JSAVER: JavaScript Static Analyzer via ECMAScript Representations

Jihyeok Park
Oracle Labs
Brisbane, Australia
jihyeok.park@oracle.com

Seungmin An
KAIST
Daejeon, South Korea
h2oche@kaist.ac.kr

Sukyoung Ryu
KAIST
Daejeon, South Korea
sryu.cs@kaist.ac.kr

ABSTRACT

This document describes the artifact package accompanying the ESEC/FSE 2022 paper "Automatically Deriving JavaScript Static Analyzers from Specifications using Meta-level Static Analysis." The artifact includes the source code of JSAVER, the accepted paper, a companion report, ECMA-262 (JavaScript language specification) open-source repository as a git submodule, and scripts to replicate the evaluation results presented in the paper. JSAVER stands for a JavaScript Static Analyzer via ECMAScript Representation. It is the first tool that automatically derives JavaScript static analyzers from language specifications using an *interpreter*-based approach called meta-level static analysis instead of traditional a *compiler*-based approach. It extends another tool JISET to extract JavaScript definitional interpreters written in IR_{ES}, an intermediate representation for ECMAScript, from diverse versions of ECMA-262.

KEYWORDS

JavaScript, definitional interpreter, meta-level static analysis

1 GETTING STARTED GUIDE

The source code of JSAVER and the dataset of our study are publicly available at https://doi.org/10.5281/zenodo.6785678, and the latest version is maintained as a GitHub repository:

```
$ git clone --recurse-submodules \
    https://github.com/kaist-plrg/jsaver.git
```

Please see `INSTALL.md` for the detailed guide on installation and how to use this artifact.

Additionally, we packaged the artifact in a docker container. If you want to skip the environment setting, we recommend you to use it. You can install the docker by following the instruction in https://docs.docker.com/get-started/ and downlaod our docker image with the following command:

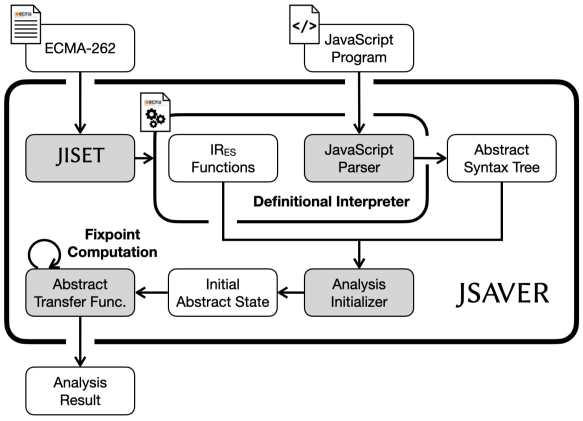
```
$ docker pull jhnaldo/icse-21-jest
$ docker run -it -m=16g --rm \
    jhnaldo/fse22-jsaver
# user: guest, password: guest
```

Note that the docker image is 3GB large thus be patient when you download it and please assign more than 16GB memory for the docker engine.

Details of the JSAVER framework are available in the original paper (`fse22-jsaver.pdf`) and the companion report for the formalization (`fse22-jsaver-report.pdf`).

2 OVERALL STRUCTURE

JSAVER consists of two phases: 1) definitional interpreter extraction and 2) meta-level static analysis.



2.1 Definitional Interpreter Extraction

We utilize another tool JISET, a JavaScript IR-based Semantics Extraction Toolchain,[1] to extract JavaScript definitional interpreters from given ECMA-262. In this artifact, we extracted the definitional interpreter from ES2021 (ES12), the latest version of ECMA-262, and manually filled out essential steps of its not-yet-compiled parts. It consists of two different main parts for semantics and syntax of JavaScript. For semantics, it compiles abstract algorithms in ECMA-262 to corresponding IR_{ES} functions. For syntax, it generates a JavaScript parser in Scala.

2.2 Meta-level Static Analysis

JSAVER performs a *meta-level static analysis* with JavaScript as its *defined*-language and IR_{ES} as its *defining*-language. Thus, it indirectly analyzes a JavaScript program by analyzing IR_{ES} functions with the AST of the program as an argument. Using the generated parser, it first parses a given JavaScript program to produce an Abstract Syntax Tree (AST). Then, Analysis Initializer constructs an initial abstract state with the extracted IR_{ES} functions and the produced AST. Finally, JSAVER computes the fixpoint of Abstract Transfer Function with the initial abstract state.

It utilizes a worklist algorithm to update the abstract state per control point, a pair of the following two components:

- A node in control-flow graph of the extracted definitional interpreter
- A view that represents an analysis sensitivity

[1] https://github.com/kaist-plrg/jiset

3 BASIC COMMANDS

You can run the artifact with the following command:

```
$ jsaver <sub-command> <option>* <filename>?
```

with the following sub-commands:

- `help` shows the help message.
- `extract` represents **Definitional Interpreter Extraction** phase that extracts a definitional interpreter from ECMA-262 defined in `ecma262/spec.html`.
  - `-extract:version=string` is given, set the git version of ecma262.
  - `-extract:genModel` is given, generate models of the extracted definitional interpreter.
- `analyze` represents **Meta-level State Analysis** phase that performs a meta-level static analysis for a given JavaScript program with the fixed definitional interpreter.
  - `-analyze:version=string` is given, set the git version of ecma262.
  - `-analyze:repl=string` is given, use a REPL for meta-level static analysis.
  - `-analyze:num=string` is given, set number domain (`flat` or `interval`).
  - `-analyze:str=string` is given, set string domain (`set-k`, `char-inc`, or `prefix-suffix`).
  - `-analyze:loop-iter=number` is given, set maximum loop iteration.
  - `-analyze:loop-depth=number` is given, set maximum loop depth.
  - `-analyze:js-k-cfa=number` is given, set k for JavaScript callsite sensitivity.
  - `-analyze:ir-k-cfa=number` is given, set k for IRES callsite sensitivity.
  - `-analyze:timeout=number` is given, set timeout of analyzer (second), 0 for unlimited.
- `collect` collects the final concrete/abstract state of a JavaScript program (used only for evaluation).
  - `-collect:concrete` is given, collect concrete state.
  - `-collect:js-k-cfa=number` is given, set k for JavaScript callsite sensitivity.

and global options:

- `-silent` is given, do not show final results.
- `-time` is given, display the duration time.

4 SIMPLE EXAMPLES

4.1 Definitional Interpreter Extraction

You can extract a definitional interpreter from ECMA-262 using the `extract` command. If you want to extract the definitional interpreter from ES12 (ES2021), please type the following script:

```
$ jsaver extract -extract:version=es2021 \
    -extract:genModel
```

You can freely set the target git version of ecma262 using the `-extract:version` option and dump the extracted definitional interpreter to use it during meta-level static analysis using the `-extract:genModel` option.

In this artifact, we extracted the definitional interpreter from ES2021 (ES12), the latest version of ECMA-262, and manually filled out essential steps of its not-yet-compiled parts. If you want to see the compiled IR_{ES} functions, please see the following directory:

```
src/main/resources/es2021/generated/algorithm
```

4.2 Meta-level Static Analysis

You can analyze a JavaScript file using the `analyze` command as follows:

```
$ jsaver analyze tests/js/addition1.js
```

It shows the final abstract state at the program exit point. In addition, it provides a read-eval-print loop (REPL) with the `-analyze:repl` option for more convenience:

```
$ jsaver analyze tests/js/addition1.js \
    -analyze:repl -silent
...

command list:
- help          Show help message.
...

RunJobs:[call: 0][loop: 0]:Entry[25909] -> {
  locals: {}
  globals: {
    SCRIPT_BODY: AST[ScriptBody](var x =...)
                 @ 0x7fb9f71f
  }
  heaps: {}
}

analyzer> continue
* Static analysis finished. (# iter: 2294)

analyzer> print -expr \
          REALM.GlobalObject.SubMap.x.Value
3.0

analyzer> exit
stop for debugging
```

5 HOW TO REPRODUCE EVALUATION RESULTS

Please see `INSTALL.md` for the detailed guide on how to reproduce evaluation results for each figure and table in the paper. All the shell scripts and data are archived in the `eval` directory.