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

TransRegex: Multi-modal Regular Expression Synthesis by Generate-and-Repair

TransRegex: 基于生成和修复的多模态正则表达式合成

Yeting Li, Shuaimin Li, Zhiwu Xu, Jialun Cao, Zixuan Chen,
Yun Hu, Haiming Chen, Shing-Chi Cheung

The 43rd IEEE/ACM International Conference on Software Engineering (ICSE 2021)

Yeting Li, 15801685206, liyt@ios.ac.cn

Background & Motivation

- Regular expressions (abbrev. regexes) have been widely used in different fields of computer science due to high effectiveness and accuracy.
- Unfortunately, despite their popularity, regexes can be difficult to understand and compose even for experienced programmers.
- To alleviate this problem, prior research has proposed techniques to automatically generate regexes.

Existing Solutions

NLP-based techniques.

- Can only generate regexes similar in shape to the training data.
- Impeded by the ambiguity and imprecision of NL even for stylized English.

Example-based techniques.

- *Rely on high quality examples provided by users.*
- *The synthesized regexes may be under- or over-fitting.*

NLP-and-example-based techniques.

- *The use of advanced NLP-based techniques can reduce the amount of required (characteristic) examples meanwhile alleviate the amount of effort from users;*
- *While the use of examples can effectively disambiguate or correct errors in the descriptions.*
- *There have been recent attempts in this direction, in which they first translated the NL description into a sketch, then searched the regex space defined by the sketch guided by the given examples.*
- *However, the forms of translated sketches are restricted. This prevents regexes from being synthesized correctly when the generated sketches are inappropriate (e.g., logically-incorrect).*

Approach

Observation.

We observe that most of the incorrect regexes generated by NLP-based techniques are very similar to the target regexes with subtle differences, and can be made equivalent to the target regexes with only minor modifications (e.g., reordering/revising characters or quantifiers).

This motivates us to view the NLP-and-example-based regex synthesis problem as the problem of NLP-based synthesis with regex repair, and develop the first framework, TransRegex, to leverage both NL and examples for regex synthesis by using NLP-based and regex repair techniques.

Algorithm 1: TRANSREGEX

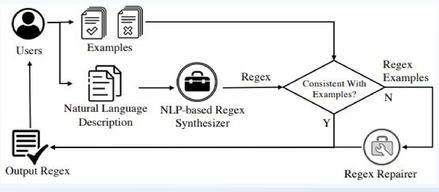
Input: a natural language description $\mathcal{NL}$, positive examples $\mathcal{P}$, and negative examples $\mathcal{N}$
Output: a regex

```
r ← S2RE(NL);
if P ⊆ L(r) and N ∩ L(r) = ∅ then return r ;
else
    r ← SYNCORR(r);
    if P ⊆ L(r) and N ∩ L(r) = ∅ then return r ;
    else return RFIXER(r);
```

Algorithm 2: SYNCORR

Input: positive examples $\mathcal{P}$, negative examples $\mathcal{N}$, and an incorrect regex r_0
Output: a regex

```
for l_max = 2 to 0 do
    r ← preprocess(r_0, l_max);
    while the stop conditions not meet do
        N(r) ← apply the transformations on r in parallel;
        r_tr ← unpreprocess(N(r), l_max);
        if f(r_tr, P, N) > f(r, P, N) then
            if f(r_tr, P, N) == 1 then return r_tr ;
            else r ← preprocess(r_tr, l_max);
        else break;
return r_0;
```



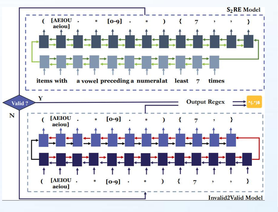
The Main Algorithm

- In the first step, NLP-based regex synthesis takes the given NL description as input and tries to synthesize a regex from NL description via an NLP-based synthesizer.
- After that, if the synthesized regex is consistent with the given examples, then TransRegex outputs the regex. Otherwise, the example-guided regex repair modifies this synthesized regex based on the provided examples by an example-guided repairer, and returns the repaired regex.

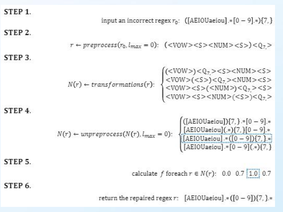
A pair of a description and examples

Natural Language Description $\mathcal{NL}$	
items with a vowel preceding a numeral at least 7 times	
Positive Examples $\mathcal{P}$	Negative Examples $\mathcal{N}$
E18043699	u.
U530136382	jz;B
U65972791327	o45
U82433805	FBcW
i3390716928	I4k,S
O789821610	U
U4765749255	I$#].
E6204251	A
e6868266	uV
O50693106874	o20m3u5817

The process of the algorithm S2RE



The process of the algorithm SynCorr



Evaluation

RQ1. Can S2RE model generate correct and valid regexes from NL descriptions?
RQ2. Can SynCorr repair incorrect regexes from examples?
RQ3. Can TransRegex synthesize regexes accurately?
RQ4. Can TransRegex synthesize regexes efficiently?

The Number of Successful Repairs by SynCorr and RFixer on Three Datasets

Approach	KB13	NL-RX-Turk	Structured Regex
RFIXER	25/45 (55.6%)	780/930 (83.9%)	245/712 (34.4%)
SYNCORR	30/45 (66.7%)	785/930 (84.4%)	346/712 (48.6%)
RFIXER + SYNCORR	35/45 (77.8%)	895/930 (96.2%)	387/712 (54.4%)

The DFA-equivalent Accuracy on Three Datasets.

Approach	KB13	NL-RX-Turk	Structured Regex
SEMANTIC-UNIFY	65.5%	38.6%	1.8%
DEEP-REGEX (Locascio et al.)	65.6%	58.2%	23.6%
DEEP-REGEX (Ye et al.)	66.5%	60.2%	24.5%
SEMREGEX	78.2%	62.3%	—
SOFTREGEX	78.2%	62.8%	28.2%
S_2RE	78.2%	62.8%	28.5%
DEEP-REGEX (Ye et al.) + EXS	77.7%	83.8%	37.2%
GRAMMARSKETCH+ MLE	68.9%	69.6%	—
DEEPSKETCH + MLE	84.0%	85.2%	—
DEEPSKETCH + MML	86.4%	84.8%	—
TRANSREGEX (S_2RE + SYNCORR)	92.7%	94.2%	63.3%
TRANSREGEX (S_2RE + RFIXER)	90.3%	94.0%	53.1%
TRANSREGEX (S_2RE + SYNCORR + RFIXER)	**95.6%**	**98.6%**	**67.4%**

The Number of Valid Regexes Generated by the Three NLP-based Models

Approach	KB13	NL-RX-Turk	Structured Regex
DEEP-REGEX (Locascio et al.)	205 (99.5%)	**2500 (100%)**	494 (49.6%)
SOFTREGEX	204 (99.1%)	**2500 (100%)**	902 (90.6%)
S_2RE	**206 (100%)**	**2500 (100%)**	**996 (100%)**

Average running time per benchmark on three datasets

Approach	KB13	NL-RX-Turk	Structured Regex
DEEP-REGEX (Locascio et al.)	2.621 s	1.104 s	2.108 s
S_2RE	3.578 s	1.656 s	3.313 s
TRANSREGEX (S_2RE + SYNCORR)	4.958 s	3.085 s	8.624 s
TRANSREGEX (S_2RE + RFIXER)	5.821 s	4.737 s	22.460 s
TRANSREGEX (S_2RE + SYNCORR + RFIXER)	6.688 s	4.011 s	13.737 s

Summary to RQ1: S2RE can achieve similar or better accuracy than the state-of-the-art NLP-based models. **Meanwhile, S2RE can synthesize more valid regexes.**
Summary to RQ2: SynCorr can more effectively repair regexes compared with the state-of-the-art tool RFixer.
Summary to RQ3: TransRegex can achieve higher accuracy than the NLP-based works with **17.4%, 35.8% and 38.9%,** and the state-of-the-art multi-modal works with **10% to 30% higher accuracy** on all three datasets.
Summary to RQ4: TransRegex can synthesize regex efficiently, especially when considering together with accuracy.

Conclusion

We propose an automatic framework **TransRegex**, for synthesizing regular expressions from both natural language descriptions and examples. To the best of our knowledge, TransRegex is the first to treat the NLP-and-example-based regex synthesis problem as **the problem of NLP-based synthesis with regex repair.**