JSTAR: JavaScript Specification Type Analyzer using Refinement

Jihyeok Park
School of Computing
KAIST
Daejeon, South Korea
jhpark0223@kaist.ac.kr

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

Wonho Shin
School of Computing
KAIST
Daejeon, South Korea
new170527@kaist.ac.kr

Yusung Sim
School of Computing
KAIST
Daejeon, South Korea
yusungsim@kaist.ac.kr

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

***Abstract*—This document describes the artifact package accompanying the ASE 2021 paper "JSTAR: JavaScript Specification Type Analyzer using Refinement." The artifact includes JSTAR source code, the accepted paper, a companion report, ECMAScript (JavaScript specification) open-source repository as a git submodule, and scripts to replicate the data presented in the paper. JSTAR performs type analysis on any commit version of ECMAScript to detect type-related specification bugs. JSTAR is an extended tool of JISET, a JavaScript IR-based Semantics Extraction Toolchain. We hope that the artifact will be useful for contributors who update JavaScript specifications by automatically detecting type-related specification bugs before releasing them.**

***Index Terms*—JavaScript, mechanized specification, type analysis, refinement, bug detection**

I. Getting Started Guide

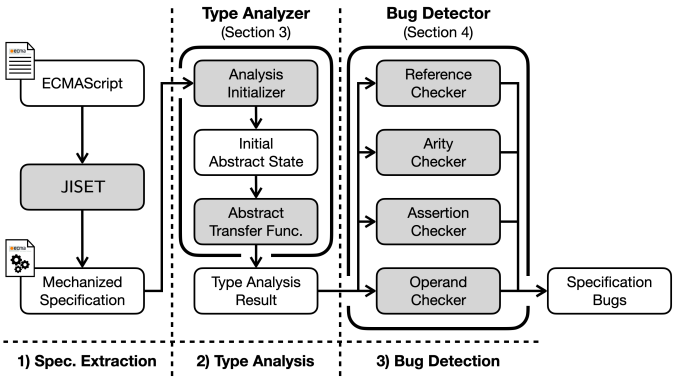
The artifact is open-source can be obtained by cloning the following git repository:

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

Please see `INSTALL.md` for the detailed guide of installation, how to use this artifact. We also provide this artifact in Zenodo with a DOI[1] and docker image as follows:

```
$ docker run -it --rm jhnaldo/jstar
```

II. Overall Structure



JSTAR consists of three phases: 1) specification extraction, 2) type analysis, and 3) bug detection. Details of the JSTAR framework are available in the paper `ase21-park-jstar.pdf` and the companion report `ase21-park-jstar-report.pdf`.

A. Specification Extraction

We utilizes another tool JISET[2], which is a **J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain, to extract a mechanized specification from given ECMAScript.

B. Type Analysis

JSTAR performs a type analysis with flow-sensitivity and type-sensitivity for arguments. Each function is split into multiple flow- and type-sensitive views, and an abstract state stores mapping from views to corresponding abstract environments. The type analyzer consists of two sub-modules: an **Analysis Initializer** and an **Abstract Transfer Function**.

- **Analysis Initializer** defines the initial abstract state and the initial set of views for a worklist.
- **Abstract Transfer Function** gets a specific view from the worklist and updates the abstract environments of the next views based on the abstract semantics for each iteration.

C. Bug Detection

To detect type-related specification bugs utilizing the type analysis, we developed four checkers in a bug detector:

- **Reference Checker** detects *reference bugs* which occur when trying to access variables not yet defined (‘UnknownVar‘) or to redefine variables already defined (‘DuplicatedVar‘).
- **Arity Checker** detects *arity bugs*. An arity bug occurs when the number of arguments does not match with the function arity (‘MissingParam‘).
- **Assertion Checker** detects *assertion failures*. An assertion failure (‘Assertion‘) occurs when the condition of an assertion instruction is not true.
- **Operand Checker** detects *ill-typed operand bugs*. An ill-typed operand bug occurs when the type of an operand does not conform to its corresponding parameter type.

[1]https://doi.org/10.5281/zenodo.5084817

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