

Macquarie University
Department of Computing

PROJECT REPORT

Static Type Analysis of Dynamically Typed Programming Language

Author: Stepan Sindelar

Student ID: 43600220

Supervisor of the project: Matthew Roberts

Study programme: exchange student

Contents

1	Introduction	2
1.1	Problem Description	2
1.2	Thesis structure	2
2	Static Analysis	3
2.1	Existing Approaches	3
2.2	Data Flow Analysis	3
2.3	The PHP Programming Language	3
3	Existing Software	4
3.1	HipHop	4
3.2	Weverca	4
4	Implementation	5
5	Results	6
6	Conslusion	7
6.1	Future Work	7
	Bibliography	8
	List of Tables	9
	Attachments	10

1. Introduction

1.1 Problem Description

Static type analysis of a dynamic language, benefits: compiler optimizations, IDE support - bug hunting analysis.

1.2 Thesis structure

2. Static Analysis

More detailed description of what static analysis is (as opposed to for example explicit model checking, verification, etc.).

2.1 Existing Approaches

List and short description of existing approaches to static analysis and related terminology (Data Flow, context-sensitive, path-sensitive, ...).

2.2 Data Flow Analysis

Detailed description of Data Flow analysis or just reference to the literature in the previous chapter and this chapter would be left out.

2.3 The PHP Programming Language

Note: maybe a new chapter?

What makes static analysis in PHP harder than of statically typed language

what are the caveats of PHP we had to deal with

PHPDoc annotations and their significance for type analysis (especially for the bug hunting)

3. Existing Software

There are lots of different incomplete solutions to slightly different but relevant problems: e.g. Weverca focuses on security vulnerabilities. How many other implementations should I mention and in what detail?

3.1 HipHop

3.2 Weverca

4. Implementation

specific constraints:

- must use Phalanger front end and its AST data structures,
- should be ready to be connected in between the Phalanger front end and back end as a middle end,
- should be ready to be used continuously: interactively re-analyze updated code when used inside IDE,

overall description:

- Control Flow graph construction
- discussion of the choice of intermediate representation (or in fact, why we do not use any intermediate representation).
- generic framework for Data Flow analyses
- type analysis

5. Results

Analyses of well known OSS PHP projects.

6. Conclusion

6.1 Future Work

Branched Data Flow analysis, integer interval analysis, integration with the compiler back-end.

Bibliography

List of Tables

Attachments