CCS 2024

FuzzFlow: Fuzzing JavaScript Engines with a Graph-based IR

Haoran Xu[1], Zhiyuan Jiang[1], Yongjun Wang[1], Shuhui Fan[1],
Shenglin Xu[1], Peidai Xie[1], Shaojing Fu[1], Mathias Payer[2]

[1] 

[2] EPFL

JS engines are everywhere

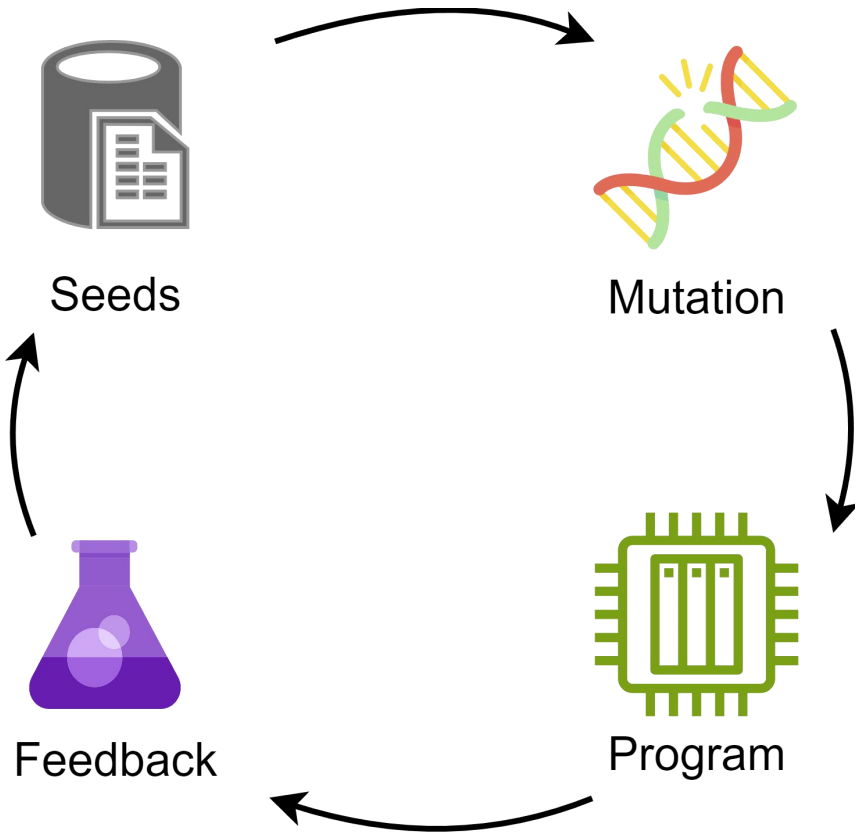
- Web browsers
- Runtime like Deno, NodeJS
- Embedded in software like PDF editor
- Mobile devices

Testing JS engines is challenging

- Extensive codebase (millions of lines of code)
- Highly structured input format
- Sufficient review and fuzzing

Mutation-based Greybox Fuzzing

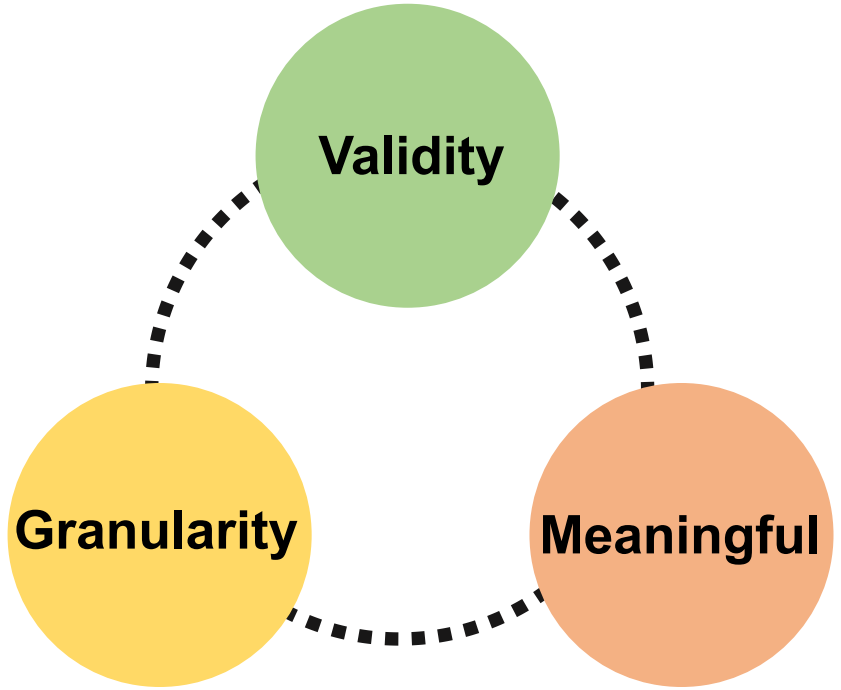
> “For finding vulnerabilities in modern JavaScript engines, especially engines with JIT compilers, better results can be achieved with mutational, coverage-guided approaches.”
>
> -- from Project Zero’s blog



Motivation

- **Requirements**

1. The mutated test case should be valid to pass the early checks

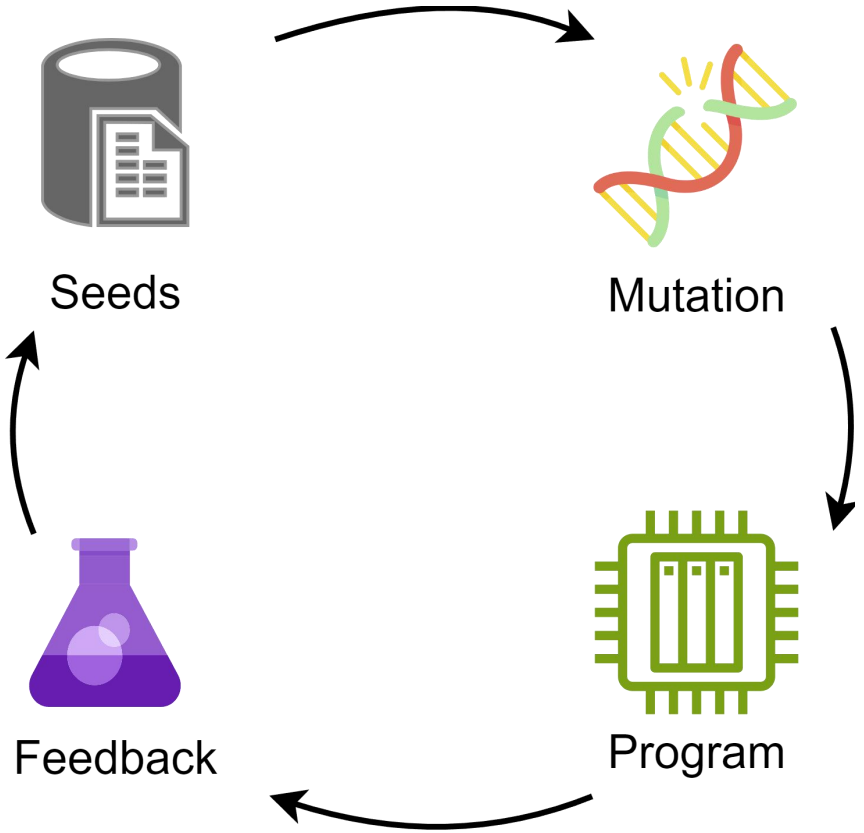


2. Changes only on the syntax will not trigger deep paths

3. Mutations only on control flow or data flow helps to trigger corner paths

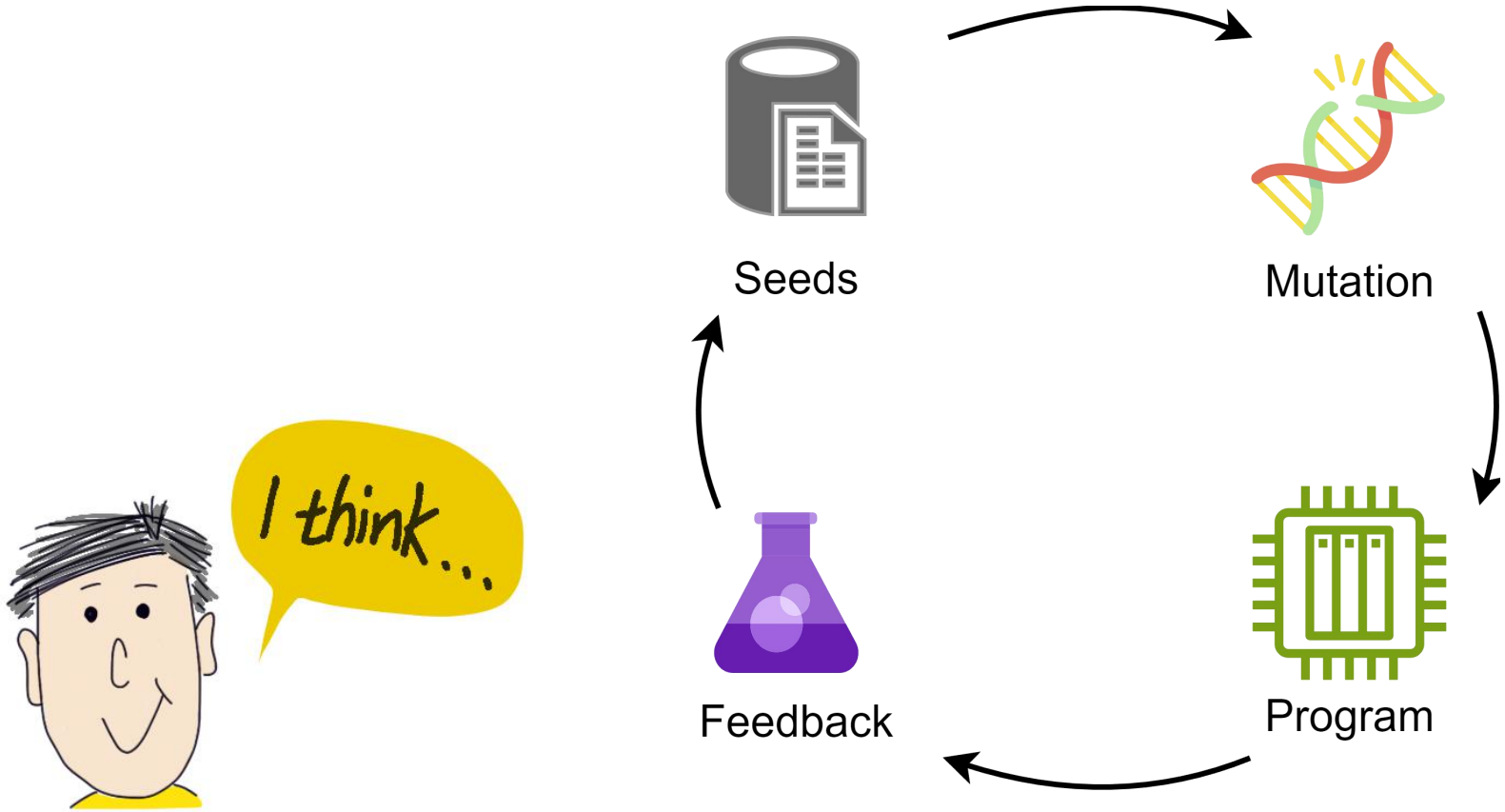
Motivation

- **Rethinking the workflow...**



Motivation

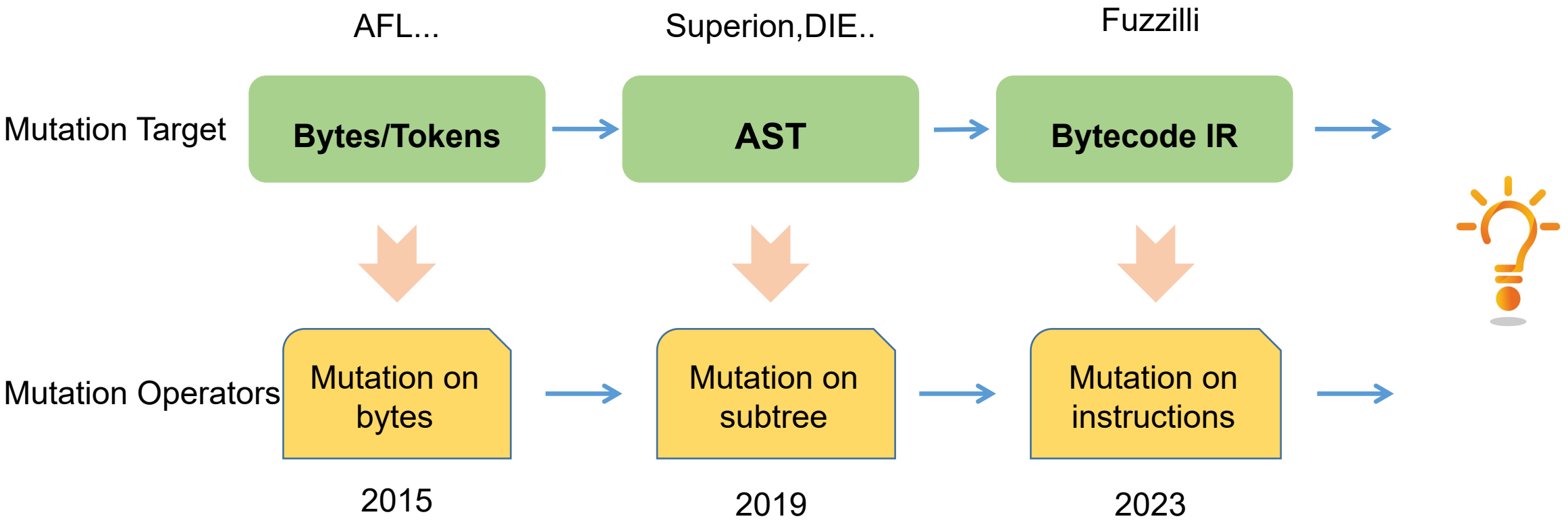
- **Rethinking the workflow...**



The representation of the mutation target is crucial.

Existing Approaches

- **Recap the history of mutation targets**



How to represent a JS program?

Existing Approaches

- **Recap the history of mutation targets**
- AST
  - test cases with semantic errors that cannot reach the backend
  - test cases with altered syntax but unchanged semantics
- Bytecode IR
  - lacks explicit control and data flow

Our Methoad: FuzzFlow

- **How to mutate the semantics directly?**

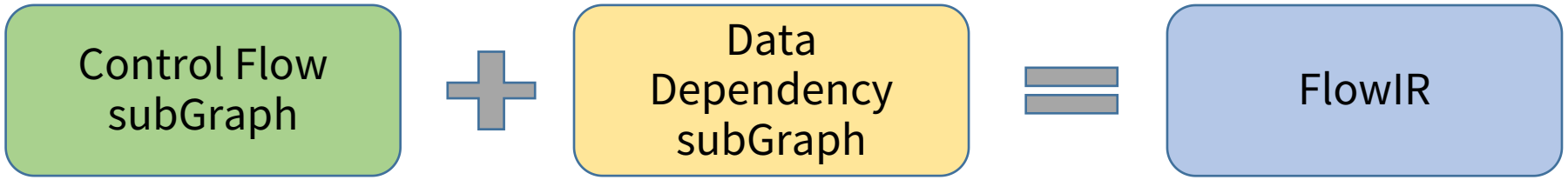
Source Code	Mutations on token/strings Token insertions, replacements...
Abstract Syntax Tree	Mutations on subtree Subtree insertions, replacements...
Bytecode IR	Mutations on instructions Instruction insertions, replacements...
Graph IR	Mutations on control and data flow Node/edge insertions, replacements...

Syntax

Semantics

FlowIR: Mutate the Semantics Directly

- A directed node-labeled, and edge unlabeled graph.
- The CFG represents the partial order on the operations.
- The DDG represents the flow of values from definition of a var to its uses.

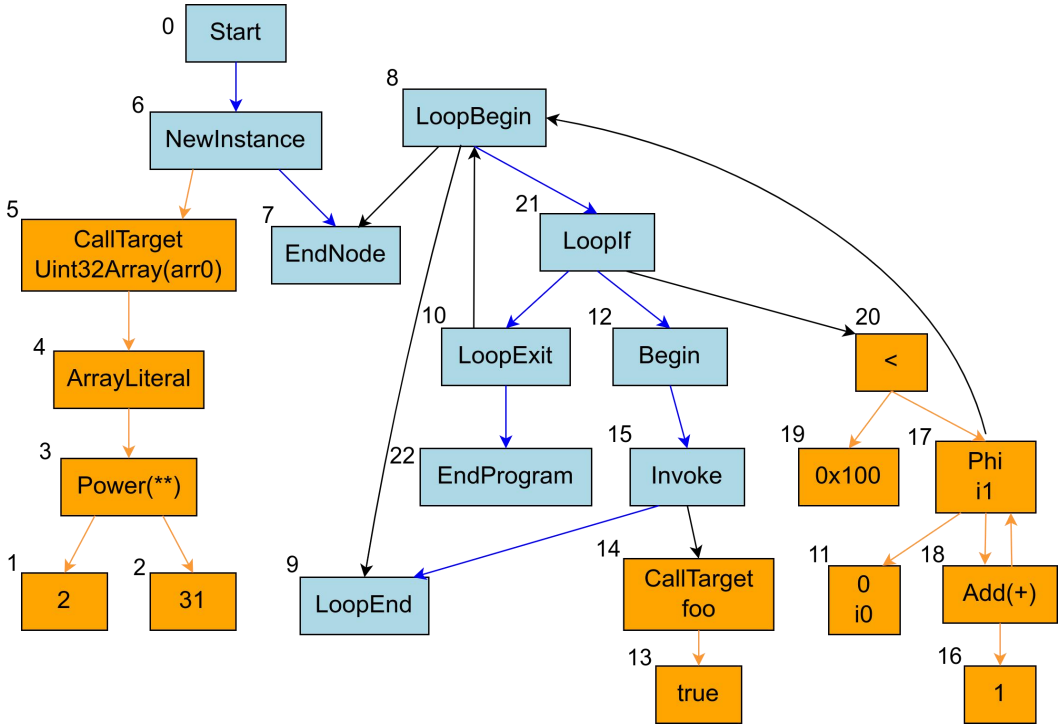


FlowIR: Mutate the Semantics Directly

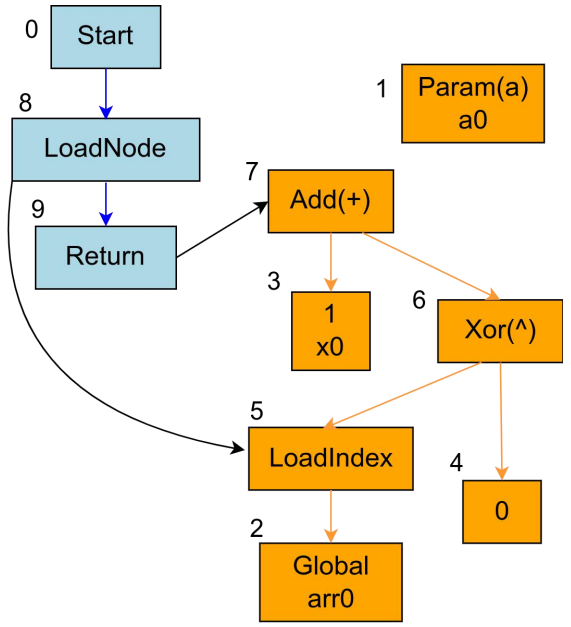
- An example of FlowIR

```
arr = new Uint32Array([2**31]);

function foo(a) {
    var x = 1;
    x = (arr[0] ^ 0) + 1;
    return x;
}

for (let i=0; i<0x100; ++i)
{ foo(true); }
```

A test case triggered
CVE-2021-21220



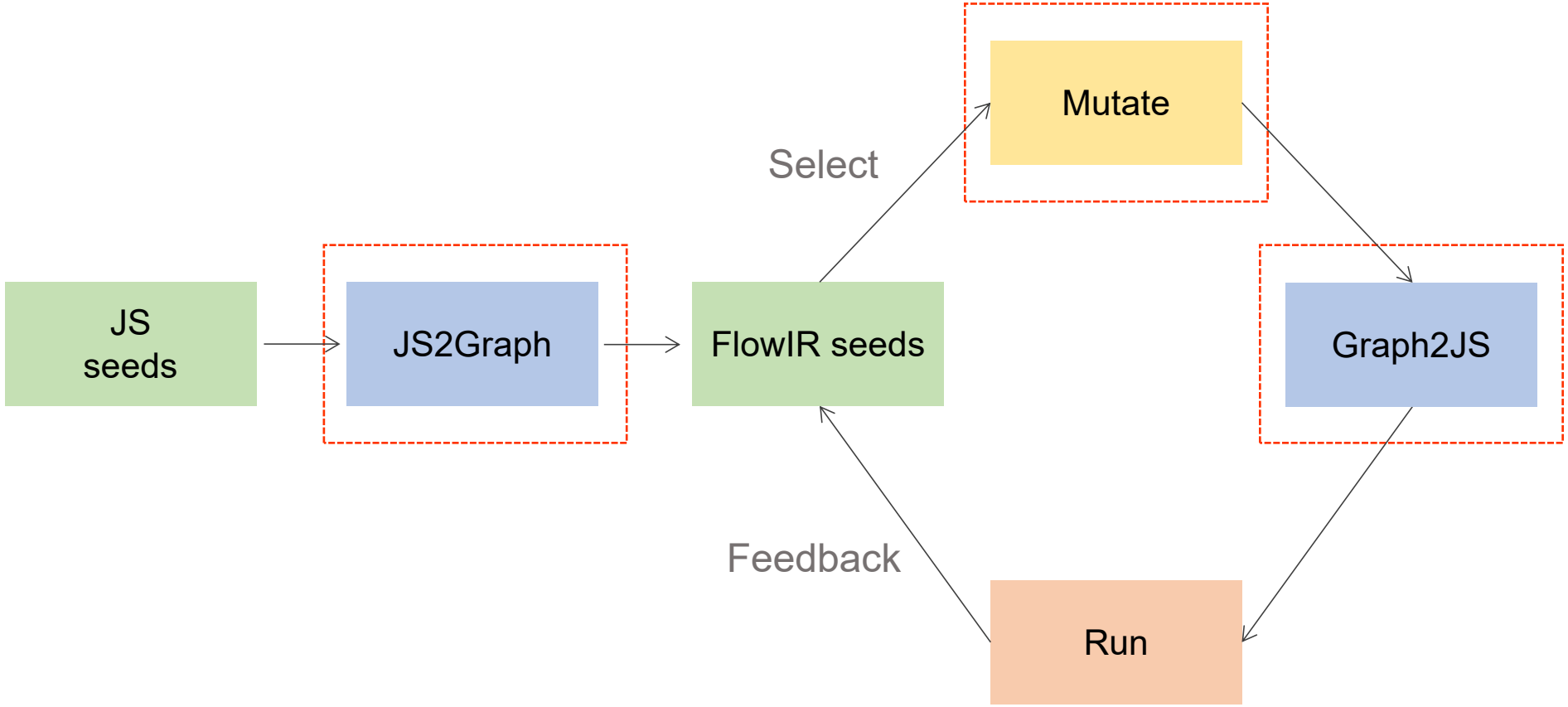
FlowIR of Main script



FlowIR of Funciton foo

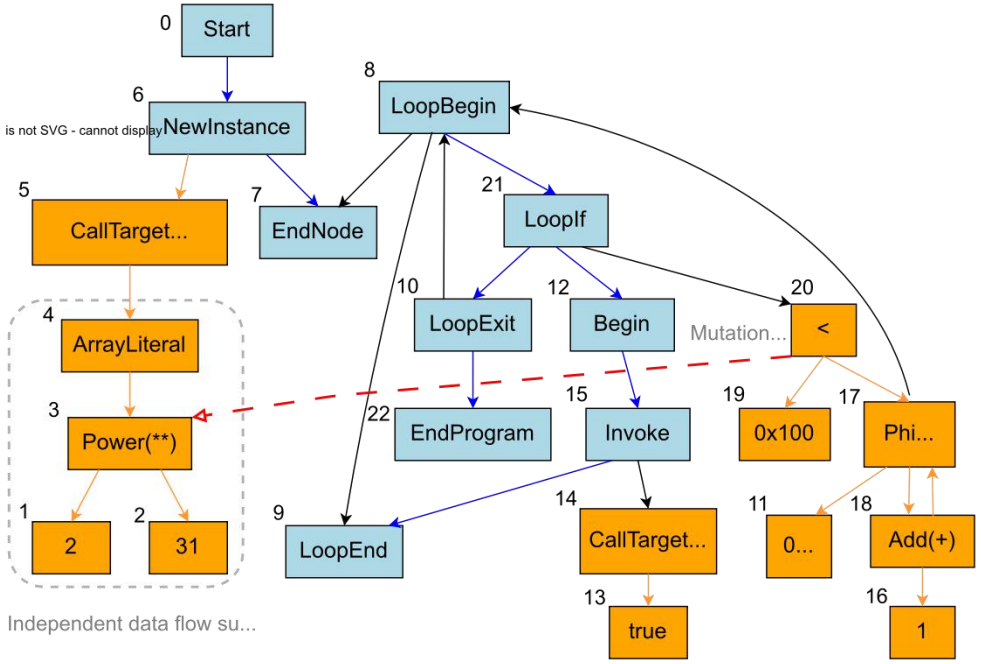
FuzzFlow Design

Overview



Mutation on Graph

- Mutations on DDG
  - Node attribute mutation
  - Node input mutation
  - Splicing independent data-flow subgraphs
  - ...
- Mutations on CFG
  - Node movement
  - Nodes group relocation
  - Branch swapping
  - ...



Evaluation

- Better code coverage.
- Better validity.
- Better mutation granularity.
- 37 new bugs found in mainstream JS engines.

Conclusion

- Fuzzing JS engines is challenging.
- The choice of the underlying representation defines the possible mutation space and subsequently influences the design of mutation operators.
- FlowIR provides a graph-based mutation target for testing JS engines.
- The representation of the mutation target deserves more attention.