iterácia ($L^* = \cup_{i=0}^{\infty} L^i$, kde $L^0 = \{\varepsilon\}$, $L^1 = L$ a $L^{i+1} = L^i L$)

$\mathcal{R}$ – trieda regulárnych jazykov

$\mathscr{L}_{CF}$ – trieda bezkontextových jazykov

$\mathscr{L}_{CS}$ – trieda bezkontextových jazykov

DKA/NKA – deterministický/nedeterministický konečný automat

LBA – lineárne ohraničený Turingov stroj

regex – regulárny výraz, ktorý môže vytvoriť najviac regulárny jazyk (základná definícia)

e-regex – regex so spätnými referenciami

le-regex – e-regex s operáciami lookahead a lookbehind

Eregex – trieda jazykov tvorená e-regexami

LEregex – trieda jazykov tvorená le-regexami

1.1 Lookahead, lookbehind

Definícia 1.1.1 (Greedy iterácia)**.**

$$L_1 \circledast L_2 = \{uv \mid u \in L_1^* \wedge v \in L_2 \wedge u \textit{ je najdlhšie také}\}$$

Definícia 1.1.2 (Minimalistická iterácia)**.**

$$L_1 {*?} L_2 = \{uv \mid u \in L_1^* \wedge v \in L_2 \wedge u \textit{ je najkratšie také}\}$$

Veta 1.1.3. $L_1 \circledast L_2 = L_1 {*?} L_2 = L_1^* L_2$

Dôkaz. $\subseteq$: Nech $w \in L_1 \circledast L_2$. Potom z definície $w = uv$ vieme, že $u \in L_1^*$ a $v \in L_2$, teda $uv \in L_1^* L_2$. Analogicky ak $x = yz \in L_1 *? L_2$, potom $yz \in L_1^* L_2$.

$\supseteq$: Majme $w \in L_1^* L_2$ a rozdeľme na podslová u, v tak, že $u \in L_1^*, v \in L_2$ a $w = uv$. Takéto rozdelenie musí byť aspoň jedno. Ak je ich viac, vezmime to, kde je u najdlhšie. Potom $uv \in L_1 \circledast L_2$. Ak zvolíme u najkratšie, tak zasa $uv \in L_1 *? L_2$. □

Dôsledok 1.1.4. *Trieda $\mathcal{R}$ je uzavretá na operácie $\circledast$ a $*?$.*

Definícia 1.1.5 (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 1.1.6 (Negatívny lookahead)**.**

$$L_1(?!L_2)L_3 = \{uv \mid u \in L_1 \wedge v \in L_3 \wedge neexistuje\ tak\acute{e}\ x, y,\ \check{z}e\ v = xy\ a\ x \in L_2\}$$

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

Definícia 1.1.7 (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 1.1.8 (Negatívny lookbehind)**.**

$$L_1(? <!L_2)L_3 = \{uv \mid u \in L_1 \wedge v \in L_3 \wedge neexituje\ tak\acute{e}\ x, y,\ \check{z}e\ u = xy\ a\ y \in L_2\}$$

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

Veta 1.1.9. *Nech $L_1, L_2, L_3 \in \mathcal{R}$. Potom $L = L_1(? = L_2)L_3 \in \mathcal{R}$.*

Dôkaz. Keďže L_1, L_2, L_3 sú regulárne, existujú $A_i = (K_i, \Sigma_i, \delta_i, q_{0i}, F_i)$ DKA také, že $L(A_i) = L_i, i \in \{1, 2, 3\}$. Zostrojím NKA $A = (K, \Sigma, \delta, q_0, F)$ pre L, kde $K = K_1 \cup K_2 \times K_3 \cup K_3$ (predp. $K_1 \cap K_3 = \emptyset$), $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, $q_0 = q_{01}$, $F = F_3 \cup F_2 \times F_3$, δ funkciu definujeme nasledovne:

$$\begin{aligned}
\forall q \in K_1, \forall a \in \Sigma &\ :\ \delta(q, a) \ni \delta_1(q, a) \\
\forall q \in F_1 &\ :\ \delta(q, \varepsilon) \ni [q_{02}, q_{03}] \\
\forall p \in K_2, \forall q \in K_3, \forall a \in \Sigma_2 \cap \Sigma_3 &\ :\ \delta([p, q], a) \ni [\delta(p, a), \delta(q, a)] \\
\forall p \in F_2, \forall q \in K_3 &\ :\ \delta([p, q], a) \ni \delta(q, a)
\end{aligned}$$

$L(A) = L$.

$\supseteq$: Máme $w \in L$ a chceme preň nájsť výpočet na A. Z definície L vyplýva $w = xyz$, kde $x \in L_1, y \in L_2$ a $yz \in L_3$, teda existujú akceptačné výpočty pre x, y, yz na DKA A_1, A_2, A_3. Z toho vyskladáme výpočet pre w na A nasledovne. Výpočet pre x bude rovnaký ako na A_1. Z akceptačné stavu A_1 vieme na ε prejsť do stavu $[q_{02}, q_{03}]$, kde začne výpočet pre y. Ten vyskladáme z A_2 a A_3 tak, že si ich výpočty napíšeme pod seba a stavy nad sebou budú tvoriť karteziánsky súčin stavov v A (keďže A_2 aj A_3 sú deterministické, tieto výpočty na y budú rovnako dlhé). $y \in L_2$, teda A_2 skončí v akceptačnom stave. Podľa δ funkcie v A vieme pokračovať len vo výpočte na A_3, teda doplníme zvyšnú postupnosť stavov pre výpočet z. Keďže $yz \in L_3$ a $F_3 \subseteq F$ (resp. $F_2 \times F_3 \subseteq F$ pre $z = \varepsilon$), automat A akceptuje.

$\subseteq$: Nech $w \in L(A)$, potom existuje akceptačný výpočet na A. Z toho vieme w rozdeliť na x, y a z tak, že x je slovo spracovávené od začiatku po prvý príchod do stavu $[q_{02}, q_{03}]$, y odtiaľto po posledný stav reprezentovaný karteziánskym súčinom stavov a zvyšok bude z. Nevynechali sme žiadne znaky a nezmenili poradie, teda $w = xyz$. Do $[q_{02}, q_{03}]$ sa A môže prvýkrát dostať len vtedy, ak bol v akceptačnom stave A_1. Prechod do $[q_{02}, q_{03}]$ je na ε, takže $x \in L_1$. Práve tento stav je počiatočný pre A_2 aj A_3. Ak $z = \varepsilon$, tak akceptačný stav A je z $F_2 \times F_3$ a $y \in L_2, y \in L_3$ a aj $yz \in L_3$. Z toho podľa definície vyplýva, že $xyz = w \in L$. Ak $z \neq \varepsilon$, potom je akceptačný stav A z F_3. Podľa δ funkcie sa z karteziánskeho súčinu stavov do normálneho stavu dá prejsť len tak, že A_2 akceptuje, teda $y \in L_2$. A_3 akceptuje na konci, čo znamená $yz \in L_3$. Znova podľa definície operácie lookahead $xyz = w \in L$. □

Veta 1.1.10. *Nech $L_1, L_2, L_3 \in \mathcal{R}$. Potom $L = L_1(? <= L_2)L_3 \in \mathcal{R}$.*

Dôkaz. Podobne ako pri lookahead. (Karteziánsky súčin stavov L_1 a L_2, ale A_2 sa pripája v každom stave A_1 - celkový NKA si potom nedeterministicky zvolí jeden moment tohto napojenia.) □

Veta 1.1.11. *$\mathscr{L}_{CF}$ nie je uzavretá na operácie lookahead a lookbehind.*

Dôkaz. Nech $L_1, L_2, L_3, L_4 \in \mathscr{L}_{CF}$. $L_1 = \{a^n b^n \mid n \geq 1\}, L_2 = \{a * b^n c^n \mid n \geq 1\}, L_3 = \{a^n b^n c * \mid n \geq 1\}, L_4 = \{ab^n c^n \mid n \geq 1\}$. Potom $d(? = L_1)L_2 = \{da^n b^n c^n \mid n \geq 1\}$ a $L_3(? <= L_4)d = \{a^n b^n c^n d \mid n \geq 1\}$, čo nie sú bezkontextové jazyky. □

Veta 1.1.12. *$\mathscr{L}_{CS}$ je uzavretá na operáciu lookahead a lookbehind.*

Dôkaz. Uzavretosť na lookahead:

Pre $L_1, L_2, L_3 \in \mathscr{L}_{CS}$ a slovo z $L = L_1(? = L_2)L_3$ zostrojíme LBA A z LBA A_1, A_2, A_3 pre dané kontextové jazyky. Najprv sa pozrime na štruktúru vstupu – prvé je slovo z L_1 a za ním nasleduje slovo z L_3, pričom jeho prefix patrí do L_2. Preto, aby A mohol simulovať dané lineárne ohraničené automaty, je potrebné označiť hranice jednotlivých slov.

Na začiatku výpočtu A prejde pásku a nedeterministicky označí 2 miesta – koniec slov pre A_1 a A_2. Následne sa vráti na začiatok a simuluje A_1. Ak akceptuje, A pokračuje a presunie sa za označený koniec vstupu pre A_1. Inak sa zasekne. V tomto bode sa začína vstup pre A_2 aj A_3, teda slovo až do konca prepíše na 2 stopy. Najprv na hornej simuluje A_2. Pokiaľ A_2 neskončí v akceptačnom stave, A sa zasekne. Inak sa vráti na označené miesto a simuluje A_3 na spodnej stope až do konca vstupu. Akceptačný stav A_3 znamená akceptáciu celého vstupného slova.

Uzavretosť na lookbehind sa dokáže analogicky. □

Teraz ukážem, ako lookahead a lookbehind nezosilnia model regexov.

Lema 1.1.13. *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. Keďže $L_1, L_2, L_3, L_4 \in \mathcal{R}$, tak pre ne existujú DKA A_1, A_2, A_3, A_4, kde $A_i = (K_i, \Sigma_i, \delta_i, q_{0i}, F_i)$ pre $\forall i$. Z nich zostrojíme NKA A pre L. Výpočet bez lookaheadov by vyzeral tak, že by sme simulovali A_1, potom po jeho akceptácii A_3 a odtiaľ by sa išlo v rámci iterácie naspäť na A_1. Zároveň z A_1 by sa dalo na ε prejsť na A_4, čo by znamenalo koniec iterácie (ošetruje aj nulovú iteráciu). Pozrime sa na to, ako a kam vsunúť lookahead. Problém je, že pri každej ďalšej iterácii pribúda nový, teda ďalší A_2. Vieme ich však simulovať všetky naraz, keď vezmeme do úvahy, že vždy pracujeme nad konečnou abecedou a K_2 je konečná. Z toho vyplýva, že aj $\mathcal{P}(K_2)$ je konečná.

Konštrukcia: $A = (K, \Sigma, \delta, q_0, F) : K = (K_1 \cup K_3 \cup K_4) \times \mathcal{P}(K_2)$, kde $K_1 \cap K_3 \cap K_4 = \emptyset$ (množiny v stavoch možno reprezentovať napr. 0-1 reťazcom dĺžky $|K_2|$, kde 1 na i–tom mieste symbolizuje, že nejaká inštancia A_2 je v i–tom stave), $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4$, $q_0 = (q_{01}, \emptyset)$, $F = F_4 \times \emptyset$, δ–funkcia:

- $\forall q \in K_i\ i = 1, 3, 4, \forall U \in \mathcal{P}(K_2), \forall a \in \Sigma : \delta((q_i, U), a) \ni (\delta_i(q_i, a), V)$, kde $\forall q \in U\ \delta_2(q, a) \in V'$ a $V = V' \setminus F_2$
- $\forall q_A \in F_1, \forall U \in \mathcal{P}(K_2) : \delta((q_A, U), \varepsilon) \ni (q_{03}, U)$
- $\forall q_A \in F_3, \forall U \in \mathcal{P}(K_2) : \delta((q_A, U), \varepsilon) \ni (q_{01}, U)$
- $\forall U \in \mathcal{P}(K_2) : \delta((q_{01}, U), \varepsilon) \ni (q_{04}, U)$

Automat A akceptuje až keď akceptuje A_4. Je zrejmé, že ak v simulácii A_i príde písmenko, ktoré do Σ_i nepatrí, automat sa zasekne.

$L(A) = L(\alpha)$.

$\subseteq$: Nech $w \in L(A)$, potom preň existuje akceptačný výpočet na A. Podľa stavov vieme určiť počet iterácií, časti z L_1, L_3, L_4 a takisto vznikajúce a akceptujúce výpočty na A_2 – každý takýto výpočet totiž začína s výpočtom na A_3 a keďže A_2 je deterministický, existuje práve jeden výpočet, ktorý musí byť akceptačný. Teda vieme povedať, že $w = x_1y_1x_2y_2 \ldots x_ny_nz$, kde n je počet iterácií, $\forall i = 1, 2, \ldots, n :\ x_i \in L_1, y_i \in L_3$ a $z \in L_4$. Zároveň vieme, že v mieste, kde začína y_i takisto začína podreťazec slova w, ktorý patrí do L_2. Z toho vidíme ako vyzerá zhoda regexu α.

$\supseteq$: Majme $v \in L(\alpha)$, teda vieme nájsť zhody podslov v pre všetky L_i (rovnaká dekompozícia slova ako v predošlej inklúzii). Keďže poradie jazykov je rovnaké ako ε–ové napojenie stavov v A (akceptačný–počiatočný, akceptačný–akceptačný pri L_4), vieme správne poprepájať akceptačné výpočty jednotlivých A_i do celkového výpočtu automatu A. □

Veta 1.1.14. *Regexy sú uzavreté na lookahead.*

Dôkaz. Regexy pokrývajú triedu regulárnych jazykov a tá je na lookahead uzavretá 1.1.9. Keďže pracujeme s množinou operácií, treba overiť, či nejaká ich kombinácia nie je náhodou silnejšia. Ak regex umiestnime do lookaheadu, či pred alebo za neho, vždy to bude regulárny jazyk a celý regex bude tiež definovať regulárny jazyk. Teda nás zaujíma vloženie lookaheadu dovnútra vykonávanej operácie. V tomto prípade prichádza do úvahy $*, +$ a ?.

? veľa nespraví – lookahead tam buď bude 1x alebo nebude vôbec. + je prípad $*$ s jedným lookaheadom istým. No a podľa lemy 1.1.13 vieme, že ani táto kombinácia nestačí na zložitejší jazyk. □

Nejaké vlastnosti lookaheadu. Tu ukážem, že dávať do lookaheadu prefixový jazyk nemá zmysel. Vytvorme čisto jazyk všetkých rôznych prefixov, aké obsahuje. Do lookaheadu stačí vložiť regex pre tento jazyk a celkový akceptovaný jazyk zostane rovnaký(zhodovať sa s...? TODO!!!). Samozrejme, to isté platí aj pre lookbehind a sufixové jazyky.

Veta 1.1.15. *Nech L je ľubovoľný jazyk a $L_p = L \cup \{uv \mid u \in L\}$. Nech α je ľubovoľný regulárny výraz taký, že obsahuje $(? = L_p)$. Potom ak prepíšeme tento lookahead na $(? = L)$ (nazvime to α'), bude platiť $L(\alpha') = L(\alpha)$.*

Dôkaz. $\subseteq$: triviálne.

$\supseteq$: Majme $w \in L(\alpha)$ a nech x je také podslovo w, ktoré sa zhodovalo práve s daným lookaheadom. Potom $x \in L^p$, teda $x = uv$, kde $u \in L$. Ak $v = \varepsilon$, $x \in L$ a máme čo sme chceli. Takže $v \neq \varepsilon$. Ale celá zhoda lookaheadu sa môže zúžiť len na u, keďže $u \in L_p$, a bude to platná zhoda s w. Čo znamená, že $w \in L(\alpha')$. □

Veta 1.1.16. *Nech α je regulárny výraz, ktorý obsahuje nejaký taký lookahead $(? = L)$ (lookbehind $(? <= L)$), že $\varepsilon \in L$. Nech je α' regulárny výraz bez tohto lookaheadu (lookbehindu). Potom $L(\alpha') = L(\alpha)$.*

Dôkaz. Uvedomme si, že lookaround nie je fixovaný na dĺžku vstupu - musí sa zhodovať s nejakým podslovom začínajúcim sa (končiacim sa) na konkrétnom mieste. Tým pádom akonáhle si môže regulárny výraz vnútri tejto operácie vybrať ε, bude hlásiť zhodu vždy. □

1.2 Spätné referencie

Rozšírme regexy o spätné referencie, túto množinu operácií nazvem e-regex. V tejto časti ma bude zaujímať, čo sa stane, ak k tomuto modelu pridáme ešte lookahead a lookbehind. Najprv však uvediem základné informácie o triede e-regexov.

Spätná referencia (angl. *backreference*) je v e-regexoch označená ako $\backslash m$. Očíslujme okrúhle zátvorky zľava doprava podľa poradia ľavej zátvorky a zoberme podslovo, ktoré akceptoval výraz vnútri m-tých zátvoriek. $\backslash m$ bude predstavovať presne tento reťazec (pri inom slove teda môže byť iným reťazcom). Budem predpokladať, že spätná referencia s číslom m sa bude nachádzať až za pravou zátvorkou s číslom m.

Triedu jazykov e-regexov budem nazývať Eregex, presná definícia sa nachádza v článku [CSY03]. (Autori ju pôvodne nazvali extended regex resp. EREG, avšak pre lepšiu prehľadnosť v tejto práci som názov upravila.)

Uvediem najprv niektoré fakty o tejto triede. Trieda Eregex je podmnožinou $\mathscr{L}_{CS}$, ale existujú jazyky z $\mathscr{L}_{CF}$ aj $\mathscr{L}_{CS}$, ktoré do nej nepatria. Je uzavretá na homomorfizmus a nie je uzavretá na komplement, inverzný homomorfizmus, konečnú substitúciu, shuffle s regulárnym jazykom [CSY03] a prienik [CN09].

Definícia 1.2.1. *Množinu e-regex obohatenú o pozitívny lookahead a pozitívny lookbehind budem nazývať le-regex.*

Definícia 1.2.2. *Triede jazykov tvorenej le-regexami budem hovoriť LEregex.*

Tu si definujme model na reprezentáciu le-regexov.

TODO!!!just idea; uvidim este co s touto definiciou (je to naviazanie na def. z jedneho clanku)

Definícia 1.2.3. *Nech α je le-regex. Nech α' je rovnaký ako α, akurát bez operácií lookahead a lookbehind. Potom $\alpha' \in Eregex$ a teda je reprezentovaný stromom $T_{\alpha'}$ podľa definície triedy Eregex. Zostrojme konečný (orientovaný, usporiadaný) výpočtový strom S_α z $T_{\alpha'}$ tak, že pridáme lookahead/lookbehind do vrcholu, kam zapadne pri postupnom rozkladaní regexu od koreňa a urobíme s ním nasledovné:*

- *α obsahuje lookahead, teda S_α má niekde vrchol $(u, (? = \beta_1)\beta_2)$. Potom tento vrchol bude mať dvoch synov - vrchol (u, β_2), ktorý pokračuje ako v $T_{\alpha'}$, a virtuálny vrchol $(u, \beta_1(.*))$, ktorý sa ďalej rozloží podľa definície.*
- *α obsahuje lookbehind, teda S_α má niekde vrchol $(u, \beta_1(? <= \beta_2))$. Potom tento vrchol bude mať dvoch synov - vrchol (u, β_1), ktorý pokračuje ako v $T_{\alpha'}$, a virtuálny vrchol $(u, (.*)\beta_2)$, ktorý sa ďalej rozloží podľa definície.*
- *všetky ostatné vrcholy sú rovnaké ako v $T_{\alpha'}$*

Jazyk popísaný le-regexom α je definovaný nasledovne:

$$L(\alpha) = \{w \in \Sigma^* \mid (w, \alpha) \textit{ je koreňom nejakého výpočtového stromu } S_\alpha\}$$

Veta 1.2.4. *Eregex $\subsetneq$ LEregex*

Dôkaz. $\subseteq$ vyplýva z definície.

Jazyk $L = \{a^i b a^{i+1} b a^k \mid k = i(i+1)k' \textit{ pre nejaké } k' > 0, i > 0\}$ nepatrí do triedy Eregex [CN09, Lemma 2], ale patrí do LEregex:

$$\alpha = (a*)b(\backslash 1a)b(? = (\backslash 1) * \$)(\backslash 2) * \$$$

□

Veta 1.2.5. *LEregex $\subseteq \mathscr{L}_{CS}$*

Dôkaz. Vieme, že *Eregex* $\in \mathscr{L}_{CS}$ [CSY03, Theorem 1], teda ľubovoľný e-regex vieme simulovať pomocou LBA. Ukážem, že ak pridáme operáciu lookahead/lookbehind, vieme to simulovať tiež. Nech α je le-regex. Potom A je LBA pre α, ktorý ignoruje lookaround (t.j. vyrábame LBA pre e-regex). Teraz vytvorím LBA B pre e-regex vnútri lookaroundu. Z nich vytvoríme LBA C pre úplný le-regex α tak, že bude simulovať A a keď príde na rad lookaround zaznačí si, v akom stave je A a na ktorom políčku skončil a simuluje do toho miesta B. Pokiaľ B akceptoval, vráti sa naspäť k výpočtu na A. C akceptuje práve vtedy, keď A.

Všimnime si, že samotný lookaround môže obsahovať le-regex, t.j. vnorený lookaround. Tých však môže byť iba konečne veľa, keďže každý le-regex musí mať konečný zápis. A teda si vieme pre ne naznačeným postupom postupne vybudovať LBA, ktorý ich dokáže simulovať. □

Veta 1.2.6. *LEregex je uzavretá na prienik.*

Dôkaz. Nech $L_1, L_2 \in LEregex$, potom $L_1 \cap L_2 = (? = L_1\$)\, L_2\$$. □

TODO!!!zla veta, zla zla zla!!!

Veta 1.2.7. *LEregex NIE je uzavretá na homomorfizmus.*

Dôkaz. TODO!!!dokaz pribudne casom... □

Triedu LEregex obohatenú o negatívny lookahead a negatívny lookbehind budeme nazývať L!Eregex.

Veta 1.2.8. *L!Eregex je uzavretá na komplement.*

Dôkaz. Nech $L_1 \in LEregex$, potom $L_1^c = (?!L_1\$)\,.*\$$. □

Veta 1.2.9.

Dôkaz. □

Záver

Literatúra

[CN09] BENJAMIN CARLE and PALIATH NARENDRAN. On extended regular expressions. In *Language and Automata Theory and Applications*, volume 3, pages 279–289. Springer, April 2009. http://www.cs.albany.edu/~dran/my_research/papers/LATA_version.pdf [Online; accessed 19-March-2013].

[Cox07] Russ Cox. *Regular Expression Matching Can Be Simple And Fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)*, 2007. http://swtch.com/~rsc/regexp/regexp1.html [Online; accessed 30-December-2012].

[CSY03] CEZAR CÂMPEANU, KAI SALOMAA, and SHENG YU. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14(06):1007–1018, 2003. http://www.worldscientific.com/doi/abs/10.1142/S012905410300214X [Online; accessed 19-March-2013].

[doc12] Python documentation. *Regular expressions operations*, 2012. http://docs.python.org/3.1/library/re.html [Online; accessed 30-December-2012].