Analyzing JavaScript: The Bad Parts

Nicholas R. Labich

University of Maryland

JavaScript is a popular, powerful, and highly dynamic programming language. It is the arguably the most widely used and ubiquitous programming language, has a low barrier to entry, and has vast amounts of code in the wild. JavaScript has grown from a language used primarily to add small amounts of dynamism to web pages into one used for large-scale applications both in and out of the browser–including operating systems and compilers. As such, automated program analysis tools for the language are increasingly valuable. Almost all of the research to date targets ECMAScript 3, a standard that was succeeded by the most recent version, 5.1, over four years ago. Much of the research targets well-behaved subsets of JavaScript, eliding the darker corners of the language (the bad parts) [5, 4]. In this work, we demonstrate how to statically analyze full, modern JavaScript, focusing on uses of the language's so-called bad parts. In particular, we highlight the analysis of scoping, strict mode, property and object descriptors, getters and setters, and eval.

Speed, precision, and soundness are the basic requirements of any static analysis [7]. To obtain soundness, we began with LambdaS5, a small, functional language developed by Guha et al. that, through desugaring, encompasses the entirety of the ECMAScript 5.1 standard [3]. Its small size and deliberate desugaring make it a tractable intermediate representation for the analysis of JavaScript. Leveraging the semantics of LambdaS5, we build an abstract machine interpreter. Following the abstracting abstract machines recipe developed by Van Horn and Might [6], a small-step abstract machine is easily and soundly transformed into an abstract interpreter.

Once we have a sound abstract interpreter, speed and precision can be configured from a number of angles. Optimization techniques developed by Johnson et al. make any AAM-style analyzer faster [1]. By parameterizing the analyzer's allocation function we gain another axis of control [2]. A modular design of the analyzer's values makes it possible to plug-and-play different lattices, which is necessary for testing different abstractions of JavaScript's values and non-trivial objects. The objects of the 5.1 specification are no longer simple key-value mappings, but map their string keys to either data or accessor properties, each of which hold their own metadata. Further, data properties can be converted to and from accessor properties. The objects themselves hold metadata that changes the semantics of modifying both their mappings and metadata, causing the same line of code to either update a value, silently have no effect, or raise an exception. In the setting of analysis, a map with abstract spaces for both keys and values presents its own challenges to retain soundness while remaining performant. As an example of a potential lattice, we discuss an abstraction of JavaScript's objects that significantly improves performance at the cost of a loss in precision.

References

1. Johnson, J.I., Labich, N., Might, M., Van Horn, D.: Optimizing abstract abstract machines. In: Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming. pp. 443–454. ACM (2013)
2. Might, M., Manolios, P.: A posteriori soundness for non-deterministic abstract interpretations. In: Verification, Model Checking, and Abstract Interpretation. pp. 260–274. Springer (2009)
3. Politz, J.G., Carroll, M.J., Lerner, B.S., Pombrio, J., Krishnamurthi, S.: A tested semantics for getters, setters, and eval in JavaScript. In: Proceedings of the 8th Symposium on Dynamic Languages. pp. 1–16. ACM (2012)
4. Richards, G., Hammer, C., Burg, B., Vitek, J.: The eval that men do - a large-scale study of the use of eval in JavaScript applications. In: ECOOP 2011 – Object-Oriented Programming. Lecture Notes in Computer Science, vol. 6813, pp. 52–78. Springer (2011)
5. Richards, G., Lebresne, S., Burg, B., Vitek, J.: An analysis of the dynamic behavior of JavaScript programs. In: Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 1–12. ACM (2010)
6. Van Horn, D., Might, M.: Abstracting abstract machines. In: Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming. pp. 51–62. ACM (2010)
7. Vardoulakis, D.: CFA2: Pushdown Flow Analysis for Higher-Order Languages. Ph.D. thesis, Northeastern University (2012)