OOPSLA'25 R2 Artifact Evaluation: ESMeta

Introduction

This tool is an extension of the **ESMeta**, **E**CMAScript **S**pecification **Meta**language. This framework extracts a mechanized specification from a given version of the ECMAScript/JavaScript specification (ECMA-262) and automatically generates language-based tools. It supports various features, including differential testing and static type checking for the language specification itself. Though this artifact still supports most of the features, the focus below is on performing type checking, as in the paper.

- Previously, ESMeta only performed syntactic refinement, which led to many false alarms due to insufficient information propagation. Occurrence typing was proposed to address this issue but struggled with mutation and non-boolean return types. Our work introduces a new technique using type guards, which achieves:
  - A substantial reduction in false alarms. Specifically, our tool shows (as seen in Figure 19) that applying our technique dramatically decreases false alarms (red area).
    - (a) **Base** (`-tycheck:no-refine`): baseline with no refinements.
    - (b) **Syn** (`-tycheck:infer-guard=false`): original ESMeta using only syntactic refinement.
    - (c) **Pre** (`-tycheck:use-syntactic-kill`): emulates *Occurrence Typing Modulo Theories (Kent et al., PLDI 2016)* by skipping analysis on potentially mutated variables.
    - (d) **Bool** (`-tycheck:use-boolean-guard`): restricts demanded types to true and false, illustrating the necessity of generalized demanded types.
    - (e) **Ours** (no extra flag): our full type guard implementation, reducing false alarms from an average of 263.63 (original *Syn*) to 26.7.
  - No significant performance degradation, as demonstrated in Figure 20.
  - Provenance information (**Prov**, `-tycheck:provenance`) is automatically extracted to show how type guards are derived and applied, helping readers better understand the specification.

Hardware Dependencies

This tool does not require high-end hardware but benefits from:

- A reasonably fast CPU. Since the analysis is single-threaded, multicore performance is not critical.
- At least 16GB of RAM, with 32GB or more recommended.

Tests were conducted on **Linux** systems with **AMD64** architecture.

Getting-Started guide (Kick-the-tire)

1. Download a Docker image from here.
2. Run `docker load -i esmeta.tar.gz` to load the image. It will show `Loaded image: esmeta:latest`.
3. Run a Docker with `docker run -it esmeta`, which gives you an interactive shell.
4. Done. You can execute esmeta with the `esmeta` command, or directly execute the jar(`java -jar bin/esmeta`).

    - You might want to test the artifact runs well with the esmeta tycheck command - it prints the analysis result of ES2024 with the latest version, which should yield 64 alarms.
    - Run `./benchmark.sh` or `./benchmark-test.sh` to get a result for the full/test benchmark. Each takes approximately 3 hours / 5 minutes.
    - FYI, `docker cp $(docker ps -lq):/opt/esmeta/result ./` copies the result directory from the Docker container to your local machine.

Step-by-Step guide

Running the benchmark

Run `./benchmark.sh` from the root directory (`/opt/esmeta`) to perform the full benchmark. This takes approximately 3 hours depending on hardware. To perform a quick test, use `./benchmark-test.sh`, which runs on just the first three ECMA-262 versions (around 5 minutes).

This process produces results covering ES2024 up to the latest internal ECMA-262 version, for all experimental settings (base, syn, pre, bool, ours, prov) in the `result/` directory. Each contains analyses of 167 versions, summarized in `summary.tsv`. We recommend opening these files in a spreadsheet tool.

Manual Execution

If running a benchmark is too slow, or you just need data for a single version of the ECMA-262, you may want to run a single analysis.

In the interactive shell, run `esmeta tycheck {options}`. Useful options include:

- `-tycheck:detail-log`: Generates detailed logs in opt/esmeta/logs/analyze.
    - `summary.yml` provides the overall analysis summary.
    - `errors` lists the alarms.
    - With `-tycheck:provenance`, a `provenance-logs` file is created, detailing provenance for each refinement point.
- `-extract:target={string}`: Specifies the ECMA-262 version by tag or hash.

Recommend to Check

Please check that those work well:

- Useful options include:
    - `-tycheck:detail-log`: Generates detailed logs in opt/esmeta/logs/analyze.
        - `summary.yml` provides the overall analysis summary.
        - `errors` lists the alarms.
        - With `-tycheck:provenance`, a `/provenance` directory is created, detailing provenance for each refinement point.
    - `-extract:target={string}`: Specifies the ECMA-262 version by tag or hash.

Reusability Guide

This artifact will be merged into the main ESMeta repository and integrated into the CI/CD for ECMA-262. ESMeta is designed to provide consistent type checking for the specification with minimal human intervention, so future ECMA-262 updates should require little to no additional effort.

The implementation of our occurrence typing techniques is mainly in `src/main/scala/esmeta/analyzer/tychecker`: `TypeGuard.scala`, `SymTy.scala`, `AbsTransfer.scala`, `AbsValue.scala`, `Effect.scala`.

These files embody the core of our method and closely align with the paper's formalization.

Possible change after the Revision

Kick-the-tire phase

May require an additional package (Python or GraphViz), but we will set these dependencies on the Docker image accordingly. We expect that reviewers will **not need to make any changes** in this phase.

Full Review

- We anticipate more substantial changes after the revision:
  - A list of confirmed true positives (real bugs), including PRs and logs.
  - Possible minor adjustments to the evaluation targets.
  - Enhanced figures for direct comparison with the paper.
  - For provenance, refined tree visualizations to better illustrate explainability.