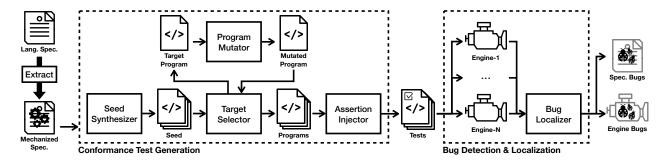
JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification

Artifact Evaluation

Jihyeok Park Seungmin An Dongjun Youn Gyeongwon Kim Sukyoung Ryu KAIST, South Korea

1 Artifact Description



The artifact consists of following modules to perform N+1-version differential testing of JavaScript engines and specification:

- SeedSynthesizer synthesizes an initial seed programs using the language syntax.
- TargetSelector selects a target program in the program pool that potentially increases the coverage of the language semantics by the pool.
- **ProgramMutator** generates a new program by mutating a given target program in order to increase the coverage of the language semantics by the program pool.
- AssertionInjector generates conformance tests by injecting assertions to the synthesized programs to check their final program states.
- **DifferentialTester** detects bugs in the specification and implementations via executing the conformance tests on multiple implementations.
- BugLocalizer localizes bugs on the specification using statistical information.

2 Getting Started Guide

The artifact is open-source can be obtained by cloning the following git repository:

```
$ git clone https://github.com/jhnaldo/jest.git
$ cd jest
```

To build and execute the artifact, you should follow the instructions in the INSTALL file in the artifact. Since we implement the artifact in Scala, it requires sbt, which is an intereactive build tool for Scala. Moreover, for differential testing, you also need to install four different JavaScript engines: V8 (v8.5), GraalJS (v20.1.0), QuickJS (2020-04-12), and Moddable XS (v10.3.0).

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 downland our docker image with the following command:

```
$ docker pull jhnaldo/icse-21-jest
```

\$ docker run -it jhnaldo/icse-21-jest # user: guest, password: jest

WARNING: The docker image is X.XGB large thus be patient when you download it.

3 Basic Commands

You can run the artifact with the following command:

```
$ jest <sub-command> <option>*
```

with the following sub-commands:

- help shows the help message.
- sample represents SeedSynthesizer and dumps seed programs to result/seed.
- generate loads seed programs from result/seed, repeatedly performs **ProgramMutator** with **Target-Selector**, dumps generated programs to result/programs. You can change the maximum iteration via the option -generate:iter=<number> (default: 10).
- inject loaded programs from result/programs and dumps results of AssertionInjector to result/tests.
- check performs DifferentialTester for result/tests and records bugs to result/bugs.
- localize performs **BugLocalizer** for found bugs in result/bugs. When the option -localize:answer is given, it reads answers from answer and shows their ranks.
- run integrates all modules to perform N+1-vesion differential testing at once.

and global options:

- -time shows duration time.
- -bugfix uses semantics extracted from bug-fixed ECMAScript.
- -detail prints intermediate processes.

4 Step-by-Step Instructions

4.1 RQ1. Coverage of Generated Tests

Execute the following command to check the size of seed programs and their syntactic coverage.

```
$ jest sample
```

Then, check the basic program generation with the following commmand.

```
$ jest generate
```

It shows the semantics coverage changes (Figure 4), and the number of generated programs and covered branches of mutation methods (Table I) during program generation. Even though it is impossible to exactly reproduce results because of the randomness in the program generation, you can check the tendencies by running the program generation with a large maximum iteration ($\geq 1,000$).

```
$ jest generate -generate:iter=1000
```

4.2 RQ2. Accuracy of Bug Localization

To reproduce the result in Figure 5, we provide the data used in the evaluation including programs generated by a single process and example programs invoke specification/engine bugs we found via the artifact. Type the following command:

```
$ rm -r result/programs
$ cp -r data result/programs
$ jest inject && jest check && jest localize -localize:answer
```

Moreover, we provide detailed data of each bug detected by the artifact in bugs.md. You can check the table in Section IV.B with this file.

4.3 RQ3/4. Bug Detection in JavaScript Engines/Specification

The file bugs.md also explains the Table II and Table III.