Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 331 (2017) 87–99

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Abstract Similarity Analysis

Mila Dalla Preda and Vanessa Vidali

Dipartimento di Informatica, Università degli studi di Verona, Italy

Abstract

Code similarity is an important component of program analysis that finds application in many fields of computer science. Graph based representations of programs, such as control flow graphs and dependency graphs, are often used as a basis for deciding code similarity. Indeed, many similarity algorithms observe particular properties of these graph-based representations of programs in order to decide whether two programs are similar or not. In this work we propose a general framework for similarity analysis where the similarity of programs is expressed in terms of abstractions of their control flow graphs representation. In particular, we consider abstractions of the basic blocks of a control flow graph.

Keywords: Code similarity, abstract interpretation.

1 Introduction

Code similarity studies if two programs are similar or if one program is similar to a portion of another program (code containment). Code similarity is an important component of program analysis that finds application in many fields of computer science, such as reverse engineering of big collections of code fragments [13,16], clone detection [2,8], identification of violations of the intellectual property of programs [1,17,12], malware detection [9,10,15], software maintenance [11,19], software forensics [2,14]. In these applications, when comparing two fragments of code it is important to take into account changes due to code evolution, compiler optimization and post-compile obfuscation. These code changes give rise to fragments of code that are syntactically different while having the same intended behavior. This means that it is important to recognize modifications of the same program that are obtained though compiler optimization or code obfuscation as similar. To this end we need to abstract from syntactic changes and implementation details that do not alter the intended behavior of programs, namely that preserve to some extent the semantics of programs.

[1] Email: mila.dallapreda@univr.it, vanessa.vidali@studenti.univr.it

[2] This work has been supported by the MIUR FIRB 2013 project FACE RBFR13AJFT.

http://dx.doi.org/10.1016/j.entcs.2017.02.006

In order to consider both semantic meanings and syntactic patterns, existing tools for similarity analysis often employ mixed syntactic/symbolic and semantic representations of programs, as for example control flow graphs and dependency graphs that express the flow of control or the dependencies among program instructions. Recently, in [7] the authors investigate the use of symbolic finite automata (SFA) and their abstractions for the analysis of code similarity. SFAs have been introduced in [18] as an extension of traditional finite state automata for modeling languages with a potential infinite alphabet. Transitions in a SFA are modeled as constraints interpreted in a given Boolean algebra, providing the semantic interpretation of constraints, and therefore the (potentially infinite) structural components of the language recognized (see [5,18]). In [7] the authors show how SFAs can be used to represent both the syntax and the semantics of programs written in an arbitrary programming language, the idea is to label transitions with syntactic labels representing program instructions, while their interpretation is given by the semantics of such instructions. Thus, SFAs provide the ideal formal setting in order to treat within the same model the abstraction of both the syntactic structure of programs and their intended semantics. A formal framework for the abstraction of syntactic and semantic properties of SFAs and therefore of programs represented as SFAs is presented in [7]. This formal framework turns out to be very useful in the understanding of existing similarity analysis tools, and in the development of similarity analysis tools based on semantic and syntactic properties of programs.

The work presented in this paper and the prototype implementation that we propose can be seen as a first step towards an experimental validation of the general theory for software similarity proposed in [7]. In this paper we consider the standard control flow graph (CFG) representation of programs that is a simplified model with respect to the SFA representation of programs proposed in [7]. Indeed, a CFG can be easily translated into an SFA where the automata is isomorphic to the CFG, basic blocks label the automata transitions and the interpretation of the basic blocks is the identity function (this is because the CFG representation of programs does not account for the interpretation of basic blocks). In the attempt to build a similarity analysis tool parametric on program's properties we decide to start form the simpler case of CFGs. Thus, in this work we present a general framework for code similarity where the notion of being similar is formalized in terms of abstraction, in the abstract interpretation sense, of the CFG of programs. We propose a methodology for testing code similarity that is parametric on the property of the basic blocks of the CFG that is used for deciding similarity. Indeed, many existing algorithms for similarity analysis can be interpreted in our framework by formalizing the abstract property of the blocks of the CFG that they consider. This allows us to compare existing similarity algorithms by comparing the abstractions that characterize them. Moreover, the precision of the proposed similarity test can be tuned by modifying the abstract property of the CFG that is being observed. We provide a prototype implementation of the proposed methodology that allows us to test the similarity of two fragments of code with respect to three different abstract properties of basic blocks.