変数のデータフローによるAPI利用コード例の検索

竹之内 啓太　石尾 隆　井上 克郎

ソフトウェア開発の効率を向上させるため，ライブラリなどの API (Application Programming Interface) が利用されている．一方で，巨大化・複雑化した API の利用は必ずしも容易ではない．そこで本研究では，API の理解を支援するコード検索手法を提案する．提案手法の特徴として，(1)「変数のデータフロー」を (独自の検索クエリを用いて) 指定し，API の理解に有益なコード例を検索する点，(2) 検索対象となるソースコードを既存のコード検索エンジンから取得することで，さまざまな API の検索に対応している点，(3) 有限オートマトンを利用した軽量なアルゴリズムを用いることでウェブアプリケーションとしての実装を実現している点が挙げられる．評価実験では，提案手法が API の理解を有効に支援する場合があることや，検索クエリの記述が比較的容易であること，検索時間が実用的な範囲に収まることを確認した．

To improve software productivity, software reuse with API (Application Programming Interface) such as library is common. However, APIs are not always easy to use because some of them are too large or complicated. Code example search is a common approach against it. In this paper, we propose a novel code example search method to help developers to understand how APIs should be used. Our approach has three features. (1) Variable data-flow can be specified with our original query to show helpful code examples. (2) Searching for various APIs is realized by downloading search targets on demand from the existing code search engine. (3) It is implemented as a web application with a low-cost algorithm using finite automaton. A result of an experiment shows that the proposal method helps API comprehension effectively in some cases, our original query is relatively easy to write, and the search time is practical.

1 はじめに

ソフトウェア開発において，ライブラリなどの API (Application Programming Interface) の利用は欠かせないものとなっている [8]．API を利用することで，第三者が過去に作成したソフトウェアを再利用することができ，開発効率の向上につながる．しかしながら， API が巨大化・複雑化している場合は，その利用が困難となることが知られている [3] [4]．そのため，コード例検索により実際に API を利用しているコード例を見つけ出し，それを見て学習することが一般的である．コード検索エンジンはウェブサービスとして利用できるため導入の手間がいらず，検索結果を即座に取得できるため，広く利用されている [6]．

しかしながら， API に対する検索の要望は常に 1 つであるとは限らない．そのため，一般的なコード検索エンジンで利用されている単語単位の検索では，要望を上手く表現することが容易ではない場合がある．たとえば，`Java` の標準ライブラリに含まれる `Statement` インターフェースの `executeQuery` メソッドはデータベースを操作する SQL 文を実行するためのメソッドである．このメソッドの戻り値である `ResultSet` 型の変数から SQL 文の実行結果を取り出したい場合を想定する．一般的なコード検索エ

Searching for API Usage Examples Focusing on Variable Data-flow

Keita Takenouchi, Katsuro Inoue, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.

Takashi Ishio, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

コンピュータソフトウェア, Vol.29, No.1 (2012), pp.78–84.

[研究論文 (レター)] 2017 年 2 月 15 日受付.

本論文は第 23 回ソフトウェア工学の基礎ワークショップ (FOSE2016) の発表論文を発展させたものである．

```
1:  ResultSet rs = stat.executeQuery(sql);
2:  while (rs.next()) {
3:    String commenter = rs.getString(2);
4:    String comment = rs.getString(3);
5:    Comment c = new Comment(commenter, comment);
6:    t.addComment(c);
7:  }
8:  rs.close();
```

図 1 提案手法の検索結果の例

ンジンを利用する場合，“executeQuery”というメソッド名で検索し，その戻り値に対する操作に関するコードを閲覧することになる．しかし，メソッド名のコメント行への出現や同名のメソッドの定義など，メソッド名を含んではいるが API の理解には有益でないコードも検索結果として提示されてしまう．

そこで本研究では，開発者にとって有益なコード例を適切に提示する手法を考案した．具体的には，独自の検索クエリを利用することで「変数のデータフロー」を表現し検索するという方式を提案する．この検索クエリは基本的には検索対象のプログラミング言語のトークンの列であるが，3 つの特殊な意味を持つワイルドカードを導入することで検索に柔軟性を持たせている．先に挙げた executeQuery メソッドの戻り値の使用方法を調査するという場合であれば，たとえば以下のような検索クエリによって要望を表現できる．

```
$a = _ . executeQuery ?? $a
```

ここで“$a”は $変数と呼ばれるワイルドカードであり，1 つの変数名にマッチする．また，“??”は改行含む任意のトークン列にマッチするワイルドカード，“_”は 1 つの文の内部におけるトークン列にマッチするワイルドカードである．この検索の結果，図 1 のようなコード例が検出される．“$a”には 1 行目の変数“rs”がマッチしており，図ではそれを太字で表現している．クエリ中の“_”には 1 行目の変数 stat が，“??”には 1 行目の executeQuery に続く“(”から 7 行目の“}”までが，最後の“$a”には 8 行目の先頭に出現する変数 rs がそれぞれマッチする．それ以降のコードはマッチの範囲には含まれないが，行末までを検索結果としての表示に含めている．

図 1 のコード例から，executeQuery メソッドの戻り値が変数 rs に「代入」され，next, getString, close とった SQL の実行結果を取り出すための一連のメソッド呼び出しに「使用」されていることが分かる．このような変数の代入から使用までの流れのことを本研究では「変数のデータフロー」と呼ぶ．そして，変数のデータフローを考慮することにより複数の API メソッドの関係を表現し検索することが本研究の目的である．本手法は，検索対象となるソースコードを既存のコード検索エンジンから取得することで，幅広い検索の要望に対応している．また，検索クエリの設計と検索のマッチングアルゴリズムを工夫し，検索の計算コストを抑えることで，応答性のよいウェブアプリケーションとしての実装を実現している．

以降，2 章では提案手法の概要について，3 章ではそのアルゴリズムについて述べる．4 章では評価実験について述べ，最後に 5 章ではまとめと今後の展望について述べる．

2 提案手法の概要

本研究では， API の利用方法の理解を支援するため，変数のデータフローを考慮したコード例検索手法を提案する．一部の先行研究（たとえば[7]）と同様に，コード検索エンジンを利用することによるスケーラビリティを最大限に活かすため，コードの収集をあらかじめ行うのではなく，検索クエリに応じてコード検索エンジンから必要なコード片を収集し開発者にコード例を提示する．本研究ではコード検索エンジンとして searchcode.com [1] を用いた．

検索エンジンを使用すると，開発者が検索クエリを送信してから結果が返ってくるまでの応答時間が長くなる傾向にある．そこで本研究では，提示するコード例のマイニング等による選定を行わず，開発者の要望を検索クエリにより表現するという方式をとる．検索クエリによりコード中に現れる「変数のデータフロー」を指定できるようにすることで，従来のコード検索エンジンに比べ，より限定的な文脈を持つ検索を実現した．3 つの特殊な意味を持つワイルドカードを用いたトークン列のマッチングにより，近似的なデータフローを指定する方式をとっている．正規表現を書くように，特定のトークン列の並びを指定するだけで

データフローを表現した検索クエリを記述することが可能である．

検索対象となるソースコードは，コメントや空白，改行のようなレイアウト情報を取り除いたトークン列として扱う．また，プログラミング言語の構文についての情報として，出現順序に対応関係の制約を持つトークン（たとえば “{” と “}”，“(” と “)”）と，変数への代入を意味するトークン (“=”)，文の区切り文字 (“;”) についての情報は与えられるものとする．トークン列への変換は字句解析器，あらかじめ与えるトークンの情報は文法定義にのみ依存するため，本研究で構築した実装は Java, C/C++, C#, Ruby, JavaScript の 5 つのプログラミング言語の検索に対応している．本論文では Java を例として説明する．

本手法では，検索クエリを指定することにより，検索対象から特定のパターンを持つトークン列を検出する．以下に述べる 3 つの特殊トークン以外は，トークンの種類や識別子名が完全に一致した場合にのみ，2 つのトークンが等しいと判定する．たとえば，検索クエリ中で “foo” という識別子を持つトークンが指定されると，そのクエリは “bar” や “FOO” という識別子を持つトークンにはマッチしない．したがって，検索クエリとして特殊トークンを使用せずに検索を行った場合，コードクローンの中でもタイプ 1 [5] のクローン検出と同等の検索となる．

以下に特殊トークンについて述べる．これらはマッチする対象が互いに異なるワイルドカードである．

「**$変数**」 “$” の後に識別子名が続くようなトークンであり，任意の 1 つの識別子名にマッチするワイルドカードである．たとえば，“$a” や “$list” のようなトークンは $変数であり，それぞれ 1 つの識別子名にマッチする．検索クエリにおいて，同一の $変数が複数箇所に出現する場合，すべての出現箇所に対し同じ識別子名がマッチしなければならない．このトークンはデータフローを追いかけたい変数を指定するのに用いる．

「**_（アンダースコア）**」 長さ 0 以上の開き括弧と閉じ括弧の対応関係が取れたトークン列にマッチするワイルドカードである．ただし，マッチする範囲は 1 つの文の中に限る．たとえば，“= _.get” という検索クエリは “= foo.get” や “= getFoo().get” のようなトークン列にマッチする．一方で，“= 0 ; a = foo.get” のような複数の文にまたがる (“;” を含む) トークン列にはマッチしない．このトークンにより，文の中におけるトークン列の差を吸収する検索クエリを記述することが可能である．

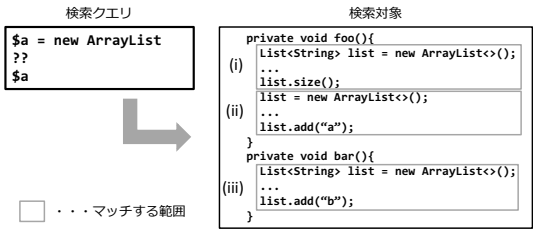


図 2 “??” がマッチする範囲の例．
(i), (ii), (iii) の 3 つのコード片にマッチする．

「**??**」 長さ 0 以上の任意のトークン列にマッチするワイルドカードである．“??” は $変数のスコープが有効である範囲において最長マッチする．すなわち，検索結果として得られるコード片は “{” より多くの “}” を含まず，$変数への再代入が行われないようなものとする．これにより，複数の箇所に出現する同じ名前を持つ変数が，異なるスコープを持つようなコード片を出力から排除することが可能である．たとえば，図 2 では，(i), (ii), (iii) の 3 つのコード片にマッチする．(i), (ii) と (iii) のコード片はそれぞれ異なるメソッドのスコープを持つため，別のコード片としてマッチする．また，(i) と (ii) が別のコード片としてマッチするのは，(ii) の最初の行において $a にマッチした変数 list への代入が行われている (list の後に “=” が続く) ため，$変数にマッチした変数の生存区間がひと続きでないとみなすからである．このようなマッチングの仕様によって，検索クエリに $変数と “??” を記述するだけで自然に変数の生存区間を考慮した検索が可能となる．すなわち，変数のデータフローに注目した検索が可能となる．

開発者にとって理解が容易であると考えられるコード例を提示するため，本手法では別名を考慮した変数

のデータフローを含むコード例は検出の対象外である．たとえば，$v が実際の変数 x にマッチしている状態で，y = x という代入があったとしても，$v に変数 y がマッチするということにはならない．このようなコード例は変数の代入関係が自明ではなく，開発者にとって API の利用方法を学習する目的には適当ではない．

なお，本手法では検索結果のランキングとコード例の整形を行っている．検索クエリに含まれる $変数がデータフローを追いかけたい（すなわち開発者の関心のある）変数であることから，変数を多く含むものを上位に表示するといった低コストなランキングと整形を用いる．詳細は [9] に記載している．

3 アルゴリズム

提案手法は，検索クエリを非決定性有限オートマトンで表現し，変数名や括弧の対応関係を確認するためのスタックを用いた状態遷移を用いてクエリのマッチを行う．以下，その手順を説明する．

手順 1．NFA の構築

まずはじめに検索クエリのトークン化を行う．トークン化には構文解析ツール ANTLR v4[†1] を使用した．特殊トークンを認識するため，検索対象のプログラミング言語の構文規則に 3 つの特殊トークンに対応する規則を追加し，検索クエリに対する字句解析器を生成した．

つぎに，図 3 のように検索クエリから NFA (非決定性有限オートマトン) を構築する．この手順は正規表現検索の一般的なアルゴリズムに類似している．NFA の入力記号は検索対象言語のトークン集合である．NFA の初期状態を生成した後，検索クエリのトークン列を先頭から読み込み，読み込んだトークン t に応じて新たな状態と遷移を追加する．直前に作成した状態を s_p とする．トークン t が “??” または “_” であるとき， s_p の任意の入力に対する遷移先として状態 s_p 自身を設定する．トークン t が “??”, “_” 以外であるとき，新たな状態 s_n を作成し， s_p の入力トークン t に対する遷移先として s_n を追加する．トークン列を末尾まで処理した後，最後に作成した状態を受理状態とすれば NFA の構築が完了する．

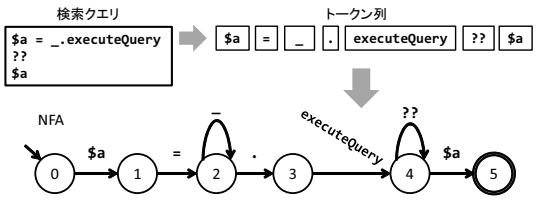


図 3 検索クエリからの非決定性有限オートマトンの構築

手順 2．コードの取得と NFA へのマッチング

本手順ではまず初めに，コード検索結果エンジンからソースコードを取得する．本研究で用いるコード検索エンジン searchcode.com では，一般的なメソッド名やクラス名などの単語単位での検索を行うための Web API が存在している．まず，検索クエリに含まれる識別子名をトークン化の段階で認識しておき，その識別子名すべてが含まれるような複数のソースコードを API 経由で取得する． そして，取得したソースコードを ANTLR v4 を用いてトークン化し，NFA に入力する．本手法で使用する NFA は一般的な NFA と同様，入力に応じてアクティブな状態が遷移するモデルである．検索対象ファイルのトークン列を先頭から NFA へ入力していくが，トークンを入力するたびに NFA の初期状態にアクティブな状態を 1 つ追加する．そして，すべてのトークン列を読み終えるまでに，NFA の受理状態に到達するアクティブな状態の集合を求めると，それが出力候補となるコード片の集合に対応する．ただし「アクティブな状態」は，その時点までに入力された括弧の対応関係を保持するためのスタックや，$変数への束縛関係などの情報を保持し，その状況に応じて消滅するルールを持つ．この情報の詳細と遷移の例，計算量等については [9] に記載されている．

手順 3．検索結果の選択

まず，出力候補から $変数への再代入が行われているコード例を除外する．束縛情報から $変数へマッ

†1 http://www.antlr.org/

チした識別子名（仮に “id” とする）を取得し，出力候補のトークン列を先頭から見ていったとき “id”, “=” というトークンの並びが二度出現した時点で出力候補から除外する．この処理は手順 2 において NFA へのマッチング中にも行うことができるが，アクティブな状態が保持する情報と消滅する条件が複雑化するのを避けるため，出力候補のフィルタリングという形をとっている．

つぎに，$変数の生存区間がより長いものを検索結果として表示するため，コードの行数が重複している出力候補同士の中でもっとも大きいもののみを出力する．そのため，受理状態に到達した 2 つのアクティブな状態 a_1, a_2 の包含関係として，以下の半順序関係 $\subseteq$ を定義する．ただし，$a.start$ はアクティブな状態 a の開始行番号，$a.end$ は a の終了行番号とする．

$$a_1 \subseteq a_2 \quad \Leftrightarrow \quad a_1.start \geq a_2.start \quad \wedge \quad a_1.end \leq a_2.end$$

この半順序関係 $\subseteq$ により，受理状態に到達したすべてのアクティブな状態集合は半順序集合となる．この半順序集合における極大元にあたるアクティブな状態が，検索結果として表示するコード例となる．この処理により，より長い $変数の生存区間を持つ，すなわち開発者にとってより有益なコード例を検索結果として提示することを実現している．

4 評価実験

本研究では，API の理解をともなうタスクを設定し，被験者実験を行うことで手法の評価を行った．

4.1 実験の設計

被験者は，第一著者と同一の研究室に所属する博士前期課程の学生 8 人であり，M1 が 4 人, M2 が 4 人から成る．被験者は主に研究に利用するツール等の開発のため Java によるプログラミング作業を日常的に行っている．そのため，実験のタスクに使用するプログラミング言語は Java とした．比較対象となる既存手法として，コード検索エンジン serachcode.com [1] を選択した．これは提案手法のソースコードの取得元であるため，検索対象となるソースコード集合に差がなく，対照実験の比較対象として適切であると考えられる．実験に使用するタスクは API の理解を伴う必要がある．恣意的なタスク設定を避けるため，実験に使用するタスクは Moreno ら [2] の実験に用いられた 2 つのプログラミングタスクをほぼそのまま使用した．被験者のタスクの達成度を，完全な達成を 100 とする得点率 (%) として評価する．なお，タスクの設計や 2 つのタスク詳細は [9] に記載している．

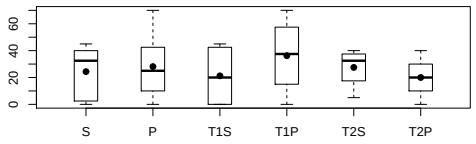


図 4　各タスクの得点率 (%) の箱ひげ図．左端の S, P はタスク 1, 2 の合計である．また，黒点は平均を示している．

4.2 実験結果と考察

実験結果として，各プログラミングタスクの得点率を図 4 の箱ひげ図として示す．この図は，横軸がタスクと使用したコード検索手法の組み合わせ (T1/T2 がタスク，S が既存手法，P が提案手法を意味する)，縦軸がその得点率 (%) である．この結果より，タスク 1 において，平均値・中央値ともに提案手法の方がよい傾向が見られる．一方で，タスク 2 ではその傾向は見られず，むしろ既存手法の方がよいという傾向が得られた．2 つのタスクを合わせた結果では，平均値は提案手法の方がよく，中央値は既存手法の方がよいという結果になった．以下に，これらの実験結果についての考察を述べる．

タスク 1 は「 HttpClient インターフェースの execute メソッドを用いて，指定された URL から PDF ファイルをダウンロードせよ.」というサブタスクから始まる．そのため，被験者はまず HttpClient インターフェースの実装クラスや，そのインスタンスの生成方法を特定しなければならない．既存手法を用いて HttpClient インターフェースの実装クラスを探す場合，API のドキュメントに記載されている複数の実装クラスを順番に見ていくか，コード検索エンジンで「 HttpClient 」というキーワードで検索し，周辺のコードを見るかの選択肢に限られる．一方で，

ERAService.java

```
1:  HttpClient httpClient = new DefaultHttpClient();
2:  HttpGet httpget = new HttpGet(service + id);
3:  httpget.setHeader("Accept", "application/json");
4:  HttpResponse response = httpClient.execute(httpget);
```

AnsServices.java

```
1:  HttpClient httpClient = new DefaultHttpClient();
2:  HttpResponse response = httpClient.execute(httpPost);
```

図 5 タスク 1 における検索結果の例

提案手法において記述された検索クエリの中には以下のようなものが見られた．

```
HttpClient $a = ?? $a.execute
```

この検索クエリは， HttpClient インターフェースの execute メソッドの呼び出しがある箇所を“$a.execute”と表現し，そのレシーバとなるオブジェクトの生成を“HttpClient $a =”と表現したものであると推測される．どちらも共通の $変数“$a”を指定することで，これにマッチする変数のデータフローを考慮した検索が可能となっている．この検索クエリを用いて検索すると，検索クエリを送信してから約 5 秒程度で検索結果が返され，上から 2 つ目と 3 つ目のファイルのコード例として図 5 が見られた．太字の変数が $変数にマッチした変数である．これらのコード例の 1 行目はともに同じであり，実装クラス DefaultHttpClient のインスタンスを生成している．実際，このコード例に倣って実装すれば，タスクを完了することが可能である．このような形で API の理解支援を行うことができたため，タスク 1 では提案手法を利用した方が得点率が高くなったと考えている．

タスク 2 において提案手法の有用性が見られなかったのは，タスク 2 は主にユーティリティメソッドを使用するサブタスクから構成されるが原因であると考えている．メソッドの多くが static メソッドであるため，オブジェクトの状態等を考える必要がなく，使い方はあまり難しくない．そのため，コード例による API の理解支援を必要とせず，提案手法と既存手法の差がほとんど生まれなかったと考えられる．

タスク終了後，アンケートにより提案手法の検索クエリの記述の容易さを 4 段階で評価してもらったところ，被験者 8 人のうち 7 人が「どちらかというと容易である」と回答した．提案手法の検索クエリに対し「最初は検索クエリの記述の仕方が分からなかったが使っているうちに理解してきた」という意見や，「検索に慣れてくると欲しいコード例を検索できて便利だと感じた」という意見が見られた．また，タスク中に送信された検索クエリは，中央値で 4.3 秒，平均値で 5.0 秒の検索時間を要することが分かった．これは一般的な開発において十分実用的な範囲であると考えている．

5 まとめと今後の展望

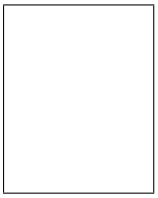
本研究では API 理解を支援するための技術として，独自の検索クエリを用いて変数のデータフローを指定することによるコード例検索手法を提案した．

今後の課題としては，実装したウェブアプリケーションを外部に公開し，実際の開発者からのフィードバックを受け取ることが挙げられる．

謝辞 本研究は JSPS 科研費 JP25220003, JP26280021, JP15H02683 の助成を受けたものです．

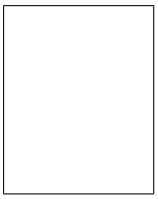
参考文献

[1] searchcode.com. https://searchcode.com.
[2] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus. How can i use this method? In *Proceedings of ICSE*, pages 880–890, 2015.
[3] M. P. Robillard. What makes APIs hard to learn? answers from developers. *IEEE Software*, 26(6):27–34, 2009.
[4] M. P. Robillard and R. Deline. A field study of API learning obstacles. *Empirical Software Eng*, 16(6):703–732, 2011.
[5] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74:470–495, 2009.
[6] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How well do search engines support code retrieval on the web? *ACM TOSEM*, 21(1):4:1–4:25, 2011.
[7] S. Thummalapenta and T. Xie. Parseweb: A programmer assistant for reusing open source code on the web. In *Proceedings of ASE*, pages 204–213, 2007.
[8] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei. MAPO: Mining and recommending API usage patterns. In *Proceedings of ECOOP*, pages 318–343, 2009.
[9] 竹之内 啓太. 変数のデータフローを考慮した API 利用コード例の検索手法. 大阪大学修士論文, http://sel.ist.osaka-u.ac.jp/lab-db/Mthesis/archive/128/128.pdf, 2017.



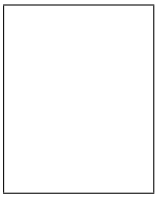
竹之内啓太

2015 年大阪大学基礎工学部情報科学科卒業．2017 年同大学大学院情報科学研究科博士前期課程修了．同年株式会社エヌ・ティ・ティ・データ入社．在学中，ソースコードの静的解析に関する研究に従事．



石 尾　隆

2003 年大阪大学大学院基礎工学研究科博士前期課程修了．2006 年同大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．2007 年大阪大学大学院情報科学研究科助教．2017 年奈良先端科学技術大学院大学情報科学研究科准教授．博士（情報科学）．プログラム解析，プログラム理解に関する研究に従事．



井上 克郎

1979 年大阪大学基礎工学部情報工学科卒業．1984 年同大学大学院博士課程修了．同年同大学基礎工学部助手．1984〜1986 年ハワイ大学マノア校情報工学科助教授．1989 年大阪大学基礎工学部講師．1991 年同助教授．1995 年同教授．2002 年大阪大学大学院情報科学研究科教授．工学博士．ソフトウェア工学，特に，ソフトウェア開発手法，プログラム解析，再利用技術の研究に従事．