Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

# PROGRAM DEPENDENCE FLOWCHART GENERATOR AND PLAGIARISM DETECTION

Dr H D Phaneedra[1] , Kumar Biplav[2], Bhanu Shankar[3], Golden Kumar[4], Kunal Sinha[5]

[1, 2,3,4,5] Information Science and Engineering,National Institute of Engineering, Mysore, Karnataka, India- 570008

**Abstract**— Understanding a source code can be difficult but its pictorial representation i.e flowchart will be much easier to understand. Flowchart will help in understand the control flow of the program and it will also help in plagiarism detection of the source code which is a major problem for programmer. Textual based comparison which is generally used will not work for programs as programmer can easily change the programs code in such a way that it will not get detected . Plagiarism based on AST allow higher level of similarities to be detected by comparing the logic of the program not the syntax , as it can normalize the default syntax of the program. We are presenting a system which will generate the approximate flowchart of the program based on abstract syntax tree and compare the similarities between the two programs based on it and calculate the plagiarism percentage.

## I.INTRODUCTION

Since the computer was invented, software plagiarism has always been a serious problem. People can copy other's painstaking efforts, and pretend it is written by them oruse it arbitrarily. Plagiarism is even easier to commit in the age of the Internet. At the same time,
plagiarism is not easy to catch. Thus, we would like to automate the discovery of cases of plagiarism. Currently, the techniques for plagiarism detection, such as Attribute Counting System, Measure of Software Similarity (MOSS), Yet Another Plague (YAP) and JPlag are focusing on text patterns. In this Project, we will use graph to analyze the syntactic structure of program, using their abstract syntax tree (AST). The reason we chose to compare the AST of
programs rather than the source code itself is because the AST describes the structure of a computer program. We think that the syntactic structure of program is important in detecting plagiarism, because this structure holds repetitive patterns that only occur in ASTs of similar, potentially plagiarized software. That is, there will be no two similar syntactic structures unless the source codes are similar. Our aim, we will create a tool to analyze the AST of computer programs. Our system will work by extracting repetitive patterns from ASTs of programs, then compares these patterns. High similarity between the patterns would mean high likelihood of plagiarism.

## II. SYSTEM STUDY AND ANALYSIS

### A. EXISTING SYSTEM

Existing System:- In programming courses, there are always students who hand in code that does not belong to them. Looking at someone elses code can be a good starting point in
learning how to program, but directly copying some ones code and handing in as ones own work is an act of plagiarism. Using code from the web is a popular choice, but there are also cases when a fellow student has been victimized.Traditionally people do word by word and line by line checking of the programs to check similarities and any form of disguises. It takes a lot of time and manual matching of complex data which many time gives wrong result. So
current system is not efficient enough to detect plagiarism. Need of the hour is a better method and tool to detect program plagiarism.

 

B. PROPOSED SYSTEM

System that analyses intermediate forms such as token strings, syntax trees,graph representations and flowchart representation have shown to be more effective than using simple textual matching methods. In this project report we discuss how flowcharts, an abstract representation of a programs semantics, can be used to find similar procedures. We also present an implementation of a system that constructs approximated flowcharts from the abstract syntax tree representation of a program. Under a scenario based evaluation our system is a popular plagiarism detection tool.

## III. DESCRIPTION

Unlike the existing structure-metric plagiarism detecting methods, we chose to examine the structure of programs in the form of the abstract syntax tree (AST). In our system, we first
send the source code to the compiler, which parses the codeand generates the AST. Then, we extract repetitive patterns from the AST. Lastly, we compare the patterns reported on pairs of ASTs, to arrive at the measure of similarity.The compiler to be used in the system is dependent .of the programming language used. We restricted our study to the C language, and used the GNU C compiler, gcc. The output from gcc must be converted to a graph format, which is accomplished using a simple format converter. After the graph files are analyzed the results will be compared by a graph matching program that computes the similarity between graphs. Figure  shows a diagram of the system, and how it isapplied to compare two computer programs.Our system consists of four major parts: AST extraction, graph conversion, Graph generation, and the matching system.

## IV. ARCHITECTURE DESIGN

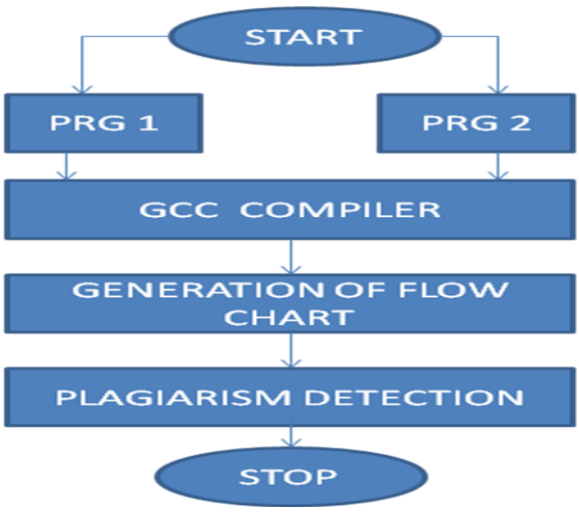


*Fig 1.1*

## V. GENERATING THE ABSTRACT SYNTAX TREE

If a language is a set of strings, then a string is a finite sequence of symbols. The symbols themselves are taken from a finite alphabet. Therefore, in the lexical analysis phase of a
compiler the source code is converted to regular expressions, and then these regular expressions are converted to deterministic finite automata (DFA). The reason we use DFA
versus non-deterministic finite automata (NFA) is because no two edges leading from the same state are labeled with the same symbol, and DFA are easy to implement by a computer
language. For parsing, we can view the string as a source program, the symbols as lexical tokens, and the alphabet as the set of token types returned by the lexical analyzer. The parse stage analyzes the phrase structure of the program. It uses a context-free grammar to describe the programming language. After the parse tree is built, the compiler will construct a syntax tree representation of the

 

input program. It indicates its relationship to the actual syntax and parse tree. Since the compilation process is driven by the syntactic structure of asource program, in this tree, the compiler needs to do semantic processing.

FRAMA-C

Frama-C stands for Framework for Modular Analysis of C programs. Frama-C is a set of interoperable program analyzers for C programs. Frama-C enables the analysis of C programs without executing them. Frama-C is Open Source software.

It works on Windows and Unix (Linux, Mac OS X,…).

Frama-C can be used for the following purposes: to understand C code which you have not written. In particular Frama-C enables to: observe a set of values, slice the program into shorter programs, navigate in the program.To prove formal properties on the code. Using specifications written in ANSI/ISO C Specification Language enables .To ensure properties of the code for any possible behavior. to instrument C code against some security flaws

Frama-C has a modular plugin architecture.Frama-C relies on CIL (C Intermediate Language) to generate an abstract syntax tree.The abstract syntax tree supports annotations written in ANSI/ISO C Specification Language (ACSL).

CIL

CIL is a front-end for the C programming language that facilitates program analysis and transformation. CIL will parse and typecheck a program, and compile it into a simplified subset of C. For example, in CIL all looping constructs are given a single form and expressions have no side-effects. This reduces the number of cases that must be considered when manipulating a C program. CIL has been used for a variety of projects, including CCured, a tool that makes C programs memory safe CIL supports ANSI C as well as most of the extensions of the GNU C and Microsoft C compilers. A Perl script acts as a drop in replacement for either gcc or Microsoft's cl, and allows merging of the source files in your project. Other features include support for control-flow and points-to analyses. Compared to C, CIL has fewer constructs. It breaks down certain complicated constructs of C into simpler ones, and thus it works at a lowerlevel than abstract-syntax trees. But CIL is also more high-level than

typical intermediate languages (e.g., three-address code) designed forcompilation. As a result, what we have is a representation that makes it easy to analyze and manipulate C programs, and emit them in a form that resembles the original source. CIL’s conceptual design tries to stay close to C, so that conclusions abouta CIL program can be mapped back to statements about the source program.Additionally, translating from CIL to C is fairly easy, including reconstructionof common C syntactic idioms.

Finally, a key requirement for CIL is the ability to parse and represent thevariety of constructs which occur in real-world systems code, such as compilerspecific extensions and inline assembly. CIL supports all GCC and MSVC ex-tensions except for nested functions, and it can handle the entire Linux kernel.

OCAML

OCaml originally known as Objective Caml, is the main implementation of the Caml programming language.

OCaml extends the core Caml language with object-oriented constructs.OCaml's toolset includes an interactive top level interpreter, a bytecode compiler, and an optimizing native code compiler. It has a large standard library that makes it useful for many of the same applications as Python or Perl,

 

as well as robust modular and object-oriented programming constructs that make it applicable for large-scale software engineering. ML-derived languages are best known for their static type systems and type-inferring compilers.

OCaml unifies functional, imperative, and object-oriented programming under an ML-like type system.

This means the program author is not required to be overly familiar with pure functional language paradigm in order to use OCaml.OCaml's static type system can help eliminate problems at runtime.However, it also forces the programmer to conform to the constraints of the type system,

Which can require careful thought and close attention.

A type-inferring compiler greatly reduces the need for manual type annotations (for example, the data type of variables and the signature of functions usually do not need to be explicitly declared, as they do in Java). Nonetheless, effective use of OCaml's type system can require some sophistication on the part of the programmer.

OCaml features: a static type system, type inference, parametric polymorphism, tail recursion, pattern matching,

First class lexical closures, functors (parametric modules), exception handling, and incremental generational automatic garbage collection. A foreign function interface for linking to C primitives is provided, including language support for efficient numerical arrays in formats compatible with both C and FORTRAN. OCaml also supports the creation of libraries of OCaml functions that can be linked to a "main" program in C, so that one could distribute an OCaml library to C programmers who have no knowledge nor installation of OCaml.

The OCaml distribution contains:

An extensible parser and macro language named Camlp4, which permits the syntax of OCaml to be extended or even replaced, Lexer and parser tools called ocamllex and ocamlyacc, Debugger that supports stepping backwards to investigate errors, Documentation generator, Profiler — for measuring performance, Numerous general-purpose libraries.

The following program "hello.ml":

```
print_endline "Hello World!"
```

can be compiled into a bytecode executable:

```
  $ ocamlc hello.ml -o hello
```

or compiled into an optimized native-code executable:

```
      $ ocamlopt hello.ml -o hello
```

and executed:

```
      $ ./hello
      Hello World!
 $
```

DOT

DOT is a plain text graph description language. It is a simple way of describing graphs that both humans and computer programs can use. DOT graphs are typically files that end with the .gv (or .dot) extension.The .gv extension is preferred in cases where there could be confusion with the .dot file extension used by early (pre-2007) versions of Microsoft Word. Various programs can process DOT files. Some, like OmniGraffle, dot, neato, twopi, circo, fdp, and sfdp, will read a DOT file and render it in graphical form. Others, like gvpr, gc, acyclic, ccomps, sccmap, and tred, will read a DOT file and perform calculations on the represented graph. Finally, others, like lefty, dotty, and grappa, provide an interactive interface.

The DOT language defines a graph, but does not provide facilities for rendering the graph.

 

There are several programs that can be used to render, view, and manipulate graphs in the DOT language like Graphviz - A collection of libraries and utilities to manipulate and render graphs
Most programs are part of the Graphviz package or use it internally. The following is an abstract grammar defining the DOT language. Terminals are shown in bold font and nonterminals in italics. Literal characters are given in single quotes. Parentheses ( and ) indicate grouping when needed. Square brackets [ and ] enclose optional items. Vertical bars | separate alternatives.

```
graph   :       [ strict ] (graph | digraph) [ ID ] '{' stmt_list '}'
stmt_list       :       [ stmt [ ';' ] [ stmt_list ] ]
stmt    :       node_stmt
        |       edge_stmt
        |       attr_stmt
        |       ID '=' ID
        |       subgraph
attr_stmt       :       (graph | node | edge) attr_list
attr_list       :       '[' [ a_list ] ']' [ attr_list ]
a_list  :       ID '=' ID [ (';' | ',') ] [ a_list ]
edge_stmt       :       (node_id | subgraph) edgeRHS [ attr_list ]
edgeRHS         :       edgeop (node_id | subgraph) [ edgeRHS ]
node_stmt       :       node_id [ attr_list ]
node_id         :       ID [ port ]
port    :       ':' ID [ ':' compass_pt ]
        |       ':' compass_pt
subgraph        :       [ subgraph [ ID ] ] '{' stmt_list '}'
compass_pt      :       (n | ne | e | se | s | sw | w | nw | c | _)
```

An ID is one of the following:
Any string of alphabetic ([a-zA-Z\200-\377]) characters, underscores ('_') or digits ([0-9]), not beginning with a digit;
a numeral [-]?(.[0-9]+ | [0-9]+(.[0-9]*)? );
any double-quoted string ("...") possibly containing escaped quotes ('")1;
an HTML string (<...>).
Subgraphs play three roles in Graphviz. First, a subgraph can be used to represent graph structure, indicating that certain nodes and edges should be grouped together.
This is the usual role for subgraphs and typically specifies semantic information about the graph components.
It can also provide a convenient shorthand for edges. An edge statement allows a subgraph on both the left and right sides of the edge operator. When this occurs, an edge is created from every node on the left to every node on the right.
The third role for subgraphs directly involves how the graph will be laid out by certain layout engines.If the name of the subgraph begins with cluster, Graphviz notes the subgraph as a special cluster subgraph. If supported, the layout engine will do the layout so that the nodes belonging to the cluster are drawn together, with the entire drawing of the cluster contained within a bounding rectangle.

USING CIL as LIBRARY

CIL can also be built as a library that is called from our stand-alone application.
Add cil/src, cil/src/frontc, cil/obj/x86_LINUX (or cil/obj/x86_WIN32) to our Ocaml project -I include paths.
Building CIL will also build the library cil/obj/*/cil.cma (or cil/obj/*/cil.cmxa).

 

We can then link our application against that library.
We can call the Frontc.parse: string -> unit -> Cil.file function with the name of a file containing the output of the C preprocessor
The Mergecil.merge: Cil.file list -> string -> Cil.file function merges multiple files.
We can then invoke our analysis function on the resulting Cil.file data structure.
We might want to call Rmtmps.removeUnusedTemps first to clean up the prototypes and variables that are not used.
Then we can call the function Cil.dumpFile: cilPrinter -> out_channel -> Cil.file -> unit to print the file to a given output channel. A good cilPrinter to use is defaultCilPrinter.
Here is a concrete example of compiling and linking our project against CIL.
Imagine that our program analysis or transformation is contained in the single file main.ml.

```
$ ocamlopt -c -I $(CIL)/obj/x86_LINUX/ main.ml
$ ocamlopt -ccopt -L$(CIL)/obj/x86_LINUX/ -o main unix.cmxa str.cmxa \
  $(CIL)/obj/x86_LINUX/cil.cmxa main.cmx
```

The first line compiles our analysis, the second line links it against CIL (as a library) and the Ocaml library.

After the program to be compared is given the user, Frama-c and Ocaml program will help in slicing the program and extract the abstract syntax tree of the program which done by the help of CIL. Then based on this abstract syntax tree Dot will generate the flowchart of the program. After this comparison of the abstract syntax tree will be done based on the label of the tree and the branching of the tree. If same label is connected to similar branch of the tree then it is considered to be copied.

## VI. APPLICATIONS

- The advantage of our system is partial function comparison. The current structure-metrics system cannot find if the program contains some plagiarism functions, especially when the functions are relatively small in the whole program. This problem is because their systems have to work with the entire source code; but in this research, the compiler separates the functions for us. So we can compare those separated functions to find whether plagiarism occurs in those functions or not.
- One of the major benefits of using plagiarism detection software is that it will prevent students from plagiarizing. If students know up front that their professor runs all submitted papers through a plagiarism detector, especially, then it's likely to prevent many students from even attempting to plagiarize their assignments.
- It can used in institutions like school, colleges, universities, etc for teaching and learning purposes, as pictorial representation of code is always easy to analyse and understand rather than going through the whole text.
- It can also be used inside various IT and Corporate organizations in development teams as an explanatory tool for a newcomer in the team who is going to work on a particular project or software

## REFERENCES

1. Alzahrani, S. Salim, N. Abraham, A. Understanding Plagiarism Linguistic Patterns, Textual Features and Detection Methods, preprint 2011.
2. Subject center for Information and Computer Sciences www.ics.heacademy.ac.uk
3. A Survey on Plagiarism Detection Systems A. S. Bin-Habtoor and M. A. Zaher International Journal of Computer Theory and Engineering Vol. 4, No. 2, April 2012
4. Abstract Syntax Tree Analysis for Plagiarism Detection Erik A. Nilsson Reg Nr: LIU-IDA/LITH-EX-A--12/043–SE Linköping 2012
5. SOFTWARE PLAGIARISM DETECTION USING ABSTRACT SYNTAX TREE AND GRAPH-BASED DATA MINING By HSI-YUE SEAN HSIAO Bachelor of Science Oklahoma State University Stillwater, Oklahoma 2002
6. A Source Code Similarity System for Plagiarism Detection Zoran Djuric , Dragan Gasevic *Faculty of Electrical Engineering, University of Banjaluka, Patre 5, 78 000 Banjaluka, Bosnia and Herzegovina School of Computing and Information Systems, Athabasca University, Athabasca, Canada*

 