Exploring Regular Expression Feature Usage in Practice and the Impact on Tool Design

Carl Chapman and Kathryn T. Stolee
Department of Computer Science
Iowa State University
{carl1978, kstolee}@iastate.edu

***Abstract*—Regular expressions are used frequently in programming languages for form validation, ad-hoc file searches, and simple parsing. Due to the popularity and pervasive use of regular expressions, researchers have created tools to support their creation, validation, and use. Each tool has made design decisions about which regular expression features to support, yet, there does not exist an empirical study of regular expression feature usage to inform these design decisions.**

In this paper, we explore regular expression feature usage, focusing on how often features are used and the diversity of regular expressions from syntactic and semantic perspectives. To do this, we analyzed about 4000 open source Python projects from GitHub. Our results indicate that the most commonly used regular expression features are supported by popular research tools, and explores the use

and that programmers frequently reinvent the wheel by writing identical or nearly identical regular expressions in different ways.

I. Introduction

Regular expressions are used extensively in many programming languages, for example, to search text files [3], in form validation, and for XYZ.

II. Motivation

Bugs related to regular expressions are common, resulting in tens of thousands of bug reports [4].

With regexes, there is a common saying 'now you have two problems'

III. Related Work

A. Research on Regular Expressions

Visual debugging of regular expressions [1]

Static analysis to reduce errors in building regular expressions by using a type system to identify errors like `PatternSyntaxExceptions` and `IndexOutOfBoundsExceptions` at compile time [4].

B. Research on Regular Expressions

Visual debugging of regular expressions [1]

C. Research that Depends on Regular Expression Usage

Regular expressions are used as queries in a data mining framework [2]

IV. Study

To understand how programmers use regular expressions in Python projects and the syntactic and semantic diversity among the regular expressions, we scraped X projects from GitHub, as described in Section IV-A. Next, we logged all unique regular expressions

We aim to answer the following research questions:

RQ1: How frequently are regexes used in python projects?

To address this question, we measure how often calls to the `re module` are made per file and per project in Python projects.

RQ2: How is the `re module` used in python projects?

To address this research question, we measure the frequency of usage for calls to the `re.compile` and `re.search`, `re.match`, `re.split`, `re.findall`, `re.finditer`, `re.sub`, `re.subn` in Python projects scraped from GitHub.

RQ3: Which regex language features are used most commonly in python?

Regex features are components of the regex language, such as capture groups, literals, and the kleene star. To measure feature usage, we use the X library, as described in Section IV-C.

RQ4: What is the impact of *not* supporting various regex features on tool designers and users?

TODO: clean this up Use semantic analysis to illustrate the impact of missing features on a tool's applicability. Since our semantic analysis is based on Rex, we use syntactic analysis to observe the impact of not supporting various features on this, and other, research.

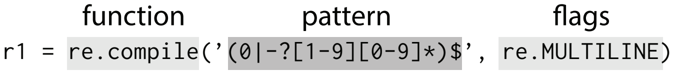
We map the regex features to each of four research tools that are commonly used for regular expressions research. To address this research question, we looked at the most popular regular expression features that Rex does not support. As Rex is used for our semantic analysis in RQ4, we were interested in the impact of not supporting these features.

A. Building the Corpus

We used the github api to page through all repositories, cloning projects that contain Python code, stopping when the scraper ran out of memory[1]. For each project, we used

[1] www.details.#better_parser

Astroid[X] to build the AST of each Python file and find uses of Python's 're' module. Here is an example of one regex *usage*, with key components labeled:



```
r1 = re.compile('(0|-?[1-9][0-9]*)$', re.MULTILINE)
```

Fig. 1. example of one regex usage

Within each project, duplicate usages (same function, pattern and flags) within the same file (same relative path) were ignored. Using git, each project was scanned at 20 evenly-spaced commits (or all commits if there were less than 20) in its history. We observed and recorded 53,894 regex usages in 3,898 projects.

B. Selecting Patterns

Our analysis focuses on the patterns found, so we ignored the 12.7% of usages using flags that can alter regex behavior. An additional 6.5% of usages contained patterns that could not be compiled because the pattern was non-static (used some runtime variable), or because of other unknown parsing failures.

The remaining 80.8% (43,525) usages were collapsed into 14,113 distinct pattern strings. The resulting set of patten strings were parsed using an antlr-based, open source PCRE parser released by Bart Kiers[2]. This parser was unable to support 0.5% (76) of the patterns due to unsupported unicode characters. Another 0.2% (27) of the patterns used regex features that we have chosen to exclude in this study[3]. The 13,912 distinct pattern strings that remain were each assigned a weight value equal to the number of distinct projects the pattern appeared in. We will refer to this set of weighted, distinct pattern strings as the *collection*.

C. Analyzing Features

After picking four large regex research projects, the big table with the features was created in order to decide which unsupported features are used most often. Our semantic analysis is dependent on the use of Rex to generate strings so we can identify semantically related clusters. For three common features unsupported by Rex, we rely on syntactic analysis to determine similarity among regular expressions containing those features. For those features supported by Rex, we cluster the regular expressions based on semantic diversity.

1) Syntactic Diversity: For the negative perspective, we picked three features: LZY, NCG, WNW that are unsupported by Rex and other projects. For each of these features, we created a subset of the *collection* where all the patterns contain that feature. Then we used syntactic analysis...to create a similarity matrix. We then used markov clustering [X] (MCL) to find clusters in the subset. We used these clusters to assist our manual search for some common use cases for the unsupported feature.

```
for each row i:
    obtain set of Rex-generated strings Ri from pattern at index i
    sRi = size of Ri
    for each col j:
        Nij = number of strings in Ri matched by pattern at index j
        M[i][j] = Nij/sRi
G = empty graph
for each row i:
    for each col j:
        SIMij = (M[i][j]+M[j][i])/2
        if SIMij ¿ 0.75:
            add edge (i,j)=SIMij to G
```

Fig. 2. Constructing Similarity Graph

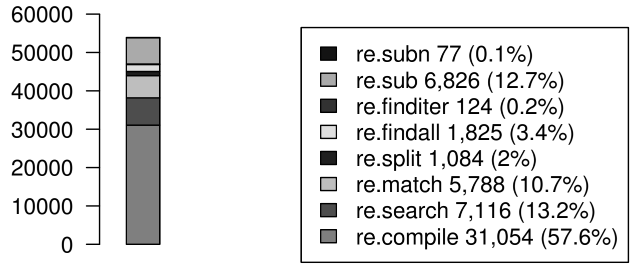


Fig. 3. How often are the 8 re functions used? (RQ2)

2) Semantic Diversity: For the positive perspective, we created another subset of patterns (XYZ patterns) where Rex was able to generate strings that the pattern matched. We then created a similarity graph with weighted, undirected edges as shown in Figure 2.

Again we used MCL to find clusters that aided a manual search for use cases strongly associated with particular features.

V. Results

A. Context and Corpus

1) Saturation: Although 42.2% of the projects observed had at least one regex usage, only 11.2% of the files observed had at least one regex usage.

From the above figure/table, we see that on average each project had 2 files containing any regex usage, out of an average of 6 files. Each of the files that did have a regex usage had an average of 1 regex usages. Because we scanned 3,898 projects, we would expect to have seen 23,388 regex usages, which is lower than the actual 53,894 usages observed.

2) Regex Functions and Flags: As seen in Figure 3 The 'compile' function encompasses 57.6% of all usages, even though every compiled regex object can only be used by calling other functions. (TODO-Why?)

87.3% of all regex usages did not use a flag or specified a non-behavioral flag (default or debug). Of all behavioral flags used, ignorecase (43.8%) and multiline (25.8%) were the most frequently used. It is also worth noting that although

[2] https://github.com/bkiers/pcre-parser

[3] www.details.#thistopic

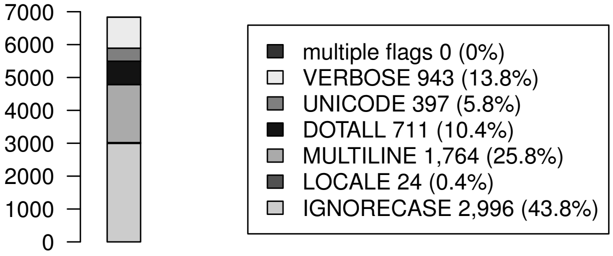


Fig. 4. Which behavioral flags are used? (RQ2)

TABLE I.

pattern	weight
'\\s+'	181
'\\s'	78
'\\d+'	70
'[\\x80-\\xff]'	69
'\nmd5_data = {\n([^}]+)}'	69
'\\\\(.)'	67
'([\\\\"]|[^\\ -~])'	66
'(-?(?:0|[1-9]\\d*))(\\.\\d+)?([eE][-+]?\\d+)?'	61
'[^]]+?\\] +([0-9.]+): (\\w+) <-(\\w+)'	60
'.*rlen=([0-9]+)'	57

multiple flags can be combined using a bitwise or, this was never observed. (remove this last part if it is observed later)

3) General Characteristics of Regexes Found: ...TODO

4) Top 10 Regex Patterns by weight:

5) All Features: Literal tokens were found in (TODO) 101% of patterns, and accounted for 75% of all tokens. Excluding literal tokens and features that were not present in any pattern, the following stats...make a sentence, these are some stats about the features:

some more text, IDK

OK now that is all for section 2. Now in section 3 I want to look at clustering by string similarity using mcl clustering algorithm. Here are the top 6 clusters using various string similarity metrics:

TABLE III.

pair	example from corpus	nTimes
CG::ADD	'(:+)'	4189
CG::KLE	'(:)*'	3983
ANY::KLE	'.*'	3709
CG::ANY	'(.)'	3160
CCC::CG	"(['])"	2665
CCC::ADD	'[ ]+'	2612
RNG::CCC	'[A-Z]'	2567
ADD::KLE	'-*(.+)'	2476
WSP::KLE	'\\s*'	2207
END::STR	'^$'	2156

TODO - multiple boxplots for all 5-6 demonstrating cluster size and then also have # of clusters, pick smallest number of clusters and then use that.

VI. Discussion

...only 11.2% of the files observed had at least one regex usage. This indicates that regex usage may usually be concentrated in just a few files.

Fun fact: while creating similarity matrix, row 5464 took 2 hours, or almost 1 second per cell avg, only suffering 18 timeouts (1.2 secs). What is this pesky pattern?

VII. Conclusion

Acknowledgment

This work is supported in part by NSF SHF-1218265, NSF SHF-EAGER-1446932, and the Harpole-Pentair endowment at Iowa State University.

References

[1] F. Beck, S. Gulan, B. Biegel, S. Baltes, and D. Weiskopf. Regviz: Visual debugging of regular expressions. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 504–507, New York, NY, USA, 2014. ACM.

[2] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: Discovering and exploiting relationships in software repositories. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 125–134, New York, NY, USA, 2010. ACM.

[3] C. L. A. Clarke and G. V. Cormack. On the use of regular expressions for searching text. *ACM Trans. Program. Lang. Syst.*, 19(3):413–426, May 1997.

[4] E. Spishak, W. Dietl, and M. D. Ernst. A type system for regular expressions. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, FTfJP '12, pages 20–26, New York, NY, USA, 2012. ACM.

TABLE II. REGEX FEATURE USAGE IN PYTHON PROJECTS AND SUPPORT BY POPULAR REGULAR EXPRESSION RESEARCH TOOLS (RQ3, RQ5)

rank	code	description	example	brics	hampi	Rex	RE2	nPatterns	% patterns	nFiles	%files	nProjects	% projects
1	ADD	one-or-more repetition	z+	●	●	●	●	6,122	44	9,330	50.3	1,209	73.5
2	CG	a capture group	(caught)	●	●	●	●	7,248	52.1	9,759	52.6	1,197	72.8
3	KLE	zero-or-more repetition	.*	●	●	●	●	6,104	43.9	8,323	44.9	1,100	66.9
4	CCC	custom character class	[aeiou]	●	●	●	●	4,581	32.9	7,808	42.1	1,027	62.4
5	ANY	any non-newline char	.	●	●	●	●	4,708	33.8	6,394	34.5	1,006	61.2
6	RNG	chars within a range	[a-z]	●	●	●	●	2,698	19.4	5,196	28	849	51.6
7	STR	start-of-line	^	○	●	●	●	3,660	26.3	5,622	30.3	847	51.5
8	END	end-of-line	$	○	●	●	●	3,258	23.4	5,549	29.9	828	50.3
9	NCCC	negated CCC	[^qwxf]	●	●	●	●	1,970	14.2	4,027	21.7	777	47.2
10	WSP	\t \n \r \b \f or space	\s	○	●	●	●	2,908	20.9	4,812	25.9	764	46.4
11	OR	logical or	a|b	●	●	●	●	2,161	15.5	4,039	21.8	711	43.2
12	DEC	any of: 0123456789	\d	○	●	●	●	2,385	17.1	4,366	23.5	694	42.2
13	WRD	[a-zA-Z0-9_]	\w	○	●	●	●	1,457	10.5	3,004	16.2	652	39.6
14	QST	zero-or-one repetition	z?	●	●	●	●	1,922	13.8	3,821	20.6	647	39.3
15	LZY	as few reps as possible	z+?	○	●	○	●	1,318	9.5	2,291	12.4	606	36.8
16	NCG	group without capturing	a(?:b)c	○	●	○	●	813	5.8	1,748	9.4	404	24.6
17	NCG	named capture group	(?P<name>x)	○	●	○	●	934	6.7	1,517	8.2	354	21.5
18	SNG	exactly n repetition	z{8}	●	●	●	●	623	4.5	1,359	7.3	340	20.7
19	NWSP	any non-whitespace	\S	○	●	●	●	490	3.5	788	4.2	271	16.5
20	DBB	$n \leq x \leq m$ repetition	z{3,8}	●	●	●	●	384	2.8	692	3.7	242	14.7
21	NLKA	sequence doesn't follow	a(?!yz)	○	○	●	○	137	1	503	2.7	184	11.2
22	NWRD	non-word chars	\W	○	●	●	●	97	0.7	315	1.7	169	10.3
23	LWB	at least n repetition	z{15,}	●	●	●	●	97	0.7	337	1.8	167	10.2
24	WNW	word/non-word boundary	\b	○	○	○	●	248	1.8	438	2.4	166	10.1
25	LKA	matching sequence follows	a(?=bc)	○	○	○	○	114	0.8	360	1.9	159	9.7
26	OPT	options wrapper	(?i)CasE	○	●	○	●	232	1.7	378	2	154	9.4
27	NLKB	sequence doesn't precede	(?<!x)yz	○	○	○	○	102	0.7	321	1.7	139	8.4
28	LKB	matching sequence precedes	(?<=a)bc	○	○	○	○	82	0.6	262	1.4	120	7.3
29	ENDZ	absolute end of string	\Z	○	○	○	●	91	0.7	154	0.8	94	5.7
30	BKR	match the i^{th} CG	\1	○	○	○	○	60	0.4	129	0.7	84	5.1
31	NDEC	any non-decimal	\D	○	●	●	●	36	0.3	92	0.5	58	3.5
32	BKRN	references NCG	\g<name>	○	●	○	○	17	0.1	44	0.2	28	1.7
33	VWSP	matches U+000B	\v	○	○	●	●	13	0.1	16	0.1	15	0.9
34	NWNW	negated WNW	\B	○	○	○	●	4	0	11	0.1	11	0.7