Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 131 (2005) 85–98

www.elsevier.com/locate/entcs

Towards a Complete Static Analyser for Java: an Abstract Interpretation Framework and its Implementation

Isabelle Pollet[1]

University of Namur

Baudouin Le Charlier[2]

Université Catholique de Louvain

Abstract

We present an abstract interpretation framework for a subset of Java (without concurrency). The framework uses a structural abstract domain whose concretization function is parameterized on a relation between abstract and concrete locations. When structurally incomptatible objects may be referred to by the same variable at a given program point, structural information is discarded and replaced by an approximated information about the objects (our presentation concentrates on type information). Plain structural information allows precise intra-procedural analysis but is quickly lost when returning from a method call. To overcome this limitation, relational structural information is introduced, which enables a precise inter-procedural analysis without resorting to inlining.
The paper contains an overview of the work. We describe parts of the standard and abstract semantics; then, we briefly explain the fixpoint algorithms used by our implementation; lastly, we provide experimental results for small programs.

Keywords: Abstract Interpretation, Java, Type Analysis, Pointer Analysis, Program Verification, Program Specialization.

[1] Email:ipo@info.fundp.ac.be
[2] Email:blc@info.ucl.ac.be

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.01.025

Introduction

There is a broad range of applications for the static analysis of Java. However, a major issue is the correctness of the analysis itself, especially when it is used in optimizing compilers. But, designing an analysis and proving its correctness is often tedious and error-prone. It is therefore reasonable attempting to design a 'generic framework' easily adaptable to perform various kinds of analyses in order to minimize the correctness proof effort. Our work is a contribution to such a 'generic framework': We define and implement a Java code analyser based on the abstract interpretation methodology [8,9,10].

We limit our analysis to a (partly arbitrary) subset of the language. This subset is, on the one hand, representative enough of the main Java features and, on the other hand, sufficiently small to be completely dealt with in a first approach. Concurrency is the main Java feature that we eliminate. This aspect is very important but rather 'orthogonal' to the object-oriented aspects. We also assume the availability of the complete source code, ignoring, at the moment, the problem of dynamic class loading (see e.g. [17]).

We apply the abstract interpretation methodology as follows: We define a straightforward standard semantics, abstract domains and an abstract semantics on those domains, which correctly approximates the standard semantics. We finally use a post-fixpoint algorithm to compute a (relevant part of) the abstract semantics.

Our abstract domains contain structural information and closely resemble to the standard domain (consisting of an environment and a store). Abstract locations may be annotated with various kind of information, making the framework generic. Structure sharing at the abstract level can be interpreted in several different ways, at the standard level, giving rise to three variants of the abstract domains [3].

The result of an analysis is a table of abstract input/output states describing method and constructor calls that can potentially arise during an actual standard execution. Such a table is similar to (and allows one to easily build) a (precise) call graph [12,26] for the whole program.

This paper presents an overview [4] of the work and is composed of five main sections. Section 1 provides a brief overview of the standard semantics. Section 2 describes the abstract domains. Section 3 sketches the abstract semantics definition. Section 4 details the results of the analysis for small programs. Section 5 discusses the related work.

[3] A preliminary presentation of the abstract domains appeared in [20].
[4] All technical definitions can be found in [19].