Automatic Fuzz Drivers for JavaScript with Type Distributions

Problem

Fuzzing libraries requires handwritten driver code which invokes libary APIs with appropriate test inputs.

Writing effective drivers is difficult and there are large bounties for new ones.

Prior work on automatically generating drivers has focused on statically typed languages, despite JavaScript being an immensely popular language.

Why Is JavaScript Different?

Fuzz driver generation searches for interesting sequences of API calls and interesting inputs for them.

For statically-typed languages the search for inputs is constrained by types. ***This is not the case for JavaScript!***

Learning Types

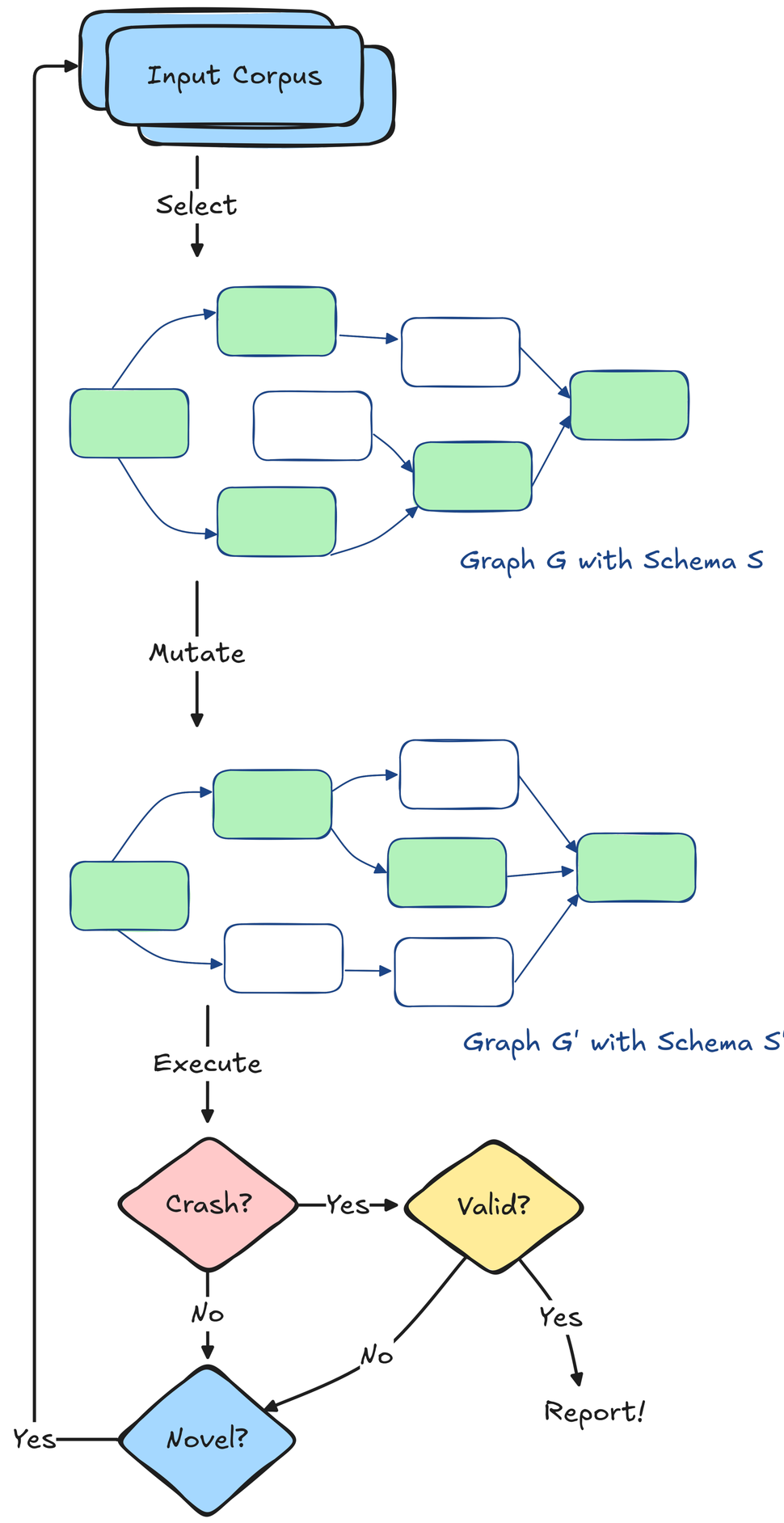
We represent ***likely*** API function signatures with type distributions, i.e., probability of a value being a certain type.

We learn probability weights for these distributions during fuzzing through ***distribution mutations*** and a ***validity oracle***.

Developers often implement run-time input validation for public APIs. Instead of reporting validation errors (false positives), we use them as feedback for our search, in addition to code coverage.

```
if (is.string(input)) { ... }
else if (is.buffer(input)) {
  if(input.length === 0) throw Error()
} else if (Array.isArray(input)) {
  if (input.length > 1) {
    if(this.options.joining) throw Error()
  }
} else throw Error()
```

Code snippet from Sharp, a JavaScript image manipulation library. Sharp accepts inputs of multiple types and performs extensive run-time validation.



Main fuzzing loop. Fuzz drivers are encoded as dataflow graphs and associated with schemas. A schema is a list of likely function signatures for the target library. Executions are considered novel if they uncover a new code path or a path previously only covered by invalid inputs. Input selection from corpus prefers valid inputs.

Early Results

We are evaluating our technique on popular open-source libraries fuzzed as part of the OSS-Fuzz project.

We have already found ***2 bugs*** in code paths unreachable by current handwritten fuzz drivers in OSS-Fuzz.

We are also evaluating how well it fills the gap left by other testing tools.

What about...?

Search-based unit test generation

Prior work requires static analysis of fast evolving JavaScript syntax

TypeScript

Unsound, especially when types and implementation are distributed separately

LLMs

Prior work involves manual specification and labelling of false positives

35 Tu-F ACM SRC

icse2025src-p8

Mayant Mukul
University of British Columbia