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 \, 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)
- $[a_1a_2\ldots a_n]$ – predstavuje ľubovoľný znak z množiny $\{a_1,\ldots,a_n\}$
- $[\hat{}a_1a_2\ldots a_n]$ – predstavuje ľubovoľný znak, ktorý nepatrí do množiny $\{a_1,\ldots,a_n\}$
- . – 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 (?...). 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 vyhovieť 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 Priorita

Pri interakcii toľkých operácií je nutné vedieť ich priority. Existujú také, ktoré sa správajú ako znak, čomu zodpovedajú [],[^],. a každá spätná referencia. Špeciálne sú lookahead a lookbehind – tie sa vykonajú hneď akonáhle na ne narazíme. Ostatné zoradíme v tabuľke:

priorita	3	2	1	0
operácia	()	$*$ + ? {}	zreťazenie	|

1.5 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 Formalizácia modelu

Pri zložitejších dôkazoch sa ukázala potreba lepšieho formalizmu, než len množiny operácií. Kvôli jednoduchosti sme vybrali len potrebné operácie – zreťazenie, alternáciu, Kleeneho $*$, spätné referencie a pozitívny a negatívny lookaround – a pokúsili sa ho vyjadriť ako model, ktorý pracuje v krokoch podobne ako Turingov stroj.

Základným prvkom je **konfigurácia**. Je to dvojica regex $r_1 \dots r_n$ a vstupné slovo $w_1 \dots w_m$, pričom v oboch reťazcoch sa navyše nachádza ukazovateľ pozície $\lceil$ (ako hlava Turingovho stroja): $(r_1 \dots \lceil r_i \dots r_n, w_1 \dots \lceil w_j \dots w_m)$. Špeciálne rozoznávame počiatočnú konfiguráciu $(\lceil r_1 \dots r_n, \lceil w_1 \dots w_m)$ a akceptačnú konfiguráciu $(r_1 \dots r_n \lceil, w_1 \dots w_m \lceil)$.

Najprv si definujeme potrebné pojmy indexovateľnosti a alternovateľnosti. *Indexovateľné zátvorky* sú také, kde za otváracou zátvorkou nenasleduje ? (t.j.

všetky prípady okrem lookaroundu). Tieto zátvorky budeme číslovať. *Alternovateľný regex* je taký, ktorý sa môže vyskytovať v alternácii. Sú 3 prípady: regex sa môže nachádzať naľavo od $|$, napravo od $|$ alebo je z oboch strán ohraničený $|$. Ak alternácia nie je uzavretá zátvorkami, ľavý a pravý krajný regex siaha až ku kraju slova, pretože alternácia je operácia s najmenšou prioritou. Inak sú pre nich hranicou zátvorky uzatváracie alternáciu.

Vďaka definovaniu týchto pojmov vidíme, že vieme algoritmicky zistiť, ktoré zátvorky sú indexovateľné a ktoré regexy sú alternovateľné.

Definujeme **krok výpočtu** ako reláciu $\vdash$ na konfiguráciách ...

Akceptačný výpočet je postupnosť konfigurácií $(\lceil R, \lceil W) \vdash^* (R\lceil, w\lceil)$. Ak existuje akceptačný výpočet pre daný regex R a slovo W hovoríme, že regex R matchuje slovo W respektívne slovo W vyhovuje regexu R. **Jazyk** vyhovujúci danému regexu je množina slov, pre ktoré existuje akceptačný výpočet.

Vďaka týmto definíciám sme schopný odhadnúť dĺžku výpočtu:

Veta 1. *Nech* $\alpha \in \mathscr{L}_{LERE}$ *a* $w \in L(\alpha)$. *Potom existuje akceptačný výpočet, ktorý má najviac* $O(|\alpha| \cdot |w|)$ *konfigurácií.*

Dôkaz. Vo väčšine krokov výpočtu sa posúvame dopredu buď v regexe alebo v slove alebo v oboch. Takéto kroky vedú k postupnosti dĺžky $O(|\alpha| + |w|)$.

Výpočet môže predĺžiť skákanie ukazovateľa dozadu. To nastáva iba v prípade, ak v regexe ukazujeme na Kleeneho $*$ a rozhodneme sa skočiť v regexe dozadu, aby sme urobili ďalšiu iteráciu. Zamyslime sa nad samotným akceptačným výpočtom. Ak existuje, potom existuje aj taká jeho verzia, kde každé opakovanie regexu pomocou Kleeneho $*$ matchuje aspoň 1 znak – prázdne iterácie môžeme vyhodiť, lebo ukazovateľ v slove zostal na mieste a konfigurácia skoku je rovnaká, takže sa postupnosť priamo napojí. Vieme, že regex opakovaný operáciou $*$ je dlhý $O(|\alpha|)$ znakov a opakujeme najviac $O(|w|)$-krát.

Teda dokopy spravíme najviac $O(|\alpha| + |w|) + O(|\alpha| \cdot |w|) = O(|\alpha| \cdot |w|)$ krokov. $\square$

3 Vlastnosti lookaroundu

Na začiatok sme zisťovali, čo robia samotné operácie lookaroundu.

Veta 2. $\mathscr{R}$ *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 ... $\square$

Veta 3. $\mathscr{L}_{CF}$ *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 4. $\mathscr{L}_{CS}$ *je uzavretá na pozitívny lookaround.*

Veta 5. *Trieda jazykov nad Regex s pozitívnym a negatívnym lookaroundom je ekvivalentná* $\mathscr{R}$.

4 Chomského hierarchia

Veta 6. $\mathscr{R} \subsetneq \mathscr{L}_{ERE} \subsetneq \mathscr{L}_{LERE} \subseteq \mathscr{L}_{nLERE} \subsetneq \mathscr{L}_{CS}$

Dôkaz. 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 $L_1 = \{ww \mid w \in \{a,b\}^*\} \notin \mathscr{L}_{CF}$ existuje regex z $LEregex$: $\alpha = ((a|b)*)\backslash 1$. Ani jedna z tried neobsahuje jazyk $L_2 = \{a^n b^n | n \in \mathbb{N}\} \in \mathscr{L}_{CF}$. $\square$

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

5 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 $\beta(?{<=}\hat{\ }\alpha)$.

Napak ohrozila uzavretosť na základnú operáciu – zreťazenie. Pri zreťazení 2 jazykov, ktorých regexy nutne musia obsahovať lookahead resp. lookbehind nastáva problém. Nemôžeme tieto regexy len tak položiť za seba. Ak sa napríklad v prvom z jazykov nachádza lookahead, počas výpočtu môže zasahovať aj do časti vstupu, ktorú matchuje druhý regex a tým zmeniť výsledok celého výpočtu. Nakoniec sa ukázalo:

Veta 7. *$\mathscr{L}_{LERE}$ je uzavretá na zreťazenie.*

Dôkaz. Nech $\alpha, \beta \in LEregex$. Jazyku $L(\alpha)L(\beta)$ bude zodpovedať regex

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

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 ˆ – na začiatok pridáme $\hat{}\backslash 1.*$
- s ˆ – pred ˆ pridáme $\hat{}\backslash 1$

Slovami vyjadrené, regex γ najprv rozdelí vstupné slovo na 2 podslová w_1, w_2 patriace do príslušných jazykov $L(\alpha), L(\beta)$. Potom spustí ešte raz regex α upravený tak, že jeho lookaheady sú „skrotené", pretože ich na konci donúti matchovať w_2. Rovnako lookbehindy v β' donúti na začiatku matchovať w_1, až potom normálne pokračuje ich výpočet.

Zrejme $L(\gamma) = L(\alpha)L(\beta)$. □

Veta 8. *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) $\exists k \in \mathbb{N},\ k \neq 0;\ \forall j = 1, 2, \ldots : xy^{kj} \in L(\alpha)$

Veta 9. *Jazyk všetkých platných výpočtov Turingovho stroja patrí do $\mathscr{L}_{LERE}$.*

Dôkaz. Takýto jazyk pre konkrétny Turingov stroj M obsahuje slová, ktoré sú tvorené postupnosťou konfigurácií oddelených oddeľovačom #. Každá postupnosť zodpovedá akceptačnému výpočtu na nejakom slove. Jazyk obsahuje akceptačné výpočty na všetkých slovách, ktoré sú v jazyku $L(M)$.

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 pomocou lookaheadu, 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 $\xi = \xi_1 \mid \xi_2 \mid \ldots \mid \xi_n$.

- 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.

Zrejme $L(\gamma)$ je požadovaný jazyk. □

6 Priestorová zložitosť

Veta 10. *$\mathscr{L}_{LERE} \subsetneq NSPACE(\log n)$, kde n je veľkosť vstupu.*

Dôsledok Savitchovej vety:

Veta 11. *$\mathscr{L}_{LERE} \subsetneq DSPACE(\log^2 n)$, kde n je veľkosť vstupu.*

Veta 12. *$\mathscr{L}_{nLERE} \subsetneq DSPACE(\log^2 n)$, kde n je veľkosť vstupu.*

Veta 13. *$L(regex\#word) \in NSPACE(r \log w)$, kde $r = |regex|$, $w = |word|$ a $regex \in LEregex$.*

Veta 14. *$L(regex\#word) \in DSPACE(n \log^2 n)$, kde $regex \in LEregex$ a n je dĺžka vstupu.*

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

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.