Moderné regulárne výrazy

Tatiana Tóthová*

Školiteľ: Michal Forišek†

Katedra informatiky, FMFI UK, Mlynská Dolina 842 48 Bratislava

Abstrakt: Regulárne výrazy implementované v súčasných programovacích jazykoch ponúkajú omnoho viac operácií ako pôvodný model z teórie jazykov. Už konštrukciou spätných referencií bola prekročená hranica regulárnych jazykov. Náš model obsahuje naviac konštrukcie lookahead a lookbehind. V článku uvedieme zaradenie modelu zodpovedajúcej triedy jazykov do Chomského hierarchie, vlastnosti tejto triedy a výsledky z oblasti priestorovej zložitosti.

Kľúčové slová: regulárny výraz, regex, lookahead, lookbehind, spätné referencie

1 Úvod

Regulárne výrazy vznikli v 60tych rokoch v teórii jazykov ako ďalší model na vyjadrenie regulárnych jazykov. Z takéhoto popisu ľudský mozog rýchlejšie pochopil o aký jazyk sa jedná, než zo zápisu konečného automatu, či regulárnej gramatiky. Ďalšou výhodou bol kratší a kompaktný zápis.

Vďaka týmto vlastnostiam boli implementované ako vyhľadávací nástroj. Postupom času sa iniciatívou používateľov s vyššími nárokmi pridávali nové konštrukcie na uľahčenie práce. Nástroj takto rozvíjali až do dnešnej podoby. My sa budeme opierať o špecifikáciu regulárnych výrazov v jazyku Python [Foundation, 2012].

Ako čoskoro zistíme, nové regulárne výrazy vedia reprezentovať zložitejšie jazyky ako regulárne, preto je dobré ich nejako odlíšiť. V literatúre sa zaužíval výraz „regex” z anglického *regular expression*, ktorý budeme používať aj my.

1.1 Základná definícia

Regulárne výrazy sú zložené zo znakov a metaznakov. Znak a predstavuje jazyk $L(a) = \{a\}$. Metaznak alebo skupina metaznakov určuje, aká operácia sa so znakmi udeje. Základné operácie sú zreťazenie (je definované tým, že regulárne výrazy idú po sebe, bez metaznaku), Kleeneho uzáver ($(0-\infty)$-krát zopakuj výraz, metaznak $*$) a alternácia (vyber výraz naľavo alebo napravo, metaznak $|$). Naviac sa využíva metaznak $\backslash$, ktorý robí z metaznakov obyčajné znaky a okrúhle zátvorky na logické oddelenie regulárnych výrazov.

Pre regulárny výraz α a slovo $w \in L(\alpha)$ hovoríme, že α vyhovuje slovu w resp. α matchuje slovo w. Tiež budeme hovoriť, že α generuje jazyk $L(\alpha)$.

1.2 Nové jednoduché konštrukcie

- $+$ – Kleeneho uzáver opakujúci $(1-\infty)$-krát
- $\{n,m\}$ ($\{n\}$) – opakuj regulárny výraz aspoň n a najviac m-krát (opakuj n-krát)
- $[a_1 a_2 \dots a_n]$ – predstavuje ľubovoľný znak z množiny $\{a_1, \dots, a_n\}$
- $[\hat{}a_1 a_2 \dots a_n]$ – predstavuje ľubovoľný znak, ktorý nepatrí do množiny $\{a_1, \dots, a_n\}$
- . – predstavuje ľubovoľný znak
- ? – 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]
- ˆ – metaznak označujúci začiatok slova
- \$ – metaznak označujúci koniec slova
- (?# komentár) – komentár sa pri vykonávaní regexu úplne ignoruje

Všetky tieto konštrukcie sú len „kozmetickou” úpravou pôvodných regexov – to isté vieme popísať pôvodnými regulárnymi výrazmi, akurát je to dlhšie a menej prehľadné

Rozdiely medzi minimalistickou a greedy verziou operácií vníma iba používateľ, pretože ak existuje zhoda regexu so slovom, v oboch prípadoch sa nájde. Viditeľné sú až pri výstupnej informácii pre používateľa, ktorú môže použiť ďalej.

* tothova166@uniba.sk

† forisek@dcs.fmph.uniba.sk

[1] všetky spomenuté operácie sú implementované greedy algoritmom

1.3 Zložitejšie konštrukcie

Spätné referencie

Najprv potrebujeme očíslovať všetky zátvorky v regexe. Číslujú sa všetky, ktoré nie sú tvaru $(?\dots)$. Poradie je určnené podľa otváracej zátvorky.

Spätné referencie umožňujú odkazovať sa na konkrétne zátvorky. Zápis je $\backslash k$ a môže sa nachádzať až za k-tymi zátvorkami. Skutočná hodnota $\backslash k$ sa určí až počas výpočtu – predstavuje posledné podslovo zo vstupu, ktoré matchovali k-te zátvorky.

Lookahead

Zapísaný formou (?=...), vnútri je validný regex.

Keď v regexe prídeme na pozíciu lookaheadu, zoberieme regex vo vnútri. V slove sa snažíme matchovať ľubovoľný prefix zostávajúcej časti slova. Ak sa to podarí, pokračujeme v regexe ďalej a v slove od pozície, kde lookahead začínal (tzn. akokeby v regexe nikdy nebol).

Má aj negatívnu verziu – negatívny lookahead (?! ...). Vykonáva sa rovnako ako lookahead, ale má otočnú akceptáciu. Teda ak neexistuje prefix, ktorý by vedel matchovať, akceptuje.

Lookbehind

Zapísaný formou (?<=...), vnútri je validný regex.

Pri výpočte zoberieme regex vnútri lookbehindu a snažíme sa vyhoviet' ľubovoľnému sufixu už matchovanej časti slova. Ak vyhovieme, pokračujeme v slove a regexe akoby tam lookbehind vôbec nebol.

Aj lookbehind má negatívnu verziu – negatívny lookbehind (?<! ...) – a pracuje analogicky ako negatívny lookahead.

Lookahead a lookbehind (spolu nazývané jedným slovom lookaround) sú v rôznych implementáciách rôzne obmedzované, aby výpočet netrval príliš dlho. V teórii tieto obmedzenia ignorujeme a prezentujeme model v plnej sile – výsledky tak prezentujú hornú hranicu toho, čo implemetnácie dokážu.

1.4 Triedy a množiny

Kvôli porovnávaniu a vytvoreniu hierarchie sme rozdelili operácie do niekoľkých množín:

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

Eregex – *Regex* so spätnými referenciami

LEregex – *Eregex* s pozitívnym lookaroundom

nLEregex – *LEregex* s negatívnym lookaroundom

$\mathscr{L}_{RE}$ – trieda jazykov nad *Regex* ($= \mathscr{R}$)

$\mathscr{L}_{ERE}$ – trieda jazykov nad *Eregex*

$\mathscr{L}_{LERE}$ – trieda jazykov nad *LEregex*

$\mathscr{L}_{nLERE}$ – trieda jazykov nad *nLEregex*

Trieda $\mathscr{L}_{LERE}$ už bola hlbšie preskúmaná a výsledky čerpáme z článkov [Câmpeanu et al., 2003] a [Carle and Nadendran, 2009].

2 Vlastnosti lookaroundu

Na začiatok sme zisťovali, čo robia samotné operácie lookaroundu.

Veta 1. *$\mathscr{R}$ je uzavretá na negatívny a pozitívny lookaround.*

Dôkaz. Nech $L_1, L_2, L_3 \in \mathscr{R}$. Chceme ukázať, že $L_1(?{=}L_2)L_3, L_1(?{<=}L_2)L_3, L_1(?!\ L_2)L_3, L_1(?{<}!\ L_2, L_3 \in \mathscr{R}$. Pre každé L_i, $i \in \{1,2,3\}$ existuje deterministický konečný automat A_i, ktorý ho akceptuje. Spojíme automaty dohromady nasledujúcim spôsobom:

$L_1(?{=}L_2)L_3$: Začne výpočet A_1. Ak akceptuje, spustí sa naraz výpočet A_2 a A_3 ... □

Veta 2. *$\mathscr{L}_{CF}$ nie je uzavretá na pozitívny lookaround.*

Vieme totiž vygenerovať jazyk $a^n b^n c^n$ prienikom jazykov $a * b^n c^n$ a $a^n b^n c*$.

Veta 3. *$\mathscr{L}_{CS}$ je uzavretá na pozitívny lookaround.*

Veta 4. *Trieda jazykov nad Regex s pozitívnym a negatívnym lookaroundom je ekvivalentná $\mathscr{R}$.*

3 Chomského hierarchia

Veta 5. $\mathscr{R} \subsetneq \mathscr{L}_{ERE} \subseteq \mathscr{L}_{LERE} \subsetneq \mathscr{L}_{nLERE} \subsetneq \mathscr{L}_{CS}$

Všetky $\subseteq$ triviálne platia.

$\mathscr{L}_{ERE} \subsetneq \mathscr{L}_{LERE}$: Nerovnosť dokazuje jazyk $L = \{a^i b a^{i+1} b a^i k \mid k = i(i+1)k'$ pre nejaké $k' > 0, i > 0\}$. $L \notin Eregex$ podľa pumpovacej lemy z [Carle and Nadendran, 2009] a tu je regex z *LEregex* pre L: $\alpha = \underset{1}{(}a*\underset{1}{)}b\underset{2}{(}\backslash 1a\underset{2}{)}b(?{=}(\backslash 1)*\$)(\backslash 2)*$

$\mathscr{L}_{LERE}, \mathscr{L}_{nLERE} \subsetneq \mathscr{L}_{CS}$: Triedy $\mathscr{L}_{LERE}$ a $\mathscr{L}_{nLERE}$ sú neporovnateľné s $\mathscr{L}_{CF}$. K jazyku $L_1 = \{ww \mid w \in \{a,b\}^*\}$ existuje regex z *LEregex*: $\alpha = ((a|b)*)\backslash 1$. Ani jedna z tried neobsahuje jazyk $L_2 = \{a^n b^n | n \in \mathbb{N}\}$.

Intuitívne by malo platiť aj $\mathscr{L}_{ERE} \subsetneq \mathscr{L}_{LERE}$, pretože negatívny lookaround pridáva uzavretosť na komplement. Jazyk dokazujúci nerovnosť by mohol byť napríklad regex $\alpha = (?!\ (aa+)(\backslash 1)+\$\ )$, pričom $L(\alpha) = \{a^p \mid p \text{ je prvočíslo}\}$.

4 Vlastnosti triedy $\mathscr{L}_{LERE}$

Očividne operácia lookahead/lookbehind pridala uzavretosť na prienik. Nech $\alpha, \beta \in LEregex$, potom $L(\alpha) \cap L(\beta) = L(\gamma)$, kde $\gamma = (?{=}\alpha\$)\beta$ alebo $\beta(?{<=}\hat{}\alpha)$.

Veta 6. *Nech $\alpha \in LEregex$ nad unárnou abecedou $\Sigma = \{a\}$, že neobsahuje lookahead s \$ ani lookbehind s ˆ vnútri iterácie. Existuje konštanta N taká, že ak $w \in L(\alpha)$ a $|w| > N$, potom existuje dekompozícia $w = xy$ s nasledujúcimi vlastnosťami:*

(i) $|y| \geq 1$

(ii) $\exists k \in \mathbb{N},\ k \neq 0;\ \forall j = 1,2,\dots : xy^{kj} \in L(\alpha)$

Veta 7. *Jazyk všetkých platných výpočtov Turingovho stroja patrí do $\mathscr{L}_{LERE}$.*

Takýto jazyk pre konkrétny Turingov stroj obsahuje slová, ktoré sú tvorené postupnosťou konfigurácií oddelených oddeľovačom #, ktoré zodpovedajú akceptačným výpočtom na všetkých možných vstupoch.

Turingov stroj má konečný zápis, preto je možné regex pre takýto jazyk vytvoriť. Konštrukcia regexu: $\alpha = \beta(\gamma) * \eta$, kde β predstavuje počiatočnú konfiguráciu[2] a η akceptačnú konfiguráciu. Ak q_0 je akceptačný stav, potom na koniec α pridáme $|(\#q_0.*\#)$. $\gamma = \gamma_1 \mid \gamma_2 \mid \gamma_3$. Prvok γ_i generuje validnú konfiguráciu a zároveň kontroluje, či nasledujúca konfigurácia môže podľa δ-funkcie nasledovať. Rozpíšeme si iba jednu možnosť: $\gamma_1 = (\underset{k}{(}.*\underset{k}{)}xqy\underset{k+1}{(}\ .*\ \underset{k+1}{)}\ \#)(? = \xi\#)$ platí pre $\forall q \in K,\ \forall y \in \Sigma$ a kde $\xi = \xi_1 \mid \xi_2 \mid \dots \mid \xi_n$.

- Ak $(p,z,0) \in \delta(q,y)$, potom $\xi_i = (\backslash k\, xpz \backslash k+1)$ pre nejaké i
- Ak $(p,z,1) \in \delta(q,y)$, potom $\xi_i = (\backslash k\, xzp \backslash k+1)$ pre nejaké i
- Ak $(p,z,-1) \in \delta(q,y)$, potom $\xi_i = (\backslash k\, pxz \backslash k+1)$ pre nejaké i

γ_2 a γ_3 sú podobné ako γ_1, ale matchujú krajné prípady, kedy je hlava Turingovho stroja na ľavom alebo pravom konci pásky.

5 Priestorová zložitosť

Pod'akovanie

Ďakujem školiteľovi za cenné rady a pripomienky.

Literatúra

[Carle and Nadendran, 2009] Carle, B. and Nadendran, P. (2009). On extended regular expressions. In *Language and Automata Theory and Applications*, volume 3, pages 279–289. Springer.

[Câmpeanu et al., 2003] Câmpeanu, C., Salomaa, K., and Yu, S. (2003). A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14(06):1007–1018.

[Ehrenfeucht and Zeiger, 1975] Ehrenfeucht, A. and Zeiger, P. (1975). Complexity measures for regular expressions. *Computer Science Technical Reports*, 64.

[Ellul et al., 2013] Ellul, K., Krawetz, B., Shallit, J., and wei Wang, M. (2013). Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics u (v) w, x–y.*

[Foundation, 2012] Foundation, P. S. (2012). *Regular expressions operations.*

[Tóthová, 2013] Tóthová, T. (2013). Moderné regulárne výrazy. Bachelor's thesis, FMFI UK Bratislava.

[2] Musí byť previazaná s nasledujúcou konfiguráciou, aby spĺňala δ-funkciu. Spraví sa to podobne ako v γ_1.