

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 navyše konštrukcie lookahead a lookbehind. V článku uvidíme 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 vyhľadávacie nástroje. 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??.

Ako čoskoro zistíme, nové regulárne výrazy vedú reprezentovať zložitejšie jazyky ako regulárne, preto je dobré ich nejakým odlišiť. 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 zret'azenie (je definované tým, že regulárne výrazy idú po sebe, bez metaznakom), Kleeneho uzáver ($(0 - \infty)$ -krát zopakuj výraz, metaznak $*$) a alternácia (vyber výraz naľavo

alebo napravo, metaznak $|$). Navyše 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 $\{a_1, \dots, a_n\}$
- $[\wedge a_1 a_2 \dots a_n]$ – 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
- $^$ – metaznak označujúci začiatok slova
- $\$$ – metaznak označujúci koniec slova
- $(?# \text{komentár})$ – 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.

1.3 Zložitejšie konštrukcie

Spätné referencie

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

*tothova166@uniba.sk

†forisek@dcs.fmph.uniba.sk

¹všetky spomenuté operácie sú implementované greedy algoritmom

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 $(?= \dots)$, vnútri je validný regex.

Keď v regexe prideme 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. ako keby v regexe nikdy nebol).

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

Lookbehind

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

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

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

Lookahead a lookbehind 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 implementácie 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

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

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

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

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

2 Význam lookaroundu

Tu rozoberiem, že samotný nespraví nič, ale so spätnými referenciami je silný.

3 Chomského hierarchia

4 Vlastnosti triedy \mathcal{L}_{LERE}

5 Priestorová zložitosť

Pod'akovanie

Ďakujem školiteľovi za cenné rady a pripomienky.

Literatúra

- [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.