Syntax Sensitive and Language Independent Detection of Code Clones

Kazuaki Maeda

Abstract—This paper proposes a new technique to detect code clones from the lexical and syntactic point of view, which is based on PALEX source code representation. The PALEX code contains the recorded parsing actions and also lexical formatting information including white spaces and comments. We can record a list of parsing actions (shift, reduce, and reading a token) during a compiling process after a compiler finishes analyzing the source code. The proposed technique has advantages for syntax sensitive approach and language independency.

Keywords—Code Clones, Source Code Representation, XML, Parser, Parser Generator

I. INTRODUCTION

“Cut and paste” and “copy and paste” were transferred into the context of computer-based editing systems in 1970’s[1]. Those user-interface actions eliminate duplicated works and improve the productivity. It is also useful in the context of program development during editing source code. The “copy and paste” activity is called “code cloning,” and the copied and pasted fragment of source code is called “code clone.”

Many researches show that a significant amount of source code contains the code clones. One of the works shows that 19% of the source code is cloned in the complete source of the X Window System[2]. Another work[3] shows that the average percentage of code clones is 12.7% of all subsystems. In an extreme case, the average percentage of code clones is 59%[4].

The code clones offer a simple way to reuse source code. If a fragment of source code is already tested, there are fewer bugs than source code which is written from scratch. In another case, the use of particular application program interfaces (APIs) often require ordered series of procedure calls to achieve desired behaviors. For example, to develop GUI applications using the Java Swing APIs, similar orderings are common with the libraries. Device drivers of operating systems usually contain large code clones because all the drivers have the same APIs and most of them implement a similar logic. In the case of developing a new driver for a new hardware model, cloning the existing driver prevents the risk of changing the existing driver[5].

Many techniques to detect the code clones have been proposed in the past. Sophisticated approaches in the proposed techniques need to parse source code[3]. Their approaches provide powerful code clone detection, but they need a parser for each target programming language, and they are dependent on the programming languages.

Based on my experiences, there are two approaches to develop a parser. One approach is to develop a parser from scratch by reading the programming language specification. There are some cases where it takes more than one week to develop just only a parser with high quality because the specification of recently popular programming languages is complex. Second approach is to get grammar definitions from major web sites, or find them using web search engines. There are some web sites including collections of public grammars. The collections in the web sites are very useful to improve the productivity of parser development. On the other hand, many grammar rules contain some errors and there is no guarantee that they are strictly correct. As a result, we must laboriously check correctness of the grammars to improve the quality.

The sophisticated approaches using parsers are not appropriate to detect code clones if we need language independent techniques. In the paper[4], they describe that

Most of the approaches are based on parsing techniques and thus rely on having the right parser for the right dialect for every language that is used within an organization.

Language dependency is a big obstacle when it comes to the practical applicability of duplication detection. We have thus chosen to employ a technique that is as simple as possible and prove that it is effective in finding duplication.

Language independency is one of the most important point to apply the code clone detection in real applications because there are a lot of programming languages which are currently available.

This paper describes a technique which detects code clones from the perspectives of lexical and syntactic analysis, and also realizes independency of programming languages. It is based on source code representation, PALEX, which was proposed by the author[6]. PALEX stands for PArsing actions and LExical information in XML, and is generated by modified compilers.

Section 2 describes related works about the code clone detection. Section 3 describes briefly the source code representation PALEX and a clone code detection tool using PALEX. Section 4 summaries this paper.

II. RELATED WORKS

There are many papers related about the code clone detection since 1990’s. Typical techniques are categorized in four approaches;

• Line-based approach[2], [4],
• Token-based approach[7], [8],
• Syntax Sensitive and Language Independent Detection of Code Clones

K. Maeda is with the Department of Business Administration and Information Science, Chubu University, 1200 Matsumoto, Kasugai, Aichi, 487-8501, Japan e-mail: kaz@acm.org.
• AST-based approach[3], [9], [10], and
• Dependency-based approach[11], [12], [13]
In the paper[8], they describe that
A code clone is a code portion in source files that
is identical or similar to another.
For example, two portions (lines 4-7 and lines 11-14) in Fig.1
are textually identical, but the execution results are different.
According to their definition[8], it naturally indicates that the
two portions are code clones.

```java
1: public class Calc {
2: int x=1;
3: void foo(){
4: for(int i = 0; i < 10; i++){
5: x = x * 2 + 1;
6: }
7: System.out.println(x);
8: }
9: void bar(){
10: int x=1;
11: for(int i = 0; i < 10; i++){
12: x = x * 2 + 1;
13: }
14: System.out.println(x);
15: }
16: }
```

Fig. 1. Example of code clones

In the line-based approach, whole lines are compared each
by each. It ignores lexical and syntactic information of source
code so that all kinds of plain text file can be processed.
It is independent of programming languages and it is very
convenient for general purposes.
However, there are some limitations on the line-based
approach. If you use a code formatting tool, the tool does
not change the syntax but changes only the locations of
some tokens. Let us imagine that a developer changes the
preferences of indentation and locations of braces, and the
developer executes partially the code formatter tool. In a worst
case, the tool changes only the lines shown in Fig.2 so that
the line-based approach decides the lines of the source code
are totally different and it fails in the code clone detection.

```java
11: for (int i=0; i<10; i++)
12: { x = x * 2 + 1; }
```

Fig. 2. Example of lines after formatting the code

In the token-based approach, the entire source code is
scanned, a sequence of tokens is built, and the tokens are
compared each by each. CPD[7] is a token-based code clone
detection tool. If CPD reads the source code containing the
lines shown in Fig.2, it detects that the two portions are code
clones and it generates output shown in Fig.3. Interestingly,
in the output, the last token is a brace for end of the method.
The brace is a syntactically meaningless token for code clone
detection. If an automatic replacement of the code clones is
implemented, such a token causes problems.
In the AST-based approach, syntax sensitive analysis detects
precisely code clones. Generally, the compiler constructs AST
in syntax analysis. We can get the syntactic information of
source code from the AST. However, it takes much time to
detect code clones using the AST. The author had informally
an experiment using CloneDR1 and CCFinder[8] to analyze
millions of lines of source code. In the case of CCFinder, it
took a few minutes to detect code clones, but CloneDR spends
a lot of time to detect code clones using the AST.
The author and the colleagues are now developing a static
analyzer for Java containing CFG and DFG analysis. The total
of lines of source code is more than one hundred thousands.
We can obtain more precise results related about the code
clones if the static analysis is applied to detect the code
clones. However, if we need code clone detection to another
programming language, it takes much development cost.
All the approaches have some advantages and some dis-
advantages. This paper proposes PALEX-based code clone
detection. The PALEX-based approach is independent of pro-
gramming languages and syntax sensitive detection. Moreover
we don’t need much development costs.

III. LALR P ARSER AND PALEX SOURCE CODE
REPRESENTATION
Parser generators, such as Y acc[14] and Bison[15], make
parser development much easier. They read user-defined syn-
tax rules with action codes to be invoked when the syntax
rules are recognized, and they generate LALR parsers. The
generated parsers execute some typical actions and the actions
are called “parsing actions” in this paper.

A. LALR Parser
The LALR parser generated by Yacc or Bison uses two
tables, Action and Goto. The parser executes mainly two
actions: those are shift and reduce. When we develop a parser
using a parser generator Bison, we can build it in a debug
mode to check the actions during parsing.
When the parser generated by Bison is built in the debug
mode and it analyzes an arithmetic expression

```java
1 for(int i=0; i<10; i++){
2 x = x * 2 + 1;
3 System.out.println(x);
4 }
```

Fig. 3. Output of the code clone detection using CPD

1CloneDR is a trademark by Semantic Designs, Inc.
1 + 2 * 3

the parser writes debug information to standard output, as shown in Fig. 4. In the figure, state transitions are shown using five kinds of lines; "reading a token," "shifting token," "entering state," "reducing stack by rule," and "stack now." The PALEX code described in next section contains some of this debug information.

![Fig. 4. Debug information written by a generated parser](image)

### B. PALEX Source Code Representation

Once a compiler finishes the analysis of the source code, it can record a list of parsing actions during the parsing process, which are shift, reduce, and reading a token. PALEX is represented in XML containing the recorded parsing actions and lexical formatting information including white spaces and comments. It has two features:

- It is independent of any programming languages because there are no language-specific elements and attributes in the XML document for PALEX. Three software tools for Java, C#, and Ruby are developed to convert source code to the PALEX code.

- The original source code can be restored from the PALEX code because it contains sufficient lexical information to restore it.

Bison was modified to write out the PALEX code and it is called MoBison. MoBison reads syntax rules and generates a special parser that contains functionality to produce the PALEX code. MoBison is used to embed the functionality in another parser. Moreover, MoBison generates parsing information for other software tools to analyze the PALEX code.

MoBison reads the syntax rules and generates the parser. After the generated parser reads the arithmetic expression mentioned in the previous section and the parser analyzes it, it writes out the PALEX code as shown in Fig. 5. The figure contains the following parsing actions: sft, rdc, lex, and cst. Moreover it contains a white space using a tag wsc.

**TABLE I**

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning of the element</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsc</td>
<td>white space and/or comments</td>
</tr>
<tr>
<td>lex</td>
<td>reading a token</td>
</tr>
<tr>
<td>sft</td>
<td>shift action</td>
</tr>
<tr>
<td>rdc</td>
<td>reduce action</td>
</tr>
<tr>
<td>cst</td>
<td>change from a state to another state</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning of the attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>state number</td>
</tr>
<tr>
<td>fr</td>
<td>source state number for shift action</td>
</tr>
<tr>
<td>to</td>
<td>destination state number for shift action</td>
</tr>
<tr>
<td>tk</td>
<td>type of a token</td>
</tr>
<tr>
<td>va</td>
<td>string image of a token</td>
</tr>
<tr>
<td>li</td>
<td>line number</td>
</tr>
<tr>
<td>co</td>
<td>column number</td>
</tr>
<tr>
<td>ru</td>
<td>syntax rule number</td>
</tr>
<tr>
<td>ln</td>
<td>length of right hand side in a rule</td>
</tr>
</tbody>
</table>

![Fig. 5. Snippet of the PALEX code](image)
C. Code Clone Detection

PALEX has a linear list structure containing lexical and syntactic information. To detect code clones using PALEX, a suffix-tree matching algorithm[16] is basically used to analyze sequences of tokens. The suffix tree is a data structure that represents the suffixes of a given string. For example, if the string is ABCDABC$ ($ means an end of the string), then it is split into seven suffixes:
- ABCDABC
- BCDABC$
- CDABC$
- DABC$
- ABC$
- BCS
- CS

and a suffix tree is built shown in Fig.8. As a result, we can find that ABC, BC and C are duplicated substrings. If the suffix tree is applied to a sequence of tokens, it is easy to find code clones.

It we can get a sequence of tokens, token-based approach is independent on programming languages. However, there are some limitations on the token-based approach. It ignores...
After it pops the three states (state 8, state 7, and state 9) from the stack, it returns to state 6 and pushes the next state on the stack.

Once the parsing actions are recorded after analyzing source code, we know when and which syntax rule is reduced. In Fig. 9, the reduce action is applied at the state 9. If we go backward from the reduce action, we can decide the unique rule at each state. For example, the following rules in Fig. 9

- \( T \rightarrow T . M L T F \) (in state 8),
- \( T \rightarrow T M L T . F \) (in state 7), and
- \( T \rightarrow T M L T F \) (in state 9)

are decided after parsing. That is, the region between the stack state 0,2,6 and the stack state 0,2,6,8,7,9 is the timing during parsing the rule \( T \rightarrow T M L T F \). If we want to watch code clones of only a specified syntax, only the specified region is analyzed.

IV. CONCLUSIONS

This paper proposes a new technique to detect code clones from the syntactic point of view, which is based on PALEX source code representation. It also realizes language independency. The PALEX code contains the recorded parsing actions and also lexical formatting information including white spaces and comments. We can record a list of parsing actions (shift, reduce, and reading a token) during a compiling process after a compiler finishes analyzing the source code. Now source code of commercial products is trying to analyze for checking the power of this paper’s approach. The development and results will be published in a future paper.

REFERENCES


Kazuaki Maeda He is an associate professor of Department of Business Administration and Information Science at Chubu University in Japan. He is a member of ACM, IEEE, IPSJ and IEICE. His research interests are Compiler Construction, Domain Specific Languages, Object-Oriented Programming, Software Engineering and Open Source Software.