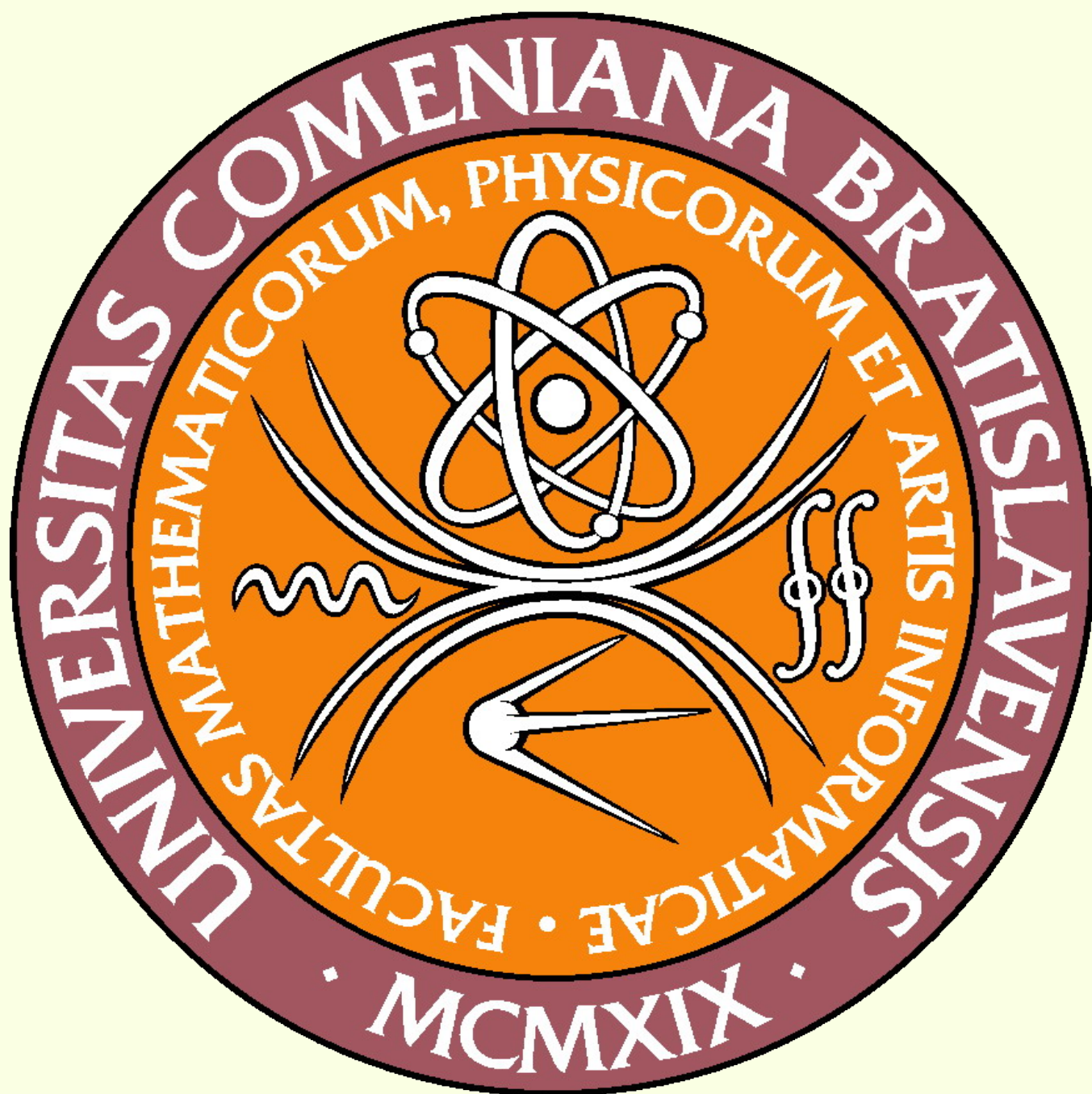


Moderné regulárne výrazy

Tatiana Tóthová

Školiteľ: RNDr. Michal Forišek PhD.

Katedra informatiky, FMFI UK, Mlynská Dolina, 842 48 Bratislava



Úvod

Regulárne výrazy vznikli v 60tych rokoch v teórii jazykov ako ďalší model na vyjadrenie regulárnych jazykov. Z ich 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 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 [Python documentation, 2012]. Nové regulárne výrazy vedia reprezentovať zložitejšie jazyky ako regulárne, preto je dobré ich odlíšiť. V literatúre sa zaužíval výraz „regex“ z anglického *regular expression*, ktorý budeme používať aj my.

Definícia moderných regulárnych jazykov

Základné regulárne výrazy sa skladajú zo znakov a metaznakov. Znak je regulárny výraz predstavujúci sám seba. Metaznak (alebo skupina metaznakov) určuje, čo sa s regulárnym výrazom udeje. Základný model obsahuje operácie

- **zreťazenie** ($\alpha\beta$)
- **alternácia** ($\alpha|\beta$)
- **Kleeneho uzáver** (α^* = opakuj α ($0 - \infty$)-krát)

Okrem toho používame **zátvorky** (**()**) na špecifikovanie poľa pôsobnosti operácií. Každý regex je zložený z konečného počtu operácií.

Moderné regulárne výrazy, nazývané aj **regexy**, majú navyše metaznak pre ľubovoľný znak $.$, začiatok slova \wedge a koniec slova $\$$. Bolo zavedené číslovanie zátvoriek – zľava doprava podľa otváracej zátvorky a konštrukcie tvaru $(? \dots)$ sa nečíslujú. Pribudli aj nasledujúce zložitejšie konštrukcie:

- **Spätné referencie** ($\backslash k$) sa odkazujú na k -te zátvorky (môže sa nachádzať až za nimi). Pri matchovaní slova si zapamätáme, aké podslovo matchovali k -te zátvorky a toto podslovo predstavuje konštrukcia $\backslash k$. Napr. pre regex $\alpha \left(\begin{smallmatrix} \beta \\ k \end{smallmatrix} \right) \gamma \backslash k \delta$ by výpočet na slove w vyzeral takto:

$$w = \underbrace{x_1 \dots x_{i-1}}_{\alpha} \underbrace{\overbrace{x_i \dots x_{j-1}}^{w_k}}_{\left(\begin{smallmatrix} \beta \\ k \end{smallmatrix} \right)} \underbrace{x_j \dots x_{l-1}}_{\gamma} \underbrace{\overbrace{x_l \dots x_{m-1}}^{w_k}}_{\backslash k} \underbrace{x_m \dots x_n}_{\delta}$$

a musí platiť: $w_k = x_i \dots x_{j-1} = x_l \dots x_{m-1}$. Ak existuje viac podslov ku k -tym zátvorkám, berie sa vždy to posledné.

- **Lookahead** (nazeranie dopredu $(?= \dots)$) musí matchovať nejaký prefix od aktuálneho pracovného miesta. Napr. pre regex $\alpha(? = \beta)\gamma$ máme:

$$w = \underbrace{x_1 \dots x_{i-1}}_{\alpha} \underbrace{\overbrace{x_i \dots x_j}^{\beta}}_{\gamma} x_{j+1} \dots x_n$$

- **Lookbehind** (nazeranie dozadu $(? <= \dots)$) musí matchovať nejaký suffix od aktuálneho pracovného miesta. Napr. pre regex $\alpha(? <= \beta)\gamma$ máme:

$$w = \underbrace{x_1 \dots x_{i-1}}_{\alpha} \underbrace{\overbrace{x_i \dots x_j}^{\beta}}_{\gamma} x_{j+1} \dots x_n$$

- Operácie lookahead a lookbehind majú aj **negatívne** verzie, kde regex vnútri konštrukcie nesmie matchovať žiaden prefix/suffix.

Pre lookahead a lookbehind sa zaužíval spoločný názov **lookaround**.

Zaviedli sme nasledujúce množiny operácií:

<i>Regex</i>	základné operácie	$\mathcal{L}_{RE} = \mathcal{R}$
<i>Eregex</i>	+ spätné referencie	\mathcal{L}_{ERE}
<i>LEregex</i>	+ lookahead, lookbehind	\mathcal{L}_{LERE}
<i>nLEregex</i>	+ negatívny lookahead, negatívny lookbehind	\mathcal{L}_{nLERE}

Formálny model

Moderné regulárne výrazy obsahujú množstvo operácií, ktoré spolu rôzne interagujú. Preto sme vytvorili formálny model, ktorý postupuje po krokoch podobne ako Turingov stroj. **Konfigurácia** pre regex $\alpha = r_1 \dots r_n$ a slovo $w = w_1 \dots w_m$ je definovaná (\lceil ukazuje pracovnú pozíciu):

$$(r_1 \dots \lceil r_i \dots r_n, w_1 \dots \lceil w_j \dots w_m)$$

V symboly v slove majú niekoľko poschodí, do ktorých si zapamätáme pomocnú informáciu počas výpočtu. **Krok výpočtu** je definovaný:

$$(r_1 \dots \lceil (\dots r_n, w_1 \dots \lceil w_j \dots w_m) \vdash (r_1 \dots (\lceil \dots r_n, w_1 \dots \lceil \overbrace{w_j}^k \dots w_m)$$

Celú definíciu nájdete v [Tóthová, 2015].

Vlastnosti lookaheadu a lookbehindu

Trieda \mathcal{R} je uzavretá na negatívny aj pozitívny lookahead. Aj trieda *Regex* s lookaroundom stále pokrýva iba regulárne jazyky – takúto kombináciu operácií totiž vieme simulovať konečnými automatmi. Z výsledkov hierarchie tried však vyplýva nasledovné:

$$\mathcal{L} \left(\begin{array}{c} \textit{Regex} \\ +\text{lookahead} \\ +\text{lookbehind} \end{array} \right) \subsetneq \mathcal{L} \left(\begin{array}{c} \textit{Regex} \\ +\text{spätné referencie} \end{array} \right) \subsetneq \mathcal{L} \left(\begin{array}{c} \textit{Regex} \\ +\text{spätné referencie} \\ +\text{lookahead, lookbehind} \end{array} \right)$$

To znamená, že lookahead je síce sám slabá operácia, ale v kombinácii so spätnými referenciami máme silnejší model ako bez lookaroundu. A teda má zmysel vsímať si túto operáciu.

Čo sa týka uzáverových vlastností, pozitívny lookahead pridáva uzavretosť na **prienik** $L(\alpha) \cap L(\beta) = L((?= \alpha \$)\beta)$ a negatívny lookahead pridáva uzavretosť na **komplement** $L(\alpha)^c = L((?! \alpha \$).*)$. Avšak ohrozená je základná operácia regulárnych výrazov – **zreťazenie**. Trieda \mathcal{L}_{LERE} však na zreťazenie uzavretá je, $L(\alpha)L(\beta)$ vyjadríme regexom:

$$(?= \left(\begin{smallmatrix} \alpha \\ 1 \end{smallmatrix} \right) \left(\begin{smallmatrix} \beta \\ 1 \end{smallmatrix} \right) \backslash k + 2 \left(\begin{smallmatrix} \$ \\ k + 2 \end{smallmatrix} \right) \alpha' \backslash k + 2 (? <= \wedge \backslash 1 \beta')$$

Prvý lookahead rozdelí vstupné slovo w na w_1, w_2 , $w = w_1 w_2$. α' je regex α upravený tak, že jeho lookaheady na konci matchujú w_2 a β' má upravené lookbehindy tak, že na začiatku matchujú w_1 . [Tóthová, 2013]

Chomského hierarchia

Triviálne platí, že v definovaných triedach je predchádzajúca podmnožinou nasledujúcej.

$$\mathcal{R} \subsetneq^{(1)} \mathcal{L}_{ERE} \subsetneq^{(2)} \mathcal{L}_{LERE} \subsetneq^{(3)} \mathcal{L}_{nLERE} \subsetneq^{(4)} \mathcal{L}_{CS}$$

Teraz uvedieme jazyky, ktoré dokazujú nerovnosť množín. Zároveň to považujeme za malú ukážku toho, čo moderné regulárne výrazy dokážu.

(1) jazyk $L(\alpha) = \{ww | w \in \{a, b\}^*\} \in \mathcal{L}_{CS}$: $\alpha = (a|b)^* \backslash 1$

(2) jazyk $L(\beta) = \{a^i b a^{i+1} b a^i k \mid k = i(i+1)k', \text{ kde } k' > 0, i > 0\} \in \mathcal{L}_{LERE}$:

$$\beta = \left(\begin{smallmatrix} a * \\ 1 \end{smallmatrix} \right) b \left(\begin{smallmatrix} \backslash 1 a \\ 2 \end{smallmatrix} \right) b (? = (\backslash 1) * \$) (\backslash 2) *$$

(3) nerovnosť je otvoreným problémom. Dobrým kandidátom na jej ukázanie sú jazyky:

$$L(\gamma) = \{a^z | z \text{ je zložené číslo} \} \in \mathcal{L}_{LERE}, \quad \gamma = (aaa^*) \backslash 1 (\backslash 1)^*$$

$$L(\delta) = L(\gamma)^c = \{a^p | p \text{ je prvočíslo} \} \in \mathcal{L}_{nLERE}, \quad \delta = (?! \gamma \$).*$$

(4) \mathcal{L}_{LERE} a \mathcal{L}_{nLERE} sú neporovnateľné s \mathcal{L}_{CF}

Priestorová zložitosť

Toto je oblasť, kvôli ktorej sme vymysleli formálny model. Všetky informácie v slove (ukazovateľ a vyššie poschodia symbolov) sa dajú zapísať vo forme adries. Adresy vieme zapísať v logaritmickej priestore od dĺžky vstupného slova. Pre regex ich potrebujeme konštantne veľa:

$$\mathcal{L}_{LERE} \subseteq NSPACE(\log n)$$

Zo Savitchovej vety vyplýva: $\mathcal{L}_{LERE} \subseteq DSPACE(\log^2 n)$

Myšlienka dôkazu Savitchovej vety spočíva v tom, že testujeme, či sa vieme dostať z jednej konfigurácie do druhej na istý počet krokov. Formálny model má definované konfigurácie, ktoré sme využili a dokázali tak výsledok aj pre negatívny lookahead:

$$\mathcal{L}_{nLERE} \subseteq DSPACE(\log^2 n)$$

V praxi je bežné, že užívateľ zadáva na vstup regex aj text na vyhľadávanie, preto sme si zadefinovali jazyk L_U , ktorý akceptuje slová tvaru *regex#word*, kde *regex* $\in U$, *regex* matchuje slovo *word* a U je množina operácií.

$$L_{LEregex} \in NSPACE(n \log n)$$

Nech U je *Eregex* alebo *LEregex* s konečným počtom vnorení lookaroundov, potom:

$$L_U \in DSPACE(n \log^2 n)$$

Pojmy a skratky

$\mathcal{L}(\dots)$ – trieda jazykov nad $\dots \mathcal{R}$ – trieda regulárnych jazykov

\mathcal{L}_{CF} – trieda bezkontextových jazykov

\mathcal{L}_{CS} – trieda kontextovných jazykov

Literatúra

[Cămpeanu et al., 2003] Cămpeanu, 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.

[Python documentation, 2012] Python documentation (2012). *Regular expression operations*. Python Software Foundation. <http://docs.python.org/2/library/re.html>.

[Tóthová, 2013] Tóthová, T. (2013). Moderné regulárne výrazy. Bachelor's thesis, FMFI UK Bratislava. <https://github.com/Tatianka/bak>.

[Tóthová, 2015] Tóthová, T. (2015). Moderné regulárne výrazy. Master's thesis, FMFI UK Bratislava. <https://github.com/Tatianka/dip>.