2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

JSFox: Integrating Static and Dynamic Type Analysis of JavaScript Programs

Tian Huat Tan[¶], Yinxing Xue[†], Manman Chen[*], Shuang Liu[‡], Yi Yu[§], Jun Sun[*]
[¶]Acronis, Singapore, [*]SUTD, Singapore, [†]NTU, Singapore, [‡]SIT, Singapore, [§]NII, Japan

Abstract—**JavaScript is a dynamic programming language that has been widely used nowadays. The dynamism has become a hindrance of type analysis for JavaScript. Existing works use either static or dynamic type analysis to infer variable types for JavaScript. Static type analysis of JavaScript is difficult since it is hard to predict the behavior of the language without execution. Dynamic type analysis is usually incomplete as it might not cover all paths of a JavaScript program. In this work, we propose JSFox, a browser-agnostic approach that provides integrated type analyis, based on both static and dynamic type analysis, which enables us to gain the merits of both types of analysis. We have made use of the integrated type analysis for finding type issues that could potentially lead to erroneous results. JSFox discovers 23 type issues in existing benchmark suites and real-world Web applications.**

I. Introduction

JavaScript is arguably one of the most used programming languages [3], [5]. It has been supported by all modern Web browsers. It can be executed on almost all kinds of platforms (mobile, PC, tablets, etc.) with various operating systems (Windows, Mac OS, Linux, etc.). Since the advent of Web 2.0, many functionalities which were originally implemented on the server, have migrated to the client-side with the help of JavaScript. Nowadays, many complicated Web applications, such as Gmail or Google Docs, are written entirely in JavaScript.

JavaScript is a dynamic, weakly typed language with many flexible features such as dynamic code evaluation, function variadicity, and constructor polymorphism. This can ease the programmer's job for writing compact code. Unfortunately, the freedom and dynamism of Javascript is a double-edged sword. No compile-time warnings are shown when variables with inconsistent types are used. Even worse, some values are coerced into another type, leading to incorrect behavior without any obvious sign of misbehavior. Type analysis for JavaScript can help mitigate these issues and improve the development efficiency. Type analysis is crucial for capturing representation errors, e.g., misuse a number as the array. Moreover, type information is the basis for program analysis methods like symbolic execution [13], and can serve as an abstraction for analysis methods like testing and model checking [9].

Existing works use either static [7], [14], [18] or dynamic [16] analysis to infer variable types in JavaScript programs. However, both approaches have their limitations. For static type analysis, the analysis is often restricted to a subset of JavaScript language features in order to achieve soundness. To perform analysis for the entire JavaScript program, unsound static type analysis has been adopted (e.g., [15]). Even so, many dynamic features of JavaScript remain very difficult to infer statically. Dynamic type analysis can address the challenges caused by the dynamic features of JavaScript. Nevertheless, unlike static type analysis, dynamic type analysis is mostly incomplete, as dynamic execution might not cover all program paths in the JavaScript program.

In short, we present a novel approach that combines static and dynamic analysis to infer types for JavaScript programs, that could be used for various purposes such as detecting type issues in JavaScript programs. JSFox has been evaluated on several real-world Web applications and a popular JavaScript benchmarks collection, i.e., JetStream [2], which includes benchmarks from the SunSpider 1.0.2 [4] and Octane 2 [11]. JSFox has shown to be effective in identifying type issues so that developers can easily fix the problem and improve the code. There are 23 type issues reported in existing benchmark suites and real-world Web applications.

We summarize the contributions in the following.

- We propose an integrated type analysis for JavaScript programs which facilitate the usage of dynamic type analysis in refining static type analysis, for the purpose of inferring types and finding type issues.
- We have developed JSFox, which is browser-agnostic and targeted at all features of Javascript.
- We have evaluated our approach on popular JavaScript benchmarks, and several real-world Web applications. The evaluation shows that our method have detected 23 type issues with a low false positive rate.

II. Architecture of JSFox

The architecture of JSFox is shown in Figure 1. The input of JSFox is a JavaScript program and the output would be the type valuation of the JavaScript program. The input JavaScript program is first normalized. A JavaScript program is first normalized into a three-address-code-like [6] format that is amenable for analysis. The normalized program is then analyzed using static and dynamic components of JSFox respectively. The

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.91

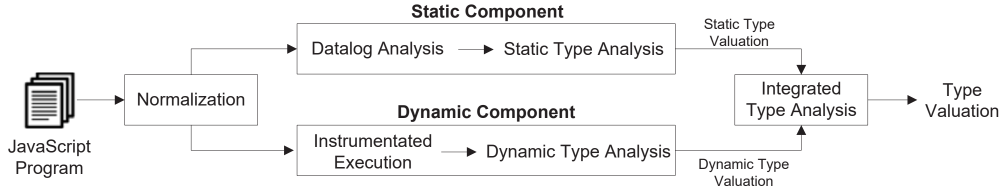

Fig. 1: Architecture of jsFox

output of these components will be static and dynamic type valuations. Both kinds of type valuations are then integrated by integrated type analysis to yield the final type valuation.

Static component is composed of two steps: Datalog analysis and then static type analysis. Datalog analysis includes control flow analysis and pointer analysis. An important part of control flow analysis is call-graph discovery. Our proposed static analysis is field-sensitive, flow-sensitive, context-insensitive, and path-insensitive

For dynamic component, there are also two steps: instrumented execution and dynamic type analysis. In the instrumented execution, we instrument the normalized JavaScript program in order to obtain the variable values and dynamic call graph edges. For collection of variable values, we record the values of variables that have been assigned at a particular line.

In the integrated type analysis, we make use of dynamic type analysis in refining static type analysis to make the analysis more complete and precise. It is very hard to obtain information in the static analysis like variadic functions. Whereas such information can be easily obtained from dynamic analysis. Therefore, our integrated analysis makes use of the information from dynamic analysis to refine the static analysis. In particular, two important pieces of information from dynamic analysis are used to refine the static analysis.

- Dynamic call graph edges, E_d – The instrumented program will be able to capture the call graph edges that cannot be easily discovered by static analysis.
- Dynamic type valuation, Q_d – It is the output of dynamic type analysis. Dynamic analysis can discover variable values that are obtained through dynamic features of JavaScript, e.g., runtime code evaluation.

III. Evaluation

We evaluate jsFox on popular JavaScript benchmarks and real-world Web applications to show our efficiency. In the following, we will present our evaluation in details. We have implemented jsFox in C#. The parsing and normalization of JavaScript programs are done by Esprima [12] and JS_WALA [8] respectively. We use Z3 [10] for solving the Datalog. The JavaScript program is instrumented with Jalangi [17].

As a baseline comparison, we compare jsFox with the state-of-art approach presented in [16]. The approach in [16] is based on pure dynamic analysis. We evaluate our tool on a popular JavaScript benchmarks collection and real-world Web applications. We introduce them in the following:

- JetStream [2]: We use JetStream version 1.1, which includes benchmarks from the SunSpider 1.0.2 [4] and Octane 2 [11]. We exclude *earley-boyer*, *typescript*, *zlib* and *coad-load* because they are obfuscated, which makes the source code difficult to analyze. evaluated in [16].
- Web Applications: Five real-world Web applications are taken from open source JavaScript frameworks and their test suites [1].

As the experiment result, we have discovered 23 type issues, and out of them 8 cases can only be detected by integrated type analysis. The approach in [16] has identified 12 of them, which are all included in our pure dynamic type analysis. We explain the reason in the following. For dynamic type analysis, we use different representation from [16] to record the observed values during execution. In particular, [16] uses type graphs, and we use lattices. Nevertheless, since both approaches record the observation of variable values during an execution, this makes their reasoning power almost equivalent. In addition, there are only 5 false positives out of 349469 lines of programs, this conveys to us that the rate of false positive is low.

IV. Conclusion and Future Work

In this work, we have proposed jsFox that makes use of both integrated typing analysis – that leverages both static and dynamic typing analysis, which allows more precise and complete type analysis than any individual typing analysis can obtain. We have applied our methods in evaluating popular benchmarks and several real-world Web applications. The typing analysis has been shown to be able to identify 23 type issues.

As future work, we will investigate our type inference method for other analysis methods such as symbolic execution. In addition, we will also investigate other methods in combining static and dynamic analysis to provide better synergy between these analysis.

References

[1] Defensive javascript. http://www.defensivejs.com/.

[2] Jetstream. http://browserbench.org/JetStream/.

[3] Language trends on github. https://github.com/blog/2047-language-trends-on-github.

[4] Sunspider. https://webkit.org/perf/sunspider/sunspider.html.

[5] Tiobe index for april 2016. http://www.tiobe.com/tiobe_index.

[6] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[7] C. Anderson, P. Giannini, and S. Drossopoulou. Towards type inference for javascript. In *ECOOP 2005 - Object-Oriented Programming, 19th European Conference, Glasgow, UK, July 25-29, 2005, Proceedings*, pages 428–452, 2005.

[8] I. T. W. R. Center. Javascript wala, note=https://github.com/wala/JS_WALA.

[9] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[10] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[11] G. Developers. Octane. https://developers.google.com/octane/.

[12] A. Hidayat. Esprima. http://esprima.org/.

[13] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[14] F. Logozzo and H. Venter. RATA: rapid atomic type analysis by abstract interpretation - application to javascript optimization. In *Compiler Construction, 19th International Conference, CC 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 66–83, 2010.

[15] M. Madsen, B. Livshits, and M. Fanning. Practical static analysis of javascript applications in the presence of frameworks and libraries. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 499–509, 2013.

[16] M. Pradel, P. Schuh, and K. Sen. Typedevil: Dynamic type inconsistency analysis for javascript. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 314–324, 2015.

[17] K. Sen, S. Kalasapur, T. G. Brutch, and S. Gibbs. Jalangi: a tool framework for concolic testing, selective record-replay, and dynamic analysis of javascript. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, pages 615–618, 2013.

[18] T. Zhao. Polymorphic type inference for scripting languages with object extensions. In *Proceedings of the 7th Symposium on Dynamic Languages, DLS 2011, October 24, 2011, Portland, OR, USA*, pages 37–50, 2011.