Sherlock: A Profiling Tool to Find Opportunities for Early Property Initialization

Tarek Auel, David Güsewell
tarek.auel@gmail.com, d.guesewell@web.de

TECHNISCHE UNIVERSITÄT DARMSTADT

Introduction

With Sherlock, we introduce a new profiling tool to find opportunities for **early object property and array element initialization** as shown in Listing 1. In order to do this, the analysis has to keep track of the usage of the elements or properties. The **expected benefits** for such early property initializations are (i) **improved code readability**, and (ii) **improved performance** because the computation time spent in **push** and **reverse** calls is now saved.

```
1 var a = [1, 2, 3]
2 a.push(4);
3 b = a.reverse();
4 /*
5 optimization: var a = [4, 3, 2, 1]
6                b = a;
7 */
```

Listing 1: Example for early property initialization

The main goals of Sherlock are to find these opportunities for early property initialization and keep the number of false positives as low as possible at the same time. Due to this, in some use cases Sherlock prefers not to optimizing a value instead of running in a potential false positive situation.

Overview

Overall **Sherlock is an implemented plugin for Jalangi**. Jalangi is a dynamic framework for JavaScript. By transforming the code and adding hooks, it allows the user to monitor almost every operation performed by the execution. Sherlock implements the callbacks which called by Jalangi in order to perform the desired dynamic analysis. Before the code is analyzed by Sherlock it is instrumented using Esprima, Estraverse and Escodegen in order to be able to track events that are not exposed by Jalangi such as the end of a branch. This additions allows to safely optimize coding which could not have optimized otherwise.

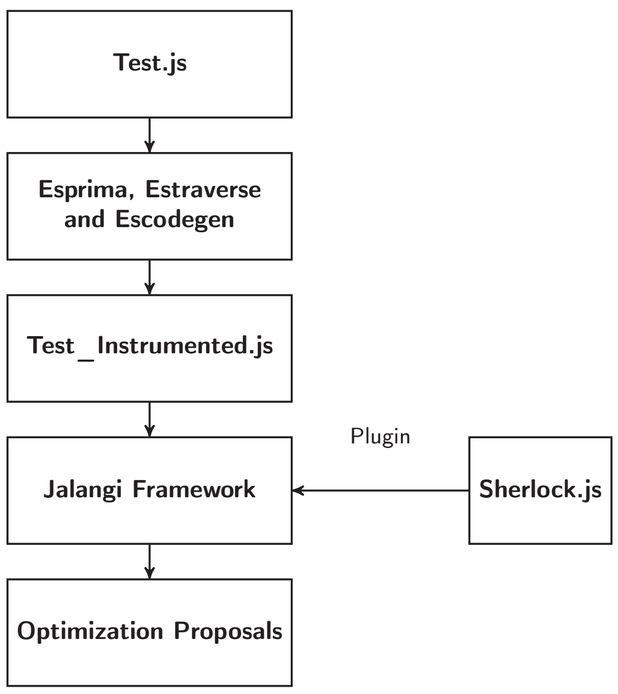


Figure 1: Optimization process using Sherlock with Jalangi

References

[1] J. M. Smith and A. B. Jones. *Book Title*. Publisher, 7th edition, 2012.

[2] A. B. Jones and J. M. Smith. Article Title. *Journal title*, 13(52):123–456, March 2013.

Sherlock

Sherlock is constantly evaluated with a set of self written test-cases. The tests proves which potential optimizations Sherlock can identify. But nevertheless, Sherlock may skip potential optimizations if it either may not be safe to optimize the code or Sherlock does not recognize that it is possible, yet.

Supported early property optimizations
The following listing shows two examples for coding that Sherlocks optimizes.

```
1 var a = [];
2 a[0] = 1;
3 //optimization: var a = [1];
4 var b = {};
5 b.a = 5;
6 //optimization: var b = {a: 5};
```

```
1 var a =[1,2,3,4];
2 a.reverse();
3 // optimization: var a = [4,3,2,1];
```

Listing 2: Examples for typical potential early property and array element initialization

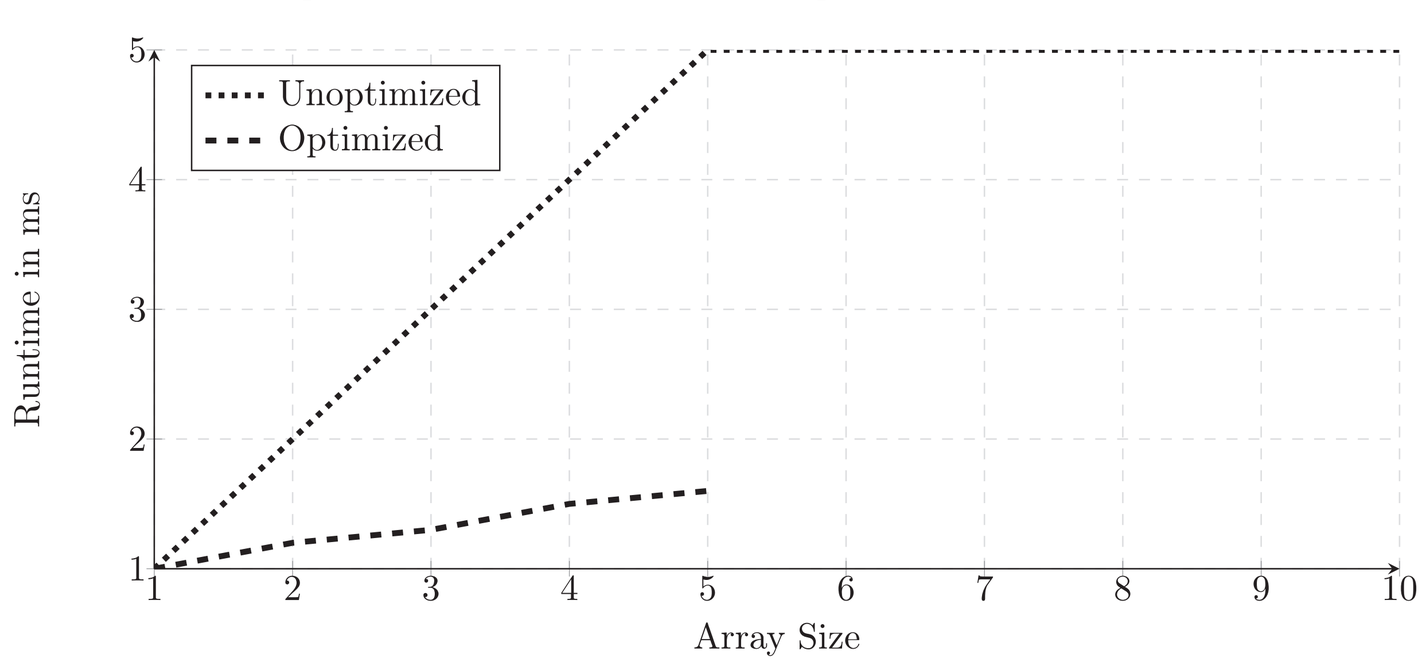
The left example of Listing 2 shows a typical potential of early property and array element initialization which is covered by Sherlock. But Sherlock does not only optimize elements or properties that are set later, but also many builtin standard JavaScript functions. For the right example, Sherlock recognizes that the array could be initialized in reverse order.
In the left example of Listing 3 the only supported optimization would be made for *a*. *b* must not be optimized, because this would change the semantics (in particular the output of line 6). *c* and *d* show two corner cases that are not supported, because Sherlock cannot distinguish using Jalangi if the *length* was used as index or as part of the assignment.
Conditioned initialization should only be done, if the optimization is always true. In the conditioned example only *b* can be optimized, because it is independent of the branching in line 2. *a* depends on *cond* and must not be optimized

```
1  var a = [];
2  a[a.length] = 0;
3  a[1] = 1;
4  //optimization: var a =[0,1];
5  var b = [1, 2, 3];
6  console.log(b.length);
7  b[3] = 4;
8  //cannot be optimized
9  var c = [];
10 c[c.length - 1] = 0;
11 //cannot be optimized
12 var d = [];
13 d[0] = d.length - 1;
14 //cannot be optimized
```

```
1 var a = []
2 if (cond) {
3   a[0] = 1;
4   var b = []
5   b[0] = 1;
6   //optimization: var b =[1];
7 }
```

Listing 3: Examples for coding that can be optimized by Sherlock

Evaluation
Sherlock still misses some potential optimizations. So far Sherlock cannot deal with objects that are created by constructors, because this introduces many situations where Sherlock has to keep track of even more properties of the coding and where often Sherlock has to discard a optimization because it may change the logic, because of side effects or other reasons.
The following figure shows the runtime difference of optimized and unoptimized array initializations. Even though Sherlock can inspect the Octane benchmarking suite, it does not find any optimization. The reason for that is the capabilities of Sherlock are still limited and that the benchmarks do not use plain objects or arrays. Especially they do not add elements within the same scope and they do not use builtin JavaScript functions that Sherlock can optimize.



Conclusion

We introduced a new profiling tool to find opportunities for early property initialization, named Sherlock. While there is still a long way to go to find all possible early property optimizations our results show that Sherlock is able to find a broad range of them.