Abstract

This dissertation describes the design and implementation of a tool that, given a collection of student computer programs written to produce the same overall functionality, provides a measure of similarity for each program compared to every other program in the collection. The main purpose of this tool was to help detect instances of plagiarism within student programming assignments. The proposed approach was to measure similarity at a machine code level, unlike contemporary software similarity measurement tools which analyse at a source code level.

Plagiarism is becoming increasingly prevalent in the field of Computer Science and Computer Engineering, especially with the influx in online learning during the COVID-19 pandemic. Plagiarism in programming is not always a copy and paste. Students use many different methods of obfuscation to avoid plagiarism detection, such as changing variable names or rearranging blocks of code. For this reason, specific tools for plagiarism detection in programming exist.

Existing tools like MOSS (Measure of Software Similarity) and Compare50, analyse programs at a source code level. This means that they are inherently language specific. The project proposes examining programs at a machine code level to side-step this limitation. This approach also automatically neutralises certain attempts at obfuscation, such as adding comments or whitespace, or changing variable names.

The approach described here compares the machine code of programs using the Levenshtein algorithm. This algorithm measures edit-distance, i.e. how many edits need to be made to convert one program to the other. The higher the edit distance, the less similar they are. This approach also performs hierarchical analysis, using a disassembly framework called Capstone. This allows individual elements of machine code instructions, such as register numbers or branch distance, to be compared and produce a more in-depth similarity measurement. An adapted Levenshtein algorithm is used to produce non-binary measures of similarity for each machine code instruction within this hierarchical analysis. The approach developed within this project is referred to as “Combined Levenshtein”.

A corpus of student programming assignments was created for experimentation, and artificial instances of plagiarism were added to the corpus. In order to assess how this approach behaved in comparison to contemporary tools, the machine code of these assignments was analysed using the Combined Levenshtein method, and the source code was analysed using Compare50. The results of these experiments were promising and indicated that it is viable to implement measures of similarity using machine code, as a method of plagiarism detection.