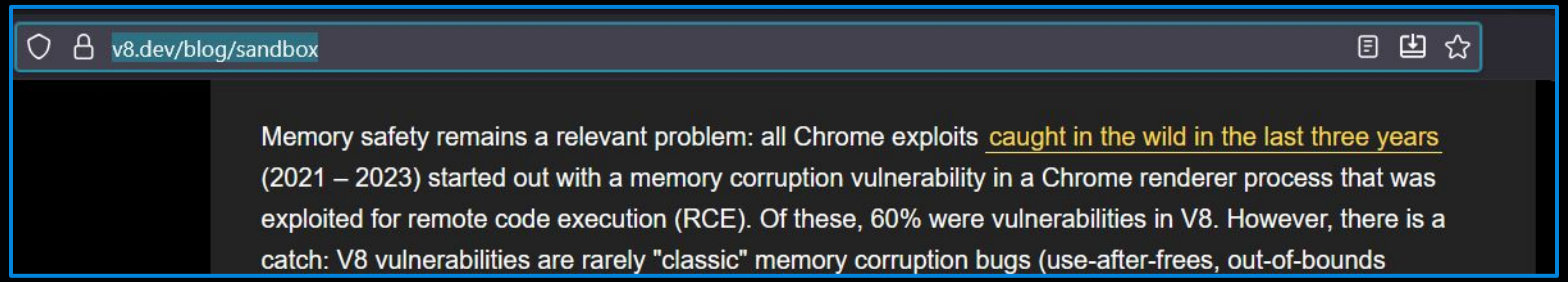
Validating the JS Rosetta Stone

Group 5
Chase Killorin and Deepak Kurian

Introduction

- Chrome, Edge, Opera use V8(Javascript Compiler)

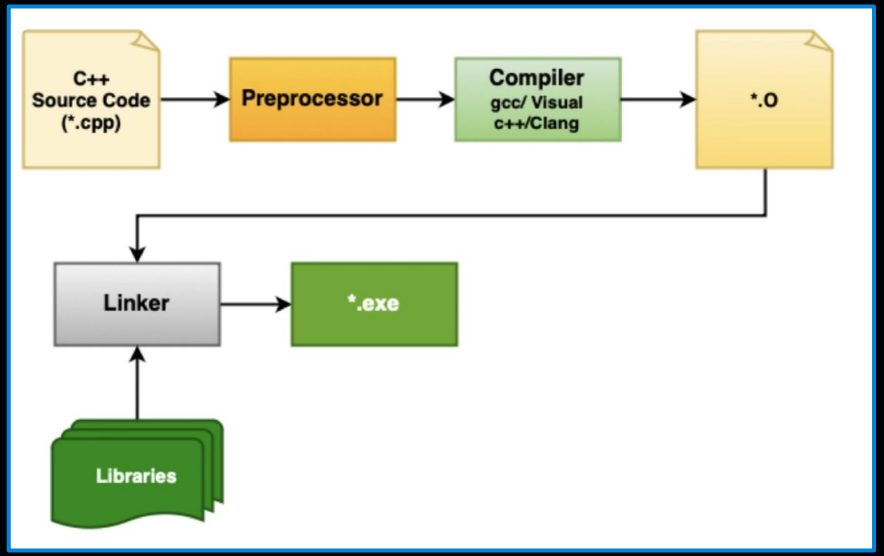


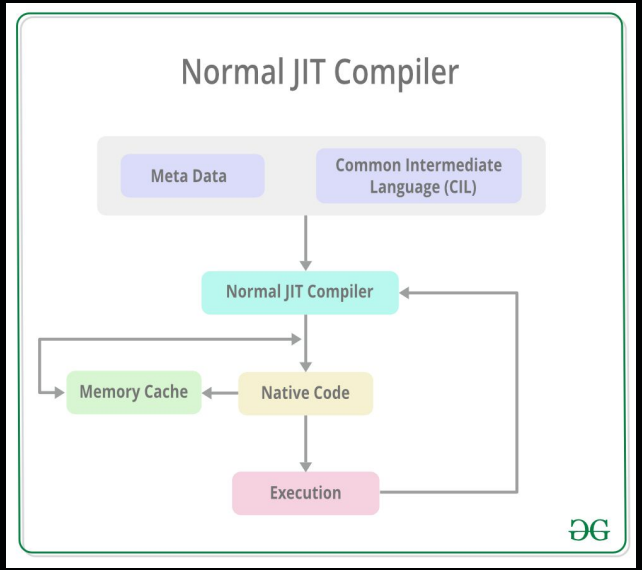
Introduction

- Research Problem
  - Improve V8 JIT security by improving existing tools (Turbo-TV)
- Research Objectives
  - Supporting Additional Opcodes in Turbo-TV
  - Adding support for the function inlining optimization

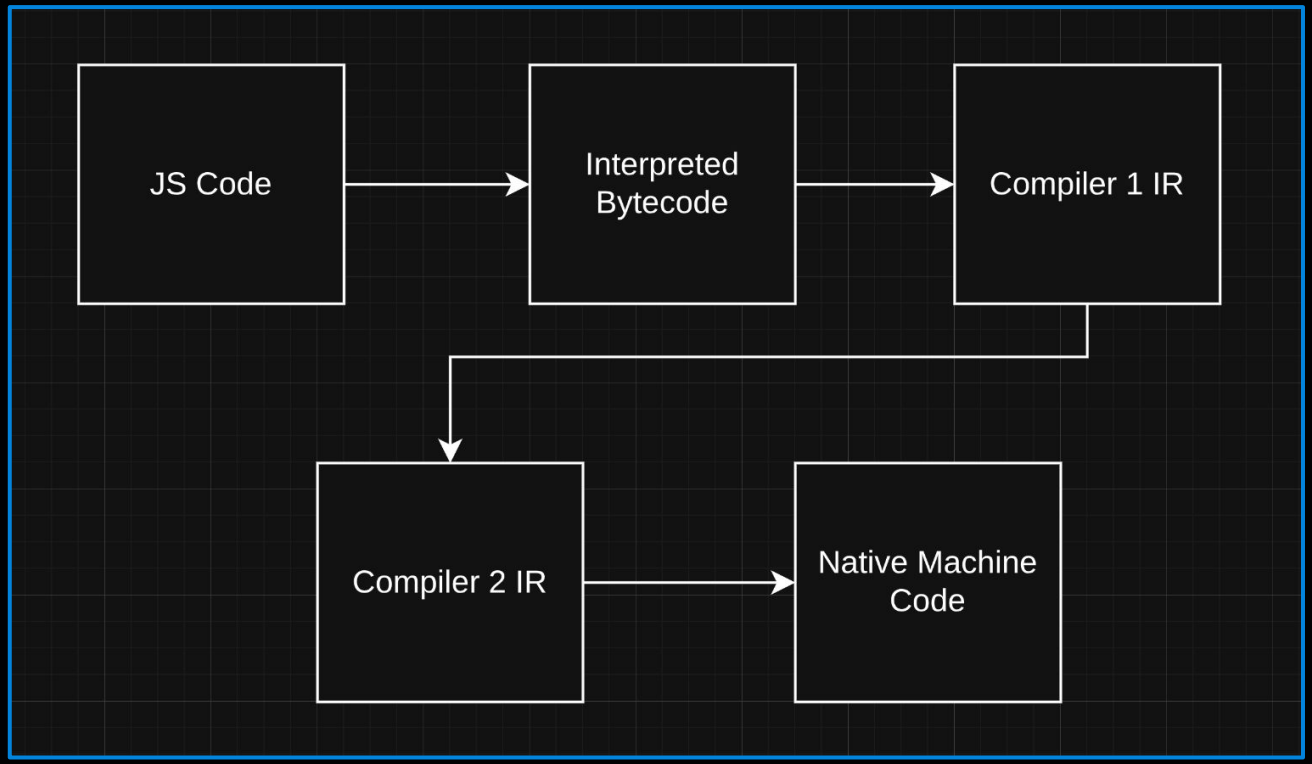
Background

- AOT (Ahead of Time) vs JIT (Just in Time) compilation





Background


JS Code
Interpreted Bytecode
Compiler 1 IR
Compiler 2 IR
Native Machine Code

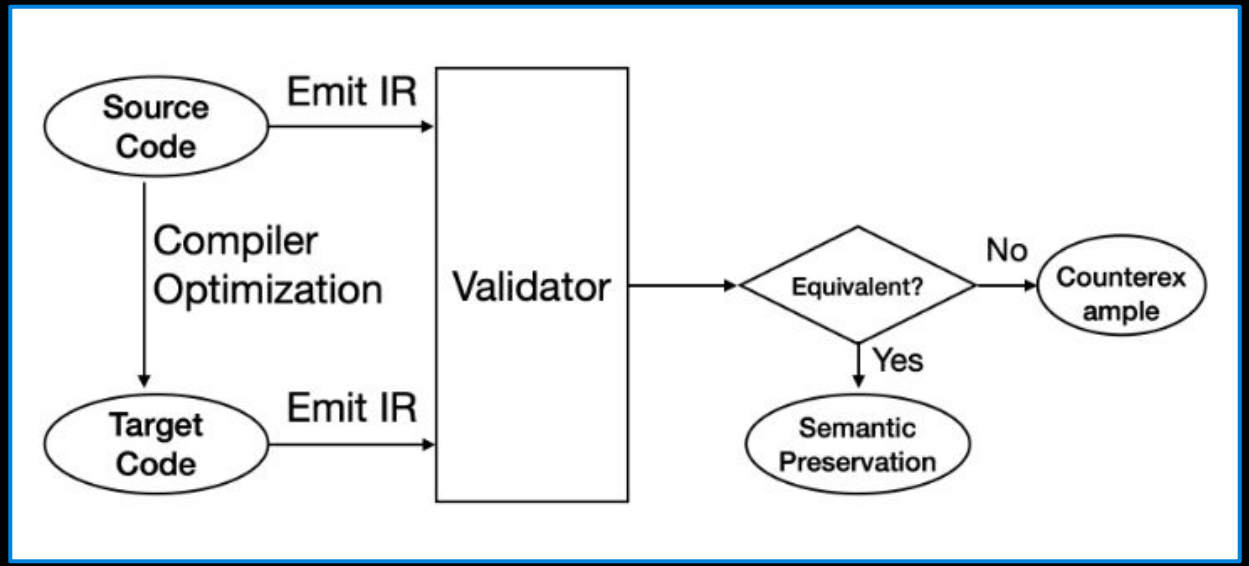
Background - Baseline

Turbo-TV - Translation Validation for JIT Compiler in the V8 JavaScript Engine

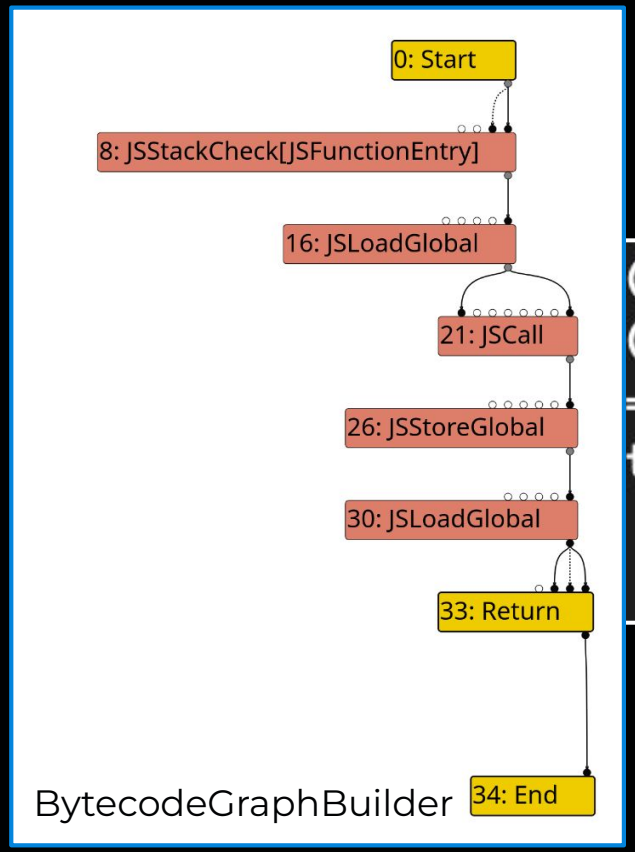
Kwon, S., Kwon, J., Kang, W., Lee, J., & Heo, K. (2024). Translation Validation for JIT Compiler in the V8 JavaScript Engine. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. Association for Computing Machinery.

Background

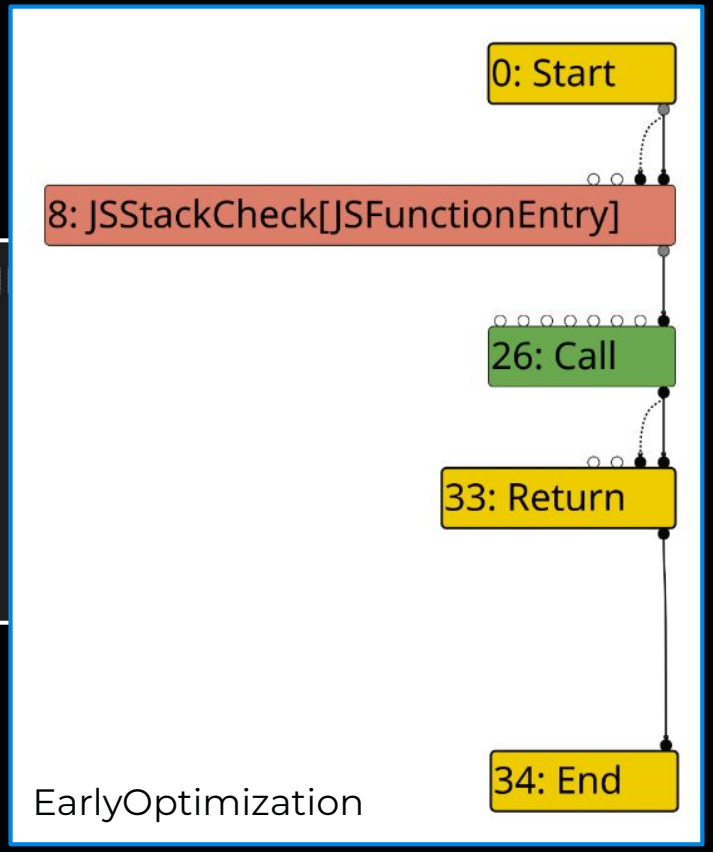
- Translation Validation



Background - Semantic equivalency

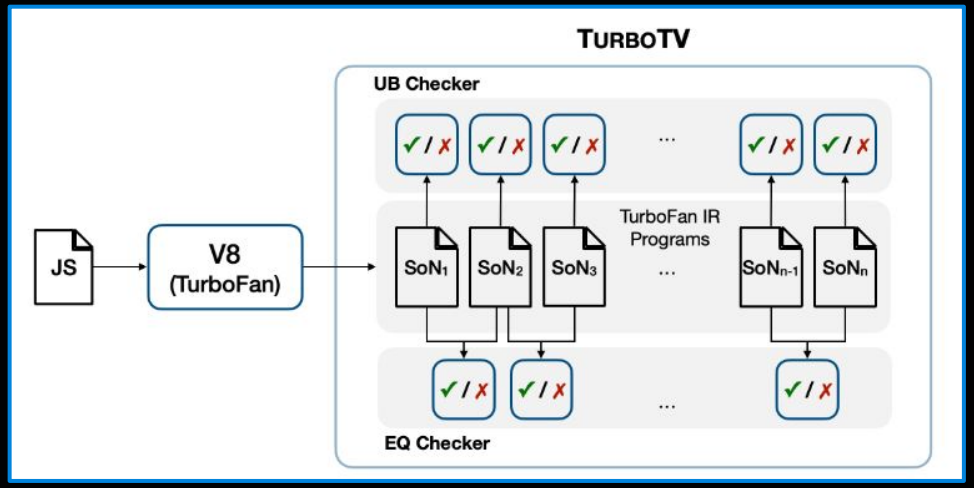



```
(x) { retu
(x) {
= x+b(x);
turn y;
```



Background - Turbo-TV Basics

- Undefined Behaviour Checker
- Equivalency checker (Our focus)



Related Work

- Towards a Verified Range Analysis for JavaScript JITs (VeRA)
  - Partial Verification
  - Implemented inside the compiler
- Alive2: Bounded Translation Validation for LLVM
  - Applied to AOT compilation
  - Creates a formalization of intended semantics

Contribution - Turbo TV

- Identifies JIT semantic bugs AOT
- Applies Translation Validation against V8 JIT
- TurboTV constructs a novel adaptation of its prior works
- Difficulty: No formal semantics for SoN IR

Contribution - Turbo TV

- Integrates equivalency checks into Fuzzilli
- Checks don’t support everything Fuzzilli can generate

Fuzzilli

A (coverage-)guided fuzzer for dynamic language interpreters based on a custom intermediate language ("FuzzIL") which can be mutated and translated to JavaScript.

Contribution - Turbo TV

Internal State Representation

$$
\begin{aligned}
\mathit{State} &= \mathit{Label} \times \mathit{RegFile} \times \mathit{Memory} \times \mathit{Deopt} \times \mathit{UB} \\
\mathit{RegFile} &= \mathit{Register} \to \mathit{Value} \\
\mathit{JSValue} &= \mathit{TaggedPointer} \uplus \mathit{TaggedSigned} \\
\mathit{Value} &= \mathit{JSValue} \uplus \mathit{Bool} \uplus \mathit{Int8} \uplus \mathit{Int16} \uplus \mathit{Int32} \uplus \cdots \\
\mathit{TaggedPointer} &= \mathit{BlockID} \times \mathit{Int32} \\
\mathit{TaggedSigned} &= \mathit{Int31} \\
\mathit{Memory} &= \mathit{BlockID} \to \mathit{Block} \\
\mathit{Block} &= \mathit{Byte}^*
\end{aligned}
$$

Methodology

- Lowering Javascript to IR

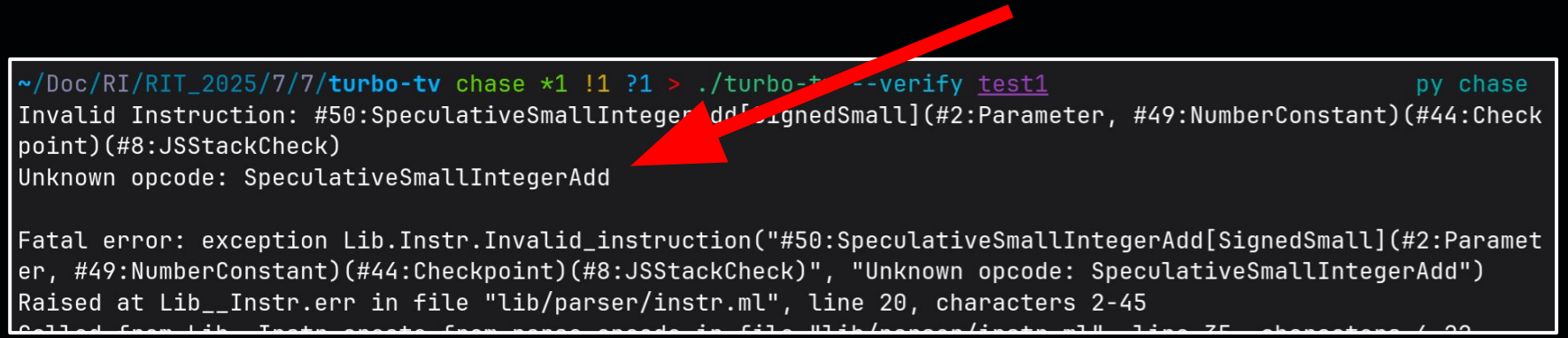
```
turbo@turbotv ~/t/test-cases (dev)> ./extract_ir.py test2.js "a" .
Running d8 with trace-turbo for function: a
Converting IR format to turbo-tv format...

Extracted IR files:
  src.ir (before optimization): ./src.ir
  tgt.ir (after optimization): ./tgt.ir

Converted
```

Methodology

- Assessing unsupported opcodes



Methodology

- Unimplemented opcodes

Methodology

- SpeculativeSmallIntegerAdd

```
(* Speculative smallInteger(smi) operations *)
'a -> E.expr -> E.expr -> 'b -> E.expr -> State.t -> State.t
let speculative_smi_add _hint lval rval _eff control state =
  checked_int32_add lval rval _eff control state
```

```
#50:SpeculativeSmallIntegerAdd[SignedSmall](#2:Parameter, #49:NumberConstant)(#44:Checkpoint)(#8:JSStackCheck)
```

Methodology

- JSStoreGlobal

```
| JSStoreGlobal ->
    (* B2V1V2E1C1: bracket operands (sloppy mode, globa
    let b1 = "sloppy" in
    let b2 = Operands.const_of_nth operands 0 in
    let v1_id = Operands.id_of_nth operands 1 in
    let v1 = RegisterFile.find v1_id rf in
    let v2_id = Operands.id_of_nth operands 2 in
    let v2 = RegisterFile.find v2_id rf in
    let _eid = Operands.id_of_nth operands 3 in
    let cid = Operands.id_of_nth operands 4 in
    let ctrl = ControlFile.find cid cf in
    js_store_global b1 b2 v1 v2 () ctrl
```

```
#26:JSStoreGlobal[sloppy, 0x38750000364d <String[1]: #y>](#25:SpeculativeSmallIntegerAdd
 #5:HeapConstant, #4:Parameter, #27:FrameState, #25:SpeculativeSmallIntegerAdd)(#8:JSSta
kCheck)(#8:JSStackCheck)
#33:Return(#32:NumberConstant, #58:NumberConstant)(#26:JSStoreGlobal)(#26:JSStoreGlobal)
```

Methodology

- Changes

```
'a -> 'b -> E.expr -> E.expr -> 'c -> E.expr -> State.t -> State.t
let js_store_global _b1 _b2 v1 v2 _eff control state =
  let mem = state.State.memory in
  let raw_ptr = TaggedPointer.to_raw_pointer v2 in
  let mem = Memory.Bytes.store Bool.tr raw_ptr 8 v1 mem in
  state |> State.update ~control ~mem
```

Results

- Support for:
    - JSStoreGlobal
    - SpeculativeSmallIntegerAdd

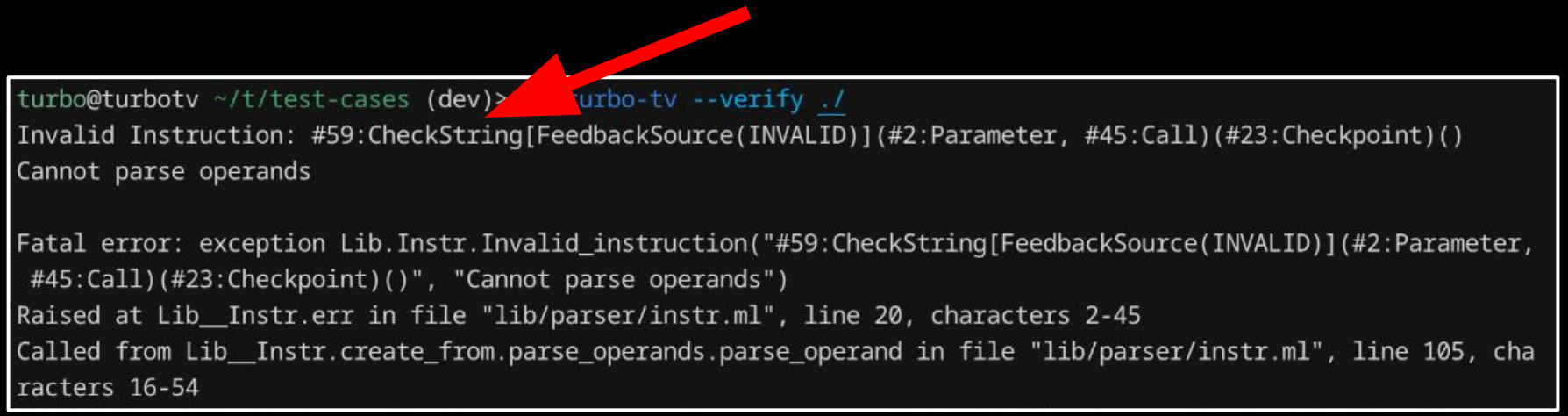
```
turbo@turbotv ~/turbo-tv (dev) [2]> ./turbo-tv --verify test-cases/
Result: Verified
```

Results - To be implemented

- Continue to add support for more instructions.
- Ultimately support JSCall for base case of function inlining.

Future Directions

- More opcode support ~> More coverage



Conclusion

- Increase coverage ~> More bugs
- Improved semantic knowledge on certain key Opcodes
  - (JSStoreGlobal)
  - Aids equivalency checks