# Profile-Guided optimization for Dynamic Languages

Manuel Serrano

2020-2021

| | |
|---|---|
| **subject** | Profile-Guided optimization for Dynamic Languages |
| **supervisor** | Manuel Serrano |
| **location** | Inria Sophia-Antipolis |
| **url** | `http://hop.inria.fr` |

JavaScript is particularly difficult to implement efficiently because most of its expressions have all sorts of different meanings that involve all sorts of different executions. To give a single prominent example, the expression "obj.prop" might denote five radically different operations. It might fetch the property "prop" from "obj". It might scan the linked list of "obj"'s prototype chain and access the property "prop" of one of these objects along the way. It might involve calling a user defined function if "prop" is an accessor. It might involve allocating an object if "obj" is a primitive value. At last, it might involve jumping through another component of the standard runtime system if "obj" is a proxy object. No syntactic element nor type information let the execution engine discover in advance which evaluation schema to use. Checking all the possible interpretations in sequence and then execute literally, that is following the prescription of the language specification, the proper one would deliver very slow performances. All mainstream implementations use alternative strategies. Amongst all the possible interpretations, they favor the one that corresponds to the most frequent situation for which they elaborate a faster execution plan, and, as importantly, for which they elaborate a fast guard that let them decide as quickly as possible, if the faster schema would preserve the original semantics or not. Typically, that what "inline caches" and "hidden classes" achieve [2, 1]. Using a single test, the comparison of the object's hidden class with the inline cache, the guard checks if the offset where to read the property is known or not. If it is, the property is directly fetched using a single indexed memory read. Otherwise, the slower execution path checking all the possible situations, for instance, the situation where the property is an accessor, is used.

The common intuition is that dynamic compilers (jit compilers) are in a better position to use this heuristics-based strategy as they have the program

and the data at hand where they have to decide which code to generate. We disagree with this intuition and we have shown in [3, 4, **?**] this is not a significant advantage because in most common situations simple guesses let aot compilers infer the most likely types and at the cost of larger codes, they can generate several versions of the program and select the one to execute using guards similar to those jit compilers would use. However, it remains that in some situations, jit compilers can use the precise knowledge about values to generate better code. For instance, a jit compiler can possibly inline a closure on a call site if it monitors that a unique function is called from that location. Gathering such a precise information is most often out of reach of pure static analyses. This lack of information could be compensated by profiling techniques. This is the main subject of this PhD thesis.

The objective of this PhD thesis is to combine the static analyses of the JavaScript hopc compiler with new profile guided optimizations. The speed of the generated should be improved and the size of the generated code should be reduced. The techniques developed in this context should apply to all the high dynamic languages. JavaScript is the main target but languages such as Python or Ruby could also benefit from this study.

## References


[1] R. Artoul. Javascript Hidden Classes and Inline Caching in V8. `http://richardartoul.github.io/jekyll/update/2015/04/26/hidden-classes.html`, Apr. 2015.

[2] 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.

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

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