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

83491340

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

Zadefinujem regulárne jazyky s operáciami, ktoré pokrývajú triedu regulárnych jazykov. Niektoré konštrukcie sú oproti teoretickému modelu nové, ale dôkaz toho, že trieda jazykov zostáva rovnaká je triviálny. Definícia pochádza z článku [CSY03].

1.

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 $L^{i+1} = L^i L$)
$\mathcal{R}$ – tradičné označenie triedy regulárnych jazykov
DKA/NKA – deterministický/nedeterministický konečný automat

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 $uv \in L_1{*?}L_2$. $\square$

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).

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

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).

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

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. Nech L_1, L_2, L_3 sú regulárne, nech $A_i = (K_i, \Sigma_i, \delta_i, q_{0i}, F_i)$ sú 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 $xyz = w \in L$. $\square$

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.) $\square$

TODO!!!lookbehind - podobne ako lookahead

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

Dôkaz. Nech $L_1, L_2, L_3, L_4 \in \mathcal{L}_{CF}$. $L_1 = \{a^n b^n \mid n \geq 1\}, L_2 = \{a * b^n c^n \mid n \geq 1\}, L_3 = \{a^n b^n c * \ \mid n \geq 1\}, L_4 = \{ab^n c^n \mid n \geq 1\}$. 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. *$\mathcal{L}_{CS}$ je uzavretá na operáciu lookahead a lookbehind.*

Dôkaz. Uzavretosť na lookahead:

Pre $L_1, L_2, L_3 \in \mathcal{L}_{CS}$ 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 zapadajú do regulárnych výrazov.

Veta 1.1.13. *Nech $L_1, L_2, L_3, L_4 \in \mathcal{R}$. Potom $L = (L_1 \, (? = L_2) \, L_3) * L_4 \in \mathcal{R}$.*

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 symbolizuje, že nejaká inštancia A_2 je v 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.akceptovať

$L(A) = L$.

$\subseteq$:

$\supseteq$: $\square$

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

Veta 1.1.14. *Nech L je ľubovoľný jazyk a $L_p = L \cup \{uv \mid u \in L\}$. Nech α je 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 $w \in L(\alpha')$. $\square$

1.2 Spätné referencie

Rozšírme regulárne výrazy o spätné referencie. 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 so spätnými refereciami.

Spätná referencia (angl. *backreference*) je v regulárnych výrazoch 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 tvorenú regulárnymi výrazmi (popisujúce triedu regulárnych jazykov) so spätnými referenciami budem nazývať E-regex, 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 E-regex je podmnožinou $\mathcal{L}_{CS}$, ale existujú jazyky z $\mathcal{L}_{CF}$ aj $\mathcal{L}_{CS}$, 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. *Triedu E-regex obohatenú o pozitívny lookahead a pozitívny lookbehind budeme nazývať LE-regex.*

Definícia 1.2.2. *Ak α je regulárny výraz, ktorý môže obsahovať aj spätné referencie a operácie lookahead a lookbehind. Potom hovoríme, že α je l-rozšírený regex a $L(\alpha)$ patrí do triedy LE-regex.*

Tu si definujme model na reprezentáciu regulárnych výrazov so spätnými referenciami a pozitívnymi operáciami lookahead a lookbehind.

Definícia 1.2.3. *Nech α je l-rozšírený regex. Nech α' je α bez operácií lookahead a lookbehind. Potom $\alpha' \in E-regex$ a teda je reprezentovaný stromom $T_{\alpha'}$ podľa definície triedy E-regex. Zostrojme konečný (orientovaný, usporiadaný) výpočtový strom S_α z $T_{\alpha'}$ postupným rozkladaním výrazu od vrchola (w, α):*

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

Jazyk popísaný l-rozšíreným regexom α je definovaný nasledovne:

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

Lema 1.2.4 (Pumpovacia lema)**.**

Dôkaz. □

Veta 1.2.5. $E - regex \subsetneq LE - regex$

Dôkaz. $\subseteq$ vyplýva z definície.

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

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

$L(\alpha) = L$. □

Veta 1.2.6. $LE - regex \subseteq \mathcal{L}_{CS}$

Dôkaz. Vyplýva z vety 1.1.12 a toho, že $E - regex \in \mathcal{L}_{CS}$. □

Veta 1.2.7. *LE-regex je uzavretá na prienik.*

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

Triedu LE-regex obohatenú o negatívny lookahead a negatívny lookbehind budeme nazývať L!E-regex.

Veta 1.2.8. *L!E-regex je uzavretá na komplement.*

Dôkaz. Nech $L_1 \in LE - regex$, 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].