Hint-typing, optimistic compilation of Dynamic Languages

Manuel Serrano

2020-2021

subject	Hint-typing, optimistic compilation of Dynamic Languages
supervisor	Manuel Serrano
location	Inria Sophia-Antipolis
url	`http://hop.inria.fr`

Nowadays, JavaScript is no longer confined to the programming of web pages. It is also used for programming server-side parts of web applications, compilers [7], and there is a growing trend for using it for programming internet-of-things (IoT) applications. All major industrial actors of the field are looking for, or are already providing, JavaScript based development kits (IoT.js, Espruino, JerryScript, Kinoma.js, ...). In this application domain, JavaScript programs execute on small devices that have limited hardware capacities, for instance only a few kilobytes of memory. Just-in-time (JIT) compilation, which has proved to be so effective for improving JavaScript performances [5, 3, 4, 2], is unthinkable in these constrained environments. There would be just not enough memory nor CPU capacity to execute them at runtime. Furthermore memory write operations on executable segments are sometimes impossible on the devices, either because of the type of memory used (ROM or FLASH) or simply because the operating system forbids them (iOS for instance). Pure JavaScript interpreters are then used, but this comes with a strong performance penalty, especially when compared to assembly or C programs, that limits the possible uses.

When JIT compilation is not an option and when interpretation is too slow, the alternative is static compilation, also known as ahead-of-time (AOT) compilation. However, this implementation technique seems not to fit the JavaScript design whose unique combination of antagonistic features such as functional programming support, high mutation rates of applications, introspection, and dynamicity, makes most known classical AOT compilation techniques ineffective.

Indeed, JavaScript is hard to compile, much harder than languages like C, Java, and even harder than Scheme and ML two other close functional languages. This is because a JavaScript source code accepts many more possible interpretations than other languages do [6]. It forces JavaScript compilers to adopt a defensive position by generating target codes that can cope with all the possible, even unlikely, interpretations.

Fortunately, the general principles of JIT compilation that prove to be so efficient [1] can be accomated to AOT compilation. The static compiler can generate several versions for each function: a generic version that can cope with all the possible interpretations, and optimized customized versions, specialized for specific data representations. The open questions are *when* to decide to duplicate functions and *how* duplicating them.

We have conducted a first experiment that we have implemented in the hopc compiler for JavaScript [8, 9]. In order to decide which customized versions to generate, the compiler extracts as much as possible information from the source code. Modular compilation, *a.k.a.,* separate compilation, prevents it to always being able to make such deductions. In that case, it *speculates* beforehand on the data structures that are likely to be used by the exported functions. The key principle of the speculation is the following assumption. *The most likely data structure a program will use is the one for which the compiler is able to produce its best code.* This first experiment shows promising results because it let hopc being competitive with industrial JIT compiler for JavaScript on many programs.

The subject of this internship is to improve and to formalize the new typing discipline that we have coined *hint typing* that let the compiler implements its optimistic code generation.

References

[1] C. Chambers and D. Ungar. Customization: Optimizing compiler technology for SELF, a dynamically-typed object-oriented programming language. In *Conference Proceedings on Programming Language Design and Implementation*, PLDI '89, New York, NY, USA, 1989. ACM.

[2] M. Chang, E. Smith, R. Reitmaier, M. Bebenita, A. Gal, C. Wimmer, B. Eich, and M. Franz. Tracing for web 3.0: trace compilation for the next generation web applications. In *In Proceedings of the International Conference on Virtual Execution Environments*, 2009.

[3] M. Chevalier-Boisvert and M. Feeley. Simple and effective type check removal through lazy basic block versioning. In *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic*, 2015.

[4] M. Chevalier-Boisvert and M. Feeley. Interprocedural type specialization of javascript programs without type analysis. In *30th European Conference on Object-Oriented Programming, ECOOP 2016, July 18-22, 2016, Rome, Italy*, 2016.

[5] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Franz. Trace-based just-in-time type specialization for dynamic languages. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*, 2009.

[6] L. Gong, M. Pradel, and K. Sen. Jitprof: Pinpointing jit-unfriendly javascript code. Technical Report UCB/EECS-2014-144, Aug. 2014.

[7] Microsoft. TypeSscript, Language Specification, version 0.9.5, Nov. 2013.

[8] M. Serrano. Javascript aot compilation. In *14th Dynamic Language Symposium (DLS)*, Boston, USA, Nov. 2018.

[9] M. Serrano and M. Feeley. Property Caches Revisited. In *Proceedings of the 28th Compiler Construction Conference (CC'19)*, Washington, USA, feb 2019.