ORACLE

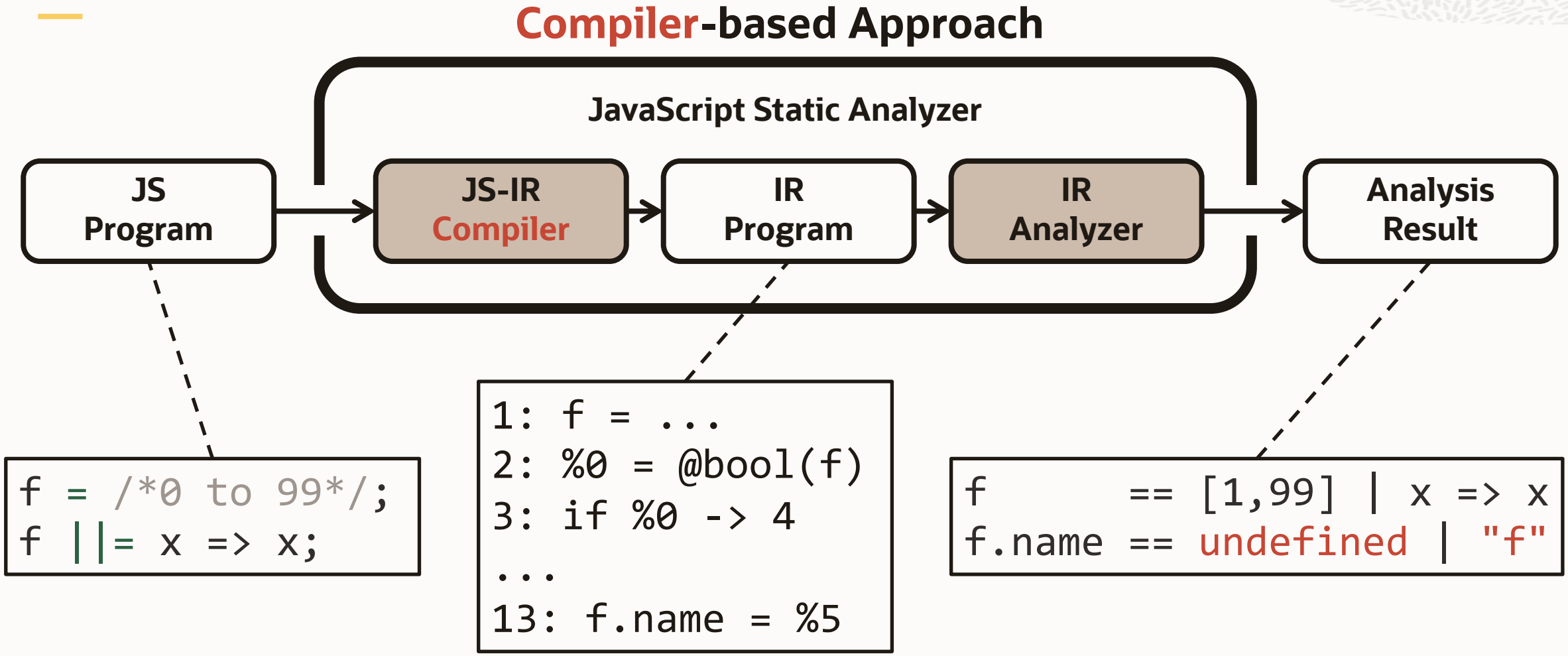
Automatically Deriving JavaScript Static Analyzers from Specifications using Meta-level Static Analysis

Jihyeok Park [1], Seungmin An [2], and Sukyoung Ryu [2]

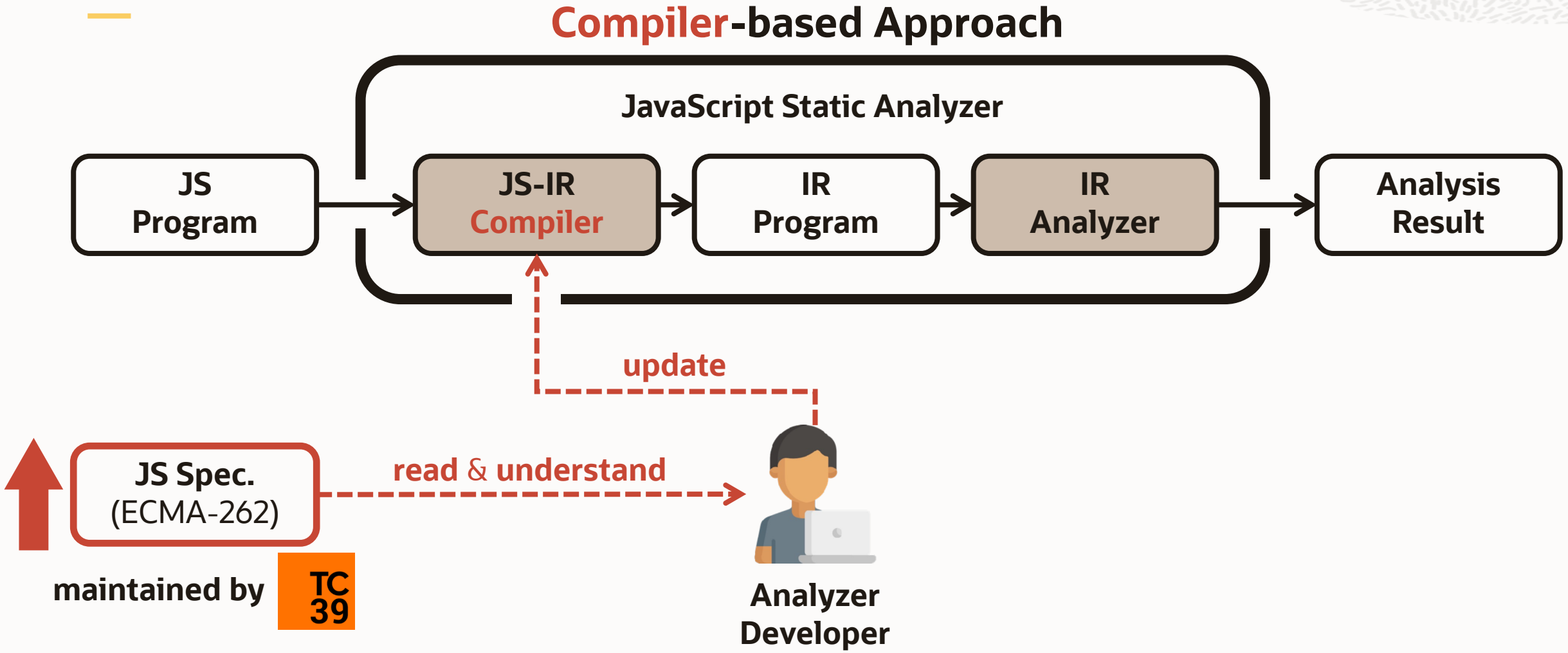
[1] Oracle Labs, Australia

[2] KAIST, South Korea

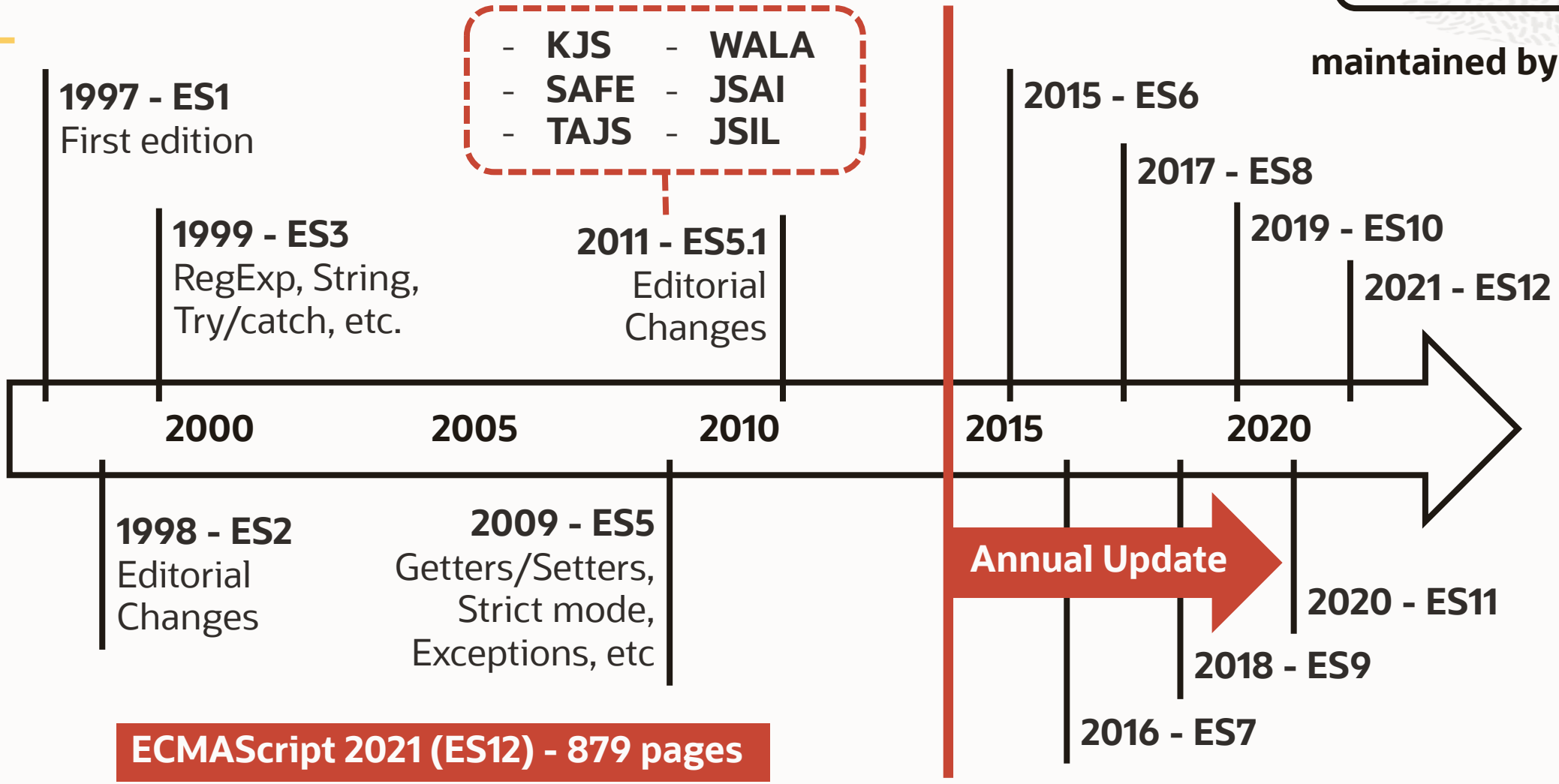
Background - JavaScript Static Analysis



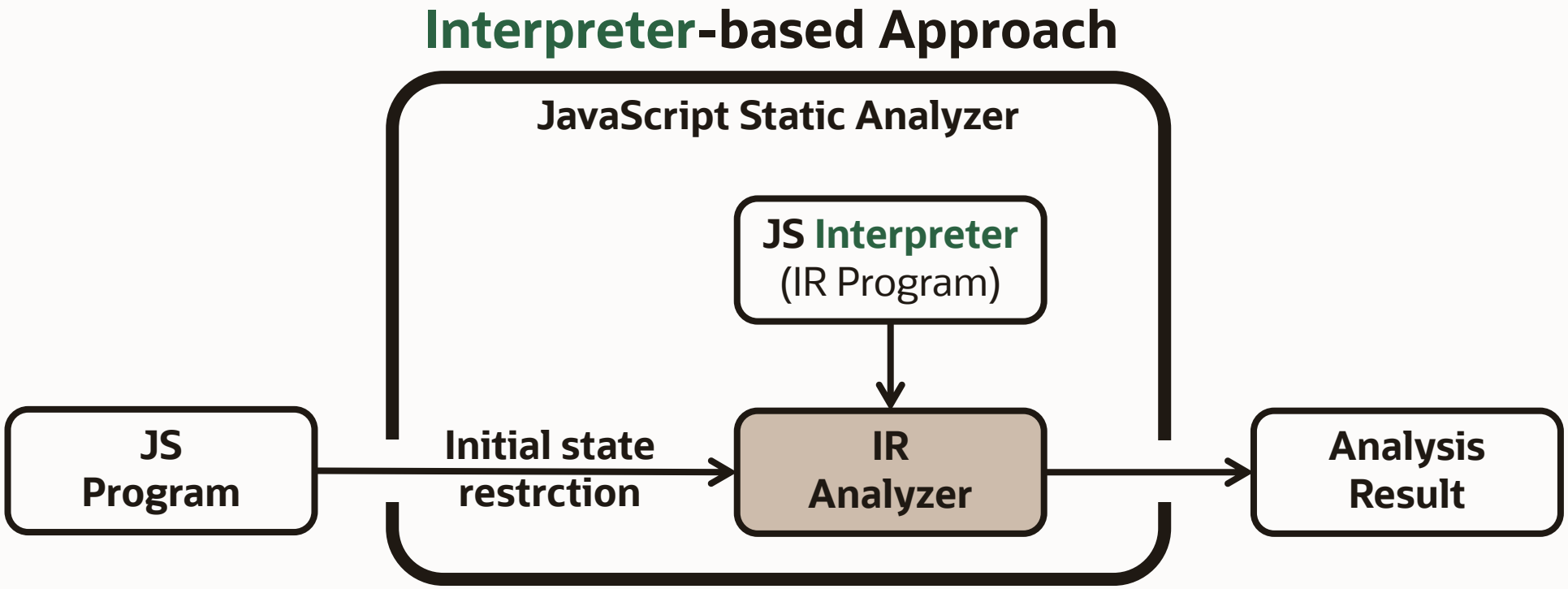
Problem - Manual Update of JS-IR Compiler



Problem - Fast Evolving JavaScript



Core Idea - Meta-level Static Analysis



Core Idea - Meta-level Static Analysis

- **Why Interpreter-based Approach?**
  - JavaScript specifications are written in an interpreter-based style

13.15.2 Runtime Semantics: Evaluation

AssignmentExpression : *LeftHandSideExpression* `||=` *AssignmentExpression*

1. Let *lref* be the result of evaluating *LeftHandSideExpression*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *lbool* be ! ToBoolean(*lval*).
4. If *lbool* is **true**, return *lval*.
5. If IsAnonymousFunctionDefinition(*AssignmentExpression*) is **true** and IsIdentifierRef of *LeftHandSideExpression* is **true**, then
   a. Let *rval* be NamedEvaluation of *AssignmentExpression* with argument *lref*.[[ReferencedName]].
6. Else,
   a. Let *rref* be the result of evaluating *AssignmentExpression*.
   b. Let *rval* be ? GetValue(*rref*).
7. Perform ? PutValue(*lref*, *rval*).
8. Return *rval*.

Evaluation **algorithm** for *logical OR assignments* in **ES12 (ES2021)**

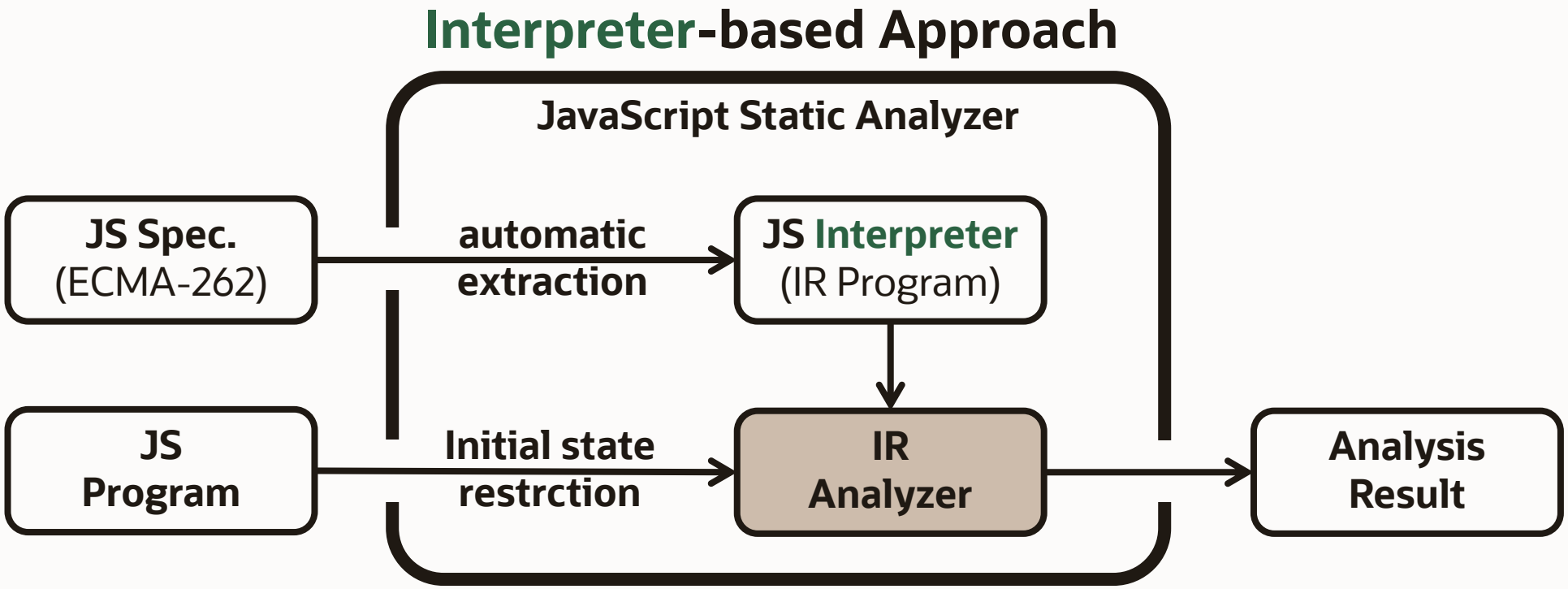
Core Idea - Meta-level Static Analysis

- **Why Interpreter-based Approach?**
  - JavaScript specifications are written in an interpreter-based style
  - **JISET**: JavaScript IR-based Semantics Extraction **(ASE 2020)**
    - Extracting JavaScript **definitional interpreters** as **IR_{ES} programs** from JS Lang. Spec. (ECMA-262).

```
1  syntax def AssignmentExpression[8].Evaluation(
2    this, LeftHandSideExpression, AssignmentExpression
3  ) { /* entry */
4    let lref = (LeftHandSideExpression.Evaluation)
5    let lval = [? (GetValue lref)]
6    let lbool = [! (ToBoolean lval)] /* #1 */
7    if (= lbool true) { /* #2 */ return lval } else {} /* #3 */
8    if (&& (IsAnonymousFunctionDefinition AssignmentExpression)
9        (LeftHandSideExpression.IsIdentifierRef)) { /* #4 */
10     let rval = (AssignmentExpression.NamedEvaluation
11                 lref.ReferencedName)
12   } else { /* #5 */
13     let rref = (AssignmentExpression.Evaluation)
14     let rval = [? (GetValue rref)]
15   } /* #6 */
16   [? (PutValue lref rval)]
17   return rval
18 } /* exit */
```

Extracted **IR_{ES} function** for *logical OR assignments* via **JISET**

Core Idea - Meta-level Static Analysis

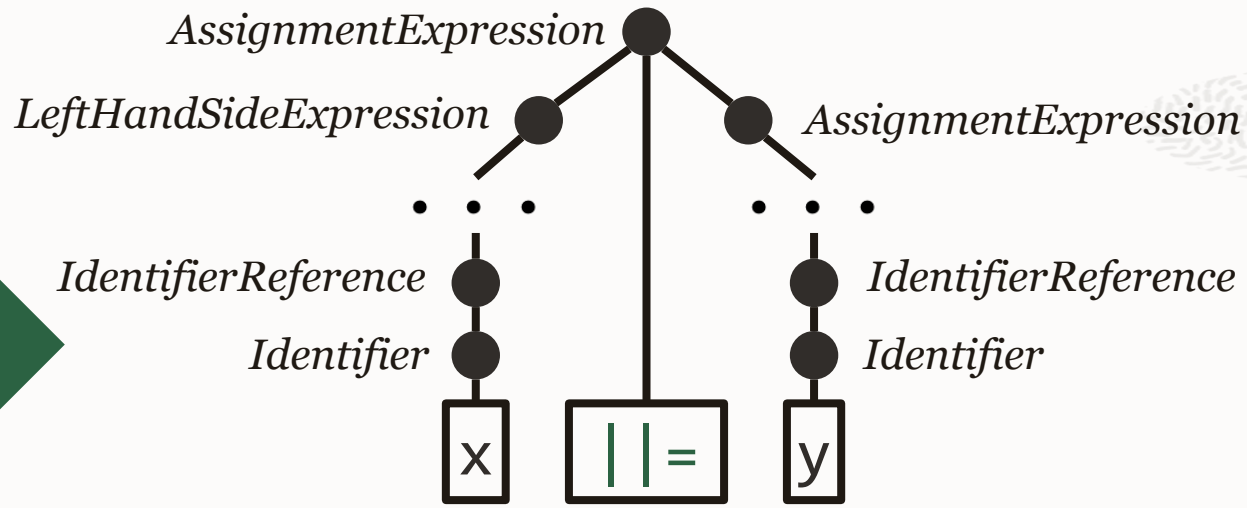


Example

```
x ||= y
```

JavaScript

PARSE



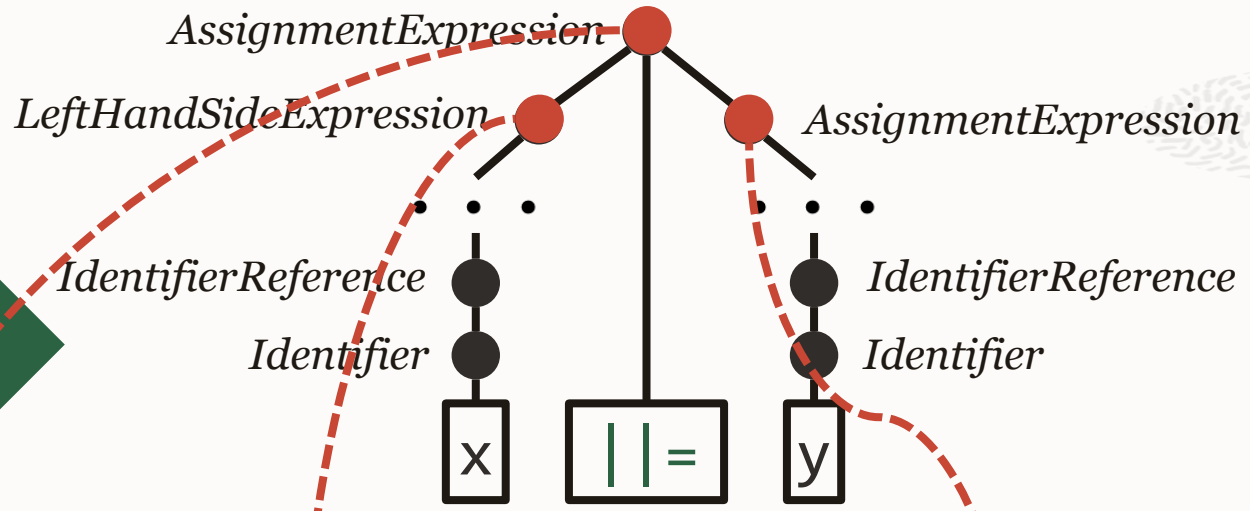
IR$_{ES}$

```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

Example

`x ||= y`

PARSE



JavaScript

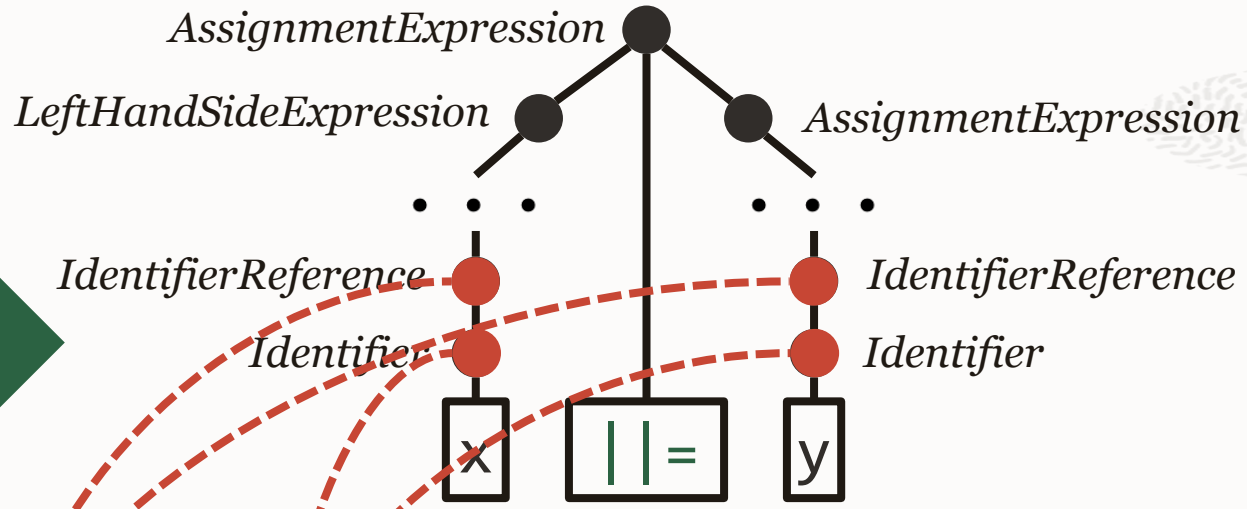
IR$_{ES}$

```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

AST Sensitivity

```
x ||= y
```

PARSE

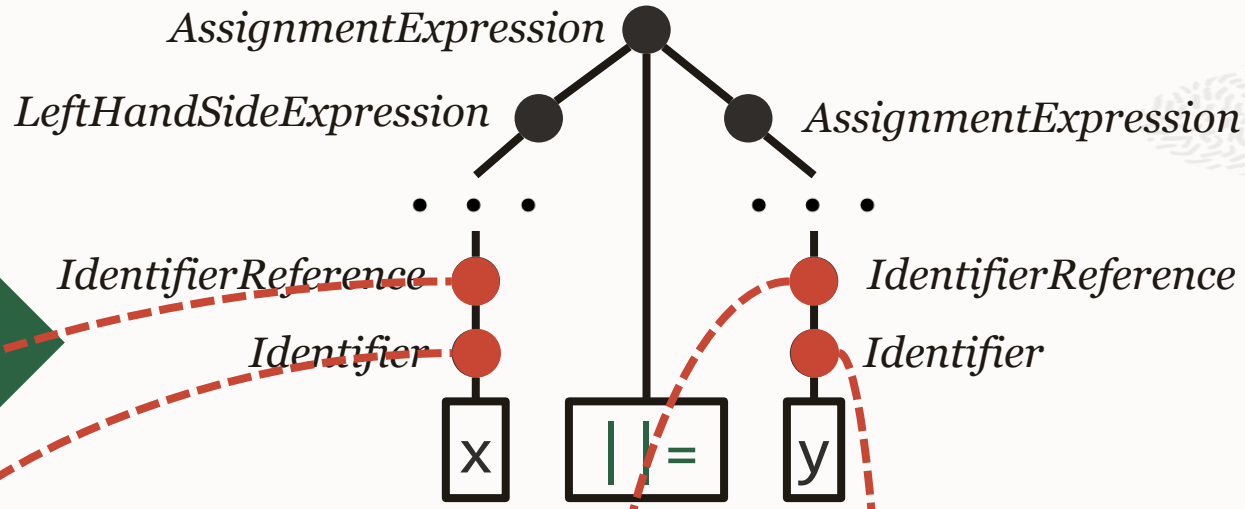


JavaScript

IR$_{ES}$

```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

AST Sensitivity




```
x ||= y
```

PARSE

JavaScript

IR$_{ES}$

```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

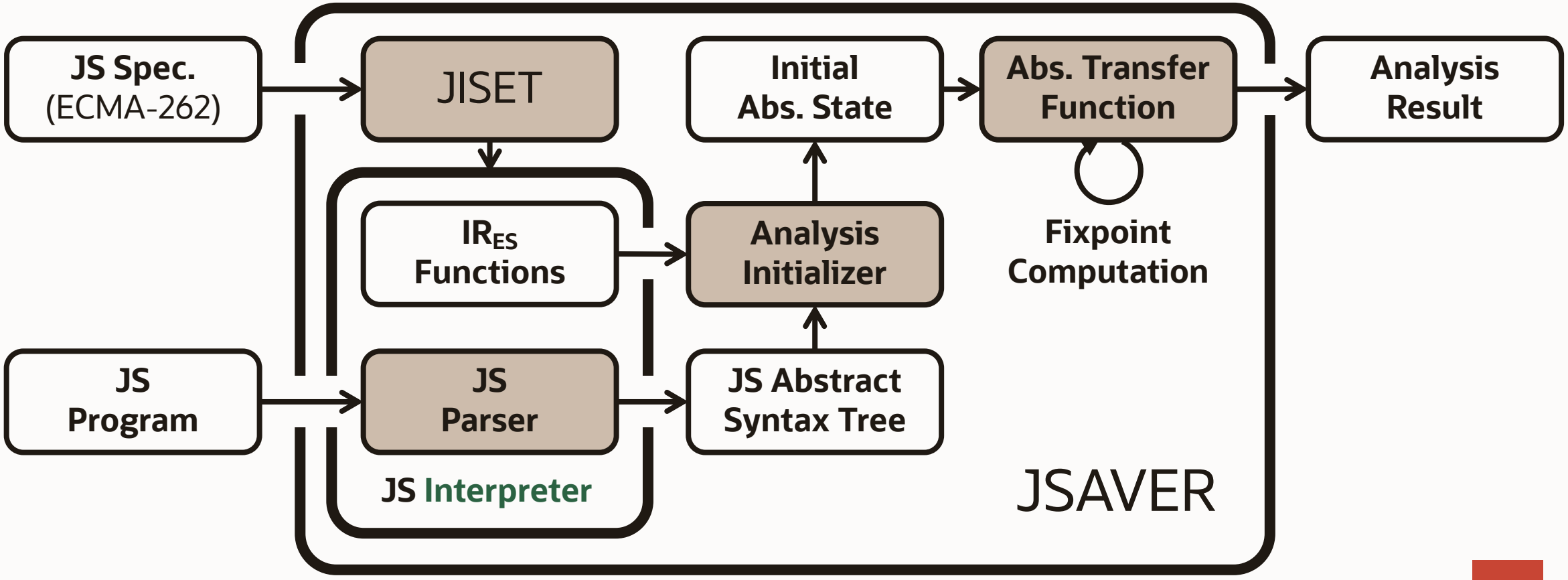
```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

AST Sensitivity

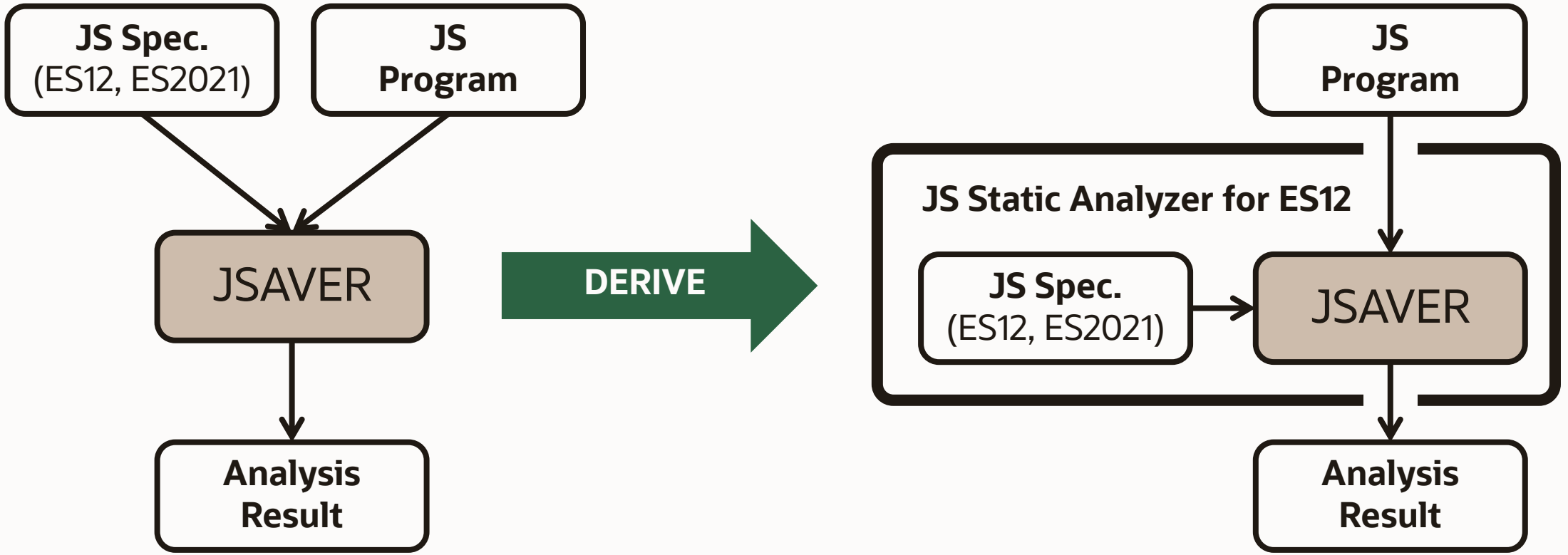
JavaScript	AST Sensitivity in IR_{ES}
Flow-Sensitivity	$\delta^{\text{flow}}(t \in \mathbb{T}) = \{\sigma = (\_, c) \in \mathbb{S} \mid \text{syntax-ctxt}(c) = c' \wedge c'(\texttt{this}) = t\}$
k-Callsite-Sensitivity	$\delta^{k\text{-cfa}}(\bar{t} \in \mathbb{T}^{\leq k}) = \{\sigma = (\_, c) \in \mathbb{S} \mid \bar{t} = [t_1, \cdots, t_n] \wedge n \leq k \wedge (\nexists(\text{syntax-ctxt} \circ \text{js-call-ctxt})^{n+1}(c) \vee n = k) \wedge \forall 1 \leq i \leq n. ((\text{syntax-ctxt} \circ \text{js-call-ctxt})^{i}(c) = c_i \wedge c_i(\texttt{this}) = t_i)\}$

Our Tool - JSAVER

- **J**avaScript **S**tatic **A**nalyzer **v**ia **E**CMAScript **R**epresentation



JS Static Analyzer Derivation via JSAVER

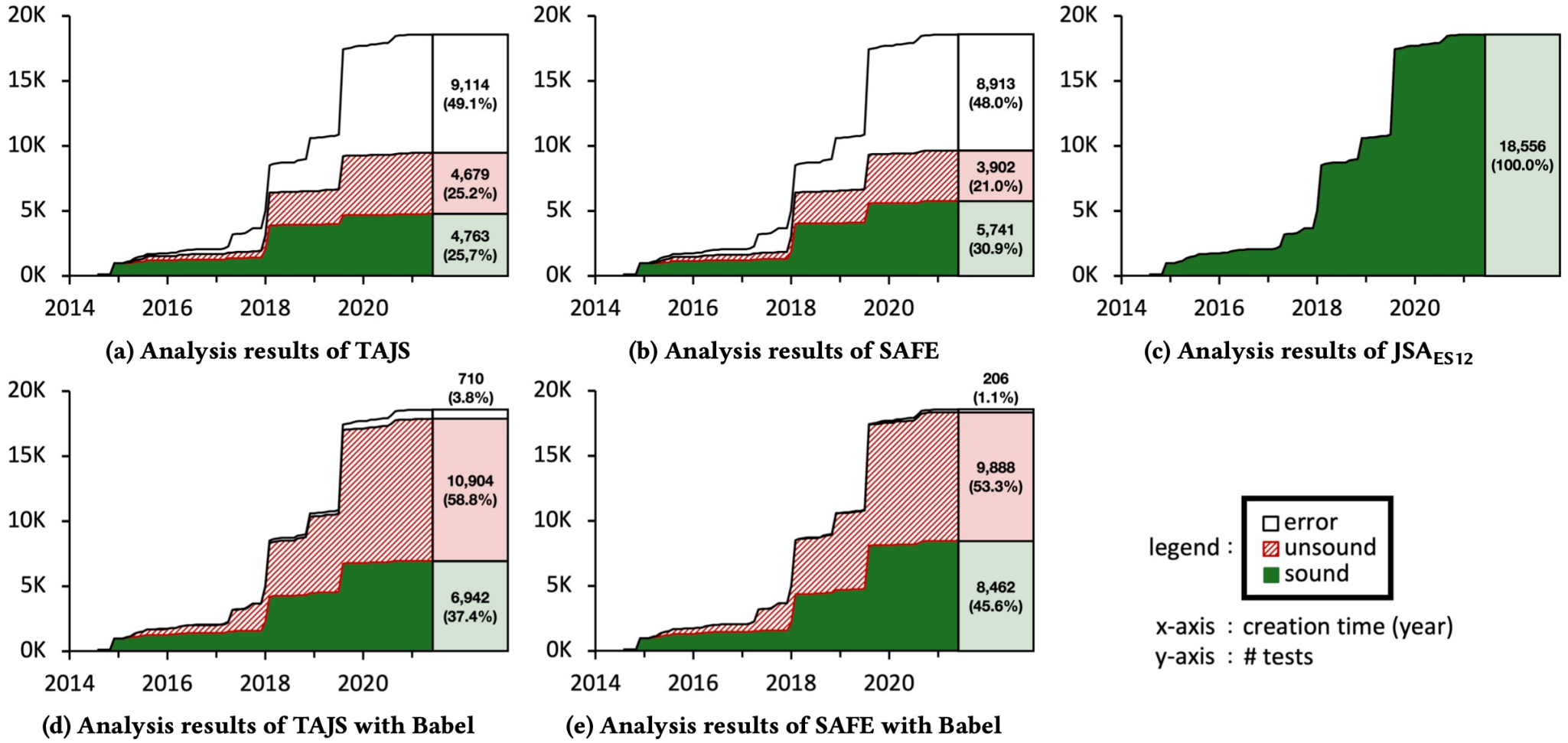


Evaluation Setting

- **Derived Analyzer - JSA_{ES12}**
  - JavaScript Static Analyzer derived from **ES12** (ES2021) via **JSAVER**
- **Comparison Targets**
  - State-of-the-art JavaScript Static Analyzers
    - TAJS / SAFE
- **Analysis Targets**
  - **Test262** (Official Conformance Test Suite) maintained by TC39
    - Used 18,556 applicable conformance tests
- **Experiment Environment**
  - An Ubuntu machine
    - 4.2GHz Quad-Core Intel Core i7 and 32GB of RAM.

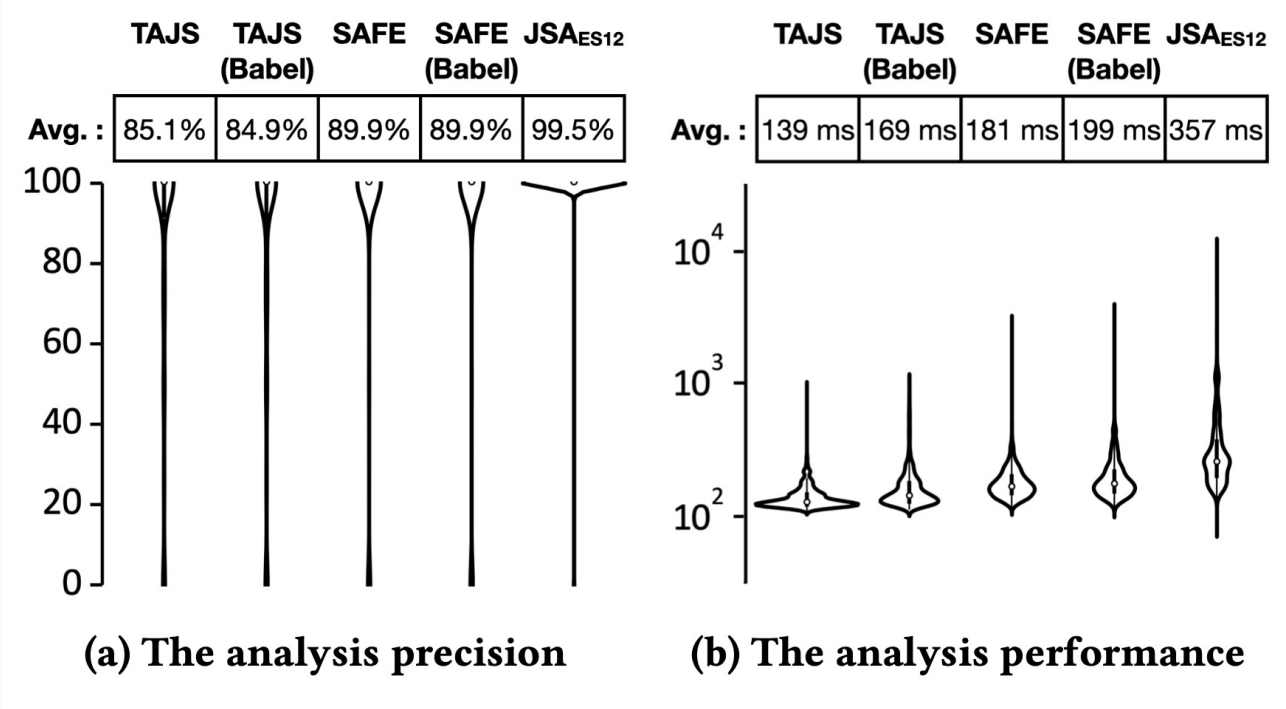
RQ1) Soundness

- Can JSA_{ES12} analyze JavaScript programs using new language features in a sound way?



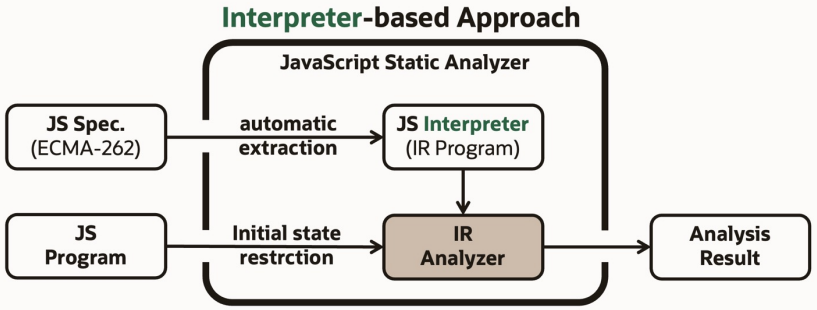
RQ2) Precision & Performance

- Can JSA_{ES12} precisely analyze JavaScript programs compared to the existing static analyzers?
  - **Targets**: 3,878 programs soundly analyzable by all of five analyzers



(a) The analysis precision (b) The analysis performance

Core Idea: Meta-level Static Analysis

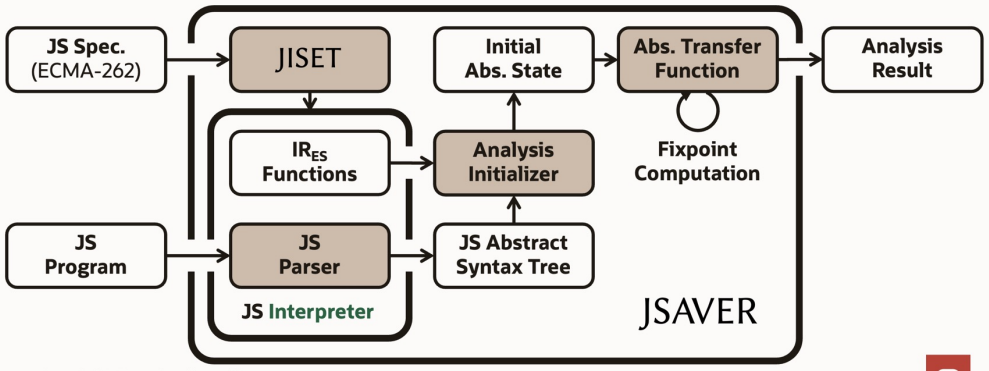


AST-Sensitivity

JavaScript	AST-Sensitivity in IR_{ES}
Flow-Sensitivity	$\delta^{\text{flow}}(t \in \mathbb{T}) = \{\sigma = (\_, c) \in \mathbb{S} \mid \text{syntax-ctxt}(c) = c' \wedge c'(\texttt{this}) = t\}$
k-Callsite-Sensitivity	$\delta^{k\text{-cfa}}(\bar{t} \in \mathbb{T}^{\leq k}) = \{\sigma = (\_, c) \in \mathbb{S} \mid \bar{t} = [t_1, \cdots, t_n] \wedge n \leq k \wedge (\nexists(\text{syntax-ctxt} \circ \text{js-call-ctxt})^{n+1}(c) \vee n = k) \wedge \forall 1 \leq i \leq n. ((\text{syntax-ctxt} \circ \text{js-call-ctxt})^{i}(c) = c_i \wedge c_i(\texttt{this}) = t_i)\}$

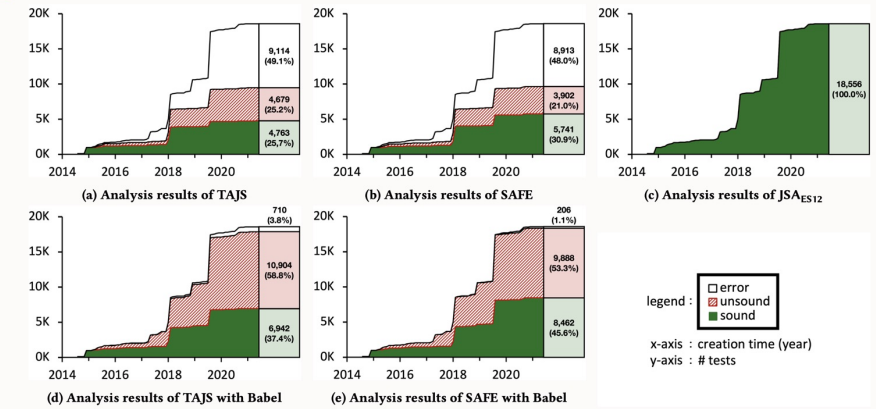
Our Tool - JSAVER

- **J**avaScript **S**tatic **A**nalyzer **v**ia **E**CMAScript **R**epresentation



RQ1) Soundness

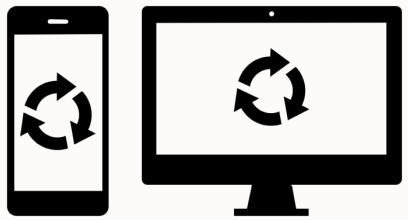
- Can JSA_{ES12} analyze JavaScript programs using new language features in a sound way?



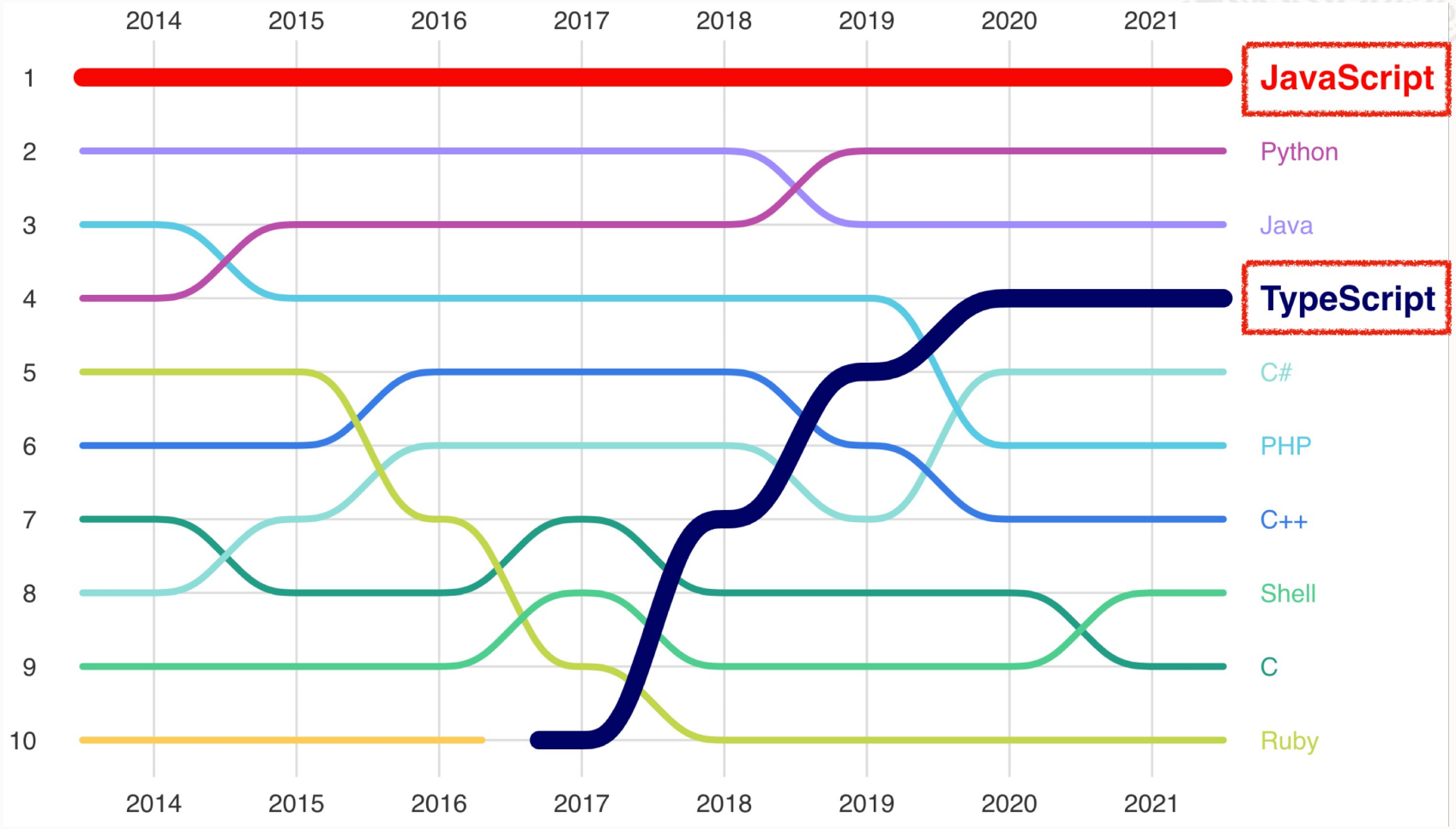
JavaScript is Everywhere

client-side

server-side



mobile/desktop apps



https://octoverse.github.com/

JavaScript Complex Semantics

```
function f(x) { return x == !x; }
```

Always return `false`?

NO!!

```
f([]) -> [] == +[]
      -> [] == false
      -> +[] == +false
      -> 0 == 0
      -> true
```

The *Interpreter*-based Nature of JavaScript

- In the 1990s, JavaScript semantics are defined with **reference interpreters**:

> *Guy Steele would ask a question about some edge-case feature behavior. [. . .] they would each turn to their **respective implementation** and **try a test case**. If they got the same answer, that became the specified behavior.*
>
> A. Wirfs-Brock and B. Eich, "*JavaScript: The First 20 Years*," HOPL, Article 77, 189 pages.

RQ3) Configurability / RQ4) Adaptability

- **RQ3) Configurability** - Can we configure abstract domains and analysis sensitivities for JavaScript in JSA_{ES12}?
  - Abstract Domains
    - Three Different String Domains – 1) String Set, 2) Character Inclusion, and 3) Prefix-Suffix
  - Analysis Sensitivities
    - Flow- and k-Callsite-Sensitivity
- **RQ4) Adaptability** - Can JSAVER adapt to new language features not yet introduced in ES12?
  - Pipeline Operator (`|>`)
  - `Observable` Built-in Library