May 10, 2018

Side-effect free program state evaluation

Erik Luo

Aleksei Koziatinskii

Yang Guo

Side-effect free program state evaluation

ABSTRACT

The ability to inspect the state of program execution at a specified stage by the evaluation of a specific expression is crucial for operations such as debugging. Executing the expression of interest may itself result in changing the program state that the evaluation of the expression is supposed to inspect. Such side effects of evaluating the expression on the program state reduce the effectiveness and reliability of debugging by evaluating the expression to inspect program state. The techniques of this disclosure enable evaluation of an expression for inspecting program state that is free from side effects.

KEYWORDS

- Debugging
- Program state
- JavaScript
- Directed graph

BACKGROUND

The ability to inspect the state of program execution at a specified stage by the evaluation of a specific expression is crucial for operations such as debugging. In programming languages such as JavaScript, executing the expression of interest may itself result in changing the program state that the evaluation of the expression is supposed to inspect. Some IDEs may provide enhanced debugging capabilities which reevaluating the expression at specific time intervals, e.g. 300 milliseconds. This capability, without ensuring that each evaluation is side effect free, may significantly change inspected program state and make debugging ineffective. Side effects of

evaluating the expression on the state of the program reduce the effectiveness and reliability of debugging and require extra efforts on part of a developer or test engineer.

DESCRIPTION

The techniques of this disclosure enable the evaluation of an expression in a programming language, such as JavaScript, that is free from any side effects. For example, such side effects include observable changes in the state of the program resulting from the evaluation of an expression meant to inspect the state of the program. To achieve evaluation that is free from side effects, a programming instruction and a built-in function is prevented from changing the program state during the evaluation of the specified expression. The execution of an expression that attempts a change that produces any side effects is terminated.

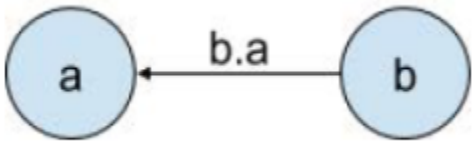

Fig. 1: Directed graph representation of program state

The techniques involve representing the program state as a directed graph, with each vertex of the directed graph representing an object within the program state. Fig. 1 illustrates an example directed graph of a program state with vertices a and b, representing objects A and B, respectively, such that:

```
a = {};
b = {a:a};
```

The direction of the arrow linking vertex b to vertex a indicates that object B can be obtained using the reference to the object A.

All new objects created during the evaluation of the expression of interest are considered temporary objects and located inside an independent component of the program state graph. The program instruction and the built-in function that creates the directed edge in the program state graph shown in Fig. 1 between object A and object B is enabled to terminate execution of an expression if it attempts to create a directed edge from a non-temporary object to a temporary object, thus avoiding side effects.

Further, the program instruction and the built-in function that changes a function object may potentially lead to side effects in the program state of the receivers of the function object, i.e., the arguments of the function call. The evaluation of an expression that attempts to call the function with a non-temporary object as a receiver, i.e., an argument to the function, is terminated, thus avoiding side effects.

The techniques of this disclosure support side-effect-free evaluation for nearly all expressions and built-in functions via temporary objects instead of a limited subset of side-effect-free functions for non-temporary objects. In a practical implementation of the techniques of this disclosure, program instructions and built-in functions that may create a directed edge in the directed graph of the program state are instrumented to abide by the operational constraints described above. Similarly, program instructions and built-in functions that may involve potential side effects on their receivers are instrumented to abide by the operational constraints described above.

The techniques of this disclosure enable the above operations by considering all object allocation during the evaluation of the expression of interest as temporary. The temporary objects are allocated and reused in a runtime heap profiler within a virtual machine. Built-in into most JavaScript virtual machines, the heap profiler is reused to track temporary objects during

evaluation. The heap profiler starts when the evaluation of the expression of interest begins and stops upon completion of the evaluation. Since any evaluation attempts that may cause side effects lead to termination of the execution of the expression, an expression that reaches completion of the evaluation is free from side effects on program state.

The heap profiler is utilized only for side-effect-free evaluation, thus avoiding runtime overhead for the evaluation of other expressions. Moreover, checks of side effects of functions can be carried out by reusing existing corresponding hooks for the function call that are provided by common debuggers for stepping purposes. When not stepping through the code, the hooks typically involve minimal to zero runtime overhead. Additionally, for programming languages that feature low-level instructions, the corresponding handlers may be patched only during side-effect-free evaluation. As a result, implementing the techniques of this disclosure for side-effect-free evaluation requires little runtime overhead. The techniques can be implemented for evaluation of programs in several programming languages.

The techniques of this disclosure enable features that allow the user to type expressions that provide live results with guarantees that the program state will not change due to evaluating the expression to obtain the live result.

CONCLUSION

Executing the expression of interest may itself result in changing the program state that the evaluation of the expression is supposed to inspect. Such side effects of evaluating the expression on the state of the program reduce the effectiveness and reliability of debugging. The techniques of this disclosure support side-effect-free evaluation for nearly all expressions and built-in function via temporary objects instead of a limited subset of side-effect-free functions for non-temporary objects. The expressions and built-in functions are instrumented to terminate the

evaluation of expressions that attempt to create a directed edge from a non-temporary object to a temporary object within the program state or call a function with a non-temporary object as a receiver. The techniques utilize existing features of debuggers and programming languages and can be implemented with minimal runtime overhead.