



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



# Diseño y desarrollo de un fragmentador de programas para Java basado en el System Dependence Graph

Trabajo Fin de Máster

Máster Universitario en Ingeniería  
y Tecnología de Sistemas Software

**Autor:** Javier Costa Rosa  
**Tutor:** Josep Francesc Silva Galiana  
Valencia, septiembre de 2019

## **Abstract**

The abstract

## **Resumen**

El resumen

***Keywords*** — Java, Program Slicing, System Dependence Graph

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Program Slicing . . . . .	3
1.2	Goals . . . . .	4
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Slicing approaches of Java programs . . . . .	4
<b>3</b>	<b>The Java language and JavaParser</b>	<b>4</b>
3.1	The Java language . . . . .	5
3.2	The Abstract Syntax Tree . . . . .	5
3.3	JavaParser . . . . .	5
3.3.1	Declarations . . . . .	5
3.3.2	Statements . . . . .	5
3.3.3	Expressions . . . . .	5
<b>4</b>	<b>Developing the slicer</b>	<b>5</b>
4.1	Scope . . . . .	5
4.2	Design . . . . .	5
4.3	Implementation . . . . .	5
<b>5</b>	<b>Tests/Performance (?)</b>	<b>5</b>
<b>6</b>	<b>Conclusions</b>	<b>5</b>
<b>7</b>	<b>Future work</b>	<b>5</b>

# 1 Introduction

## 1.1 Program Slicing

Program Slicing is a debugging technique firstly introduced by Mark Weiser in 1981 [4] which consists in obtaining a reduced version of a program (the slice) obtaining the parts of it that are relevant for a given slicing criterion. Some of the main applications of Program Slicing are debugging and performing program analysis, among others.

A slicing criterion gives information about what variable or variables are targeted and in what point of the program the slicing will be performed for them, i.e., given a program  $p$ , a slicing criterion  $\langle s, v \rangle$  specifies a statement  $s$  and a set of variables  $v$  in  $p$ .

Depending on the variant of program slicing applied, the slicing criterion may have to give more information. There are many of this variants, but however, they all can be classified into static or dynamic, and forward or backward:

- The static slicing do not take into account any particular execution. As a consequence, the slice will result in all the possibly relevant parts of the code, whatever the program input is. On the other hand, the dynamic program slicing is always related to a particular execution of the program and, therefore, an input for that execution. The slicing criterion will vary depending on whether the slicing is static or dynamic. If it is dynamic, the slicing criterion will have to provide, in addition to the statement and the set of variables, the input of the program
- The backward slicing will get all the parts of the program that can affect the statement specified in the slicing criterion. On the other hand, the forward slicing will do the same but for all the parts that can be affected by the specified statement

Based on this classification, many variants of program slicing have been specified. However, they will not be discussed here, as this work is focused on performing a **static**, **backward** slicing of a Java program.

### Example 1

```
1  void main() {  
3      int x = 1  
        int y = 2;  
5  }
```

To be able to perform slicing to a program, the dependencies between its statements must be computed first. There are three data structures commonly used to compute these dependencies, where each one of them represents different dependencies:

- The Control Flow Graph (CFG) is a directed graph that stores the control flow dependencies. That is, the order in which the statements are executed in the program

- The Program Dependence Graph (PDG) is an oriented graph that stores the control and data dependencies. A statement is control-dependent of other when the second one has to be executed in order to be able to execute the first one. A data dependency occurs when the statement a uses a variable that has been defined in statement b, a can be reached from b in the CFG and in this path the variable is not redefined (i.e. there is a path that connects them without redefining the variable). The PDG is able to represent only intraprocedural control and data dependencies, but not interprocedural
- The System Dependence Graph (SDG) is a graph that is able to represent interprocedural control and data dependencies. Its construction is, in essence, an interconnection of PDGs (each procedure has its own PDG and the SDG is built by connecting them)

## 1.2 Goals

The main goal of this work is to develop a slicer that is able to take a Java program and a slicing criterion as input and make a static, backward slicing of it. As a result, the output will consist in:

- The generated graph (or graphs)
- The executable slice (i.e. the reduced program)

## 2 Background

### 2.1 Slicing approaches of Java programs

slicers with their differences table

## 3 The Java language and JavaParser

Java

Abstract Syntax Tree

JavaParser builds the AST

- 3.1 The Java language
- 3.2 The Abstract Syntax Tree
- 3.3 JavaParser
  - 3.3.1 Declarations
  - 3.3.2 Statements
  - 3.3.3 Expressions
- 4 Developing the slicer
  - 4.1 Scope
  - 4.2 Design
  - 4.3 Implementation
- 5 Tests/Performance (?)
- 6 Conclusions
- 7 Future work

## References

- [1] Citation needed
- [2] Matthew Allen, Susan Horwitz. *Slicing Java Programs that Throw and Catch Exceptions*. 2003.
- [3] Mark D. Weiser. *Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. 1979.
- [4] M. Weiser. *Program Slicing* 1981.