

# 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, vlastností tejto triedy a výsledky z oblasti priestorovej zložitosti.

**Keľúčové 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 [Foundation, 2012].

Ako čoskoro zistíme, nové regulárne výrazy vedú 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 zret'azenie (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  $|$ ). Navyše sa využíva metaznak  $\backslash$ , ktorý robí z metaznakov obyčajné znaky a okružle zátvorky na logické oddelenie regulárnych výrazov.

Pre regulárny výraz  $\alpha$  a slovo  $w \in L(\alpha)$  hovoríme, že  $\alpha$  vyhovuje slovu  $w$  resp.  $\alpha$  matchuje slovo  $w$ . Tiež budeme hovoriť, že  $\alpha$  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  $*, +, ?, \{n, m\}$ )<sup>1</sup>
- $^$  – 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

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@dcf.fmph.uniba.sk

<sup>1</sup>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 určené 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  $(?= \dots)$ , 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 (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 implementácie 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 vykonávajú 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

$\mathcal{L}_{RE}$  – 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*

Trieda  $\mathcal{L}_{LERE}$  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  $\uparrow$  (ako hlava Turingovho stroja):  $(r_1 \dots \uparrow r_i \dots r_n, w_1 \dots \uparrow w_j \dots w_m)$ . Špeciálne rozoznávame počiatočnú konfiguráciu  $(\uparrow r_1 \dots r_n, \uparrow w_1 \dots w_m)$  a akceptačnú konfiguráciu  $(r_1 \dots r_n \uparrow, w_1 \dots w_m \uparrow)$ .

Najprv si definujeme potrebné pojmy indexovateľnosti a alternovateľnosti. *Indexovateľné zátvorky* sú také, kde za otváracou zátvorkou nasleduje  $?$  (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 algoritmicke zistiť, ktoré zátvorky sú indexovateľné a ktoré regexy sú alternovateľné.

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

**Akceptačný výpočet** je postupnosť konfigurácií  $(\lceil R, \lceil W) \vdash^* (\lceil R, \lceil w)$ . 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  $\alpha \in \mathcal{L}_{LRE}$  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. Zamyslíme 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 vyhodit', lebo ukazovateľ v slove zostal na mieste a konfigurácia skoku je rovnaká, takže sa postupnosť priamo napojí. Vieme, že regex opakovanej 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 zist'ovali, čo robia samotné operácie lookaroundu.

**Veta 2.**  *$\mathcal{R}$  je uzavretá na negatívny a pozitívny lookaround.*

*Dôkaz.* Nech  $L_1, L_2, L_3 \in \mathcal{R}$ . Chceme ukázať, že  $L_1(?=L_2)L_3, L_1(?<=L_2)L_3, L_1(!L_2)L_3, L_1(?<!L_2)L_3 \in \mathcal{R}$ . Pre každé  $L_i, i \in \{1, 2, 3\}$  existuje determinický konečný automat  $A_i$ , ktorý ho akceptuje.

Konečné automaty vieme vhodne pospájať dohromady. Spravíme konštrukciu pre prienik regulárnych jazykov, ale mierne upravenú tak, že akonáhle automat pre  $L_2$  v pozitívnom lookaheade akceptuje, vo výpočte bude pokračovať už len samotný  $A_3$ , kým dočíta slovo. Podobne pre pozitívny lookbehind –  $A_1$  začne sám a nedeterministicky v nejakom kroku začne výpočet aj  $A_2$ . Akceptovať musia spolu.

Pre negatívne formy musíme navyše upraviť akceptáciu. Ak  $A_2$  pre lookahead akceptuje, celý automat sa zasekne a zamietne.  $A_2$  musí dočítať slovo bez dosiahnutia akceptačného stavu. Pre lookbehind v každom kroku  $A_1$  spúšťame ďalšiu  $A_2$  a držíme si množinu stavov, v ktorých sa všetky nachádzajú. Úspech je, ak  $A_1$  akceptuje a množina stavov pre automaty  $A_2$  neobsahuje akceptačný stav.

Odhládnuť teraz od lookaroundu, máme zreteľnejšie  $L_1$  a  $L_3$ . Preto prepojíme akceptačný stav  $A_1$  s počiatočným stavom  $A_3$ . Výsledný automat sa nedeterministicky rozhoduje, či z akceptačného stavu  $A_1$  pokračuje ďalej v  $A_1$  alebo  $A_3$ .  $\square$

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

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

## 4 Chomského hierarchia

**Veta 6.**  *$\mathcal{R} \subsetneq \mathcal{L}_{ERE} \subsetneq \mathcal{L}_{LRE} \subseteq \mathcal{L}_{nLRE} \subsetneq \mathcal{L}_{CS}$*

*Dôkaz.* Všetky  $\subseteq$  triviálne platia.

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

$\mathcal{L}_{LRE}, \mathcal{L}_{nLRE} \subsetneq \mathcal{L}_{CS}$ : Triedy  $\mathcal{L}_{LRE}$  a  $\mathcal{L}_{nLRE}$  sú neporovnateľné s  $\mathcal{L}_{CF}$ . K jazyku  $L_1 = \{ww \mid w \in \{a, b\}^*\} \notin \mathcal{L}_{CF}$  existuje regex z  $LEregex$ :  $\alpha =$

$((a|b)^*) \setminus 1$ . Ani jedna z tried neobsahuje jazyk  $L_2 = \{a^n b^n | n \in \mathbb{N}\} \in \mathcal{L}_{CF}$ .  $\square$

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

## 5 Vlastnosti triedy $\mathcal{L}_{LRE}$

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

Napak ohrozila uzavretosť na základnú operáciu – zret'azenie. Pri zret'azení 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.**  $\mathcal{L}_{LRE}$  je uzavretá na zret'azenie.

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

$$\gamma = (?=(\alpha) (\beta) \$) \alpha' \setminus k+2 (?<=\setminus 1 \beta')$$

1   1 k+2   k+2

V  $\alpha, \beta$  treba vhodne prepísať označenie zátvoriek (po poradí).  $\alpha'$  je  $\alpha$  prepísaný tak, že pre každý lookahead:

- bez  $\$$  – na koniec pridáme  $.* \setminus k+2\$$
- s  $\$$  – pred  $\$$  pridáme  $\setminus k+2$

$\beta'$  je  $\beta$  prepísaný tak, že pre každý lookbehind:

- bez  $\wedge$  – na začiatok pridáme  $\wedge 1.*$
- s  $\wedge$  – pred  $\wedge$  pridáme  $\wedge 1$

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

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

**Veta 8.** Nech  $\alpha \in LRegex$  nad unárnou abecedou  $\Sigma = \{a\}$ , že neobsahuje lookahead s  $\$$  ani lookbehind s  $\wedge$  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 9.** Jazyk všetkých platných výpočtov Turingovho stroja patrí do  $\mathcal{L}_{LRE}$ .

*Dôkaz.* Takýto jazyk pre konkrétny Turingov stroj  $M$  obsahuje slová, ktoré sú tvorené postupnosťou konfigurácií oddelených oddel'ovačom  $\#$ . 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  $\beta$  predstavuje počiatočnú konfiguráciu<sup>2</sup> a  $\eta$  akceptačnú konfiguráciu. Ak  $q_0$  je akceptačný stav, potom na koniec  $\alpha$  pridáme  $|(\#q_0.*\#)$ .  $\gamma = \gamma_1 \mid \gamma_2 \mid \gamma_3$ . Prvok  $\gamma_i$  generuje validnú konfiguráciu a zároveň kontroluje pomocou lookaheadu, či nasledujúca konfigurácia môže podľa  $\delta$ -funkcie nasledovať. Rozpíšeme si iba jednu možnosť:

$$\gamma_1 = ((.*)_k x q y_k (.*)_k \#) (? = \xi \#)$$

k   k   k+1   k+1

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  $\xi_i = (\setminus k x p z \setminus k+1)$  pre nejaké  $i$
- Ak  $(p, z, 1) \in \delta(q, y)$ , potom  $\xi_i = (\setminus k x z p \setminus k+1)$  pre nejaké  $i$
- Ak  $(p, z, -1) \in \delta(q, y)$ , potom  $\xi_i = (\setminus k p x z \setminus k+1)$  pre nejaké  $i$

$\gamma_2$  a  $\gamma_3$  sú podobné ako  $\gamma_1$ , ale matchujú krajné prípady, kedy je hlava Turingovho stroja na ľavom alebo pravom konci pásky.

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

<sup>2</sup>Musí byť previazaná s nasledujúcou konfiguráciou, aby spĺňala  $\delta$ -funkciu. Spraví sa to pomocou lookaheadu, podobne ako v  $\gamma_1$ .

## 6 Priestorová zložitosť

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

Dôsledok Savitchovej vety:

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

**Veta 12.**  $\mathcal{L}_{nLERE} \subsetneq DSPACE(\log^2 n)$ , 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.

## Podakovanie

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.