(CLSCR) CROSS LANGUAGE SOURCE CODE REUSE DETECTION USING INTERMEDIATE LANGUAGE

Dimpal Shah
Gujarat University,
Ahmedabad,
India.
Dimpalshah38@gmail.com

Heena Jethani
Gujarat University,
Ahmedabad,
India.
heenahjethani@gmail.com

Shri Hardik Joshi
Gujarat University,
Ahmedabad,
India.
joshee@acm.org

ABSTRACT

In today's digital era information access is just a click away. so computer science students also have easy access to all the source codes from different websites thus it has become difficult for academicians to detect source code reuse in students programming assignments. The new trend in the area of source code reuse is using the source code by translating it in another programming language popularly known as cross language plagiarism.

Our CLSCR addresses this problem. CLSCR mainly has two components: A compiler that compiles and translates the language specific source code into a tool specific internal format and The Similarity calculator that computes similarity between internal formats of different programs.

Keywords

Cross Language; CLSCR; Tokenization; Learning Management System.

1. INTRODUCTION

Identifying if students programming assignments are their original work or have been plagiarized from internet is of sole importance to the academicians. To address this problem many tools have been developed till date. Some of the tools are Sherlock, MOSS, JPLAG etc. All of these tools detect mono language plagiarism

Mono language plagiarism:

It is the act of producing source code file from another source code file of same language just by doing text edit operation and not understanding the granularities of the program.

With advance developments and research in the field of information retrieval new techniques of plagiarism have also emerged. One such technique is cross language plagiarism it is a modern and smart way of plagiarism.

Cross language plagiarism comes into picture when students want a source code for particular functionality in language A and while surfing the internet students come across the exact source code for the functionality but in language B so student decides to plagiarize by translating syntax of commands on A to syntax of B without understanding the working of the code.

Our tool CLSCR detects this type of plagiarism CLSCR basically works in 3 phases that are language detection, internal format conversion, similarity computation. All are explained in Section 4.

.

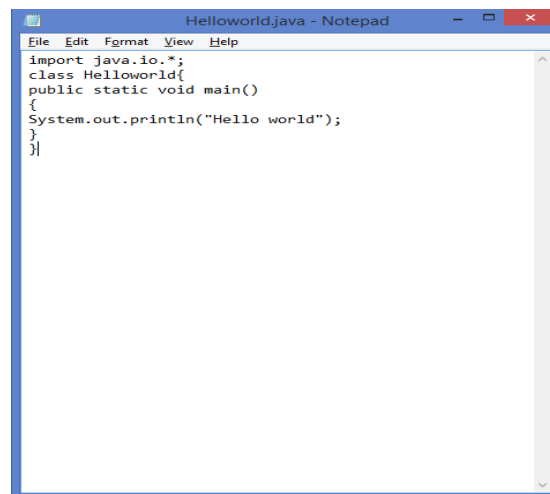## 2. DEFINITION

Cross language source code reuse:

Cross language plagiarism is also known as translation plagiarism. Let A1 and A2 be two programming languages and A1!=A2. Cross language source code reuse is stated as the translation of a source code P1 € A1 into P2 € A2.

.

3. RELATED WORK

3.1 Tokenization

It is the preprocessing technique that CLSCR performs before its actual implementation. It is the process of converting the source code in to tokens. Token is the smallest unit that holds meaning in a program. Tokens include:

(1) Identifiers (Variable types, Functions and Labels).

(2) Literals.

(3) Operators (For example +, -, / etc).

(4) Keywords (For example for, While, If etc).


```
Helloworld.java - Notepad
File  Edit  Format  View  Help
import java.io.*;
class Helloworld{
public static void main()
{
System.out.println("Hello world");
}
}
```

Figure 1. Example of Java File

Import
Java
Io
Class
HelloWorld
{
Public
Static
....

Figure 2. Tokenization of java file

4. DESIGN

CLSCR mainly works in 3 phases.

```
Language Detection
        |
        v
Intermediate language generation
        |
        v
Similarity computation
```
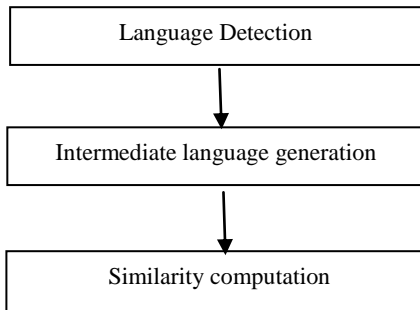
Figure 3. Design of CLSCR

PHASE 1: Language detection

The tokenized source code file is given as input to phase1. It detects the programming language of the file by comparing it with the predefined database consisting of keywords of different programming languages.
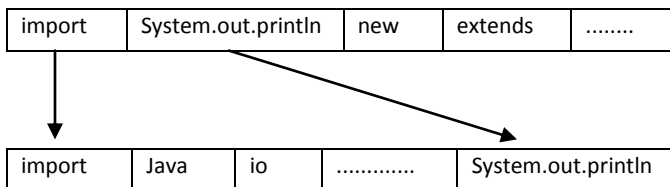
import	System.out.println	new	extends	....

import	Java	io	....	System.out.println

Figure 3. Comparison between tokens and database

After detecting the programming language in the phase1 automatically moves the input files to specific predefined folders. For example, it will move C++ language program file to the C++ folder and java files to java folder.

PHASE 2: Intermediate language (Internal Format) Generation.

Phase2 gets its input files from different folders for example C++ folder and java folder. Input files are then processed by compiler. For example, we have C++ conversion file a part of compiler for translating C++ folder files and java conversion file for translating java folder

Files. These translations produce common internal format for files.

Internal format is the compiler specific language file.

Figure 4. Intermediate language generation

As shown in figure 4 the internal format files are in monolingual. In short CLSCR performs translation of different programming language source codes to an intermediate language.

PHASE 3: Similarity Computation

It is the last phase of CLSCR. Phase2 generated internal format files is then compared to compute similarity.

This phase uses open source plagiarism detector SHERLOCK for calculating similarity percentage between internal format files.

4.1 Sherlock

SHERLOCK tool allows an instructor to examine a collection of submitted programs for similarities. Each program is stored as a single file, and is written using a specific *predefined language* [1] Here our predefined language is our *internal format*. It uses the concept of runs and anomalies to detect similarity.

Runs and Anomalies: The *Tool* defines *run* as a sequence of

common lines in two files, where the sequence might not be quite contiguous. There may be a number of extra or deleted lines interrupting the sequence. The allowable size of interruptions is called *anomalies. Similarity Percentage* is calculated on the basis of *length of run* and *anomalies.*

Table 1. Runs and anomalies

Sequence1	Sequence2	Sequence3
Begin	Begin	Begin
Line2	Lin2	Extra line
Line3	Extra line	Line3
Line4	Line3	Extra line
Line5	Line4	Line4
Line6	Line5	Extra line
Line 7	Line7	Another line
Line8	Line8	Line7

Sequence 1 and 2 form run with 2 anomalies.

Sherlock Usage: To use Sherlock we downloaded sherlock.C file which is available online. Then we compiled sherlock.C to generate exe file. All files that need to be compared for plagiarism and the exe file are placed in same folder. Then we run Sherlock a command-line program to generate result file containing similarity percentage of the files.

sherlock *.java > results.txt

5. EXPERIMENTS AND RESULTS

As our initial effort we have just focused on two object oriented programming languages C++ and java. but it can be implemented to detect many other languages. Evaluation of our tool is done through a data set that is checked for originality and degree of plagiarism is computed. For testing, the dataset used was collected from third party organization. Dataset consisted of 1000+ java and C++ program for now we have tested this tool only on two object oriented languages java and C++.

But with slight modification this tool can be implemented to detect plagiarism between many other languages.

After passing all source code files to 3 phases of CLSCR, the results obtains shows the similarity percentage between various files.
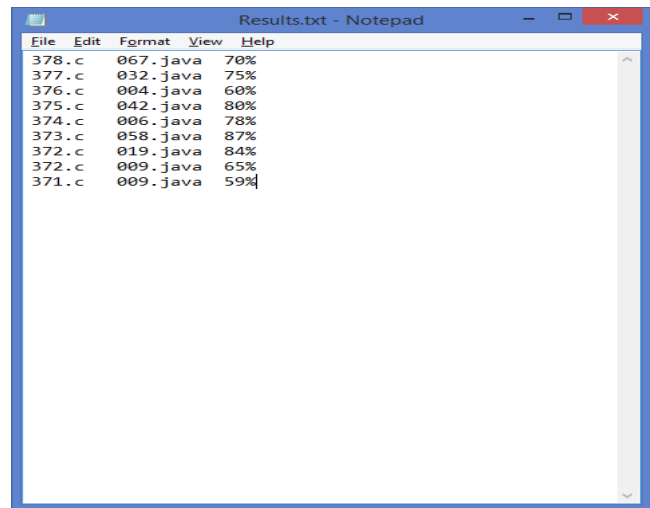

Figure 5. Similarity Computation

6. IMPROVED EFFECIENCY

CLSCR by default compares all files of C++ folder with java folder files. These folders may contain 500+ files resp.

Comparing this large number of files is a tedious task and may take some amount of time. To improve efficiency addition processing phase can be introduced.

Preprocessing phase: This phase is implemented before phase 2. Before converting the source code into intermediate language, Attribute comparison among different source codes is performed.

Attributes are general properties of source code files. They include number of classes, number of functions, number of objects, number of constructers, number of variables etc.

We have assigned weight to all general properties as per their importance in plagiarism detection.

Weight of class = cl

Weight of constructor = co

Weight of function = f

Weight of variable = v

Weight of object = o

Then we calculate weight total of properties

Weight_total = cl(no of class) + co(no of constructor) +o(no of object) + v (no of variable) + f (no of function).

All those files having similar weight total are only compared. As files with large difference in weight total have different properties thus the degree of similarity is very less. Thus they are ignored.

7. CONCLUSION

A software system that automatically detects cross language plagiarism between C++ and java files has been proposed and presented in this paper. This is basically a desktop application to detect plagiarism between different language source code files.

Academicians can install the application and by just uploading the collection of assignments of the students can detect the degree of plagiarism between the programs.

The system accepts the .txt, .java, .cpp all extensions of the source code so the overhead of converting the programs to a specific extension is also removed.

The proposed system has potential for becoming the comprehensive plagiarism detection system for universities. As CLSCR being able to detect cross language plagiarism additionally can even be used to detect mono language plagiarism. Although some of the processing of CLSCR would be worthless when in attempt of detecting mono language plagiarism but the result of detection would be accurate.

This software has been tested for large number of programming assignments of all categories with accurate results. This system can efficiently handle huge data set and can be seamlessly integrated with any learning management system.

This system can overall improve the quality of education imported in different computer science institution.

8. ACKNOWLEDGEMENTS

9. REFERENCES

[1] Joy, Mike, and Michael Luck. "Plagiarism in programming assignments." *Education, IEEE Transactions on* 42.2 (1999): 129-133.

[2] Đurić, Zoran, and Dragan Gašević. "A source code similarity system for plagiarism detection." *The Computer Journal* (2012): bxs018.

[3] Jadalla, Ameera, and Ashraf Elnagar. "PDE4Java: Plagiarism Detection Engine for Java source code: a clustering approach." *International Journal of Business Intelligence and Data Mining* 3.2 (2008): 121-135

[4] Juričić, Vedran, Tereza Jurić, and Marija Tkalec. "Performance evaluation of plagiarism detection method based on the intermediate language." (2011).

[5] Gabel, Mark, and Zhendong Su. "A study of the uniqueness of source code." Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2010

[6] Chae, Dong-Kyu, et al. "Software plagiarism detection: a graph-based approach." Proceedings of the 22nd ACM international conference on Conference on information & knowledge management. ACM, 2013

[7] Flores, Enrique, et al. "DeSoCoRe: detecting source code re-use across programming languages." Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstration Session. Association for Computational Linguistics, 2012.

[8] Flores, Enrique, et al. "Towards the detection of cross-language source code reuse." *Natural Language Processing and Information Systems*. Springer Berlin Heidelberg, 2011. 250-253

[9] Flores, Enrique, Rosso, Paolo, Moreno, Lidia and Villatoro-Tello, Esaú: PAN@FIRE 2015:" *Overview of CL-SOCO Track on the Detection of Cross-Language SOurce COde Re-use*". In Proceedings of the Seventh Forum for Information Retrieval Evaluation (FIRE 2015), Gandhinagar, India, 4-6 December (2015)