社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

意味情報と入出力例を用いた正規表現の用例検索システム resem の提案

竹重　拓輝†　　柗本　真佑†　　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{h-takesg,shinsuke,kusumoto}@ist.osaka-u.ac.jp

あらまし　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て再利用するという方法が考えられる．しかし，再利用するパターンの検索には課題が存在する．Web 検索はその検索対象の広さから効率的な検索は難しい．API 検索やスニペット検索は対象をソースコードに絞っているが，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．

キーワード　正規表現，用例検索，動的解析，静的解析，再利用，文字列処理

1. はじめに

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．正規表現では，任意の文字列集合を特殊な文字列（以降，パターンと呼ぶ）を用いて表現する．例えば，メジャー番号とマイナー番号で構成されるバージョン番号を受理するパターンは，`\d+\.\d+`(注1)のように表される．正規表現を用いることで，文字列の一致判定や置換，部分文字列の抽出などの様々な処理を，文字列のみを用いて容易に実現できる．正規表現は，ほぼ全てのプログラミング言語で利用可能 [1] であり，また Python や JavaScript プロジェクトの 30% 以上が正規表現を利用しているとの報告 [2] [3] もある．

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 [1]．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 [1]．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．正規表現関連の不具合を調べた文献 [4] によると，その 46% がパターンの挙動そのものに起因した不具合であることが報告されている．

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て再利用するという方法が考えられる．Google 等を用いた Web 記事の検索は，近年の情報処理における基本技術となりつつあるが，検索対象が広すぎるため効率的とはいえない [5]．他方，ソースコードに特化した検索の仕組みとして，API 検索 [6] [7] やスニペット検索 [8] [9] [10] 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，パターンが呼び出し箇所にリテラルとして記述されていればその読み取りは容易である．しかし，パターンが変数として渡されている場合もありうる．この場合，メソッドの呼び出し実行時にどのようなパターンがその変数に格納されているかは，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，12 名による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．

(注1)：`\d` は数字一文字，`+` は直前文字の 1 回以上の繰り返し，`\.` はドット一文字を表す．

2. 準　　備

2.1 正規表現

正規表現は文字列処理において，処理対象の指定に文字列集合を使用する仕組みである．特殊文字を含むパターンを用いて文字列集合を表し，入力文字列がこの文字列集合に含まれるかの判定（マッチング）を行う．このマッチング結果を用い，入力文字列の書式判定や部分文字列の置換，抽出が可能である．

正規表現は広く使用されており，多様なプログラミング言語やソフトウェアにおいて正規表現を使用するための機能が提供されている．例えば Java では文字列を格納する際に用いる `String` クラスは `matches` メソッドを持つ．このメソッドは引数として正規表現のパターンを受け取り，インスタンスが格納する文字列がこのパターンにマッチする場合 `true` を返す．以降，このようにプログラミング言語において正規表現を使用するために用意されているインターフェースを正規表現 API と呼ぶ．

2.2 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 [1]．誤った記述により想定と異なるマッチングが行われ，不具合の原因となることがある．実際，文字列処理に関連する不具合のうち 37% は正規表現が原因との報告 [11] がある．また，正規表現の使用に関連する不具合のうち 61% は使用されたパターンの記述に問題があったという報告 [4] もされている．

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．

2.3 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 [1] [12]．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 [13] 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 [14] 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 [15] [16] [17] 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．これは過去の用例を検索し，必要に応じて一部を修正し再利用するという方法である．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 [5]．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 [1]．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 [12] 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 [6] [7] やスニペット検索 [8] [9] [10] 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．

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．意味の背景色の濃淡は用例における重みを表しており，より関連度の高い意味の背景色が濃くなる．この用例では version が最も関連度が高く，acceptable が最も低い．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．

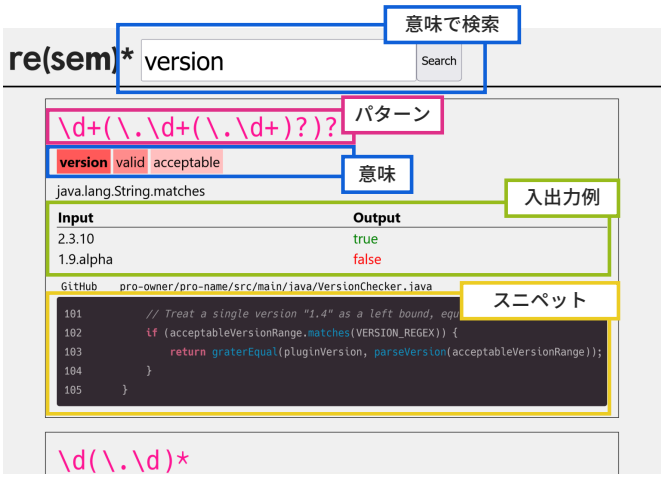

図 1　resem の外観

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必要がある．さらに，検索システムとして利用するため，収集した意味を次の 2 つの観点で重み付けする．1 つは tf-idf，もう 1 つは正規表現 API との関係性である．以降は意味の収集と重み付けについて説明する．

3.2.1　意味の収集方法

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．このとき，複数の単語から構成されている識別子は分割し，単語として収集する．例えば，`versionString` という識別子は `version` と `string` に分割する．

3.2.2　tf-idf

収集した意味に対し，tf-idf を用いた重み付けを行う．tf-idf は文書集合に対し，各文書に含まれる単語がどれだけ文書の特徴を表すかの指標である．本手法では各用例を文書と捉え，それぞれの用例において収集した意味を，文書中の単語として tf-idf を適用する．

tf-idf は単語の出現頻度（Term Frequency; tf）と逆文書頻度（Inversed Document Frequency; idf）の積であり，tf はある単語が文書中で繰り返し出現するほど高く，idf は出現する文書が少ないほど高くなる．提案手法では識別子の使用における以下の特徴により，tf-idf を用いるとよりパターンの意味を表す単語に高い重みを付与できると考える．API の使用箇所周囲において繰り返し出現する単語は API の処理に強く関連し，使用されているパターンとの関連度も高いと考えられる．このような単語の tf は高くなる．一方，識別子の中には `str` など慣例的に使用され，処理内容を表さない単語もある．このような単語は多数の API 使用箇所で現れると考えられる．よってこのような単語の idf は低くなる．

3.2.3　API 呼び出しとの関係性

周囲の識別子は API 呼び出しとの関わり方によって正規表現との関連度が異なると考えられる．例えば，パターンを格納していた変数名はパターンの意味を強く表すことが多い．よって，そのような変数名から収集した意味には高い重みを与える．一方，API 呼び出しを含むメソッド名やクラス名は API の処理内容との関連度が低い場合がある．このため，メソッド名やクラス名から収集した意味には低い重みを与える．

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，3.2.1 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．

4.　システム実装と用例収集の実行

4.1　用例収集の流れ

用例収集における処理の流れを説明する．収集は入力プロジェクトに対する静的解析と動的解析によって行う．

まず，プロジェクトの静的解析を行う．これは意味の収集と動的解析に向けたコードの埋め込みが目的である．プロジェクトのソースコードから，正規表現 API の呼び出しを検出する．検出した API 呼び出しから，その周囲にある意味を収集する．さらに，API 呼び出し周辺に動的解析で用いるコードを埋め込む．このコードは，API 呼び出しの実行時に API の入出力をファイルに書き出す．

次に動的解析を行う．静的解析においてコードを埋め込んだプロジェクトのテストを実行する．埋め込んだコードによって出力されるファイルを解析し，API 呼び出しの入出力を収集する．

4.2　収 集 実 行

用例の収集を行い，70 件のプロジェクトから 1,111 件の用例を

収集した．入力とするプロジェクトは GitHub 上の公開プロジェクトを使用した．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 プロジェクトのうち，スター数上位 1,000 件である．また，対象とする正規表現 API は `String` クラスの `matches`，`split`，`replaceAll`，`replaceFirst` メソッドとした．正規表現を用いる API としてはこのほかに，`Pattern` クラスや `Matcher` クラスのメソッドが存在するが，現段階では対象外とした．両クラスへの対応は今後行う予定である．

5. 実　　験

パターン記述における提案手法の有用性を確認するため 2 つの実験を行った．

収集用例の分析　提案手法によって検索システム上で有用な用例を収集可能か調査する．

被験者実験　提案手法による用例の提示がパターンの記述に与える影響，および検索システムを使用した被験者の印象を調査する．

5.1 収集用例の分析 概要

提案手法による収集で検索システムが提示する用例として有用な用例が収集できるか調査するため収集した用例を分析する．分析は特殊文字を含むパターンを使用する用例数，一意なパターン数，適切な意味が付与された用例の割合の 3 つの指標で行う．

特殊文字を含むパターンを使用する用例数は，検索システムが再利用に値する用例を提示できるか確認するために設定する．正規表現 API の用例にはパターン内に特殊文字を含まない場合がある．2.2 節で述べたように，パターンの再利用を行う狙いの一つは特殊文字使用の難しさである．よって特殊文字を用いない用例は，パターンの再利用を目的とした検索の際には不要だと考えられる．例えば，`String#split(",")` のような場合がこれに該当する．

一意なパターン数は検索結果の多様性を確認するための指標である．検索システムのユーザーが正規表現を使用する目的は様々であり，求める用例はそれぞれ異なる．このため，検索システムとして有用であるためには，ユーザーに対し，多様なパターンを提示するべきだと考える．前述の特殊文字を含むパターンから重複を除き，一意なパターン数を調査する．

適切な意味が付与された用例の割合は，提案手法によってパターンの意味を回収できたか確認するために調査する．提案手法において意味の収集元としている正規表現 API 周囲の識別子からは，パターンの意味とは無関係の単語しか収集できない場合も考えられる．よって，記号を含むパターンを使用する用例のうち，パターンの意味として適切な単語が 1 つ以上収集されている用例の割合を調べる．用例に対して意味が適切であるかは第一著者が目視によって判断する．調査対象は記号が使用されている用例から無作為に抽出した 100 件とした．

5.2 収集用例の分析 結果

各指標の調査結果を表 1 に示す．特殊文字を含むパターンを使用する用例数は 314 件であり，1,111 件の用例のうち 7 割以上が特殊文字を含まない用例であった．これは，収集の対象とした正規表現 API が `String` クラスのもののみであることが原因だと考えられる．`String` クラスでは，正規表現 API に特殊文字を含まないパターンを与えることで特定の文字列を対象とした置換などが行われる．`Pattern` クラスや `Matcher` クラスといった今回対象外としたクラスの用例収集に対応すると，特殊文字を含む用例は増えると考えられる．これらのクラスはグルーピング文字によって指定された部分文字列の後方参照が可能であり，参照される文字列は特殊文字により一般化して指定されると考えられるためである．

また，特殊文字を含むパターンから重複を除外し，一意なパターン数を調査した．その結果，一意なパターンは 252 件であった．減少した件数が 20% 程度であることから，収集した用例はある程度多様性を持つと判断できる．

また，適切な意味が付与された用例の割合は 36% であった．よって特殊文字を使用している用例 314 件のうち，100 件程度が適切な意味を付与されていると予想される．この結果から，割合としては少ないが，提案手法による意味の収集は可能であると考えられる．

これらの値について，検索システムとしての十分性の客観的な評価は行えていない．しかし，これらの値は必要に応じ，収集対象とするプロジェクト数の拡大や対応する言語の追加によって補えると考えられる．よって，提案手法における用例収集は検索システムに有用な用例の収集が可能だと判断できる．

5.3 被験者実験 概要

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．

5.4 被験者実験 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

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

表 1　収集用例の分析において用いた指標とその結果

指標	値
特殊文字を含むパターンを使用する用例数	314 件
一意なパターンの数	252 件
適切な意味が付与された用例の割合	36%

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         return
   ----------------------
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

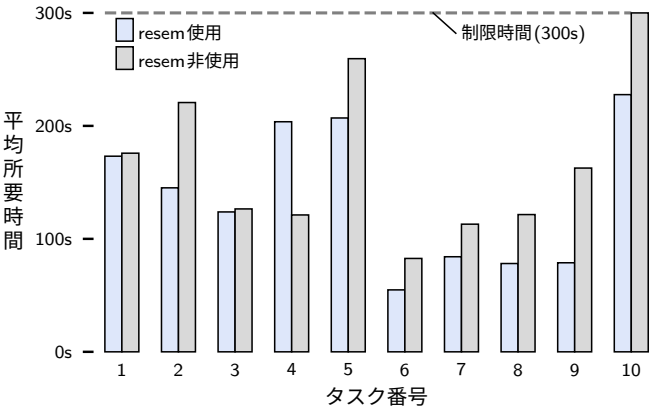

図 3　resem 使用時と非使用時の平均所要時間

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．

5.5　被験者実験 被験者

被験者は大阪大学大学院情報科学研究科所属の教員 1 名，修士学生 8 名，及び大阪大学基礎工学部学部生 3 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．

5.6　被験者実験 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

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                    return
   ---------------------------------------------
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

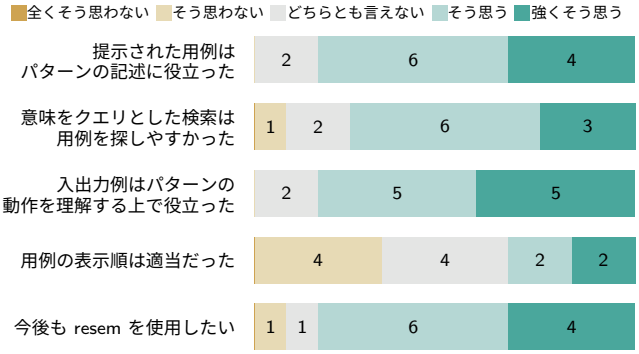

図 5　アンケート結果（グラフ中の数字は各選択肢の回答者数）

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 は先読みと後読みという正規表現の機能を使用しなければ解答できない．よってこのタスクを 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理解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．今後も resem を使用したいと回答した被験者は 12 名中 10 名であった．

一方，表示順については半数を超える被験者がどちらとも言えない，または適当でないと回答した．よって，重み付けやその検索への反映が適切に行えていない可能性がある．

このほか，resem 使用時の利点として次のような回答を得た．

> 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

> 正規表現の用例だけを検索でき，（用例の再利用に）関係のない情報がヒットしないため用例を探しやすかった

一方改善すべき点として次のような回答を得た．

> 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

> 用例に付与された意味情報の一覧などがあれば，検索しやすくなると感じた

5.7 被験者実験 考察

実験 2 の結果より，Qa に対する解答として 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できると考えられる．

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．正規表現特化のメリットを主張するコメントがあった点から web 検索よりも探しやすいと感じた被験者がいたと考えられる．

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．

6. 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，評価実験を行い，用例収集の可能性，及び resem の有用性を確認した．その結果，提案手法による用例収集が可能であることを確認した．また，resem の使用によりパターンの記述に要する時間を削減でき，正規表現の利用における resem の有用性を確認した．

今後の課題として収集する正規表現 API の拡充，及び意味に対する重み付けの評価と精度向上が挙げられる．`Pattern` クラスや `Matcher` クラスの用例収集に対応することで，より多くの正規表現の用例を収集できると考えられる．また，実験結果から重み付けの精度が低い可能性が示されており，これに対する評価，および改善が必要である．

謝辞 本研究の一部は，JSPS 科研費（JP21H04877）による助成を受けた．

文　献

[1] L.G. Michael, J. Donohue, J.C. Davis, D. Lee, and F. Servant, “Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions,” Proc. International Conference on Automated Software Engineering, pp.415–426, 2019.

[2] C. Chapman and K.T. Stolee, “Exploring regular expression usage and context in python,” Proc. International Symposium on Software Testing and Analysis, pp.282–293, 2016.

[3] J.C. Davis, C.A. Coghlan, F. Servant, and D. Lee, “The impact of regular expression denial of service (redos) in practice: an empirical study at the ecosystem scale,” Proc. Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.246–256, 2018.

[4] P. Wang, C. Brown, J.A. Jennings, and K.T. Stolee, “An empirical study on regular expression bugs,” Proc. International Conference on Mining Software Repositories, pp.103–113, 2020.

[5] M.M. Rahman, J. Barson, S. Paul, J. Kayani, F.A. Lois, S.F. Quezada, C. Parnin, K.T. Stolee, and B. Ray, “Evaluating how developers use general-purpose web-search for code retrieval,” Proc. International Conference on Mining Software Repositories, pp.465–475, 2018.

[6] M.H. Asyrofi, F. Thung, D. Lo, and L. Jiang, “Ausearch: Accurate api usage search in github repositories with type resolution,” Proc. International Conference on Software Analysis, Evolution and Reengineering, pp.637–641, 2020.

[7] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, “Mapo: Mining and recommending api usage patterns,” Proc. European Conference on Object-Oriented Programming, pp.318–343, 2009.

[8] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, “Sourcerer: mining and searching internet-scale software repositories,” Data Mining and Knowledge Discovery, vol.18, no.2, pp.300–336, 2009.

[9] S. Chatterjee, S. Juvekar, and K. Sen, “Sniff: A search engine for java using free-form queries,” Proc. International Conference on Fundamental Approaches to Software Engineering, pp.385–400, 2009.

[10] GitHub, “Code Search,” accessed 2022-02-13. `https://github.com/search`.

[11] A. Eghbali and M. Pradel, “No strings attached: an empirical study of string-related software bugs,” Proc. International Conference on Automated Software Engineering, pp.956–967, 2020.

[12] J.C. Davis, L.G. Michael IV, C.A. Coghlan, F. Servant, and D. Lee, “Why aren’t regular expressions a lingua franca? an empirical study on the re-use and portability of regular expressions,” Proc. European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.443–454, 2019.

[13] JetBrains, “Regular expressions assistance,” accessed 2022-02-07. `https://www.jetbrains.com/help/rider/Regular_Expressions_Assistance.html`.

[14] F. Dib, “RegEx101,” accessed 2022-02-07. `https://regex101.com/`.

[15] E. Spishak, W. Dietl, and M.D. Ernst, “A type system for regular expressions,” Proc. Formal Techniques for Java-like Programs, pp.20–26, 2012.

[16] E. Larson, “Automatic checking of regular expressions,” Proc. Source Code Analysis and Manipulation, pp.225–234, 2018.

[17] E. Larson and A. Kirk, “Generating evil test strings for regular expressions,” Proc. International Conference on Software Testing, Verification and Validation, pp.309–319, 2016.