正規表現の記述支援を目的とした意味情報に基づく用例検索システム

竹重 拓輝　柗本 真佑　楠本 真二

プログラミングにおける強力かつ汎用的な文字列処理の仕組みとして，正規表現が広く知られている．一方で，その利用には一定の難しさがあるとされている．正規表現の利用を支援する方法の一つとして，過去の用例を参考にした再利用が考えられる．しかし，再利用するパターンの検索には課題が存在する．ソースコードを対象とした API 検索やスニペット検索は正規表現パターンの再利用というシナリオにおいてはその利用は適当であるとはいえない．本研究では，正規表現におけるパターンの効率的な再利用を目的とし，正規表現の用例検索システム Resem を提案する．Resem はパターンの意味を検索クエリとして受け付け，検索結果の用例には対応する入出力例を併記する．これらの特徴からパターン作成における，目的の処理の分析や特殊文字の読み取りに係るユーザーの負担を軽減する．また，Resem の有用性を評価するため被験者実験を行い，パターン記述に要する時間を減少させることを確認した．

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. As an evaluation result, we confirm that Resem decreased the time required for describing patterns by 16%.

1　はじめに

正規表現は高い汎用性を持つ一方で，その利用には一定の難しさがあることが知られている．この難しさの要因の一つは，パターン内における非直感的な特殊文字（メタ文字）の存在にある [10]．代表的な特殊文字としては文字クラス`\s\d\w`や，量指定文字`*+?`，グルーピング文字 `()[]` などが挙げられる．わずか数文字で文字列処理の様々な制御が可能である一方で，特殊文字と処理内容との対応は直感的とはいえない．また，処理対象データと行いたい処理の内容を理解し，一般化するという問題分析の難しさも存在する [10]．この分析が不十分な場合，パターンの過剰適合（正しくない文字を受理）や過小適合（正しい文字を非受理）という問題に繋がる．

パターンの作成に係る負担を軽減する方法の一つとして，過去の用例を参考にした再利用が考えられる．ソースコードに特化した検索の仕組みとして，API 検索 [2] [13] やスニペット検索 [9] [3] が存在する．これらの手法を正規表現の用例検索に応用することも可能であるが，正規表現パターンの再利用というシナリオにおいては適当であるとは言えない．これは，その検索結果がコードスニペットであるためである．再利用するパターンを入手したい開発者が API 検索やスニペット検索を用いる場合，正規表現を使用するメソッドの呼び出し箇所を検索すると考えられる．このとき，パターンが変数としてメソッドに渡されている場合，実行時にどのようなパターンがその変数に格

Semantic-Based Retrieval System for Regular Expression Usages

Hiroki Takeshige, Shinsuke Matsumoto, Shinji Kusumoto, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.

[研究論文 (レター)] 2023 年 03 月 29 日受付.

納されているかは，プログラムの動作を解析しなければ読み取れない．

本研究の目的は，正規表現におけるパターンの効率的な再利用である．この目的を達成するために，正規表現に特化したパターンの用例検索システム Resem を提案する．Resem は，正規表現の意味や使用目的を検索クエリとして受け付ける．これによりパターンの内容ではなく，パターンの持つ意味や解釈による検索を実現する．さらに検索結果には，その用例に対応する入出力例を提示する．具体的な入出力例はパターンの理解を補助する強力な情報となるといえる．例えば，ある用例が`\d+\.\d+`であった場合，`1.2` → `accept` や `1.2beta` → `reject` などの例を併記する．これらの特徴から目的の処理の分析や特殊文字の読み取りに係るユーザーの負担を軽減する．Resem の有用性を評価するために被験者実験を行った．その結果，Resem の使用によりパターンの記述に要する時間が 16%減少することを確認した．現在，Resem のプロトタイプを `https://tyr.ics.es.osaka-u.ac.jp/resem` にて公開している．

2 関連研究

2.1 正規表現利用の課題

正規表現は文字列処理に広く採用されている一方，そのパターンの記述は容易ではないと指摘されている [10]．この原因としてパターンに使用する特殊文字が直感的でないこと，及び処理対象の文字列と処理内容について正確な分析が必要であることが挙げられる．正規表現で使用される特殊文字はわずかな文字数で様々な処理の制御を行う．このため，処理内容とその表現が乖離し，その振る舞いを想像することが難しい場合がある．また，パターンを作成する際は，対象とする入力文字列を分析し，期待通りの出力を得るためにどのような一般化が必要か検討しなければならない．分析が不十分であれば，一般化を過剰や過少に行ってしまい，パターンの動作が期待と異なるものとなり，不具合の原因となる場合がある．

2.2 正規表現利用の支援とその課題

不具合の発生を抑えつつ効率的に正規表現を利用するため，多くの開発者は正規表現の記述支援の利用や，過去に作成されたパターンの再利用を行っていることが報告されている [10] [4]．記述支援としてはユーザーが記述したパターンに対し，シンタックスハイライトの適用 [6] や指定した入力とのマッチング結果のリアルタイムでの表示 [5] がある．これらのツールは正規表現の構文理解に係る負担を軽減する．一方，パターンの入力が必要であるため，ユーザーはまず自力でパターンを作成しなければならない．この他の多くの記述支援手法 [12] [7] [8] においても同様にパターンの入力を求められるため，これらの手法はパターン作成の負担を取り除いているとは言えない．

正規表現パターンの作成に係る負担を軽減する方法の一つとして，過去に作成されたパターンを再利用するという方法が考えられる．新たに実装する処理に対して適当なパターンであるかはその都度検証しなければならないが，パターンを自力のみで作成するよりも容易に記述できる．

しかし，再利用する用例の検索には課題点が存在する．ソースコードの記述に関連する Web 検索は一般的な内容の検索よりも多数のページを探索しなければならず，所要時間などの面で高いコストを要するとされている [11]．また，正規表現においては定義した文字列集合を Web 検索のクエリにすることが難しいとされている [10]．これは検索クエリの作成においても目的とする処理を分析，分解し，言語化しなければならないためである．さらに，正規表現には言語や実装によって特殊文字の用法に違いが存在 [4] し，自身が使用する環境と一致するかを考慮して検索しなければならない．

ソースコードに特化した検索として，API 検索 [2] [13] やスニペット検索 [9] [3] が存在する．API 検索は API 名や引数を指定し，その使用例を提示するシステムである．API を指定して検索するため，動作環境を指定しやすく，また提示される検索結果がソースコードであるため，Web ページと比べて正規表現に関する記述を探しやすいといえる．スニペット検索はソースコード中の単語をクエリとしてその周囲

をコードスニペットとして提示するシステムである．クエリとして探したいコードの周囲の単語を用いられることから，API を使用する目的に関連した単語で検索することで，これを考慮した検索が可能である．

しかし，API 検索とスニペット検索の双方に共通する課題点として，得られたソースコードからパターンを再利用することが難しい点がある．検索結果として得られたソースコードのどの部分にパターンが記述されているかはユーザーが自分で探さなければならない．よって変数への代入を辿るなど，実行時の動作を解析しなければならない場合がある．また，得られた検索結果から目的とするパターンを選択するにはパターンがどう動作するか読み取らなければならない．これはパターンにおける直感的でない特殊文字の使用から，その作成と同様の負担が発生する．

本研究と同様のアプローチとして，正規表現に特化したデータベースである RegExLib [1] が存在する．RegExLib はユーザーによって投稿されたパターンを検索できるシステムである．ユーザーはパターンとともにパターンのタイトルや説明，入出力例をともに投稿する．しかし，パターンの投稿は 2010 年以来行われていない．パターン収集をユーザーの投稿に依存したシステムではデータベース拡大に限界が存在するといえる．

3 提案手法

3.1 概要

本研究では前章で述べた課題の解決を目的として，正規表現に特化した用例検索システム Resem を提案する．Resem はオープンソースソフトウェアのソースコードを対象として正規表現の用例を収集する．収集の際，各用例に対してソースコードの解析によって得られた意味を設定し，その意味を検索文字列とした用例の検索機能を提供する．Resem は次の 2 つの特徴を持つ．1 つはパターンが持つ意味による検索が可能である点（F1）．もう 1 つはパターンの入出力例を提示する点（F2）である．これらの特徴から正規表現の利用経験が浅いユーザーでも参考とする用例を効率的に検索できるようになる．

Resem の外観を図 1 に示す．ユーザーは上部の入力エリアに検索クエリを入力する．このとき，検索クエリは受理したい文字列集合の意味を用いる．図の例ではバージョン番号を表す文字列を受理するパターンを検索するため，“version” と入力している．1 件目の検索結果として`\d+(\.\d+(\.d+)?)?`というパターンを用いる用例が出力されている．1 つの用例はパターン，意味，入出力例，API 使用箇所スニペットから成る．ユーザーは出力された用例のうち自身の目的に合ったものを参考にしてパターンを記述する．

図 1 Resem の外観

以降では Resem の 2 つの特徴を説明する．

3.2 F1 意味による検索

Resem は検索クエリとしてパターンが受理する文字列の意味を用いる．この特徴からユーザーはパターンをどう組み立てるかではなく，何を実現したいかで検索できる．例えば，セマンティックバージョニングを採用しているソフトウェアのバージョン番号を受理するパターンを検索することを考える．このとき，ユーザーは version や semantic といった単語を入力すればよく，数字や繰り返しを正規表現でどのように表現するかは考える必要がない．

この特徴を実現するため，パターンが受理する文字列が持つ意味を集める．パターンが受理する文字列の意味は正規表現 API の使用箇所周囲にある識別子から収集する．パターンや入力文字列を格納する変数，API 呼び出しを行っているメソッドの名前はパターンが持つ意味に関連するものが設定されていると仮

定する．抽象構文木の解析によりプログラム中の正規表現を用いる API の呼び出しを検出し，前述の識別子を収集する．

3.3 F2 入出力例の提示

正規表現 API に対する入力文字列とそれに対する出力を，使用されたパターンに対する入出力と表現する．Resem は検索結果として正規表現の用例を出力する際，同時に各パターンに対する入出力例をともに出力する．入出力の例示により，パターンが含む記号を解釈せずともそのパターンが受理する文字列集合がどのようなものであるか想像できる．この特徴はユーザーが検索結果として出力された用例から，参考とする用例を選択する際の効率を向上させる．例えば検索結果として`\d+\.\d+\.\d+`というパターンが得られた場合，`1.2.3` や `17.8.10` のような具体的な文字列をともに出力する．ユーザーは`\d` や`+`が何を表すかを気にせず，自身が受理させたい文字列に類似する文字列を受理するパターンを探せばよい．

提示する入出力例にはソフトウェアテスト実行時に与えられた入力と，これに対する出力を用いる．これらは収集対象のプログラムのテスト実行時に正規表現 API に与えられるパターン，入力文字列，およびその出力の解析によって得る．

入出力の解析の流れは以下の通りである．まず，3.2 節で検出した正規表現 API の使用箇所に入出力収集用のコードを埋め込む．このコードは実行されると API に対して与えられるパターン，入力文字列，API の出力をファイルなどに書き出す．次に，埋め込みを行ったコードをテストを用いて実行する．最後に，埋め込みコードの出力を解析することでパターンに対する入出力を収集する．

3.4 用例収集結果

GitHub 上の公開プロジェクトから用例の収集を行い，68 件のプロジェクトから 3,120 件の用例を収集した．使用したプロジェクトは README.md ファイルに文字列 “gradle” を含む Java プロジェクト 1,000 件である．これはソフトウェアテスト実行時にビルドツールである Gradle を使用したためである．

シナリオ

与えられた文字列がソフトウェアのバージョン番号として有効か判定したい．バージョン番号はメジャー番号，マイナー番号，パッチ番号の3つの非負整数からなり，これらがこの順で．（ドット）で連結された文字列である．このような文字列のときのみtrueを返すよう `String#matches`に渡すべき正規表現パターンを作成せよ．

入出力例

```
      input         output
-------------------------
例1: 2.10.3      → true
例2: 11.2        → false
例3: 3.0.a       → false
例4: (空文字)    → false
```

解答例

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

図 2　出題したタスクの例（タスク 2）

4 実験

実験実施にあたり，次の問いを設定する．

Qa Resem はパターンの記述に要する時間を短縮するか

Qb 意味による検索（F1）は用例の探索を容易にするか

Qc 入出力例の提示（F2）は用例の選択を容易にするか

Qa を調査するため，被験者に文字列処理タスクを出題し，Resem 使用の有無による所要時間の差を調査する．また，Qb 及び Qc を調査するため，被験者が Resem を使用した際の印象やコメントを収集する．

4.1 実験設定

本実験では被験者に対し，文字列処理を行うタスクを出題する．出題するタスクは文字列処理を行うシナリオと，そのシナリオにおける入出力例から成る．

出題したタスクを 1 題抜粋し，図 2 に示す．このタスクでは入力文字列がバージョン番号の書式に沿うかを確認するシナリオが提示され，また，シナリオの理解を補助するため入力例と期待する出力が示されている．被験者はこのシナリオに沿ったパターンを作成する．このタスクでは`\d+\.\d+\.\d+`が正解となる．

出題するシナリオは以下の条件で作成した．

1. 特定のアプリケーションに強く依存せず，一般性を持つ
2. Resem が収集した用例に参考となるものが存在する

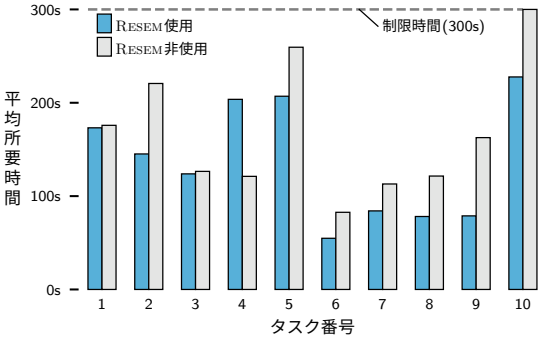

図 3 RESEM 使用時と非使用時の平均所要時間

条件 1 は特定のアプリケーションに依存したシナリオは評価に不適切と考えたため設定する．アプリケーションに依存したシナリオはそのアプリケーションの利用における有用性しか確認できない．より一般的な状況を想定し実験を行うため，特定のアプリケーションに依存したシナリオは作成しない．

条件 2 は実験目的の限定のため設定する．被験者実験は提案手法における用例提示部分の有用性の評価を目的としている．参考となる用例が収集されていないシナリオでは RESEM は用例の提示を行えず，用例提示の有用性の評価を行えない．よって，収集された用例を元にシナリオを作成する．

また，全てのタスクへの解答後，被験者に対しアンケートを行う．さらに，RESEM 対する意見を自由記述による回答で求める．

被験者は大学教員 1 名, 学生 11 名の計 12 名である．

本実験では各タスクについて，RESEM 使用者と非使用者の所要時間の差に注目する．よって被験者を 2 グループに分け，それぞれのタスクにおいて一方のグループには RESEM を使用させ，他方には RESEM を使用させない．なお，Web 検索はどちらのグループも実験を通して使用可能とする．グループ分けは事前のアンケートによってプログラミングや正規表現のスキルに偏りが生まれないようにし，タスクの半数で RESEM 使用の可否を入れ替えることで実験結果が個人のスキル差の影響を受けないようにする．

シナリオ

アルファベットと数字が混ざった文字列がある．この文字列をアルファベットのみからなる部分文字列と数字のみからなる部分文字列が交互に並ぶよう切り分けたい．String#split に渡すべき正規表現パターンを作成せよ．

入出力例

```
      input                  output
-----------------------------------------------------
例1: 2022January31        → [2022, January, 31]
例2: ac33o4k22            → [ac, 33, o, 4, k, 22]
例3: 空文字                → []
```

解答例

```
(?<=\w)(?=\d)|(?<=\d)(?=\w)
```

図 4 タスク 10

4.2 結果

各タスクにおける，グループごとの平均所要時間の差を図 3 に示す．横軸はタスクの番号であり，縦軸は各タスクの平均所要時間である．青は RESEM を使用可能な状態で解答したグループ，灰色は使用を禁止したグループである．不正解及び制限時間に達した被験者の所要時間は 300 秒として扱っている．

タスク 4 を除く全てのタスクにおいて RESEM を使用したグループの平均所要時間が短い．全タスクの減少率の平均は約 16%だった．また，タスク 10 は RESEM 非使用のグループには正解者がいなかったが，使用したグループは 6 人中 2 人が正解した．タスク 10 を図 4 に示す．タスク 10 は与えられた文字列をアルファベットと数字に分割するタスクである．このタスクへの解答には先読みと後読みという正規表現の機能が必要であり，Web 検索によって解答するにはそれらの機能についてある程度の知識が必要だったと予想される．一方 RESEM を使用しこのタスクに正解した被験者は “alphabet num” を検索クエリとして，参考とする用例を発見していた．これより，RESEM では正規表現に対する知識が少なくとも，高度な機能を使用する用例の検索が可能であるといえる．

タスク 4 においては RESEM を使用したグループのほうが所要時間が長くなった．これは被験者が使用したクエリと，RESEM が収集した意味が一致しなかったためである．タスク 4 は指定の文字種以外の文字を置換する際に使用するパターンを作成するタスクであった．このタスクに対し，RESEM を使用してタスクに取り組んだ被験者は “space” や “replace”, “not contain” などのクエリで検索していた．一方，参

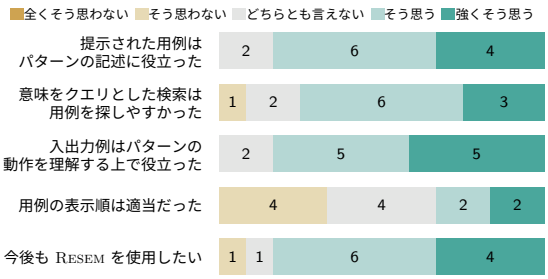

図 5　アンケート結果（数字は各選択肢の回答者数）

考にされると想定した用例に付与された意味は“suite”や“name”といったものであり，被験者が検索に使用した意味は付与されていなかった．このため，被験者はこの用例を見付けられず，Resem を使用した時間が web 検索へ移るまでの不要な時間となったと考えられる．

また，Resem 使用後のアンケート結果を図 5 に示す．提示された用例が役立ったという回答が半数を超えていることから用例検索による記述支援は有用だったといえる．さらに，意味での検索や入出力例の提示に対して好意的な回答が多かったことから，Resem の特徴（F1，F2）は有用であったといえる．

このほか，Resem に対する意見として次の回答を得た．

> 入出力例やコードスニペットにより，パターンの意味がわかりやすかった

> 検索上位だが欲しい検索結果ではない例を読み飛ばすとき，入出力例が便利だった

> 検索クエリとして適切な単語を見つけにくい場合があった

4.3　考察

被験者実験の結果より，Qa について，Resem の使用はパターンの記述に要する時間を短縮できるといえる．難度の高いタスクにおいて大きな差が見られることから，実際の開発においても作成に時間の掛かる複雑なパターンの作成に寄与すると考えられる．

また，Qb および Qc について，Resem の特徴 F1 と F2 は用例の探索と選択を容易にすると考えられる．これは，意味による検索や入出力の例示について好意的な回答が多かった点，および Web 検索が難しいパターンの作成に効果があった点から判断した．

一方，適当な意味が付与されていない場合，Web 検索よりも所要時間が長くなる場合があった．また，検索に使用する単語の選択が難しかったという意見から，シナリオから連想しやすい意味の付与を行えていない場合があると考えられる．

5　おわりに

本研究ではパターン記述の支援を目的として意味による検索を実現するシステム Resem を提案した．さらに，被験者実験を行い，Resem の有用性を評価した．その結果，Resem は過去の用例の再利用を容易にし，パターン記述に要する時間を 16% 短縮した．

今後の課題としてパターンに対する意味付与方法の改善が挙げられる．実験結果より，正規表現使用箇所付近の識別子から意味を収集する現在の方法は検索に適した意味をパターンに付与できない可能性がある．よりユーザーの想定に近い意味を付与できればユーザーの検索効率を向上させ，さらなるパターン記述時間の短縮が期待できる．

謝辞 本研究の一部は，JSPS 科研費（JP21H04877, JP20H04166, JP21K18302, JP21K11829, JP21K11820, JP22H03567, JP22K11985）による助成を受けた．

参 考 文 献

[1] : Regular Expression Library, accessed 2022-06-21. `https://regexlib.com/`.

[2] Asyrofi, M. H., Thung, F., Lo, D., and Jiang, L.: AUSearch: Accurate API usage search in GitHub repositories with type resolution, *Proc. Int. Conf. Softw. Anal. Evol. and Reengineering*, 2020, pp. 637–641.

[3] Chatterjee, S., Juvekar, S., and Sen, K.: Sniff: A search engine for java using free-form queries, *Proc. Int. Conf. Fund. Approaches to Softw. Eng.*, 2009, pp. 385–400.

[4] Davis, J. C., Michael IV, L. G., Coghlan, C. A., Servant, F., and Lee, D.: Why aren't regular expressions a lingua franca? an empirical study on the re-use and portability of regular expressions, *Proc. Joint Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 443–454.

[5] Dib, F.: RegEx101, accessed 2022-02-07. `https:`

//regex101.com/.
[6] JetBrains: Regular expressions assistance, accessed 2022-02-07. https://www.jetbrains.com/help/rider/Regular_Expressions_Assistance.html.
[7] Larson, E.: Automatic Checking of Regular Expressions, *Proc. Source Code Anal. Manip.*, 2018, pp. 225–234.
[8] Larson, E. and Kirk, A.: Generating evil test strings for regular expressions, *Proc. Int. Conf. Softw. Testing, Verification and Validation*, 2016, pp. 309–319.
[9] Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P.: Sourcerer: mining and searching internet-scale software repositories, *IEEE Trans. Data Mining and Knowledge Discovery*, Vol. 18, No. 2(2009), pp. 300–336.
[10] Michael, L. G., Donohue, J., Davis, J. C., Lee, D., and Servant, F.: Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions, *Proc. Int. Conf. Automated Softw. Eng.*, 2019, pp. 415–426.
[11] Rahman, M. M., Barson, J., Paul, S., Kayani, J., Lois, F. A., Quezada, S. F., Parnin, C., Stolee, K. T., and Ray, B.: Evaluating how developers use general-purpose web-search for code retrieval, *Proc. Int. Conf. Mining Softw. Repos.*, 2018, pp. 465–475.
[12] Spishak, E., Dietl, W., and Ernst, M. D.: A Type System for Regular Expressions, *Proc. Formal Techn. Java-like Programs*, 2012, pp. 20–26.
[13] Zhong, H., Xie, T., Zhang, L., Pei, J., and Mei, H.: MAPO: Mining and recommending API usage patterns, *Proc. Eur. Conf. Object-Oriented Program.*, 2009, pp. 318–343.

竹重拓輝

2022 年大阪大学基礎工学部情報科学科卒業．同年より同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程在学中．正規表現に関する研究に従事．

柗本真佑

2010 年奈良先端科学技術大学院大学博士後期課程修了．同年神戸大学大学院システム情報学研究科特命助教．2016 年大阪大学大学院情報科学研究科助教．博士（工学）．エンピリカルソフトウェア工学の研究に従事．

楠本真二

1988 年大阪大学基礎工学部卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 年同講師．1999 年同助教授．2002 年同大学大学院情報科学研究科助教授．2005 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．電子情報通信学会，情報処理学会，IEEE，JFPUG，PM，SEA 各会員．