ISCAS 中国科学院软件研究所学术年会'2021 暨计算机科学国家重点实验室开放周
学术论文

FlashRegex: Deducing Anti-ReDoS Regexes from Examples
FlashRegex:从示例中推断抗ReDoS的正则表达式

Yeting Li, Zhiwu Xu, Jialun Cao, Haiming Chen, Tingjian Ge, Shing-Chi Cheung, Haoren Zhao
The 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020)
Yeting Li, 15801685206, liyt@ios.ac.cn

Background & Motivation

- Regular expressions (regexes) are widely used in different fields of computer science.
- Regexes are hard for users/experts to understand and compose, thus that is why automatic regex synthesis/repair techniques are proposed.
- However, existing works do not consider the issue of ReDoS-vulnerability in regex synthesis/repair.

This motivates the need for techniques that can automatically not only synthesize ReDoS-invulnerable regexes, but also help repair incorrect and/or ReDoS-vulnerable regexes.

Challenges

Huge search space.

For both regex synthesis and repair, the search space is extremely large because practical regexes: (i) are large, (ii) operate over very large alphabet size, and (iii)contain various operators.

Difficulty of synthesizing/repairing regexes from examples.

The problem of ReDoS-invulnerable regex synthesis- and repair-from-examples is shown to be an NP-hard problem.

Difficulty of prevention of ReDoS-vulnerabilities.

Instead of avoiding certain patterns of regexes as prerequisites of ReDoS attacks, developers or users expect to address ReDoS-vulnerability from its root cause---the ambiguity of regexes. Indeed, ambiguity can lead to catastrophic backtracking that causes ReDoS attacks.

How to avoid generating these ambiguous regexes effectively is a distinct merit of our work over existing techniques.

Approach

Regex Synthesis.

The first problem we target at is to synthesize anti-ReDoS regexes from positive and negative examples. Given a positive example set S^+ and a negative example set S^-, the goal is to learn a regex r such that (i) $S^+ \subseteq \mathcal{L}(r)$ and $S^- \cap \mathcal{L}(r) = \phi$; and (ii) r is invulnerable to ReDoS attacks.

The key of our solution to tackle this problem is the use of deterministic regexes. In particular, our solution consists of two steps, namely, *k-OA synthesis* and *regex extraction*.

- *k-OA synthesis* takes the given positive and negative examples as input and tries to synthesize a deterministic k-OA from the examples via SAT.
- After that, *regex extraction* marks the synthesized deterministic k-OA and extracts a marked regex from the marked k-OA, by calling the procedure *Soa2Sore* used in Freydenberger and Kotzing's work.

Algorithm 1: SynRegex

Input: a positive set S^+ and a negative set S^-
Output: a deterministic regex r with $S^+ \subseteq L(r)$ and $S^- \cap L(r) = \varnothing$ if solvable for k_{max}, or *null* otherwise

```
for k = 1 to k_max do
    A ← synKOA*(S+, S−, k)
    if A ≠ null then
        r ← GenRegex(A)
        if r ≠ null then
            return r
return null
```

Algorithm 2: *fast*KOA*

Input: a positive set S^+, a negative set S^-, a value k
Output: a deterministic k-OA $\mathcal{A}$ or *null*

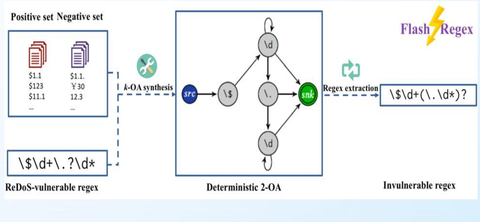
```
initialize the formula set D ← ∅
add Deter(k) to D
for w ∈ S+ do
    add Pos_a(w, k) ∧ Pos_b(w, k) to D
for w ∈ S− do
    add Neg(w, k) to D
Put D in a SAT solver
if D is satisfiable then
    convert Boolean variables (matrix) to a k-OA A
    A ← prune A w.r.t. S+ and S−
    return A
else return iKOA*(S+, S−, k);
```

Regex Repair.

The second problem is to repair an incorrect (*i.e.*, rejecting some examples in S^+ or accepting some examples in S^-) or ReDoS-vulnerable regex r (*i.e.*, ReDoS-prone) with respect to a positive example set S^+ and a negative example set S^-.

The idea is quite similar to regex synthesis: to use deterministic regexes when possible.

- Starts with a deterministic k-OA, which is converted from the given regex r.
- Then it searches for a k-OA which can accept the most positive examples and/or reject the most negative ones among those in the neighborhood (*i.e.*, those with one different value from the current k-OA).
- Keeps on searching, until it finds a deterministic k-OA that accepts all the positive examples and rejects all the negative ones, or the number of iterations exceeds a given number (set to be 200 in this paper).



Evaluation

RQ1. Evaluation of regex synthesis.
RQ2. Evaluation of incorrect regex repair.
RQ3. Evaluation of ReDoS-invulnerable regex repair.

The effectiveness and efficiency of ReDoS-invulnerable regex repair

No.	Source	SL (Sub-)Regex	RFixer			FlashRegex		
			Repaired (Sub-)Regex	Time (s)	RV	Repaired (Sub-)Regex	Time (s)	RV
#1	OWASP	(a|aa)+	(a|aa)+	0.098	V	a+	0.596	I
#2	OWASP	(a|a?)+	(a?)+	0.133	I	a+	0.028	I
#3	OWASP	([a-zA-Z]+)*	([a-zA-Z]+)*	0.057	V	([a-zA-Z])*	0.059	I
#4	StackOverflow	(x+x*)+y	(x+)+y	10.289	V	xx+y	0.183	I
#5	snyk	(\w\d\+)+C	(\w\+\d\+)+C	0.176	V	([A-Za-z_]+\d)+C	0.058	I
#6	RegExLib	(\d+(,\d+)+)+	(\d+(,\d+)+)+	0.196	V	\d+(,\d+)+	0.427	I
#7	RegExLib	([0-9a-f]+\d+)+	([0-9a-f]+\d+)+	0.204	V	([[a-f]+\d)[\d]+	0.574	I
#8	RegExLib	(\d+[\d+\.\d+])+	(\d+[\d+\.\d+])+	0.158	V	(\.?\d)+	0.040	I
#9	Davis et al. [14]	\s+#?\s+	\s+#?\s+	0.139	V	\s+(#\s+)?	0.249	I
#10	Davis et al. [14]	(\n\s+)+	(\n\s+)+	0.004	V	\n\s+	0.052	I
#11	Davis et al. [14]	[$_a-z]+([$_a-z0-9-]+	[$_a-z]+([$_a-z0-9-]+	0.003	V	[$_a-z][$_a-z0-9-]+	0.061	I
#12	CVE-2009-3277	((a{1,2}){1,2}){1,10}	((a{1,2}){1,2}){1,10}	15.763	V	a{1,40}	8.555	I
#13	CVE-2016-4055	A(B|C+)+D	A(B+|C+)+D	0.162	V	A(B|C)+D	0.063	I
#14	CVE-2017-15010	([^=;]+)\s+=\s+([^\n\r\0]+)	([^=;\s]+)\s+=\s+([^\n\0]+)	31.534	I	([^=;\s]+)\s+=\s+([^\n\0]+)	5.484	I
#15	CVE-2017-16098	\s+=\s+("")? +([\w\-]+)	\s+=\s+("")? +([\w\-]+)	3.218	V	\s+=\s+([^) +)?([\w\-]+)	20.125	I
#16	CVE-2017-16137	\s+\n\s+	[\f\r\t\v]+\n\s+	0.368	I	[\f\r\t\v]+\n\s+	0.543	I
#17	CVE-2017-18214	(\s+?[\u0600-\u06FF]+)(1,2)	(\s+?[\u0600-\u06FF]+)(1,2)	3.270	V	\s+[\u0600-\u06FF]+(\s+[\u0600-\u06FF]+)?	23.074	I
#18	CVE-2018-3737	([\n \t]+([^\n]+))?	([\n \t]+([^\n \t]+))?	183.469	V	([\n \t]+([^\n \t]+))?	0.375	I
#19	CVE-2019-17592	(\-|\+)?([1-9]+[0-9]+)	(\-|\+)?([1-9]+[0-9]+)	15.936	V	(\-|\+)?[1-9]\d+	4.305	I
#20	CVE-2020-5243	*([^;]+)+ *	*([^;]+)+ *	1.406	I	*([^;]+)+ *	2.102	I

The effectiveness and efficiency of incorrect regex synthesis

Benchmarks	Bin-Syn-Regex				Multi-Syn-Regex			
Technique	#Sol (%)	#CSol (%)	#Vul	Avg. Time (s)	#Sol (%	#CSol (%)	#Vul	Avg. Time (s)
RegexGenerator++	-	-	-	-	50 (100%)	3 (6%)	0	198.0
GP-RegexGolf	-	-	-	-	50 (100%)	7 (14%)	4	3889.6
AlphaRegex	50 (100%)	50 (100%)	21	7.6	-	-	-	-
FlashRegex-Exact	50 (100%)	50 (100%)	0	3.3	38 (76%)	38 (100%)	0	5.3
FlashRegex-LCS	36 (72%)	36 (100%)	0	1.1	29 (58%)	29 (100%)	0	3.4
FlashRegex	50 (100%)	50 (100%)	0	1.9	38 (76%)	38 (100%)	0	4.0

The effectiveness and efficiency of incorrect regex repair

Benchmarks	Pos-Rep-Regex				Pos-Neg-Rep-Regex			
Technique	#Sol (%)	#CSol (%)	#Vul	Avg. Time (s)	#Sol (%	#CSol (%)	#Vul	Avg. Time (s)
Rebele et al [44]	50 (100%)	50 (100%)	0	0.2	-	-	-	-
RFixer	35 (70%)	35 (100%)	3	2.4	1,611 (75.67%)	1,611 (100%)	349	9.3
FlashRegex	35 (70%)	35 (100%)	0	1.5	1,948 (91.50%)	1,948 (100%)	0	1.6

Summary to RQ1: **FlashRegex can synthesize regex efficiently, correctly and safely.** The results also confirmed the lack of focus on ReDoS-vulnerability in previous works, thus making further repair a necessity.
Summary to RQ2: **FlashRegex can repair incorrect regex efficiently, correctly and safely.** The efficiency is not affected significantly by negative examples, and the regex after repair is free from ReDoS-vulnerability.
Summary to RQ3: **FlashRegex can repair ReDoS-vulnerable regex efficiently and correctly.** The experiment also indicates the incapability of existing work for repairing ReDoS-vulnerable regex.

Conclusion

We propose **a PBE framework, FlashRegex,** which provides three core functionalities including regex synthesis, incorrect regex repair, and ReDoS-vulnerable regex repair. Ours is the first framework that integrates the synthesis and repair of regexes **with the awareness of ReDoS-vulnerabilities**.