Karlsruhe Institute of Technology

Institute for Theoretical Computer Science
Research Group Verification meets Algorithmics
Head: Dr. Carsten Sinz

Am Fasanengarten 5
76131 Karlsruhe, Germany

Phone: +49 721-608-4-4212
Fax: +49 721-608-4-4211
Email: carsten.sinz@kit.edu
Web: verialg.iti.kit.edu

KIT-Campus Süd | ITI | Am Fasanengarten 5 | 76131 Karlsruhe, Germany

To whom it may concern

Date: 2014-08-15

Review of the PhD Thesis of David Hauzar
"Towards Static Analysis of Languages with Dynamic Features"

Scientific Background

Web applications, which are ubiquitous nowadays, require a high level of code quality to avoid unintended behavior (resulting in user frustration or even financial losses) and to protect private data. Static analysis of programs provides a means to avoid such program errors with high reliability and ascertain a level of code quality that goes beyond what can be achieved by testing. Static analysis of programs has been a research topic for several decades, but in the past mainly focused on statically typed languages such as C or Java. In web applications, dynamic programming languages (such as JavaScript, PHP, or Perl) are predominant, both on the server and the client side. Analyzing such programs is much harder, as, e.g., variables may change type, code may be added, or objects may be modified during execution of the program. Statically analyzing such programs requires new techniques for heap and value analysis, which are developed in this thesis.

Summary of the Thesis

David Hauzar's thesis "Towards Static Analysis of Languages with Dynamic Features" deals with a wide range of research topics in the area of analyzing programs written in dynamic programming languages. Chapter 1 provides an introduction to the topic, presents an example that is used throughout the thesis (written in PHP), and identifies research goals. Chapter 2 presents the state of the art in analyzing programs written in dynamic languages, focusing on security aspects and heap analysis. Limitations of existing techniques (e.g. on decompsing updates) are also addressed. The chapter closes with an exposure of techiques to combine heap and value analysis. The short Chapter 3 revisits the goals in the light of the notions of Chapters 1 and 2, stressing the importance of a precise and sound analysis, for which an interplay between heap and value analysis is needed. Chapter 4, which can, together with Chapter 5, be considered the core of the thesis, starts with an introduction to features of dynamic languages (associative arrays with dynamic types, dynamic accesses, references and aliasing), presented by an example. Further on, it gives a formalization of the heap analysis, including associative arrays of arbitrary depth, precise modeling of multi-dimensional updates as well as aliases between variables, array indices and object properties. The formalization is accomplished via data-flow equations, including the modelling of read and write accesses as well as merging of multiple predecessor states. Chapter 4 closes by giving an informal proof of the termination and soundness of the heap analysis procedure. In Chapter 5 a framework for static analysis of dynamic languages is developed, consisting of the

heap analysis of the previous chapter, which is extended by value analysis. The value analysis uses a value domain for the heap, an interval domain for numeric values, and several other domains for other types such as String and Boolean. An intermediate representation (IR) is also presented, consisting of a graph with value and non-value nodes and flow and value edges. It serves for modeling control flow. The whole analysis framework is illustrated via several graphical representations of the heap and value components of program states. Chapter 6 presents the implementation of the ideas from Chapters 4 and 5 in a tool called WeVerca. This tool, which is implemented within the IDE Eclipse, provides simple error-detection features based on the abstract syntax tree (AST), such as detection of nested function declarations, as well as more complex code analyses based on the static analysis framework of Chapter 5, such as taint analysis for identifying security vulnerabilities. In Chapter 7 the framework and its implementation are experimentally evaluated. First, scalability of the heap analysis is examined on some (perhaps a bit artificial) examples. Thereafter, a comparison with existing tools (Pixy, Phantm) on a small weblog application written in PHP is presented, showing the superiority of the WeVerca implementation with respect to error coverage and false positive rate. Finally, a larger example of the analysis of a webmail client is presented, in which WeVerca found a previously unknown security flaw. For both the weblog application and the webmail client reported errors are explained in detail. Chapter 8 closes the thesis with a summary and an outlook on possible extensions and future work.

Evaluation

David Hauzar's thesis deals with a broad range of topics around the central theme of statically analyzing dynamic programming languages. The main contributions of his thesis are:

1. A thorough analysis of existing techniques for statically analyzing dynamic programming languages.
2. A formalization of heap and value analysis for such programming languages, taking into account the precise semantics of non-decomposable, multidimensional updates to associative arrays.
3. Development of a complete framework to statically analyze dynamic languages such as PHP, and its implementation in the tool WeVerca.
4. A detailed experimental evaluation of the tool WeVerca, comparing it with existing tools developed by other research groups.

All results have been published in peer-reviewed conference (publications [25, 44]) or workshop (publications [23, 24, 26]) proceedings.

The results are presented nicely; the dissertation is very readable and clearly structured, I could not find any errors in the presented material. David Hauzar has shown that he is able to identify important research directions, develop new approaches, formalizations and algorithms, select suitable experiments and execute them thoroughly. It becomes clear that he has an extremely broad range of abilities, from theoretical work to practical, experimental and implementation work. What particularly impressed me is the precision with which he formalized the extremely complex heap analysis of a dynamic language.

This is a very nice thesis. It clearly advances the state of the art in analyzing programs written in languages with dynamic features and shows the author's ability for creative scientific work. I expect his work to have considerable impact on future research, and lead to more secure and stable web applications.

I can thus wholeheartedly recommend his PhD thesis for the oral defense. Given the achievements, the candidate should be granted a PhD degree.

Sincerely,

Questions to be answered during the defense:

- What is the reason for the relatively high false positve rate in the experiments and how could it be improved?
- Would it be possible to formalize the heap and aliasing/references in another way, e.g., by modelling the refcount and is_ref fields of PHP's variable containers explicitly? What would be the advantages and disadvantages?
- Would it be advisable or not to use an SMT solver, e.g., for the value domain?
- Which parts of the formalization are precise and where are abstractions used? Could you use an abstraction-refinement cycle to improve abstractions?
- What would be needed for a path-sensitive analysis?
- Which aspects of your work could help to improve static analysis procedures for non-dynamic languages?