**LLVM: built-in scalable code clone detection based on semantic analysis[[1]](#footnote-1)\***

Sevak Sargsyan, Shamil Kurmangaleev, Andrey Belevantsev

sevaksargsyan@ispras.ru, kursh@ispras.ru, abel@ispras.ru

 Two code fragments are called clones if they are identical by some similarity function. Code clones detection has number of applications. It can be used for semantic mistakes detection, automatic refactoring, code quality improvements and optimizations. There are several methods for code clones detection. Textual [1] and lexical [2] approaches are not capable to detect all clone types [3]. The Algorithms based on abstract syntax tree [4] and metrics [5] have low accuracy. The algorithms based on program dependence graph (PDG) [6] allow reaching high accuracy. But these algorithms have large computational complexity, which makes them unusable for real world projects analysis.

This paper describes scalable and accurate method of code clones detection, which allow analyze million lines of source code (written in C/C++). As well it describes architecture of the tool based on this method. The tool has two basic parts. The first part is designed as LLVM pass [7] and responsible for PDGs generation. The second part designed as LLVM tool [7], which analyzes PDGs to detect clones.

The method based on PDG and has some extra features, which makes it capable for large scale projects analysis. Dependence graphs are constructed at compile-time based on intermediate representation (bitcode) of LLVM [7] (first part). This approach has number of advantages compared with analogical methods [8]. Source code parses only once (during compilation). No need for extra analysis of dependencies between compilation modules. When all graphs are generated second part of tool split them to subgraphs. Then clone detection algorithm runs for pair of these subgraphs. Therefore it is important to split graphs so that every possible clone was located inside one subgraph. Existed methods split PDG to weakly connected components. Some of them split to subgraphs, which are intersected with each other in limited number of vertices [9]. Two algorithms were designed for graphs’ partitioning. Theses improve the number of detected clones. The first method divides PDG to relatively equal pieces. Second one splits PDG to subgraphs, where corresponding source code has sequential order.

 Scalability was achieved by composition of two types of algorithms. The first type of algorithms tries to prove if two graphs do not have enough large isomorphic subgraphs. These algorithms have liner complexity. Up to 80% of PDGs are processed by them [9]. Other 20% are analyzed with approximate algorithms of maximal isomorphic subgraphs detection. They have high accuracy and computational complexity.

Target machine of testing was Intel Core i3 CPU 540 with 8 GB RAM. Mozilla Firefox (3.8 million lines of С/С++ code) – was analyzed in 8h, detected 98 clones, 7 of them were false positive (manual analysis). LLVM+CLANG (1.3 million lines of С/С++ code) – was analyzed in 6h, detected 51 clones, 2 of them were false. OpenSSL (280 000 lines of С/С++ code) – was analyzed in ~200 second, 107 clones are detected.

**References**

1. *Ducasse S.* [*at al.*] A language independent approach for detecting duplicated code // Software Maintenance. – 1999. – 109– 118p.
2. *Kamiya T.* [*at al.*] CCFinder : A multilinguistic token-based code clone detection system for large scale source code // Software Engineering. – 2002. – 654–670p.
3. *Chanchal K.* [*at al.*] Comparison and evaluation of code clone detection techniques and tools: A qualitative approach // Science of Computer Programming. – 2009. V. 74 . – 470– 495p.
4. *Jiang L.* [*at al.*] DECKARD : Scalable and accurate tree-based detection of code clones // Software Engineering. – 2007. – 96-105p.
5. *Mayrand J.* [*at al.*] Experiment on the automatic detection of function clones in a software system using metrics // Software Maintenance. – 1996. – 244– 253p.
6. *Komondoor R.* [*at al.*] Using slicing to identify duplication in source code // 8th International Symposium on Static Analysis. – 2001. – 40–56p.
7. www.llvm.org
8. M. Gabel, L. Jiang, Z. Su, Scalable detection of semantic clones, in: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008,2008,pp.321–330
9. *Baker B.* [*at al.*] On finding duplication and near-duplication in large software systems // Reverse Engineering. – 1995. – 86–95p.
1. \* The paper is supported by RFBR grant 15-07-07541 А [↑](#footnote-ref-1)