Vol.2018-SE-199 No.27
2018/7/20

構文定義記述を用いた多言語対応コードクローン検出ツールの開発

瀬村 雄一[1,a)]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：字句単位のコードクローン検出ツールである CCFinder は，対象のソースコードの字句解析を行っている．CCFinder に対応する言語を増やすにはその言語の字句解析を実装する必要があるが，多くの言語に対して 1 つずつ字句解析を実装するのは，コードクローン検出ツールの開発者に各言語への理解が求められ，手間のかかる作業である．コードクローン検出ツールの使用者が，新たな言語に容易に対応できる仕組みを作ることで，開発者の労力を減らすことができる．本研究ではコードクローン検出における字句解析に必要な情報を，構文解析器生成系の構文定義ファイルから自動的に抽出する手法を提案する．そして，その手法を用いて構文定義記述解析モジュールを開発し，既存のコードクローン検出ツールである CCFinderSW に字句解析情報を与えることで，多言語のコードクローン検出を行う仕組みを開発した．適用実験では，構文解析器生成系の 1 つである ANTLR の構文定義ファイルから，プログラミング言語のコメント記法と予約語を抽出し，CCFinderSW を用いてコードクローン検出を行い，提案手法の有効性を確認した．

キーワード：コードクローン，字句解析，ANTLR

A Tool for Multilingual Detection of Code Clones Using Syntax Definitions

Yuichi Semura[1,a)]　Norihiro Yoshida[2]　Eunjong Choi[3]　Katsuro Inoue[1]

1. はじめに

コードクローンとは，ソースコード中に存在する互いに一致，または類似したコード片を指し，主に既存のコード片のコピーアンドペーストによって生成される [3]．一般的にコードクローンの存在は，ソフトウェア保守を困難にしている大きな要因の 1 つとして挙げられている．あるコード片にバグが見つかった場合，そのコード片のコードクローンにもバグが含まれる可能性が高い．そのため，開発者はあるコード片にバグが見つかった場合，そのコード片または一致または類似するすべての箇所に対して同一の修正を行うか検討する必要がある [4]．したがって，開発者はコードクローンに関する情報を認識しておく必要があるが，大規模なソフトウェアにおいては，コードクローンにあたる箇所が膨大な数になることにより，その全てを認識しておくことは現実的でない．そこで，開発者を支援するためにコードクローン検出ツールが提案されている [2]．

コードクローン検出ツールの 1 つである CCFinder は，C，C++，Java，COBOL，Fortran の言語で書かれたソースコードからコードクローンを検出することが可能である [6]．CCFinder では字句単位のコードクローンを検出するための前処理として，ソースコードを言語の文法に沿って字句単位に分割している．この処理は一般的に字句解析と呼ばれる [13]．また CCFinderX は，CCFinder のバージョンアップとして開発されたコードクローン検出ツールであ

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[3] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[a)] y-semura@ist.osaka-u.ac.jp

り，字句解析部のユーザによる変更を可能にしている[*1]．これはコードクローン検出を行いたい言語に対応する字句解析部をユーザが用意することで，その言語のコードクローン検出が可能になるということを意味する．しかし，字句解析部を用意するにはその言語の文法を理解することが必要であるため，字句解析部の実装は手間のかかる作業である．このような場合に，多くの言語に対応できる簡単に仕組みを使うことで，ユーザの手間を減らすことが出来る [11]．

CCFinderX では特定の字句やその組み合わせについて，正規化処理を行っている．これは変数名や関数名などを全て同一の字句に置き換える処理であり，実用的に意味のあるコードクローンだけを検出するために行われる．変数名や関数名で使用できる字句は、プログラミング言語によって設定が異なるために，言語によって別の設定を行わなければならない．

CCFinderSW [12] では多言語の字句単位のコードクローン検出を目的として，ユーザによって容易に字句解析部の変更が可能になるような字句解析機構を提案している．字句解析機構の変更箇所を，主にコメント除去のルールと置き換えが必要な字句に限定した．この仕組みによってユーザによる字句解析部のコーディングの手間が省かれ，新たな言語への対応が容易になっている．

構文解析器生成系は，字句解析器や構文解析器を自動的に生成するプログラムであり，パーサジェネレータとも呼ばれる．構文解析器生成系は，構文定義記述を用いることで字句解析器・構文解析器を生成することができるため，字句解析器のコーディングの手間を省くことができる．代表的な構文解析器生成系に，Yacc [5] や JavaCC[*2]，ANTLR[*3]などがある．

本研究では，コードクローン検出ツールを多言語に対応させるために，字句解析に必要な情報を構文解析器生成系の構文定義ファイルから自動的に抽出する方法を提案する．また構文解析器生成系の中から ANTLR を対象として，ANTLR で用いられる構文定義記述からプログラミング言語のコメント記法と予約語を抽出し，字句解析を行うモジュールを開発した．このモジュールによって出力された字句解析情報を用いて，CCFinderSW でコードクローン検出を行う．

以降，2 章では本研究の背景として，コードクローン検出ツールである CCFinderX と CCFinderSW の説明を行う．3 章では本研究で対象とした構文解析器生成系の 1 つである ANTLR の構文定義記述の形式について説明し，4 章では ANTLR の構文定義ファイルから情報を抽出する役割を担うモジュールについて説明を行う．5 章では構文定義記述解析モジュールを用いたコメントと予約語の抽出実験と，コードクローン検出実験について記述する．6 章ではまとめと今後の課題について述べる．

[*1] http://www.ccfinder.net/
[*2] https://javacc.org/
[*3] http://www.antlr.org/

2. コードクローン検出ツール:CCFinderX/CCFinderSW

本章では本研究の背景として，コードクローン検出ツールである CCFinderX と CCFinderSW についての説明を行う．

コードクローンとは，ソースコード中に存在する互いに一致，または類似したコード片を指し，主に既存のコード片のコピーアンドペーストによって生成される [3]．一般的に，互いにコードクローンとなるコード片はクローンペアと呼ばれる．コードクローンに普遍的な定義は存在しない．本論文ではコードクローンの定義として以下の 2 つのタイプの分類を用いる [10]．

タイプ 1
空白，タブ文字，改行やコメントなどを除いて一致するコードクローン．

タイプ 2
タイプ 1 の条件に加えて，リテラル，型，識別子を除いて一致するコードクローン．

CCFinderX は，字句単位のコードクローンを検出するツールである．CCFinderX の特徴として以下のようなものが挙げられる．

- 変数名や関数名などの字句を同一字句に置き換えることで，タイプ 2 までのコードクローンを検出できる．
- 数百万行規模のシステムにも実用時間で解析可能である．
- 言語依存部分を取り替えることでさまざまな言語に対応している．
- しきい値を与えることで，字句数がしきい値未満のコードクローンを検出しないようにできる．

CCFinderX は，CCFinder のバージョンアップとして開発されたコードクローン検出ツールであり，字句解析部のユーザによる変更を可能にしている．これはコードクローン検出を行いたい言語に対応する字句解析部をユーザが用意することで，その言語のコードクローン検出が可能になるということを意味する．しかし，字句解析部を用意するには言語の文法を理解することが必要であり，字句解析部のコーディングは手間のかかる作業である．

CCFinderSW は，CCFinderX と同様に字句単位のコードクローンを検出するツールである [12]．CCFinderSW は多言語対応を行ったコードクローン検出ツールであり，字句解析部に対象言語のコメントルールと予約語を入力することで多くの言語に柔軟に対応できる．この仕組みによってユーザによる字句解析部のコーディングの手間が省

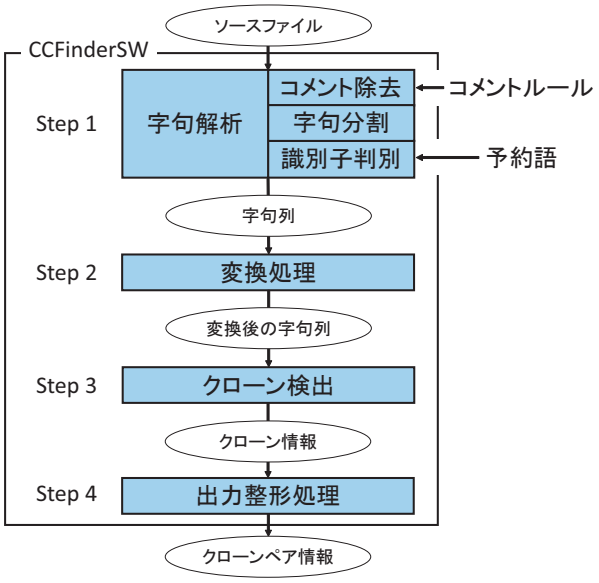

図 1　CCFinderSW の概要
Fig. 1　An overview of CCFinderSW

かれ，新たな言語への対応が容易になっている．ここで，CCFinderSW の処理概要について記述する．4 つの Step に分けられ，各 Step の詳細を説明する．

Step 1:字句解析
ソースコードをプログラミング言語の文法に沿って字句列に変換する．その際，空白とコメントは機能に影響しないので無視される．字句解析はコメント除去，字句分割，識別子判別といった処理が行われている．入力されたコメントルールはコメント除去で使用され，予約語一覧は識別子判別で使用される．

Step 2:変換処理
分割された字句列から実用的に意味のあるコードクローンだけを検出するための変換を行う．これは変数名や関数名などを全て同一の字句に置き換える処理である．

Step 3:クローン検出
変換された字句列を，比較しコードクローンを検出する．この比較には N-gram を用いたアルゴリズムを採用している．

Step 4:出力整形処理
最後に出力整形処理を行い，検出されたクローンペアについて，もとのソースコードでどのファイルに存在するか，行番号と列番号が出力される．

以降，Step 1 の詳細として 2.1 節でコメント除去について，2.2 節で字句分割についての詳細を説明する．2.3 節では，Step 1 の識別子判別と・Step 2 の変換処理について記述する．2.4 節では現在の CCFinderSW の問題点について説明する．

2.1　コメント除去
CCFinderSW の字句解析部では，まず入力されたソースコードからコメントを除去する．これはコードクローンの定義に基づいて，コードクローンにはコメントや空白は含まれないためである．CCFinderSW のコメント除去部では，ユーザによるコメントルールの入力を必要とする．除去可能なコメントのルールとして，行コメント，複数行コメント，行全体コメント，複数行全体コメントの 4 種類のルールが用意されている．ユーザはそのルールをもとにコメントルールを記述したファイルを作成し，実行時に入力としてそのファイルを CCFinderSW に与える．

2.2　字句分割
コメント除去を行った後は字句分割を行う．字句分割で使われるルールは以下の通りである．番号が小さいルールほど優先される．

(1) 文字，文字列リテラルは 1 字句とする．
(2) 空白と改行の前後で字句を分割する．
(3) 記号は 1 文字ずつで分割する．記号が複数文字で 1 つの意味を表す場合でも，1 文字で 1 字句とする．
(4) それ以外の連続した英数字列は 1 字句とする．

2.3　識別子判別・変換処理
識別子判別では字句分割で英数字列として分割された字句が，識別子か予約語かを判定するものである．これは Step 2 の変換処理のために行われる．予約語は変数名や関数名に使用できない文字列のことを指し，プログラミング言語によって異なっている．ユーザは予約語一覧を記述したファイルを作成し，実行時に入力としてそのファイルを与える．

変換処理は，実用的に意味のあるコードクローンだけを検出するために行われる．これは字句解析で識別子と判別された変数名や関数名などを，全て同一の字句に置き換える処理である．

2.4　CCFinderSW の問題点
CCFinderSW は入力としてコメントルールと予約語の入力が必要であるが，新たな言語に対応するにはコメントルールを記述するファイルと予約語の一覧を記述するファイルの 2 つをユーザが作成しなければならない．コメントルールを記述するファイルには独自の文法が存在し，ユーザはそれを理解する必要がある．また全ての予約語を記述したファイルを用意することは非常に手間である．これらのユーザによる手間を軽減させるための仕組みが必要である．

3.　構文解析器生成系：ANTLR
構文解析器生成系は，字句解析器や構文解析器を自動的

に生成するプログラムであり，パーサジェネレータとも呼ばれる．構文解析器生成系は，構文定義記述を用いることで字句解析器・構文解析器を生成することができるため，字句解析器のコーディングの手間を省くことができる．代表的な構文解析器生成系に，Yacc や JavaCC，ANTLR などがある．

構文解析器生成系の 1 つである ANTLR は，構文木の構築・探索が可能な構文解析器を生成する．Twitter での検索クエリの解析には ANTLR が採用されるなど広く使用されていて [7]，2018 年現在も開発が進められているため，対象として選択した．ANTLR が構文解析器を生成するターゲットとなる言語は C++，Java，Python，Go などがある．

本章では，本研究を行うにあたって必要となった ANTLR の構文定義記述の文法と，調査を行ったファイルに多く採用されていた表現方法について説明する．

ANTLR で構文定義記述の例として，四則演算式を表すものを掲載する．まず 1 行目に，**grammar 文法名**と書くことで，このファイルが表す文法の名前を宣言する．2 行目以降は文法を構成するルールについて記述している．ANTLR の文法を定義するルールには，字句解析ルールと構文解析ルールの 2 つが存在する．

四則演算式を表す構文定義ファイル

```
grammar Prog;
prog: expr;
expr: term ( ( '+' | '-' ) term )*;
term: factor ( ( '*' | '/' ) factor )*;
factor: INT | '(' expr ')';
INT : [0-9]+;
WS: [ \t\r\n]+ -> skip;
```

ルールの記述法は，**ルール名 : ルールブロック ;** となり，このようなルールが 1 つのファイルに複数定義されることとなる．字句解析ルールのルール名は大文字から始まり，構文解析ルールのルール名は小文字から始まる．本研究で構文解析ルールについてはほとんど使用しないため，本稿では字句解析ルールにおいてのみ説明する．

ANTLR で用いられる字句解析ルールでの字句表現について，ANTLR の開発者によるドキュメント [8] が存在する．その中から本稿に必要なものを抜粋し，表 1 に示した．先に示した四則演算式を表すファイルの中では，字句解析ルールとして 'INT' と 'WS' が存在する． 'INT' は数字列を表していて，'WS' は空白とタブ文字と改行を表している．

次に，ANTLR で使用されるコマンドという機能について説明する．ANTLR は字句解析ルールを用いてソースコード中で出現する字句を定義するが，それぞれの字句にコマンドを指定して特定の操作を施すこともできる．

表 1　ANTLR で使用される主要な字句
Table 1　Main tokens used in ANTLR

字句	説明
'リテラル'	文字，または文字列にマッチする．
[文字集合]	指定された文字のどれか 1 つにマッチする．a-z のように書くことで範囲を指定することも可能．正規表現で使用されるものと同様．
.	任意の一文字にマッチする
~x	x で記述されている集合にマッチしない任意の一文字にマッチする． x は 1 文字のリテラルや文字集合が指定される．本稿では NOT 演算子と呼ぶ
x*	x の 0 回以上の繰り返しにマッチする．
x?	x の 0 回，または 1 回の出現にマッチする．
x*?	x の 0 回以上の最短の繰り返しにマッチする．
x|y	x または y にマッチする．

コマンドは **ルール名 : ルールブロック − >コマンド**; のように記述する． 本稿に必要なコマンドの種類は 2 つである．1 つ目は skip である．skip を指定された字句は，字句解析によって読み飛ばされる．2 つ目は channel コマンドである．channel(x) のように記述し，x にはあらかじめ定義された channel 名が入る．channel コマンドは字句をある channel に渡す．例えばコマンドを channel(HIDDEN) と指定すると，HIDDEN に渡す．特に channel(HIDDEN) は skip と似た働きをしていて，このコマンドを指定された字句は Parser によって無視される [9]．

4. 構文定義記述解析モジュールの開発

本研究では，字句解析に必要な情報を，構文解析器生成系の構文定義記述ファイルから自動的に抽出する方法を提案する．そして構文解析器生成系の中から ANTLR を対象として，ANTLR で用いられる構文定義記述を解析しプログラミング言語のコメント記法と予約語を抽出するモジュールを開発した．このモジュールによって出力された字句解析に必要な情報を用いて，CCFinderSW でコードクローン検出を行う．本章では新たに開発した構文定義記述解析するモジュールと CCFinderSW の変更点について述べる．

図 2 は新たに開発を行った構文定義記述解析モジュールと CCFinderSW の関係を表したものである．本研究で新たに開発した部分は図 2 のオレンジ色でハイライトされた部分であり，既存の CCFinderSW に含まれる部分は青色でハイライトされた部分である．構文定義記述解析モジュールは，ANTLR の構文定義記述からコメントを表す正規表現と予約語を抽出するものである．また CCFinderSW の字句解析部を拡張して，コメントを表す正規表現を用いてコメント除去を行う機能を追加した．これによって，字句分割以降の処理を従来の CCFinderSW と同じ流れでソー

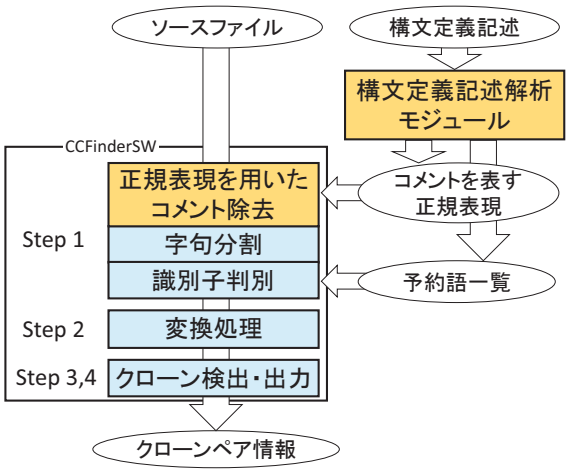

図 2　構文定義記述解析モジュールと CCFinderSW
Fig. 2　An anaylsis module for syntax definition and CCFinderSW

スファイルを扱うことができる．CCFinderSW と構文定義記述解析モジュールを合わせると，ソースファイルと構文定義記述を入力として与えることでクローンペア情報が出力されることになる．

本研究で開発した構文定義記述解析モジュールは，まず ANTLR の構文定義記述を構文解析して構文木を生成している．この構文解析には，Java 言語で動作する構文解析器を ANTLR で生成して組み込んだ．

以降，4.1 節ではコメント記述ルールの正規表現への変換について説明し，4.2 節では予約語一覧の正規表現への変換について説明する．

4.1　コメント記述ルールの正規表現への変換

本節では ANTLR の構文定義記述から，コメントを表す正規表現の抽出方法について述べる．まず正規表現で抽出する理由としては，ANTLR の構文定義記述は正規表現と近い表現がされており，変換可能であるからである．また CCFinderSW で採用されていたコメントルールの分類以外のコメントルールにも，幅広く対応出来る可能性があるからである．

正規表現の抽出は以下の 4 つの Step に分けられる．

Step A　全てのルールの中からコメントに関すると考えられるルールを選びだす．

Step B　それぞれのルールの中で，別のルールを参照している部分を再帰的に詳細化する．

Step C　詳細化されたルールを Java で使用される正規表現に変換する．

Step D　生成された正規表現を全て結合して 1 つの表現にする．

各 Step の詳細は以下のとおりである．

Step A　まず，全てのルールの中からコメントに関すると考えられるルールを選び出す．これには判断基準を 4 つ設け，そのうちの少なくとも 1 つに当てはまることでコメントに関するルールであると識別する．その 4 つの判断基準とそれらに当てはまるルールの例を以下に示す．例示された 4 つのルールは全て C 言語の複数行コメントに相当する表現である．

(1) ルール名に Comment や COMMENT などの文字が含まれている．
(2) コマンドの skip が呼ばれている．
(3) コマンドの channel(HIDDEN) が呼ばれている．
(4) コマンドの channel が呼ばれていて，channel 名に Comment や COMMENT などの文字が含まれている．

コメントの定義例

```
Comment:'/*'.*?'*/';
Block1: '/*'.*?'*/' -> skip;
Block2: '/*'.*?'*/' -> channel(HIDDEN);
Block3: '/*'.*?'*/' -> channel(BComment);
```

Step B　選び出されたコメントに関するルールを再帰的に詳細化する．3 節で示した四則演算式を表す構文定義ファイルの例では，term という名前のルールの中で factor というルールが埋め込まれている．このようにルールの中で他のルールが参照されている場合は，Step C で正規表現に変換するために参照先ルールの内容を再帰的に代入し，詳細化を行う．

Step C　詳細化されたルールを Java で使用されるような正規表現に変換する．これは本研究でのモジュールが Java で開発されていることで，ANTLR で使用される表現とは部分的に異なっているためである．Java で用いられる正規表現と ANTLR の構文定義で使用される表現の違いで，変換が必要なものは 3 つある．1 つ目はシングルクォーテーションである．ANTLR の構文定義記述では，対象言語に出現する実際のリテラルをシングルクォーテーションで囲んで記述する．正規表現ではこのシングルクォーテーションは不要であるため除去する．2 つ目は NOT 演算子である．ANTLR の構文定義記述では，ある特定の文字集合にマッチしない集合を表すために先頭に，NOT 演算子の役割をする '~' をつける． この NOT 演算子に直接的に相当するものは正規表現には存在しないため，正規表現の否定的先読みを用いて同等の表現に変換する．3 つ目は任意の 1 文字を表す '.' である．ANTLR の構文定義記述ではこの '.' は任意の 1 文字を表すのに対し，Java で使用される正規表現では改行以外の任意の 1 文字を表し，定義がそれぞれで異なっている．この定義の差異を埋めるため，構文定義記述で用いられる '.' を正規表現での同等の表現である '[\s\S]' に変換することで対応している．

Step D　生成された正規表現を全て結合して 1 つの表現にする．Step C で生成された正規表現の全ての論理和

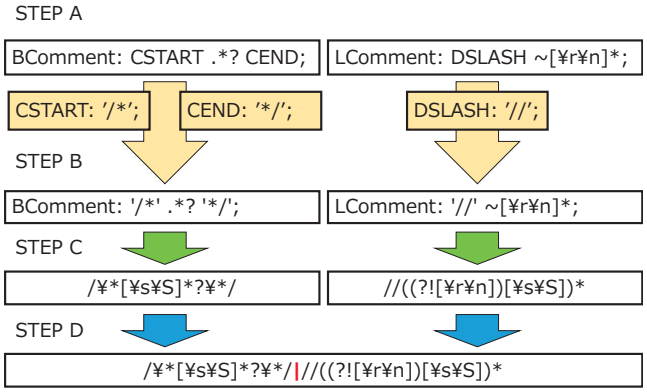

図 3　コメント記述ルールの正規表現への変換例
Fig. 3　An example of transformating comment rules into a regular expression

をとる．これは，全ての正規表現を間に '|' を挟んで結合することで行われる．この Step D を終えた時点で，コメント記述ルールの正規表現の変換が終了する．

図 3 は，以上のコメント記述ルールの正規表現への変換例である．まず Step A で *BComment* と *LComment* という 2 つのコメントを表すと考えられるルールを選択する．次に Step B では選択されたルールの詳細化を行う．*BComment* では *CSTART* と *CEND* という他のルールへの参照があるため，参照先のルールを代入して他のルール名が含まれない表現にする．Step C では正規表現での同等の表現に変換し，最後に Step D で生成された全ての正規表現を結合する．

4.2　予約語一覧の正規表現への変換

本節では ANTLR の構文定義記述から，予約語一覧の抽出方法について述べる．

本ツールを開発する際に，ANTLR の構文定義記述内での予約語の記述法を調査した結果，大きく分けて 2 種類の記述法があった．以下に 2 種類の記述法について，*while* という予約語を用いて例示する．

予約語の表記例
```
WHILE:'while';
WHILE:[wW][hH][iI][lL][eE];
```

1 つ目の記述法では，*WHILE* というルール名に *while* という単語が紐付けられている．この記述法は最も広く使われているものであった．そして 2 つ目の記述法では，正規表現で使用されるような文字クラスを用いた書き方がされている．これは *while* に含まれる 5 文字のそれぞれに大文字と小文字のどちらでもマッチングすることを許す記述法である．1 つ目の記述法の右辺も正規表現として考えらえるため，予約語一覧もコメントルールと同様に正規表現で生成することにした．予約語一覧の抽出は以下の 5 つの Step に分けられる．

Step α　全てのルールで，出現しているリテラルのうち英字列に一致するものを抽出する．

Step β　全てのルールで，別のルールを参照している部分を再帰的に詳細化する．

Step γ　Step β で詳細化されたルールを Java で使用される正規表現に変換する．

Step δ　Step γ で変換された正規表現で，英字列を表しているものを選出する．

Step ϵ　Step α で抽出されたリテラルと Step δ で選出された正規表現を全て結合して 1 つの表現にする．

Step α は，ルールブロックに出現するリテラルのうち英字列に一致するものを全て抽出し予約語であるとみなす．Step β は 4.1 節で説明したコメント抽出の Step A と同様であり，Step γ はコメント抽出の Step C と同様である．Step δ は Step γ で生成された正規表現が，英字列を表しているものを選び出す．Step ϵ はコメント抽出の Step D と同様に '|' を用いた結合を行う．

Step α と Step δ では，予約語は英字列であるという前提に基づいている．本研究の調査では予約語に含まれる文字として英字列以外に，'_' や '@' などの記号も含まれることがあったため，モジュールに抽出したい予約語に含まれる文字を指定する機能を実装した．

5.　適用実験

本章では，開発した構文定義記述解析モジュールの有用性を確認するために行った 2 つの適用実験について説明する．

5.1　コメントと予約語の抽出実験

本研究で開発したモジュールを用いて，どの程度の構文定義記述ファイルにおいてコメントと予約語の情報が抽出可能であるかを示す実験を行った．

実験の対象となるファイルは Github のリポジトリである grammars-v4[*4]を使用した．このリポジトリは ANTLR の文法ファイルを集めたものであり，約 150 種類の文法ファイルが含まれている．ANTLR の開発者である Parr も含めて，170 人以上の貢献者が存在し，現在でも更新が続けられている．実験で使用したリポジトリのスナップショットは 2017 年 12 月 14 日時点のものである．

まず本研究では，リポジトリに含まれている 154 種類の文法について調査を行った．その中から，Github の Advanced Code Search[*5]の検索対象に登録されている 43 言語を実験対象とした．次にその 43 言語の構文定義ファイルからコメントと予約語と考えられる定義についてあらかじめ記録しておき，構文定義記述解析モジュールがコメントと予約語についての情報を抽出できるかどうか判定した．

[*4] https://github.com/antlr/grammars-v4
[*5] https://github.com/search/advanced

本研究における予約語の定義について記述する．一般的に予約語とは，変数名や関数名に使用できない文字列のことを指し，それぞれのプログラミング言語によって定められている．一方，プログラミング言語におけるキーワードとは，あるプログラミング言語で使用される特別な意味を持つ文字列のことを指す．予約語とキーワードは似通った存在と言われているが，言語によってはキーワードであっても予約語ではない文字列が存在し，そもそも予約語が存在しない言語も存在する．例として，実験対象言語の 1 つである Fortran には予約語は存在せず，キーワードも変数名に使用できる．この実験では，各言語で使用されるキーワードがそのまま予約語になると定義して実験を行った．

表 2 は，各言語の構文定義ファイルに対してコメントと予約語についての情報を抽出結果を示したものである．各言語に対し，モジュールがコメントと予約語の情報が抽出可能かどうかを示している．○は抽出可能，×は抽出不可能を示し，「定義なし」は文法ファイルにコメントと予約語らしき記述がなかったことを示している．結果，プログラミング言語の文法を表す 43 の構文定義記述ファイルのうち，コメントは 93 %，予約語は 97 %のファイルから抽出することができた．

次に，×となった理由についてそれぞれ記述する．

brainf*ck　コメント情報の抽出ができない．brainf*ck は難解プログラミング言語であり，特定の 8 文字のみソースコードが記述され，それ以外の文字が無視され，コメントとみなされるという特殊な文法を持つものであった．

css　予約語情報の抽出ができない．css の構文定義ファイルの中では，抽出したい予約語の中に改行が含まれることを許すような記述がされているなど，特殊な書き方がされているため抽出不可能と判断した．

lua　コメント情報の抽出ができない．lua に用いられるコメント記号が構文定義ファイルの中で複雑に定義されており，正規表現での抽出は不可能と判断した．

rexx　コメント情報の抽出ができない．構文定義ファイルでネストを許す複数行コメントの定義が，2 つのルールをループすることで表現されているため，他の言語の構文定義ファイルとは違った特殊な書き方がされているため，抽出は不可能と判断した．

5.2　コードクローン検出実験

本節では ANTLR の構文定義記述から抽出したコメントに相当する正規表現と予約語一覧を，コードクローン検出へ適用可能であることを確認するために行った実験について記述する．

実験の手順について説明する．まず，ある言語のソースコードを 1 つ用意し，そのソースコードのコピーを同時に作成する．この時点では 2 つのソースコードは同一のため，タイプ 1 のコードクローンといえる．次にコピーのソースコードに対して，適切な箇所にコメントを追加して変数名を書き換える．このような処理によって，手動でタイプ 2 のコードクローンを作成する．最後に，このオリジナルとコピーされたファイルと，構文定義記述ファイルから抽出した情報を入力として CCFinderSW に渡し，コー

表 2　コメントと予約語の抽出結果
Table 2　Result of the extraction of comments and reserved words

言語	コメント	予約語
agc	○	○
antlrv4	○	○
apex	○	○
asm6502	○	○
aspectj	○	○
brainf*ck	×	定義なし
c	○	○
clojure	○	○
cobol85	○	○
cool	○	○
cpp14	○	○
csharp	○	○
css3	○	×
ecmascript	○	○
erlang	○	○
fortran77	○	○
golang	○	○
html	○	定義なし
idl	○	○
java9	○	○
kotlin	○	○
lua	×	○
modelica	○	○
modula2	○	○
objective-c	○	○
pascal	○	○
php	○	○
plsql	○	○
prolog	定義なし	定義なし
protobuf3	○	○
python3	○	○
r	○	定義なし
rexx	×	○
scala	○	○
smalltalk	○	○
smtlibv2	○	○
swift3	○	○
vba	○	○
verilog2001	○	○
vhdl	○	○
visualbasic6	○	○
webidl	○	○

表 3　コードクローン検出の結果
Table 3　Result of code clone detection

言語	コードクローン検出
c	○
cpp14	○
csharp	○
golang	○
java9	○
python3	○
smalltalk	○
vhdl	○
visualbasic6	○

ドクローン検出を行い，タイプ 2 のコードクローンとして検出されることを確認する．

本実験の対象として 9 言語を選出した．この言語は 5.1 節の実験でコメントと予約語のどちらも抽出可能となった言語のうち，変数名の変更とコメントの追加が十分に可能と考えられたものを複数選択した．

コードクローン検出の結果を表 3 に示した．全ての言語においてタイプ 2 のコードクローンの検出が可能であることが確認できた．

6. まとめと今後の課題

本研究ではコードクローン検出における字句解析に必要な情報を，構文解析器生成系の構文定義ファイルから自動的に抽出する手法を提案した．そして，その手法を用いて構文定義記述解析モジュールを開発し，既存のコードクローン検出ツールである CCFinderSW に字句解析情報を与えることで，多言語のコードクローン検出を行う仕組みを開発した．開発したモジュールの適用実験では，プログラミング言語の文法を表す 43 の構文定義記述ファイルのうち，コメントは 93 %，予約語は 97 %から抽出できたことを示した．そして抽出した情報を入力として CCFinderSW に与えることで，9 言語においてタイプ 2 のコードクローンの検出が可能であることが確認した．

構文定義記述は対象となるプログラミング言語の文法に依存するが，書き手にも依存する．つまり，同じ文法であっても複数の記述法で表現することができる．本研究での開発は，構文定義記述の調査において多く存在した記述法に対して行われたものである．新たな記述法に対応するためには，モジュールを拡張しなければならない．

CCFinderSW でのコメント除去を，コメントに相当する正規表現を用いることで行ったが，現在の方法では完全にコメントを削除することはできない．これはシングルクォーテーションやダブルクォーテーションを用いた文字列リテラルによる例外に対応してないためである．このような問題に対処しようとすると，正規表現が複雑になるため [1]，正規表現でのコメント削除は避けたほうがよい．今後の課題として構文定義ファイルからコメントルールと合わせて文字列リテラルの情報も取得し，多くのコメントルールに対応出来るような形式に変換して，コメント除去が行えるように CCFinderSW を拡張することが必要である．

本研究で開発したモジュールで抽出される文字列はプログラミング言語のキーワードであり，予約語かどうかの判定は行われていない．今後の課題として，抽出されたキーワードが予約語であるか判定することが挙げられる．

謝辞　本研究は JSPS 科研費 JP25220003, JP18H04094, JP16K16034 の助成を受けた．

参考文献

[1] Friedl, J. E.: *Mastering regular expressions* (2002).
[2] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌，Vol. J91-D, No. 6, pp. 1465–1481 (2008).
[3] 井上克郎，神谷年洋，楠本真二：コードクローン検出法，コンピュータソフトウェア，Vol. 18, No. 5, pp. 47–54 (2001).
[4] 肥後芳樹，神谷年洋，楠本真二，井上克郎：コードクローンを対象としたリファクタリング支援環境，電子情報通信学会論文誌，Vol. Vol.J88-D-I, No. 2, pp. 186–195 (2005).
[5] Johnson, S. C.: *Yacc: Yet another compiler-compiler*, Vol. 32, Bell Laboratories Murray Hill, NJ (1975).
[6] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670 (2002).
[7] Parr, T.: About The ANTLR Parser Generator, http://www.antlr.org/about.html.
[8] Parr, T.: antlr4/index.md at master antlr/antlr4, https://github.com/antlr/antlr4/blob/master/doc/index.md.
[9] Parr, T.: *The definitive ANTLR 4 reference* (2013).
[10] Roy, C. K., Cordy, J. R. and Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495 (2009).
[11] Sakamoto, K., Shimojo, K., Takasawa, R., Washizaki, H. and Fukazawa, Y.: OCCF: A framework for developing test coverage measurement tools supporting multiple programming languages, *Proc. of ICST 2013*, IEEE, pp. 422–430 (2013).
[12] Semura, Y., Yoshida, N., Choi, E. and Inoue, K.: CCFinderSW: Clone Detection Tool with Flexible Multilingual Tokenization, *Proc. of APSEC 2017*, IEEE, pp. 654–659 (2017).
[13] 植田泰士，神谷年洋，楠本真二，井上克郎：クローン検出ツールを用いたソースコード分析ツールの試作，電子情報通信学会技術研究報告，Vol. 101, No. 240, pp. 17–24 (2001).