Runtime Universe Type Inference

Frank Lyner
Supervising Professor: Peter Müller
Supervising Assistant: Werner Dietl

January 11, 2005 — July 10, 2005

1 Background

Aliasing occurs in object-oriented programming whenever an object is referred to by more than one object or variable. For example, the cells in a doubly-linked list are, in general, referenced by the predecessor, the successor, and possibly by the list head and iterators.

Aliasing is a fundamental principle for the flexibility and efficiency of object-oriented programs, but in some cases it is undesirable and a source of errors: the existence of aliases can lead to unforeseen manipulations of object structures, leading to inconsistencies and violations of invariants. For the implementation of a doubly-linked list you would e.g. assume that the cell structure is only manipulated by the list head. Unwanted references to cells could be used to destroy the list structure.

The Software Component Technology group has developed the Universe type system. It is an extension to the Java type system that allows the programmer to perform aliasing and dependency control in a flexible way.

2 Mission

The goal of this project is to develop an algorithm and a tool that is able to automatically annotate a normal Java program with Universe types. The ownership relations of the objects of the program are detected through observation of the program execution. This information is then used to infer the correct Universe types. Depending on the implementation the tool will either accept only Java source file or source and binary files. The output will be either annotated source files or separate JML specifications.

The static correctness of these annotations will be checked through the Universe compiler.
The core of the project consists of three parts:

1. Development of the algorithm with the focus on low complexity, correctness and completeness.

2. Implementation of a prototype.

3. Measurements of the time and memory consumption of the prototype.

For simplicity reasons, the algorithm will only annotate object fields and method signatures. The annotation of local variables and static fields is not part of the project core.

3 Extensions

There are several possible extensions:

1. *Annotation iteration:* For a given program there are typically a number of different annotations that are correct. To get the desired results, the process could be made iterative. After each automatic annotation the programmer is asked to clarify ambiguous or conflicting settings and to introduce annotations himself.

2. *Local Variables:* The local variables that are not considered in the core of the project represent a possible extension.

3. *IDE integration:* Since the goal of the project is to help the programmers, an obvious step is to integrate the tool in a widely used IDE—like Eclipse.

This represents only a selection of possible extensions. The decision which extensions are to be implemented will be taken in the course of the project.