Verification of Information Flow Properties of Java Programs without Approximations

Christoph Scheben and Peter H. Schmitt†
Karlsruhe Institute of Technology (KIT)
Am Fasanengarten 5, 76131 Karlsruhe, Germany

Abstract. In this paper we propose a methodology for the specification and verification of information flow properties for sequential Java programs. This proposal also covers declassification. We define an extension of the Java Modeling Language (JML) that significantly goes beyond previous approaches. The JML specification clauses are translated into proof obligations in Dynamic Logic. An experimental implementation within the KeY-system shows the feasibility of the approach.

1 Introduction

This paper is concerned with the specification and verification of information flow properties. A typical example of an information flow property is confidentiality: An attacker observing the public outputs of a computing system cannot gain information on secret data. The frequently quoted survey paper [19] gives a concise introduction into this area, now commonly called language-based information-flow security, and reviews a number of approaches that have been persued in the field. We will be concerned here, more precisely, with an approach to information flow security using program logic. In this approach information flow properties are recast as formulas in a general, as opposed to problem specific, language e.g., in the language of Hoare Logic or of a weakest precondition calculus as pioneered in the papers [5] and [1]. It is an appealing feature of this methodology that the formalization is in most cases a straightforward transcription of the informal definition. Another advantage is the possibility to use existing program verification systems and theorem provers to support verification of the specified properties. There are various options on how theorem proving support can be organized. The papers [16,14,21] investigate the use of abstraction while [8] is devoted to the integration of security type systems into program verification. Both lines of attack lead to approximate results. In this paper we follow the lines of [5] and avoid approximative methods.

The novel contributions of the present work are twofold. The original proposal [5] that uses theorem proving in the analysis of secure information flow properties...
the proof of concept was demonstrated for a simple theoretical programming language. All considerations in this paper refer to real world sequential Java programs. As a second contribution we extend the Java Modeling Language (JML), see [13], by new constructs that allow to specify information flow properties. Given the current state of research, the formulation of such properties at the level of the internal program logic is a necessary intermediate stage. In the end one would want a way to specify them in a way that is accessible to the programmer or domain expert without a background in logic. In particular, there should be no mentioning of concepts like self-composition at this level of abstraction. For functional properties the behavioral interface specification language JML has been successfully used for this purpose. In our approach, JML method contracts are extended with two new clauses: respects and declassify. The first clause specifies sets of sets of locations \( V \). The informal semantics of such a specification is, that for every \( V \in V \) the locations not in \( V \) do not interfere with locations belonging to \( V \). This means that an attacker which can observe one of those sets of locations won’t be able to deduce more information through the execution of the method than he already knew. The declassify-clause is used to specify permitted additional information flow to some location set by the specification of a term. If the method is executed, an observer of the specified location set may learn the value of the term before the execution. The extension of JML integrates seamlessly with functional JML specifications. This is important since a real precise calculation of information flow dependencies can only be achieved with knowledge on the functional behavior of a program or method. This also works the other way around: knowledge on information flow dependencies does improve functional verification. The annotated Java programs are translated to JAVADL proof-obligations which express non-interference by self-composition. This translation does not over-approximate dependencies between variables. The proof-obligations are then verified with the help of the KeY-System. In combination with a functional specification, which is assumed to exist for the functional verification anyway, the KeY-system was able to verify the formulas in our examples automatically. We expect the approach to work automatically at least in those cases in which the KeY-system can proof the functional specification automatically. However, at the time of writing we don’t have a bigger case-study which can substantiate this claim.

Outline. As a starting point, some basics of Java Dynamic Logic (JAVADL) will be summarised in the next section. Afterwards, a simple definition of non-interference and its formalisation in JAVADL will be considered in Section 3. The formalisation will be illustrated on a password checker example which will be used and extended throughout the paper. Section 4 gives a short introduction to JML and JML* and continues with the definition of an extension of JML* suitable for the specification of information flow properties. Section 5 describes the translation of the JML* extensions into JAVADL. Finally the approach is discussed in a conclusion in Section 6.