Praktická aplikácia regulárnych výrazov. Problémy a ťažkosti pri ich písaní a používaní.*

Denis Danilov

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

xdanilovd@stuba.sk

4. novembra 2023

Abstrakt

Regulárne výrazy, často známe ako regex, je formálny jazyk, ktorý sa používa na priraďovanie textových vzorov. Používanie regexov pomáha rýchlo a presne nájsť a extrahovať kľúčové informácie z veľkých objemov textu. Môžu Vám doslova pomôcť nájsť ihlu v kope sena, čo organizáciám umožňuje presmerovať svoje ľudské zdroje do iných oblastí. Regexy sú obzvlášť užitočné na definovanie filtrov, pretože obsahujú postupnosť znakov, ktoré definujú vzor textu, ktorý sa má zhodovať. Cieľom tejto práce je preskúmať, ako sa regulárne výrazy často používajú v praxi, identifikovať problémy a ťažkosti, s ktorými sa programátori stretávajú pri písaní a používaní regulárnych výrazov. Metódami na dosiahnutie cieľa bude analýza prieskumu medzi vývojármi a údajov o používaní regulárnych výrazov v otvorených projektoch na GitHub. Tento článok jasne ukáže výhody používania regulárnych výrazov, ako aj osvedčené postupy pri ich písaní.

1 Úvod

Regulárne výrazy, tiež známe ako Regexy, sú nástrojom na spracovanie textu, ktorý je integrovaný do všetkých súčasných programovacích jazykov a bežne používaných utilít, ako sú textové editory. Regulárne výrazy sa používajú pri úlohách vyhľadávania a nahrádzania reťazcov, ako je vyhľadávanie slov, úprava textu a analýza súborov. [4] Odhady naznačujú, že viac ako tretina projektov JavaScript a Python obsahuje aspoň jeden regulárny výraz. [5]

Regulárne výrazy sú široko používané v rôznych oblastiach výpočtovej techniky. Tu je niekoľko bežných prípadov použitia:

1. **Analýza protokolov:** Regulárne výrazy možno použiť na extrahovanie špecifických častí informácií zo systémových alebo aplikačných protokolov.
2. **Overenie údajov:** Môžu sa použiť na overenie formátu vstupných údajov, ako sú e-mailové adresy, telefónne čísla a ID používateľov.

*Semestrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2023/24, vedenie: PaedDr. Pavol Baťalík

3. **Hľadať a nahradiť:** Regulárne výrazy možno použiť na nájdenie a nahradenie konkrétnych vzorov v texte alebo kódovej základni.

4. **Premenovanie súborov:** Môžu sa použiť na dávkové premenovanie súborov na základe určitých vzorov.

5. **Analýza vstupu používateľa:** Regulárne výrazy možno použiť na analýzu vstupu používateľa alebo príkazových riadkov.

6. **Čítanie konfiguračných súborov:** Môžu sa použiť na čítanie a analýzu konfiguračných súborov.

Pochopenie a písanie regulárnych výrazov si vyžaduje znalosti aj zručnosti. [3] Jediná chyba v znaku môže spôsobiť drasticky odlišné správanie regulárneho výrazu pri porovnávaní. [7] Viac ako 80 % regulárnych výrazov napísaných v projektoch GitHub však nie je otestovaných [8], čo naznačuje, že vývojári buď regulárne výrazy netestujú, alebo namiesto testovacích prípadov používajú externé nástroje. [7]

Pre regulárne výrazy [6] existujú nástroje na generovanie testov regulárnych výrazov. Tieto nástroje vymenúvajú členy jazyka regulárnych výrazov, ale nevypočítavajú nezhodujúce sa reťazce. Nedávne výskumné úsilie skúmalo prístup založený na chybách pri generovaní testov regulárnych výrazov prostredníctvom testovania mutácií [1]; do regulárnych výrazov sa vnášajú chyby a generujú sa reťazce, ktoré sú svedkami chyby, čím sa poskytujú príklady nezhodného správania pôvodného regulárneho výrazu. Hoci je testovanie mutáciou zrelou oblasťou výskumu, na mutáciu regulárnych výrazov sa zamerali nedávno [1] [2]. Tieto snahy však definovali operátory mutácie ad hoc spôsobom, bez toho, aby zohľadnili, ako sa regulárne výrazy v praxi skutočne vyvíjajú. Preto injektované chyby nemusia byť reprezentatívne pre skutočné úpravy. [7]

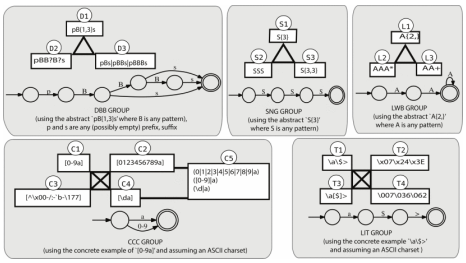
Tento článok je určený na to, aby vám pomohol pochopiť a efektívne používať regulárne výrazy. Začneme základnými pojmami a postupne prejdeme k pokročilejším príkladom.

2 Výskumné úlohy

V tejto práci používame termín reprezentácia regexu na označenie syntaktického vyjadrenia regulárneho výrazu. Funkcia je štrukturálna zložka regulárneho výrazu (napr. Kleenova hviezdička: * alebo vlastná trieda znakov: [4]). Trieda ekvivalencie je skupina behaviorálne ekvivalentných regulárnych výrazov. Pri skúmaní porozumenia regexov odpovedáme na nasledujúce výskumné otázky: [4]

Ktoré reprezentácie regexov sú najzrozumiteľnejšie?

Aby sme odpovedali na prvú otázku, uskutočnili sme štúdiu, v ktorej sme programátorom predložili regex a položili im otázky o jeho správaní pri porovnávaní.



Obr. 1: Typy tried ekvivalencie na základe vlastností jazyka. DBB = dvojnásobne ohraničená, SNG = jednoduchá ohraničená, LWB = nižšia ohraničená, CCC = vlastná trieda znakov a LIT = literálna. Na ilustráciu používame konkrétne regexy spolu s ich deterministickým konečným automatom (dfa) v reprezentáciách. A v skupine LWB (alebo B v skupine DBB, S v skupine SNG a podobne) však abstraktne reprezentujú akýkoľvek vzor, s ktorým by sa dalo pracovať modifikátorom opakovania (napr. literálne znaky, triedy znakov alebo skupiny). To isté platí pre literály používané vo všetkých reprezentáciách. [4]

Porovnaním presnosti regexov, ktoré zodpovedajú rovnakému jazyku, ale sú vyjadrené pomocou rôznych reprezentácií (napr. tri[a-f] 3 a tri (a | b | c | d | e | f)3), môžeme merať zrozumiteľnosť a identifikovať zápach kódu. [4]
Skúmame aj faktory, ktoré môžu ovplyvniť porozumenie, konkrétne dĺžku reťazca regexu, veľkosť regexu DFA (Deterministický konečný automat) a reprezentáciu triedy ekvivalencie. Táto analýza si vyžaduje identifikáciu tried ekvivalencie pre regexy. Kontrolou súboru údajov o regexoch v jazyku Python, ktorý obsahuje takmer 14 000 regexov [3], sme vytvorili počiatočný súbor piatich typov tried ekvivalencie, ktoré sme skúmali. [4]

Ktoré reprezentácie regexov majú najväčšiu podporu komunity na základe frekvencie?

Na zodpovedanie druhej otázky skúmame verejne dostupný súbor údajov o regexoch [3] a používame prítomnosť a neprítomnosť jazykových znakov ako zástupný ukazovateľ podpory komunity, pričom sa predpokladá, že častejšie používané znaky sú zrozumiteľnejšie. [4]

3 Základy písania regulárnych výrazov

3.1 Ako vyzerá regulárny výraz?

V najjednoduchšej podobe môže regulárny výraz vyzerať takto:

```
/Test/
```

V príklade "Test"vygeneroval test písmena podobný vzor vyhľadávania ako pri jednoduchom vyhľadávaní. Tieto regulárne výrazy však nie sú vždy také jednoduché. Tu je regulárny výraz, ktorý zodpovedá 3 číslam, za ktorými nasle-

duje znak ", potom 3 čísla, za ktorými nasleduje ďalší znak ä nakoniec končí 4 číslami.

```
^(?:\d{3}-){2}\d{4}$
```

Väčšinu regulárnych výrazov možno v skutočnosti zapísať viacerými spôsobmi, rovnako ako iné formy programovania. Napríklad vyššie uvedený sa dá prepísať do dlhšej, ale o niečo čitateľnejšej verzie:

```
^[0-9]{3}-[0-9]{3}-[0-9]{4}$
```

Väčšina jazykov poskytuje zabudovanú metódu na vyhľadávanie a nahrádzanie reťazcov pomocou regulárnych výrazov. Každý jazyk však môže mať iný súbor syntaxe v závislosti od toho, čo daný jazyk predpisuje.
V tomto článku sa zameriame na variant regulárneho výrazu ECMAScript, ktorý sa používa v jazyku JavaScript a má veľa spoločného s implementáciami regulárnych výrazov v iných jazykoch.

3.2 Ako čítať (a zapisovať) regulárne výrazy

3.2.1 Kvantory

Kvantifikátory regulárnych výrazov kontrolujú, koľkokrát sa má znak vyhľadať. Tu je zoznam všetkých kvantifikátorov:

- a | b - Zodpovedajúce "a" alebo "b".
- ? - Nula alebo jednotka
- + - jedna alebo viac
- * ? Nula alebo viac?
- N - Presne N-krát (kde N je číslo)
- N, - N alebo viac krát (kde N je číslo)
- N,M - N až M krát (kde N a M sú čísla a N $<$ M)
- *? - Nula alebo viac, ale zastavenie po prvej zhode

Napríklad nasledujúci regulárny výraz:

```
Ahoj \textbar Dobre
```

Zodpovedá reťazcom "Ahoj" a "Dobre".

Medzitým:

```
Ahoj?
```

Bude sledovať "j" od nuly do jednej, takže bude zodpovedať "Aho" a "Ahoj".

Alternatívne:

```
Ahoj{1,3}
```

Bude zodpovedať "Ahoj", "Ahojj", "Ahojjj", ale nie "Ahojjjj", pretože hľadá písmeno "j" od 1 do 3 krát.

3.2.2 Chamtivé porovnávanie

Jedným z kvantifikátorov regexu, ktorého sme sa dotkli v predchádzajúcom zozname, bol symbol. Tento symbol zodpovedá jednému alebo viacerým znakom. To znamená, že:+

```
Ahoj+
```

Zodpovedá všetkým slovám od "Ahoj" po "Ahojjjjjjjjjjj". Je to preto, že všetky kvantifikátory sú štandardne považované za "chamtivé". Ak ho však zmeníte na "lenivý"pomocou symbolu otáznika () na nasledujúci, správanie sa zmení?

```
Ahoj+?
```

Teraz sa porovnávač pokúsi nájsť zhodu čo najmenej krát. Keďže ikona znamená "jeden alebo viac", bude sa zhodovať len s jedným "j". To znamená, že ak zadáme reťazec"Ahojjjjjjjjjjj", porovná sa iba "Ahoj".i+
Hoci to samo o sebe nie je obzvlášť užitočné, v kombinácii so širšími zhodami, ako je symbol, sa to stáva mimoriadne dôležitým, ako si povieme v nasledujúcej časti. Symbol sa v regexe používa na nájdenie"akéhokoľvek znaku".
Ak teraz použijete:

```
A.*oj
```

Môžete porovnať všetko od "Achoj" po "Ahoj" aj po "Ahojchojhoj".

3.2.3 Zbierky vzorov

Zbierky vzorov umožňujú vyhľadať kolekciu znakov, ktoré sa majú porovnať. Napríklad pomocou nasledujúceho regexu:

```
Moja oblubena hlaska je [aeiou]
```

Môžete porovnať nasledujúce reťazce:

```
Moja oblubena hlaska je a
Moja oblubena hlaska je e
Moja oblubena hlaska je i
Moja oblubena hlaska je o
Moja oblubena hlaska je u
```

Ale nič iné.
Tu je zoznam najbežnejších zbierok vzorov:

- [A-Z] - Zodpovedá akémukoľvek veľkým písmenám od "A" po "Z".
- [a-z] - Zodpovedá akémukoľvek malým písmenám od "a" po "z".
- [0-9] - Zodpovedá akýmkoľvek číslam.
- [asdf] - Zodpovedá akémukoľvek znaku, ktorý je buď "a", "s", "d" alebo "f".
- [^asdf] - Zodpovedá akémukoľvek znaku, ktorý nie je "a", "s", "d" alebo "f".

3.2.4 Všeobecné tokeny

Nie každá postava je tak ľahko identifikovateľná. Kým kľúče ako ääž "z"majú zmysel priradiť pomocou regexu, ako je to so znakom nového riadku?

- . - Akýkoľvek znak
- \n - Znak nového riadka
- \t - Znak tabulácie
- \s - Akýkoľvek medzerníkový znak znak (vrátane medzier a iných)
- \S - Akýkoľvek znak, ktorý nie je medzerníkový
- \w - Akýkoľvek znak slova (veľké a malé písmená latinskej abecedy, číslice 0-9 a _)
- \W - Akýkoľvek znak, ktorý nie je znakom slova
- \b - Hraničný znak slova: hraničné znaky medzi w a \W, ale zhoduje sa so znakmi medzi znakmi \w \W
- \B - Hraničný znak slova: opak znaku \b
- ^ - Začiatok riadka
- $ - Koniec riadka
- \\ - Doslovný znak "\"

Ak by ste teda chceli odstrániť každý znak, ktorý začína nové slovo, mohli by ste použiť niečo ako nasledujúci regex:

```
\s.
```

A výsledky nahraďte prázdnym reťazcom. Pri tomto postupe je nasledovné:

```
Ahoj ako sa mas
```

Stáva sa:

```
Ahojkoaas
```

Kombináciou uvedených metód môžete vyriešiť väčšinu základných úloh, s ktorými sa vývojár stretáva.

4 Štúdia o zrozumiteľnosti

Táto štúdia predstavuje programátorom regexy a kladie otázky na porozumenie. Porovnaním zrozumiteľnosti sémanticky ekvivalentných regexov, ktoré zodpovedajú rovnakému jazyku, ale majú rôzne syntaktické reprezentácie, sa snažíme identifikovať zápachy kódu. Táto štúdia bola realizovaná na portáli Amazon Mechanical Turk so 180 účastníkmi. Celkovo sa hodnotilo 60 regexov, čím sa vytvorilo 42 párov porovnávania regexov. Každý regexový vzor hodnotilo 30 účastníkov. [4]

String	'RR*'	Oracle	P1	P2	P3	P4
1	"ARROW"	✓	✓	✓	✓	✓
2	"qRs"	✓	✓	×	×	?
3	"R0R"	✓	✓	✓	?	-
4	"qrs"	×	✓	×	✓	-
5	"98"	×	×	×	×	-
	Score	1.00	0.80	0.80	0.50	1.00

✓ = match, ×= not a match, ? = unsure, – = left blank

Obr. 2: Príklad metrického porovnávania [4]

4.1 Metriky

Zrozumiteľnosť regexov meriame pomocou dvoch komplementárnych metrík, porovnávania a skladania. Tieto metriky sa označujú ako metriky zrozumiteľnosti. Pre hlbší pohľad na údaje s cieľom lepšie pochopiť faktory, ktoré ovplyvňujú zrozumiteľnosť, vypočítame aj dĺžku regexu a veľkosť DFA pre každý regex. [4]

4.1.1 Zodpovedanie

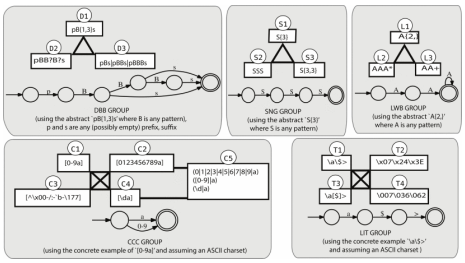
Pri danom vzore a množine reťazcov účastník kontrolou určí, ktoré reťazce sa budú zhodovať so vzorom. Pre každý reťazec existujú štyri možné odpovede: zodpovedá, nezodpovedá, nie je istý alebo je prázdny. Príklad z našej štúdie je uvedený na obrázku 2. [4]

Percentuálny podiel správnych odpovedí, bez zohľadnenia prázdnych miest a neistých odpovedí, predstavuje skóre zhody. Uvažujme napríklad regexový vzor "RR*, päť reťazcov uvedených v tabuľke I a odpovede štyroch účastníkov v stĺpcoch Pl,P2,P3 a P4. Oracle uvádza, že prvé tri reťazce sa zhodujú a posledné dva nie; Pl odpovedá správne na prvé tri reťazce a piaty, ale nesprávne na štvrtý, takže skóre zhody je 4/5=0,80. P2 si nesprávne myslí, že druhý reťazec sa nezhoduje, takže skóre je tiež 4/5=0,80. P3 označí tretí reťazec ako "neistý", takže celkový počet pokusov o zhodu je 4. P3 sa nesprávne vyjadrí o druhom a štvrtom reťazci, takže dosiahne skóre 2/4=0,50. V prípade P4 máme k dispozícii údaje len pre prvý a druhý reťazec, pretože ostatné tri sú prázdne. P4 označí druhý reťazec ako "neistý", takže sa pokúsil zodpovedať len jednu otázku; skóre za zhodu je 1/1=1,00. [4]

Do metriky boli zahrnuté prázdne miesta, pretože v štúdii sa občas vyskytli nevyplnené otázky. Neisté odpovede boli uvedené ako možnosť, aby nedošlo k skresleniu výsledkov slepým odhadom. Tieto situácie sa nevyskytovali veľmi často. Z 1 800 otázok (180 účastníkov * 10 otázok) bolo iba 1,8 % (32) ovplyvnených prázdnou alebo neistou odpoveďou (nikdy nie viac ako štyri z 30 odpovedí na vzor). [4]

4.1.2 Zloženie

Ak je daný vzor, účastník vytvorí reťazec, o ktorom si myslí, že sa mu zhoduje (otázka 7.F na obrázku 2). Ak je účastník presný, skóre zloženia je 1, inak 0. Napríklad vzhľadom na vzor (q4fab |ab) z našej štúdie sa reťazec "xyzq4fab"zhoduje a získa skóre 1, ale reťazec äcb"sa nezhoduje a získa skóre 0. [4]



Obr. 3: Otázky z jedného vzoru na jeden zásah [4]

Na určenie zhody medzi reťazcom a vzorom sa vzor skompiluje pomocou modulu re.compile v jazyku Python. Pomocou skompilovaného vzoru sa vytvorí inštancia re.RegexObject m. m.search() vráti inštanciu re.MatchObject m2 s reťazcom zadaným ako vstup do tejto funkcie. Ak m2 nie je None, potom je tento reťazec zhodný a má hodnotu 1; v opačnom prípade má hodnotu 0. [4]

4.1.3 Dĺžka regexu

Pri danom vzore sa dĺžka regexu vypočíta podľa dĺžky jeho literálneho reťazca. Napríklad regexy \072 a ab*c majú dĺžku štyri. [4]

4.1.4 Veľkosť DFA

Aby sme vypočítali veľkosť minimálneho DFA, na každom regexe spustíme brics aj Rex [6] a ručne skontrolujeme ich výsledky, aby sme zaručili ich správnosť. [4]

4.2 Dizajn

Túto štúdiu sme realizovali na platforme Amazon Mechanical Turk (MTurk), na ktorej žiadatelia vytvárajú úlohy ľudskej inteligencie (HIT), ktoré majú pracovníci dokončiť. [4]

4.2.1 Kvalifikácia pracovníka

Kvalifikovaní pracovníci museli správne odpovedať na štyri z piatich základných otázok regexu. Tieto otázky boli s výberom odpovede a pracovník mal analyzovať tieto vzory: a+, (r | z), \\\\\\\\\d, q* a [p-s]. [4]

4.2.2 Úlohy

Na základe vzorov v korpuse sme vytvorili 60 regexových vzorov, ktoré sme zoskupili do 26 skupín sémantickej ekvivalencie. V 18 skupinách boli dva regexy zamerané na rôzne hrany v triedach ekvivalencie. Ďalších osem skupín malo po tri regexy. Celkovo bolo vytvorených 42 dvojíc vzorov. Takto môžeme vyvodiť závery porovnaním reprezentácií, keďže vyhodnotené regexy boli sémanticky ekvivalentné. [4]

Na vytvorenie sémantických skupín sme vzali regex z korpusu, priradili sme ho k reprezentácii na obrázku 1, skrátili sme ho tak, aby obsahoval len o málo

viac než len funkciu, ktorá nás zaujíma, a potom sme vytvorili ďalšie regexy, ktoré sú sémanticky ekvivalentné, ale patria do iných uzlov triedy ekvivalencie. Napríklad sémantická skupina s regexmi ((q4f) 0, 1 ab, ((q4f) ?ab) a (q4fab |ab) patrí do D1, D2 a D3. Skupina s regexmi ([0-9] +) \. ([0-9] +) a (\d+) (\d+) je určená na vyhodnotenie hrany medzi C1 a C4. Poznamenávame, že ak by sme použili len regexy z korpusu, mali by sme v každom uzle regexy s rôznou sémantikou alebo s ďalšími jazykovými vlastnosťami, čo by sťažilo porovnávanie cieľových vlastností. [4]

		Average Matching		Average Composition	
	Df	F value	Pr(>F)	F value	Pr(>F)
dfa_size	1	7.632	0.0153 *	10.084	0.00674 **
len	1	3.325	0.0896 ·	0.001	0.98161
rep	15	2.062	0.0921 ·	1.224	0.35538
dfa_size:len	1	1.002	0.3339	1.384	0.25907
dfa_size:rep	14	0.709	0.7355	0.920	0.56075
len:rep	10	0.924	0.5397	0.599	0.79054
dfa_size:len:rep	3	1.163	0.3589	0.678	0.58002
Residuals	14				

$\cdot\alpha = 0.10$ $*\alpha = 0.05$ $**\alpha = 0.01$ $***\alpha = 0.001$

Obr. 4: 3-faktorová anova s priemernou presnosťou zhody alebo zloženia ako závislými premennými, pri zohľadnení reprezentácie (REP), veľkosti DFA (dfa _size) a dĺžky regexu (len) ako nezávislých premenných [4]

Pre každú z 26 sémantických skupín sme vytvorili päť reťazcov pre štúdiu, kde sa aspoň jeden zhodoval a aspoň jeden nezhodoval. Tie sa použili na výpočet metriky zhody. Po zhromaždení všetkých vzorov a zhodných reťazcov sme vytvorili úlohy pre účastníkov MTurk nasledovne: náhodne vybrať vzor z 10 z 26 sémantických skupín. Náhodne vyberte poradie týchto 10 vzorov, ako aj poradie zodpovedajúcich reťazcov pre každý vzor. Po pridaní otázky, v ktorej sa účastník pýta na zloženie reťazca, ktorý zodpovedá každému vzoru, sa tak vytvorí jedna úloha na MTurku, ako je príklad na na obrázku 2. Tento proces sa dokončil, kým sa každý zo 60 regexov neobjavil v 30 HIT, čo viedlo k celkovému počtu 180 jedinečných HIT. [4]

4.2.3 Implementácia

Pracovníci dostali 3,00 USD za úspešné absolvovanie jedného a len jedného HIT. Priemerný čas dokončenia prijatých HIT bol 682 sekúnd (11 min. 22 s). Celkovo bolo odmietnutých 54 HIT: 48 malo príliš veľa prázdnych odpovedí, štyri boli dvojnásobné podania od tých istých pracovníkov, jeden neodpovedal na otázky týkajúce sa zloženia a v jednom chýbali údaje z 3 otázok. Odmietnuté HIT-y boli vrátené na MTurk, aby ich vyplnili iní. [4]

4.2.4 Účastníci

Celkovo sa ho zúčastnilo 180 účastníkov. Väčšinu tvorili muži (83 %). Väčšina z nich mala aspoň vysokoškolské vzdelanie (72 %), boli aspoň trochu oboznámení s regexmi (87 %) a mali predchádzajúce skúsenosti s programovaním (84 %). [4]

4.3 Analýza

Na základe 30 odpovedí účastníkov sme pre každý regex vypočítali skóre zhody a zloženia. Priemerná analýza alebo priemerné zloženie sa vypočítalo spriemerovaním príslušných 26 - 30 hodnôt pre každú metriku pre každý zo 60 regexov (menej ako 30 hodnôt sa použilo, ak všetky odpovede v odpovedi na otázku o zhode boli kombináciou prázdnych a neistých). [4]

Z pôvodných 42 párov uvádzame skóre pre 41 párov. Kvôli konštrukčnej chybe boli vyhodnotené regexy, \. * a \. +. neboli sémanticky ekvivalentné (v prvom prípade chýba escape a malo by byť \. \. *), preto boli z údajov vynechané. Nakoniec sme analyzovali 58 regexov, ktoré pokrývajú 17 hrán z na obrázku 1. [4]

Aby sme lepšie pochopili, prečo môžu byť niektoré regexy zrozumiteľnejšie ako iné, pozrieme sa aj na vplyv reprezentácie z obrázku 1, dĺžky regexu a veľkosti DFA na metriky zrozumiteľnosti. Všimnite si, že pri tejto analýze zachovávame všetkých 60 regexov, pretože vlastnosti regexov skúmame jednotlivo. Vykonáme dve trojfaktorové analýzy variancií (ANOVA) s presnosťou porovnávania a presnosťou zloženia ako závislými premennými. Vykonávame aj korelačnú analýzu medzi týmito tromi faktormi a metrikami zloženia. Používame Spearmanovu koreláciu rangu a poradia, pretože nemáme žiadne apriórne znalosti o rozdelení faktorov. Keďže regresné reprezentácie sú kategorické údaje, sú z korelačnej analýzy vylúčené. [4]

4.4 Výsledky

ANOVA v tabuľke 2 ukazuje, že veľkosť DFA významne ovplyvňuje priemernú presnosť párovania aj priemerné zloženie pri α=0,05 a α=0,01. Dĺžka a zastúpenie z obrázku 1 každé z nich významne ovplyvňujú priemernú presnosť priradenia pri α=0,10. Keďže veľkosti DFA sa pri párových porovnaniach v rámci reprezentácie líšia, uvádzame naše výsledky porovnávania a skladania s použitím každej zo 41 dvojíc regexov samostatne, a nie súhrnne cez skúmané hrany tried ekvivalencie. [4]
V tabuľke 3 sú uvedené výsledky metrík porozumenia. Každý riadok predstavuje dvojicu regexov hodnotených účastníkmi štúdie. Reprezentácie pre regexy podľa obrázku 1 sú uvedené v stĺpci Hrana, podľa ktorého je tabuľka zoradená. Stĺpce Regex1 a Regex2 identifikujú regexy použité v štúdii, ktoré sa mapujú na prvú, resp. druhú reprezentáciu v stĺpci Edge. Match1 je priemerná zhoda pre Regex1 a Match2 je priemerná zhoda pre Regex2. Pomocou Mannovho-Whitneyho testu stredných hodnôt sa v nasledujúcom stĺpci sigM testuje, či existuje významný rozdiel medzi presnosťou. V stĺpci Comp1 je uvedené percento reťazcových odpovedí, ktoré boli skutočne správne priradené pomocou Regex1. V stĺpci Comp2 sú uvedené rovnaké informácie okrem Regex2. V nasledujúcom stĺpci sigC sa používa test dvoch podielov na zistenie, či sa percento účastníkov, ktorí správne zostavili reťazec pre Regex1, významne líši od percenta účastníkov, ktorí správne zostavili reťazec pre Regex2. [4]

Na ilustráciu uveďme dvojicu 16 v tabuľke 3. Jednou dvojicou regexov boli

([}{]) a (\|\) v C4 a C5 s priemerným skóre zhody 78,79 % a 70,33 % a priemerným skóre zloženia 50,00 % a 86,67 %. Rozdiel medzi skóre zloženia je významný pri α=0,01, avšak rozdiel medzi presnosťou nie je významný. V skutočnosti bolo zobrazenie C5 zrozumiteľnejšie v tom, že účastníci vedeli efektívnejšie zostaviť reťazec, ktorý by sa mu zhodoval, ale C4 je zrozumiteľnejšie v tom, že účastníci vedeli ľahšie určiť, ktorému z množiny reťazcov by sa zhodoval C4. Preto nie je ani jedna reprezentácia v stĺpci Hrany zvýraznená tučným písmom, pretože ide o konflikt. Ak obe metriky porozumenia označili preferovanú reprezentáciu, táto reprezentácia je zvýraznená tučným písmom (napr. C4 v dvojici 15). Väzby sa prerušia tak, že sa uprednostní druhá metrika. Napríklad v prípade dvojice 17 je nerozhodný výsledok v zložení, ale zhoda naznačuje uprednostnenie reprezentácie C5. Preto je C5 zvýraznená tučným písmom. [4]

Pair	Edge	Regex1	Regex2	Match1(%)	Match2(%)	SigM	Comp1(%)	Comp2(%)	SigC
1	**C1** – C2	`tri[a-f]3`	`tri[abcdef]3`	94.00	93.17		83.33	83.33	
2	**C1** – C2	`no[w-z]5`	`no[wxyz]5`	93.33	87.17		86.67	86.67	
3	C1 – **C3**	`no[w-z]5`	`no(w|x|y|z)5`	93.33	93.67		86.67	96.67	
4	C1 – **C4**	`([0-9]+)\.([0-9]+)`	`(\d+)\.(\d+)`	90.17	94.44		83.33	93.33	
5	**C1** – C4	`xgl([0-9]{1,3})%`	`xgl(\d{1,3})%`	82.67	81.33		76.67	66.67	
6	**C1** – C4	`[a-f]([0-9]+)[a-f]`	`[a-f](\d+)[a-f]`	91.17	83.33		80.00	70.00	
7	C1 – **C4**	`&([A-Za-z0-9_]+);`	`&(\w+);`	81.90	82.59		56.67	66.67	
8	**C1** – C4	`1q[A-Za-z0-9_][A-Za-z0-9_]`	`1q\w\w`	86.00	78.11		83.33	70.00	
9	**C1** – C4	`tuv[A-Za-z0-9_]`	`tuv\w`	89.17	86.00		83.33	70.00	
10	**C1** – C5	`tri[a-f]3`	`tri(a|b|c|d|e|f)3`	94.00	86.11		83.33	80.00	
11	C2 – C4	`[\t\r\f\n ]`	`[\s]`	82.99	92.41	·	3.33	0.00	
12	**C2** – C5	`tri[abcdef]3`	`tri(a|b|c|d|e|f)3`	93.17	86.11		83.33	80.00	
13	C2 – **C5**	`no[wxyz]5`	`no(w|x|y|z)5`	87.17	93.67		86.67	96.67	
14	C3 – C4	`[^0-9A-Za-z]`	`[\W_]`	64.50	61.00		46.67	53.33	
15	C3 – **C4**	`[^0-9]`	`[\D]`	58.00	73.33		63.33	73.33	
16	C4 – C5	`([}{])`	`(\{[\}])`	78.79	70.33		50.00	86.67	**
17	C4 – **C5**	`([:;])`	`(:|;)`	81.38	94.00		46.67	46.67	
18	**D1** – D2	`((q4f){0,1}ab)`	`((q4f)?ab)`	82.93	79.25		50.00	40.00	
19	**D1** – D2	`(dee(do){1,2})`	`(deedo(do)?)`	84.83	77.17		66.67	60.00	
20	D1 – **D3**	`((q4f){0,1}ab)`	`(q4fab|ab)`	82.93	84.50		50.00	60.00	
21	D1 – D3	`(dee(do){1,2})`	`(deedo|deedodo)`	84.83	90.00		66.67	63.33	
22	D2 – **D3**	`((q4f)?ab)`	`(q4fab|ab)`	79.25	84.50		40.00	60.00	
23	D2 – **D3**	`(deedo(do)?)`	`(deedo|deedodo)`	77.17	90.00	*	60.00	63.33	
24	L2 – L3	`zaa+`	`za+`	86.67	90.67		70.00	50.00	
25	L2 – **L3**	`RR+`	`R+`	86.00	91.56		66.67	66.67	
26	S1 – S2	`%([0-9A-Fa-f]{2})`	`%([0-9a-fA-F][0-9a-fA-F])`	77.78	73.44		50.00	60.00	
27	S1 – **S2**	`&d([aeiou]{2})z`	`&d([aeiou][aeiou])z`	91.28	95.34		83.33	83.33	
28	S1 – S2	`fa[lmnop]{3}`	`fa[lmnop][lmnop][lmnop]`	87.17	88.00		83.33	73.33	
29	T1 – T2	`xyz[_\[\]'\^\\]`	`xyz[\x5b-\x5f]`	77.78	78.67		86.67	56.67	*
30	**T1** – T2	`t[:;]+p`	`t[\x3a-\x3b]+p`	94.33	88.59		80.00	63.33	
31	T1 – T3	`(\$\{}\d+(:[^}]+\})`	`([$][{]}\d+(:[^}]+[}])`	81.61	75.18		63.33	73.33	
32	T1 – **T3**	`t\.\$+\d+\*`	`t[.][$]+\d+[*]`	88.67	94.00		56.67	73.33	
33	**T1** – T3	`\{\$(\d+\.\d)\}`	`[{][$](\d+[.]\d)[}]`	93.28	89.33		70.00	66.67	
34	**T1** – T4	`xyz[_\[\]'\^\\]`	`xyz[\0133-\0140]`	77.78	71.35		86.67	33.33	***
35	**T1** – T4	`t[:;]+p`	`t[\072\073]+p`	94.33	90.00		80.00	70.00	
36	**T1** – T4	`(\{[\}])`	`([\0175\0173])`	70.33	54.40	·	86.67	30.00	***
37	**T1** – T4	`([}{])`	`([\0175\0173])`	78.79	54.40	**	50.00	30.00	
38	**T1** – T4	`(:|;)`	`([\072\073])`	94.00	65.77	**	46.67	23.33	
39	**T1** – T4	`([:;])`	`([\072\073])`	81.38	65.77	·	46.67	23.33	
40	**T2** – T4	`xyz[\x5b-\x5f]`	`xyz[\0133-\0140]`	78.67	71.35		56.67	33.33	
41	T2 – **T4**	`t[\x3a-\x3b]+p`	`t[\072-\073]+p`	88.59	90.00		63.33	70.00	

·$\alpha = 0.10$ *$\alpha = 0.05$ **$\alpha = 0.01$ ***$\alpha = 0.001$

Obr. 5: Párové porovnávanie regexov. Každá hodnota porovnania alebo zloženia je vypočítaná na základe približne 30 dátových bodov od 30 účastníkov štúdie [4]

V prípade párov 16, 29, 34 a 36 je rozdiel v zložení významný pri α<0,05, čo naznačuje rozdiely v prospech C5 oproti C4, T1 oproti T2 a dvakrát v prospech T1 oproti T4. V prípade dvojíc 23, 37 a 38 je rozdiel v zhode významný pri α<0,05, čo naznačuje rozdiely uprednostňujúce D3 pred D2 a dvakrát uprednostňujúce T1 pred T4. Zaujímavé je, že v prípade dvojíc 16 a 29 sú síce rozdiely v zložení významné, ale medzi metrikami zloženia je rozpor. Je potrebné ďalšie skúmanie, aby sme pochopili, za akých okolností sú metriky vo vzájomnom kon-

flikte. Pripomeňme si, že účastníci mohli vybrať neistotu, či sa reťazec zhoduje so vzorom. Z hľadiska porozumenia to naznačuje určitú úroveň zmätku. Pre každý vzor sme spočítali počet odpovedí, ktoré obsahovali aspoň jednu neistotu. Celkovo najvyšší počet neistých odpovedí pochádzal zo vzorov T4 a T2, ktoré majú osmičkovú a šesťuholníkovú reprezentáciu znakov. Najmenší počet neistých odpovedí bol v L3 a D3. Tieto výsledky odrážajú analýzu zrozumiteľnosti, keďže T4 a T2 majú vo všeobecnosti nižšiu mieru zrozumiteľnosti a L3 a D3 vo všeobecnosti vyššiu. [4]

Hoci ANOVA naznačuje, že rozdiely v párovaní sú spôsobené všetkými tromi faktormi, reprezentáciou, veľkosťou DFA a dĺžkou regexu, nie je úplne jasné prečo. Rozptyl v zložení ovplyvňuje len veľkosť DFA. Medzi veľkosťou DFA a zložením existuje silná, pozitívna korelácia pri α=0,01 s ρ=0,354. Na prvý pohľad sa tento výsledok môže zdať kontraintuitívny, ale vzhľadom na to, že väčšie DFA môžu predstavovať obmedzenejšie regexové jazyky (t. j. jazyky, ktoré akceptujú menej reťazcov), môže byť pre ne jednoduchšie skladať reťazec. Keďže však skúmaný rozsah veľkosti DFA bol od dvoch do ôsmich uzlov, tieto výsledky sa nemusia zovšeobecniť na väčšie regexy. Žiadna z ostatných korelácií nie je významná s α<0,05. [4]

4.5 Zhrnutie

Na porovnávanie a skladanie má vplyv veľkosť DFA a na porovnávanie má vplyv aj dĺžka regexu a jeho reprezentácia, čo potvrdzuje, že reprezentácia regexu má vplyv na porozumenie. Čím väčší bol DFA, tým ľahšie bolo pre komunitu generovať reťazce, ktoré mu zodpovedajú. Zdá sa tiež, že existuje jasný trend uprednostňovania T1 pred T4. Uprednostňované sú aj reprezentácie D3 a C5. Hoci sa uprednostňuje porovnanie C1 oproti C2, C4 a C5, žiadne z nich nie je významné. [4]

5 Záver

Záver.

Literatúra

[1] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. Mutrex: A mutation-based generator of fault detecting strings for regular expressions. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 87–96, 2017.

[2] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[3] Carl Chapman and Kathryn T. Stolee. Exploring regular expression usage and context in python. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, July 2016.

[4] Carl Chapman, Peipei Wang, and Kathryn T. Stolee. Exploring regular expression comprehension. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, October 2017.

[5] Louis G. Michael, James Donohue, James C. Davis, Dongyoon Lee, and Francisco Servant. Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, November 2019.

[6] Margus Veanes, Peli de Halleux, and Nikolai Tillmann. Rex: Symbolic regular expression explorer. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 498–507, 2010.

[7] Peipei Wang, Gina R. Bai, and Kathryn T. Stolee. Exploring regular expression evolution. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, February 2019.

[8] Peipei Wang and Kathryn T. Stolee. How well are regular expressions tested in the wild? In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, October 2018.