Abstract Formal Framework for Method Overriding

Siti Hafizah, Mohd Sapiyan Baba, and Abdullah Gani

Faculty of Computer Science and Information Technologies, Universiti Malaya, 50630 Kuala Lumpur, Malaysia {sitihafizah,pian,abdullahgani}@um.edu.my

Abstract. Automated verification on method overriding is important in Object-Oriented Programming Language (OOPL) to reduce human errors during software verification process. Static verification method fails to address the issue effectively. This paper examines the issues in developing semantics for verifying method overriding in OOPL. The purpose is to identify elements or components for verification process. A study conducted on literature reveals two main issues of verifying method overriding: subtyping and class invariant. Both issues are resolvable by integrating the elements of non-reverification, modularity, and programmer intervention into a framework. We propose an abstract formal framework with the integration of the three elements by using abstract interpretation and Lazy Behavioral Subtyping (LBS). The framework shows that the integration of less restriction of LBS and abstract interpretation is possible to achieve automated verification.

Keywords: program analysis, abstract interpretation, software verification, method overriding.

1 Introduction

Software verification is an important process in software development to ensure the software specifications are achieved. The verification process proves or disproves the correctness or optimization of the semantic of the software by using formal specification of formal methods. In Object-Oriented Programming Language (OOPL), program reuses and re-forms through the use of classes. Method overriding is a concept that is used to redefine method’s definition where it affects classes and objects’ behavior. The ability of changing of behavior makes the program unstable that leads to unexpected program termination. In the context of method overriding, the unstable program is due to class, method, and data.

Since the presentation of Hoare’s seminal paper [7] on data abstraction, class invariant concept has become a popular method to verify programs. In the environment of object-oriented, the method which is called Hoare’s Logic has been enhanced to be modular by Muller [14], to control class’s specification of data hiding and encapsulation. During verification process of OOPL, programmer encounters problem in object mutability and subtyping. Therefore, Liskov [11] has presented a notion called behavioral subtyping to overcome the problem exists on relationship between methods where it affects on how the methods behave. Over the years,
Parkinson [16] has extended the class invariant by introducing Separation Logic that enables to verify pointer in OOPL and subtyping. However, those works needs human intervention by annotation during the verification process. Therefore, Logozzo [12] proposes a framework that verifies OOPL by using abstract interpretation theory. Yet, the work on using abstract interpretation on OOPL modular analysis and behavioral subtyping concentrates on superclass only. The inference rules are able to avoid the re-verification problem; however it results in over-approximation on subclass invariants.

Based on abstract interpretation theory, we design an abstract formal framework by integrating method overriding which involves late bound method calls. Abstract interpretation [5] is used compare to axiomatic or denotational semantic so that the program analysis excludes the requirement of annotation as verification tools [14][3][4][16]. Late binding is included in the framework due to flexible code reuse. However, it complicates the reasoning when the method calls at late binding cannot be statically determined. Programmers use behavioral subtyping to overcome the problem even though it restricts the specification of precondition is contravariance and postcondition is covariant. Therefore, the framework of method overriding incorporates Lazy Behavioral Subtyping [6] where the technique supports incremental reasoning and in least restriction manner.

This paper is organized as follows: section 2 discusses related works on invariants and subtyping; section 3 proposes the abstract formal framework by using the model of verifying method overriding and the architecture of abstract formal framework; section 4 presents the general definition of the verifiable semantics of method overriding; section 5 discusses the abstract formal framework. Finally section 6 presents the conclusion.

2 Related Works

Invariant is a concept from Mathematics where it is generally described as value of expression that does not change during the program execution. The purpose is to become a property that is true of all expressions of a given code at all time [18]. Approximate polynomial representation is used for the shape of an invariant. Once the shape of the invariant is predicted, a deterministic technique is used to generate the exact form of the invariant [17]. The idea of a class invariant that is first proposed by Hoare has been extended so that the structure of classes and objects will be inclusion. The class invariant is a difficult task in inheritance because the class invariant is meant for single object, but there is aggregate structure in inheritance that involves two or more objects. Re-verification of superclass every time subclass is added into the program can cause code blow-up. Therefore, Parkinson [15] proposes of using a general foundation of verification which is predicates to specify the properties of aggregate structure, after considering the complexity of peer invariants of [8] and history invariants of [9]. The predicates are represented as Separation logic which is extension to Hoare logic. However, the method involves annotation that can lead to human errors. Based on existing invariants generating techniques, Xing et al. [19] present a technique where invariants are generated automatically at each statement to ensure properties are safe and terminated. The technique does not need to use Hoare