Resem: Searching Regular Expression Patterns with Semantics and Input/Output Examples

Hiroki Takeshige(✉), Shinsuke Matsumoto, and Shinji Kusumoto

Graduate School of Information Science and Technology,
Osaka University, Osaka, Japan
{h-takesg,shinsuke,kusumoto}@ist.osaka-u.ac.jp

Abstract. Regular expression is widely known as a powerful and general-purpose text processing tool for programming. Though the regular expression is highly versatile, there are various difficulties in using them. One promising approach to reduce the burden of the pattern composition is reuse by referring to past usages. Still, several source code-specialized search engines have been proposed, they are not suitable for the scenario of reusing regular expression patterns. The purpose of this study is the efficient reuse of regular expression patterns. To achieve the purpose, we propose a usage retrieval system Resem specialized in regular expression patterns. Resem adopts two key features: search by semantics and collecting input/output examples. Resem will smoothly connect *what to do* to *how to do* in the implementation process of string manipulation.

Keywords: Regular expression · Pattern · Usage search · Input/output example · Dynamic analysis · Semantics

1 Introduction

Regular expression is widely known as a powerful and general-purpose text processing tool for programming. In regular expression, any strings can be expressed in a special character sequence. This paper calls such a character sequence pattern. An example of a pattern is `\d+\.\d+`[1] which accepts any version number consisting of a major and a minor number.

Though the regular expression is highly versatile, there are various difficulties in using them. One of the reasons is on non-intuitive metacharacters in a pattern [4]. While metacharacters enable flexible text manipulation with only a few characters, they do not intuitively represent what they mean. In addition, it is difficult to analyze and generalize the string to be processed and manipulated [4]. Wang et al. reported that 46% of regular expression-related bugs were caused by incorrect behaviors of patterns [5].

[1] `\d`, `+`, and `\.` respectively means a single digit, one or more repetitions of the preceding character, and a single dot.

One promising approach to reducing the burden of the pattern composition is reusing past pattern usages. Several source code-specialized search engines have been proposed to support the reuse of program APIs [1,6] and code snippets [2,3]. Although these engines can be used to retrieve usages of regular expression, they are not suitable for the scenario of reusing regular expression patterns. The reason is that the existing approach provides *"how to use an API"*, not *"how to compose a pattern using regular expression literals"*. Furthermore, although the existing approach queries code snippets themselves (e.g., API names), we need to query the meaning of the pattern.

The purpose of this study is the efficient reuse of regular expression patterns. To achieve the purpose, we propose a usage retrieval system, Resem, specialized in regular expression patterns. Resem adopts two key features: search by semantics and collecting input/output examples. Resem accepts patterns' meaning or purpose as a search query. This idea enables to query by the semantics of patterns rather than their contents. In addition, multiple sets of input/output examples corresponding to the usages are presented in the search result. The concrete I/O examples are powerful information that helps to understand the pattern. These features reduce the user's burden of analyzing the manipulation and reading special characters. To evaluate the effectiveness of Resem, we conducted an experiment with 12 subjects. As a result, we confirm that Resem decreased the time required for describing patterns by 16%. Currently, prototype of Resem is available on https://tyr.ics.es.osaka-u.ac.jp/resem/.

2 Resem

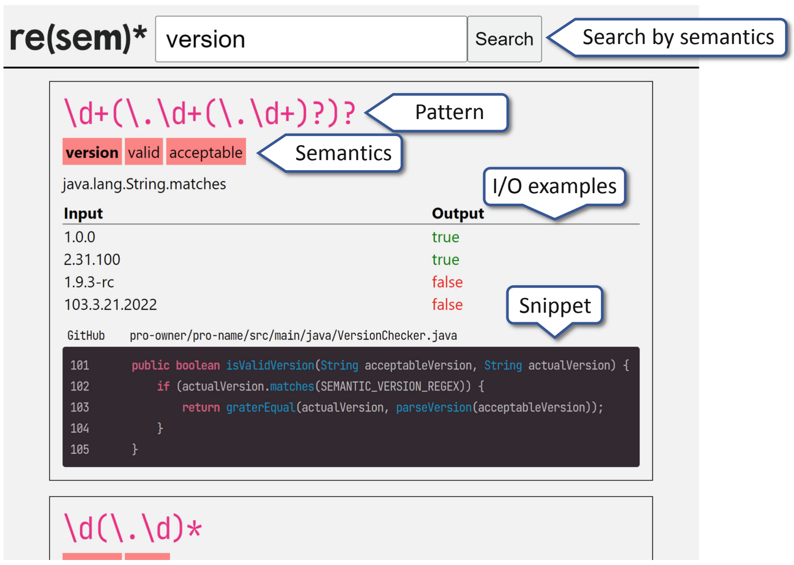
2.1 Overview

In order to alleviate the burden of composing regular expressions, this paper proposes a usage retrieval system Resem specialized in regular expression. Resem has the following two features. One is that it enables search by the meaning of the pattern (F1). The other is to present an example of inputs and outputs of a pattern (F2). These features make it possible for users who have little experience in using regular expression to search for usages efficiently.

The appearance of Resem is shown in Fig. 1. Users enter search queries in the upper input area. The search queries are the meaning of the pattern they want. In Fig. 1, "version" is entered to search for a pattern that accepts strings representing a version number. As the first search result, a usage using with the pattern `\d+(\.\d+(\.d+)?)?` is shown. One usage consists of a pattern, semantics, I/O examples, and code snippet around an API call. The users select the pattern that fits their purpose from the results and use it in their programs.

2.2 F1: Search by Semantics

Resem accepts the meanings of a pattern as a search query. This feature allows users to search by what they want to achieve rather than how to compose a pattern. For example, considering searching for a pattern that matches the version



\d+(\.\d+(\.\d+)?)?

version valid acceptable

java.lang.String.matches

Input	Output
1.0.0	true
2.31.100	true
1.9.3-rc	false
103.3.21.2022	false

GitHub pro-owner/pro-name/src/main/java/VersionChecker.java

```
public boolean isValidVersion(String acceptableVersion, String actualVersion) {
    if (actualVersion.matches(SEMANTIC_VERSION_REGEX)) {
        return graterEqual(actualVersion, parseVersion(acceptableVersion));
    }
}
```

\d(\.\d)*

Fig. 1. A screenshot of Resem

number of software that uses semantic versioning. The users can enter words such as "version" or "semantic" without having to think about how to express numbers or repetitions in regular expression.

In order to realize this feature, Resem collects meanings of patterns by static analysis. The meanings are collected from identifiers around use of a regular expression API. We assumed that variables storing patterns and input strings, and names of the methods that make API calls are set concerning the meaning of the patterns.

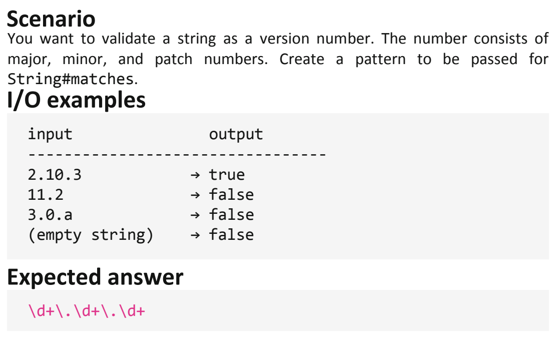
2.3 F2: Presentation of I/O Examples

Resem outputs usages of regular expression with I/O examples. The examples enable users to imagine easily what the set of strings the pattern accepts without interpreting the special characters it contains. This feature improves the efficiency when the user selects a reference pattern from the search result.

The I/O examples are gathered by dynamically analyzing patterns given to the regular expression API, input strings, and their outputs when the program is tested. The flow of I/O analysis is as follows. First, an instrumentation code is embedded in the regular expression APIs call detected by Sect. 2.2. When the instrumentation code is executed, it writes out the patterns given to the API, input strings, and outputs of the API to a file. Next, the embedded code is executed by software test. Finally, the inputs and the outputs for the patterns are collected by analyzing the output file of the embedded code.

2.4 Collecting from Open Source Projects

We collected usages from public Java projects on GitHub. Because Resem uses Gradle, a build automation tool, to run test suite, we selected target projects



Scenario

You want to validate a string as a version number. The number consists of major, minor, and patch numbers. Create a pattern to be passed for String#matches.

I/O examples

```
input               output
--------------------------------
2.10.3              → true
11.2                → false
3.0.a               → false
(empty string)      → false
```

Expected answer

```
\d+\.\d+\.\d+
```

Fig. 2. An example of tasks

which contain the word "gradle" in their README.md. The target regular expression APIs are `String` class, `Pattern` class, and `Matcher` class. As a result, 3,120 usages were collected from 68 projects. The other projects did not output any examples, or failed in running their tests.

3 Experiment

We carried out experiment with subjects to evaluate the effect of presenting usages by our system on pattern description, and to investigate the impression of subjects using the system.

We design the experiment to answer the following questions:

Qa Can Resem reduce the time to describe patterns?
Qb Can search by semantics (F1) make creating patterns easy?
Qc Can presenting I/O examples (F2) help to select a usage?

To answer Qa, we ask the subjects to perform a string processing task and compare the time required depending on whether Resem is used. In addition, to answer Qb and Qc, subjects' impressions about Resem are collected.

3.1 Experiment Design

In this experiment, subjects are asked to perform a task of string processing. The task consists of a scenario for the process and I/O examples.

One of the tasks is shown in Fig. 2. The scenario asks to verify that the input string follows the format of the version number, and provides input examples and expected output to assist in understanding the scenario. The subject creates a pattern according to this scenario. The expected answer for this task is `\d+\.\d+\.\d+`.

We prepare the scenarios under the following conditions. First, be generic and independent of specific applications. Second, some usages in the Resem database can be used for reference.

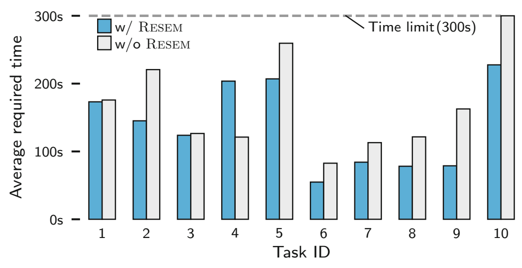


Fig. 3. Average required time for each task

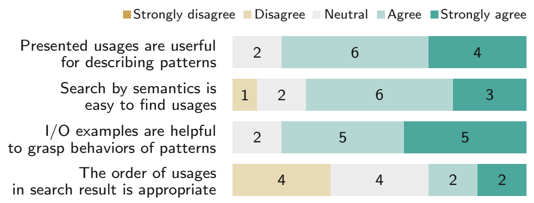


Fig. 4. Result of questionnaire

The first condition is set because a scenario that depends on a specific application is inappropriate for evaluation. Such scenarios can confirm the usefulness only when Resem is used with the application. To conduct the experiment assuming general situations, we do not make a scenario which depends on specific applications. The second condition is set to focus on the effectiveness of the usage presentation. The aim of Resem is reuse patterns in past usages. When no usage is useful for a scenario, we cannot evaluate the utility of proposing usages. Therefore, the scenarios are created based on the collected usages.

After completing all tasks, we ask the subjects to answer a questionnaire.

The subjects were 12 people, one graduate school teacher and 11 students. In this experiment, we pay attention to the difference in required time between Resem users and non-users. The subjects are divided into two groups. One group uses Resem and the other does not for each task. Both groups can use Web search throughout the experiment. In the grouping, we avoid bias in programming and regular expression skills. In addition, whether a group can use Resem is switched in half of the tasks so that the experiment results are not affected by the skill difference between the two groups.

3.2 Results and Discussion

The average time required for each task by each group is shown in Fig. 3. The horizontal axis is the number of tasks, and the vertical axis is the average time required for each task. Blue is the group in which the Resem was available, and gray is the group in which the use was prohibited. The required time for subjects who made an incorrect answer or reached the time limit is treated as 300 s, which is the same as the time limit.

The average required time of the group using Resem was short for all tasks except for Task 4. The average reduction rate of all tasks was about 16%. Though no person gave the correct answer on task 10 in the group which did not use Resem, two of six people answered correctly in the group which used it. The task cannot be solved without using lookahead and lookback. Therefore, we expected that some knowledge of these functions was necessary to answer this task by Web search. On the other hand, subjects who used Resem and correctly answered this task found the effective usage using "alphabet num" as a search query. From this, it can be said that Resem allows users with little knowledge of regular expression to search for usages that use advanced features.

The questionnaire result is shown in Fig. 4. More than half of the respondents answered that the presented usages were useful, therefore it was revealed that the description support by the usage retrieval was effective. In addition, Resem's characteristics (F1, F2) were useful because there were many favorable answers to the retrieval by the meaning and the presentation of I/O examples.

On the other hand, more than half of the subjects answered that the order in which search results were displayed was undecided or inappropriate. Therefore, it can be said that the order needs to be improved.

From the results of the subject experiment, we answer Qa, Resem can shorten the time required for the description of the pattern. The large difference in the difficult tasks suggests that this method can contribute to creating complex patterns that take a long time to be composed in actual development.

As for Qb and Qc, we judged Resem's features facilitate searching for and selecting usages because of two reasons. First, there were many favorable answers on the retrieval by the meanings and presenting I/O examples. Second, it was effective for the creation of patterns which is difficult to search from the Web.

4 Conclusion

In this paper, we propose a search system Resem, which realizes semantic search, to support regular expression pattern description. The evaluation experiment was carried out. As a result, Resem can collect regular expression usages from public projects. We confirmed that the time required for the description of the patterns was reduced by our method.

Acknowledgments. This research was partially supported by JSPS KAKENHI Japan (Grant Number: JP21H04877, JP20H04166, JP21K18302, JP21K11829, JP21K11820, JP22H03567, and JP22K11985).

References

1. Asyrofi, M.H., Thung, F., Lo, D., Jiang, L.: AUSearch: accurate API usage search in GitHub repositories with type resolution. In: Proceedings of International Conference on Software Analysis, Evolution and Reengineering, pp. 637–641 (2020)
2. Chatterjee, S., Juvekar, S., Sen, K.: Sniff: A search engine for java using free-form queries. In: Proceedings of International Conference on Fundmental Approaches to Software Engineerng, pp. 385–400 (2009)
3. Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., Baldi, P.: Sourcerer: mining and searching internet-scale software repositories. IEEE Trans. Data Mining Knowl. Discov. **18**(2), 300–336 (2009)
4. Michael, L.G., Donohue, J., Davis, J.C., Lee, D., Servant, F.: Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. In: Proceedings of International Conference on Automated Software Engineering, pp. 415–426 (2019)

5. Wang, P., Brown, C., Jennings, J.A., Stolee, K.T.: An empirical study on regular expression bugs. In: Proceedings of International Conference on Mining Software Repositories, pp. 103–113 (2020)
6. Zhong, H., Xie, T., Zhang, L., Pei, J., Mei, H.: MAPO: Mining and recommending api usage patterns. In: Proceedings of European Conference on Object-Oriented Programming, pp. 318–343 (2009)