



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

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

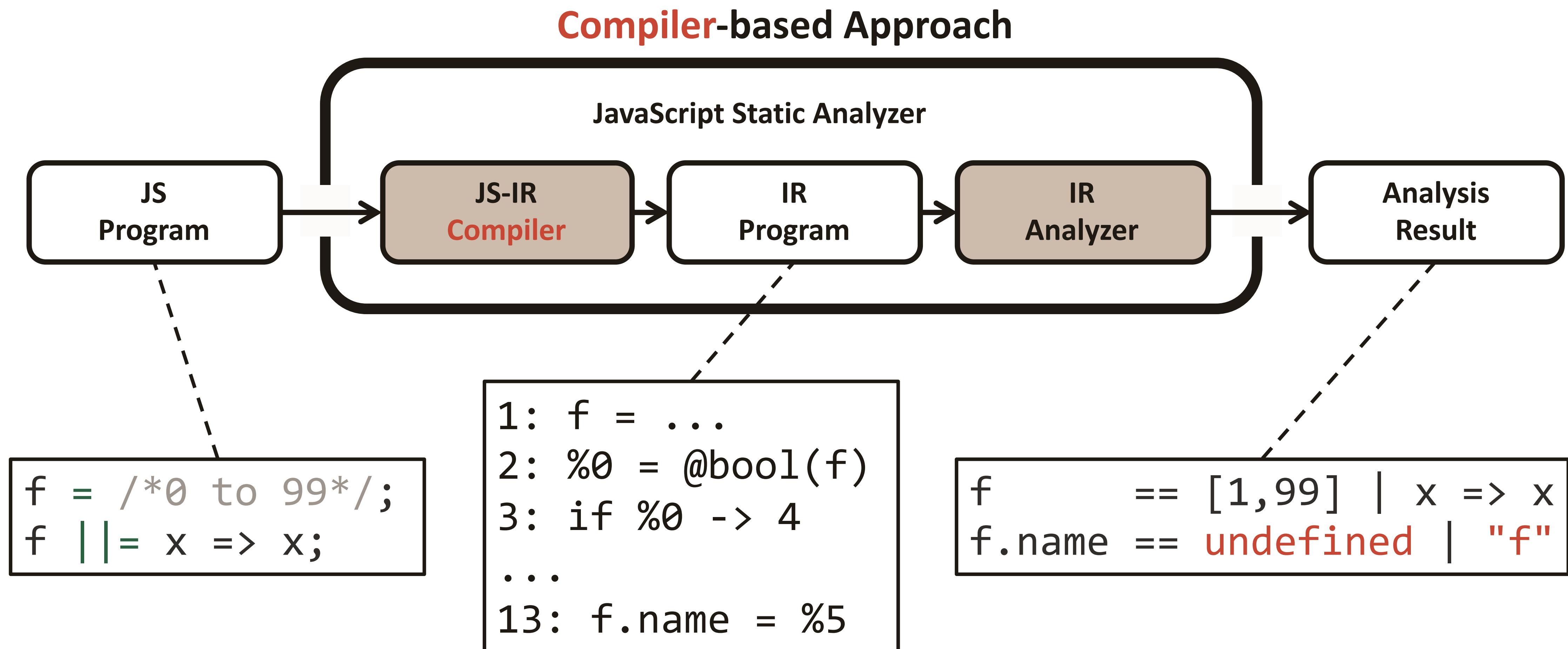
¹ Oracle Labs, Australia and ² KAIST, South Korea

KCSE 2023 우수 국제학회/학술지 초청 논문 세미나

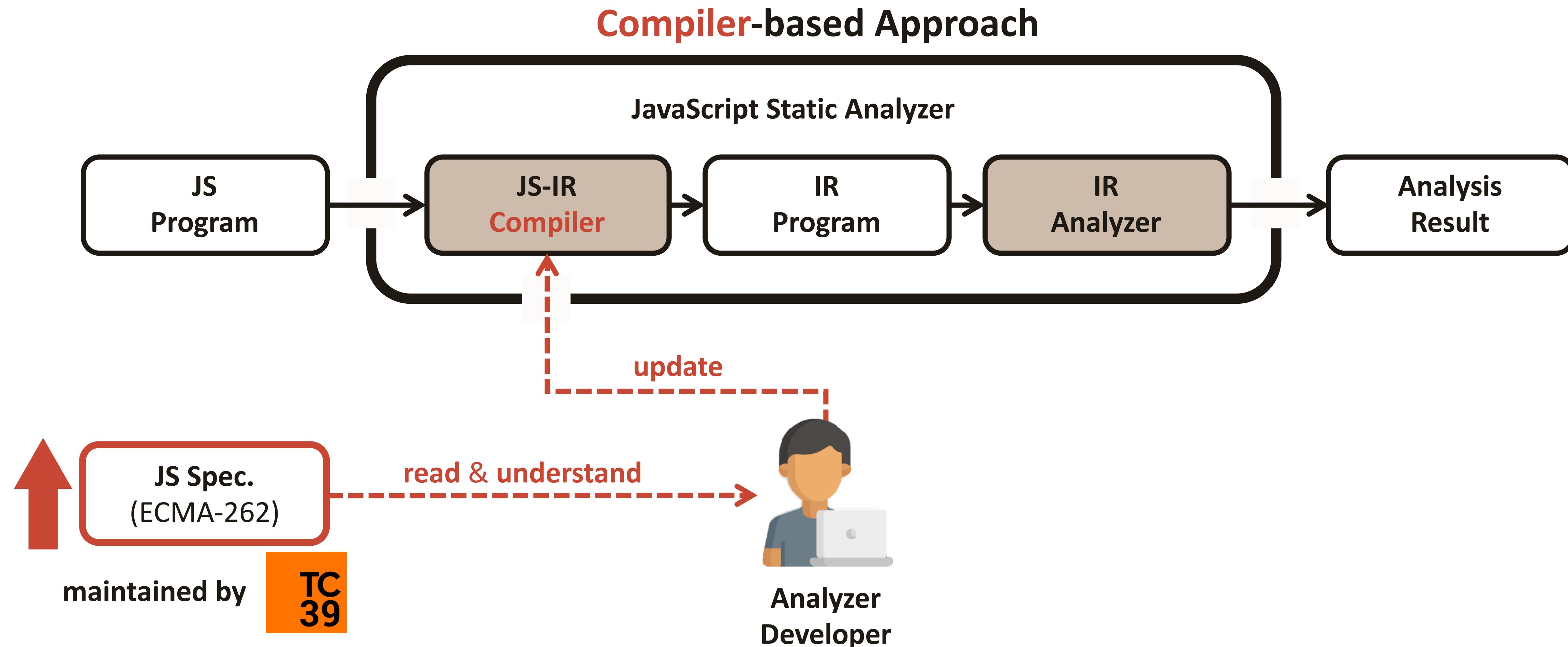
학회명: ESEC/FSE 2022

February 9, 2023

Background - JavaScript Static Analysis



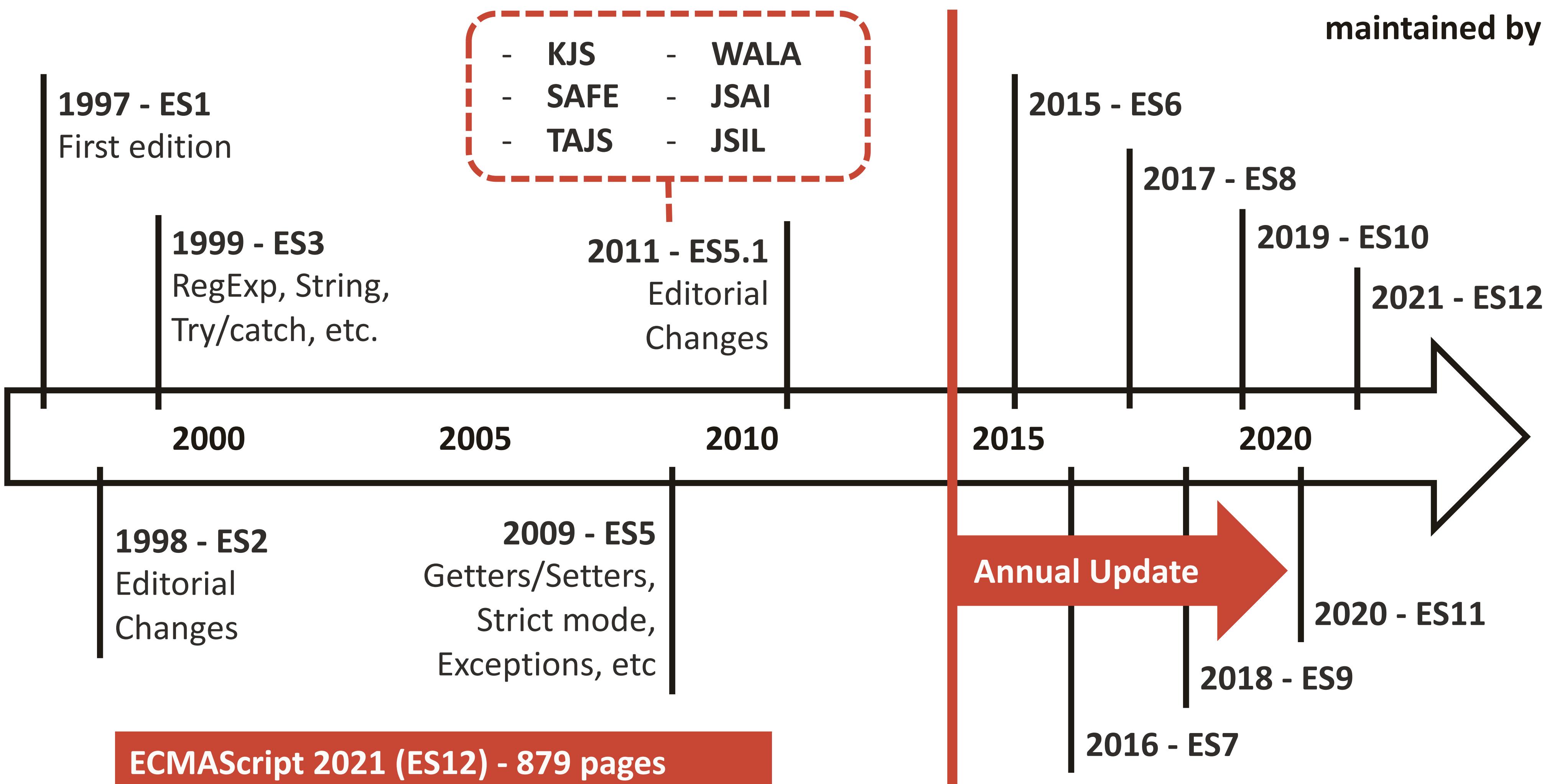
Problem - Manual Update of JS-IR Compiler



Problem - Fast Evolving JavaScript

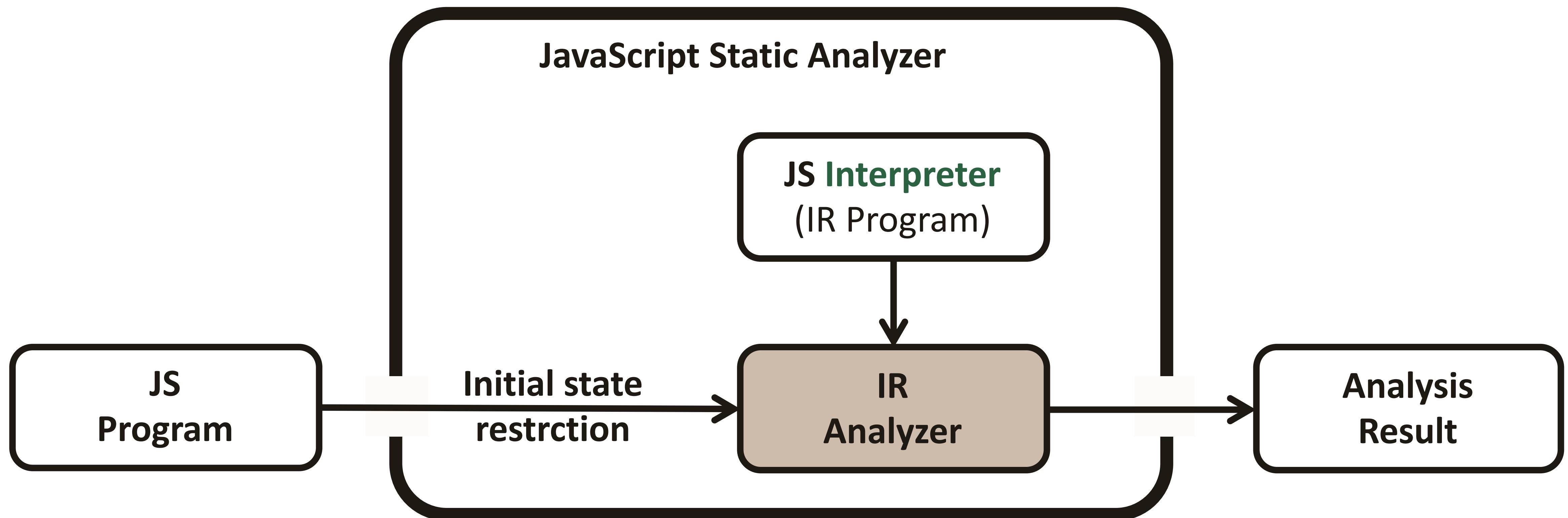
JS Spec.
(ECMA-262)

maintained by
**TC
39**



Core Idea - Meta-level Static Analysis

Interpreter-based Approach



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* $\mid\mid=$ *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*. $[[\text{ReferencedName}]]$.
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)**

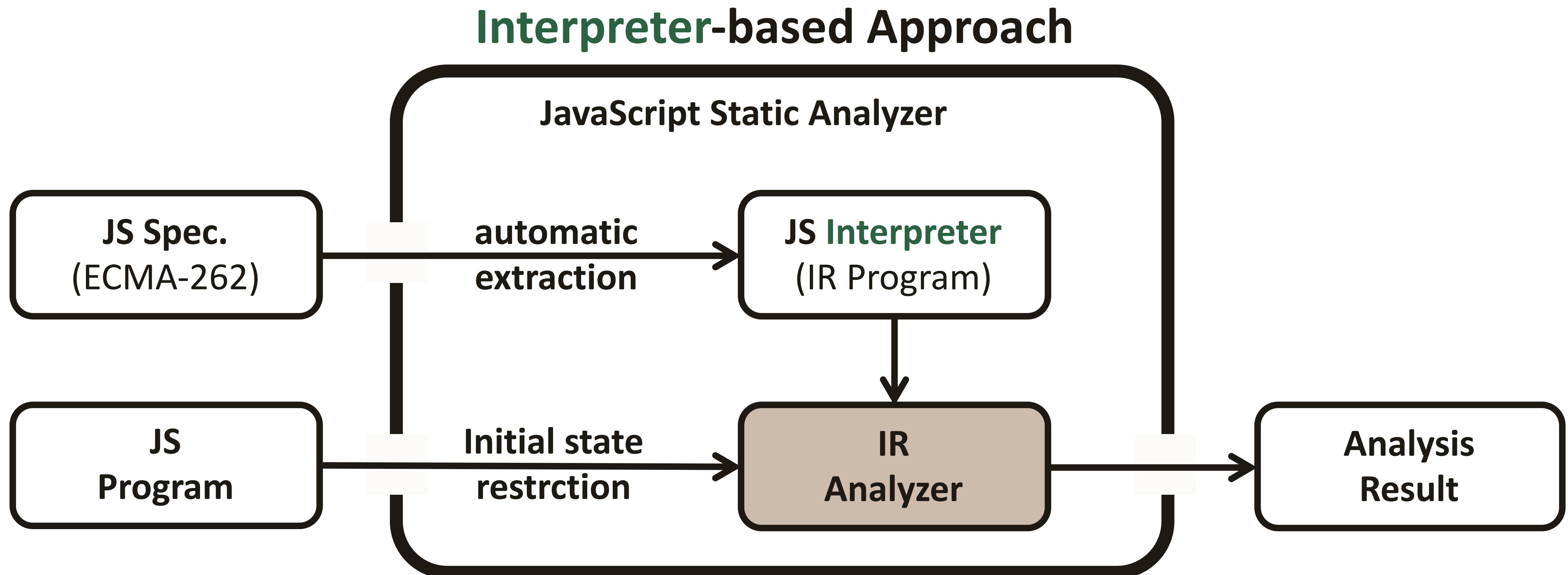
Core Idea - Meta-level Static Analysis

- 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 */
```

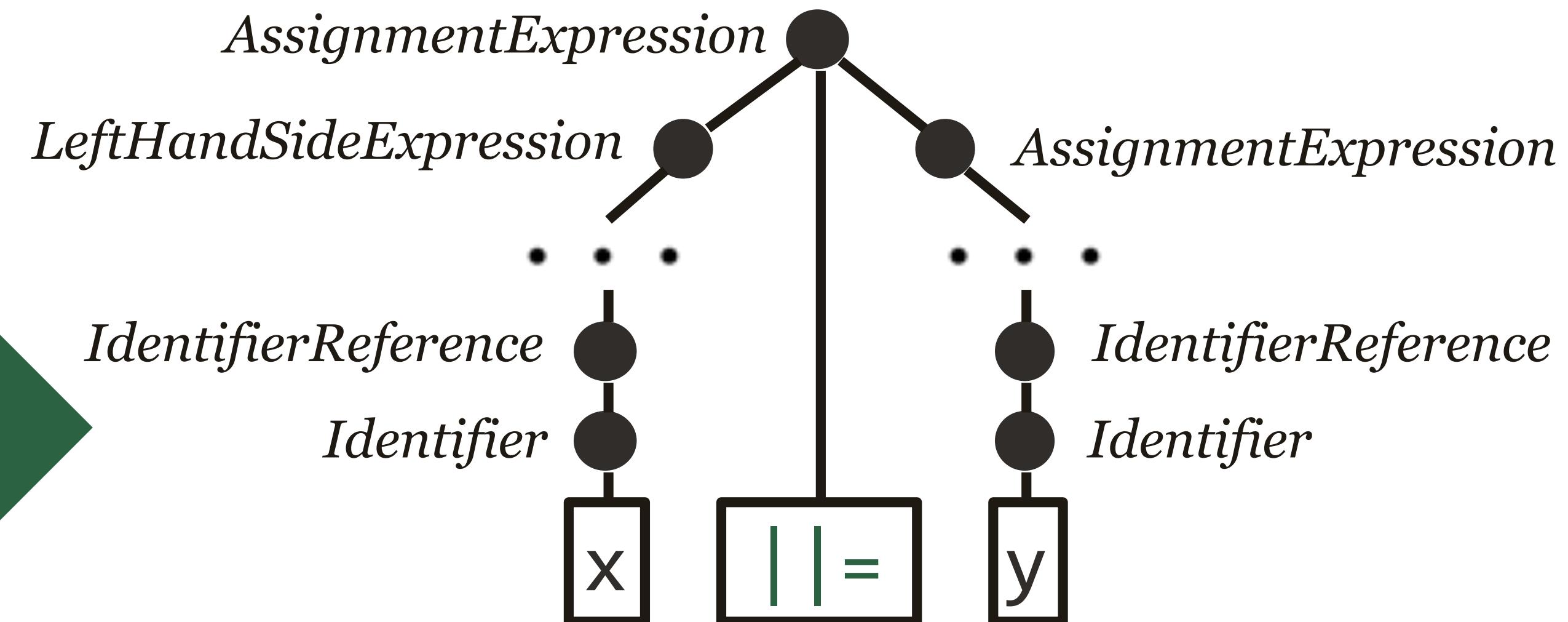
Extracted **IR_{ES}** function for *logical OR assignments* via **JISET**

Core Idea - Meta-level Static Analysis



Example

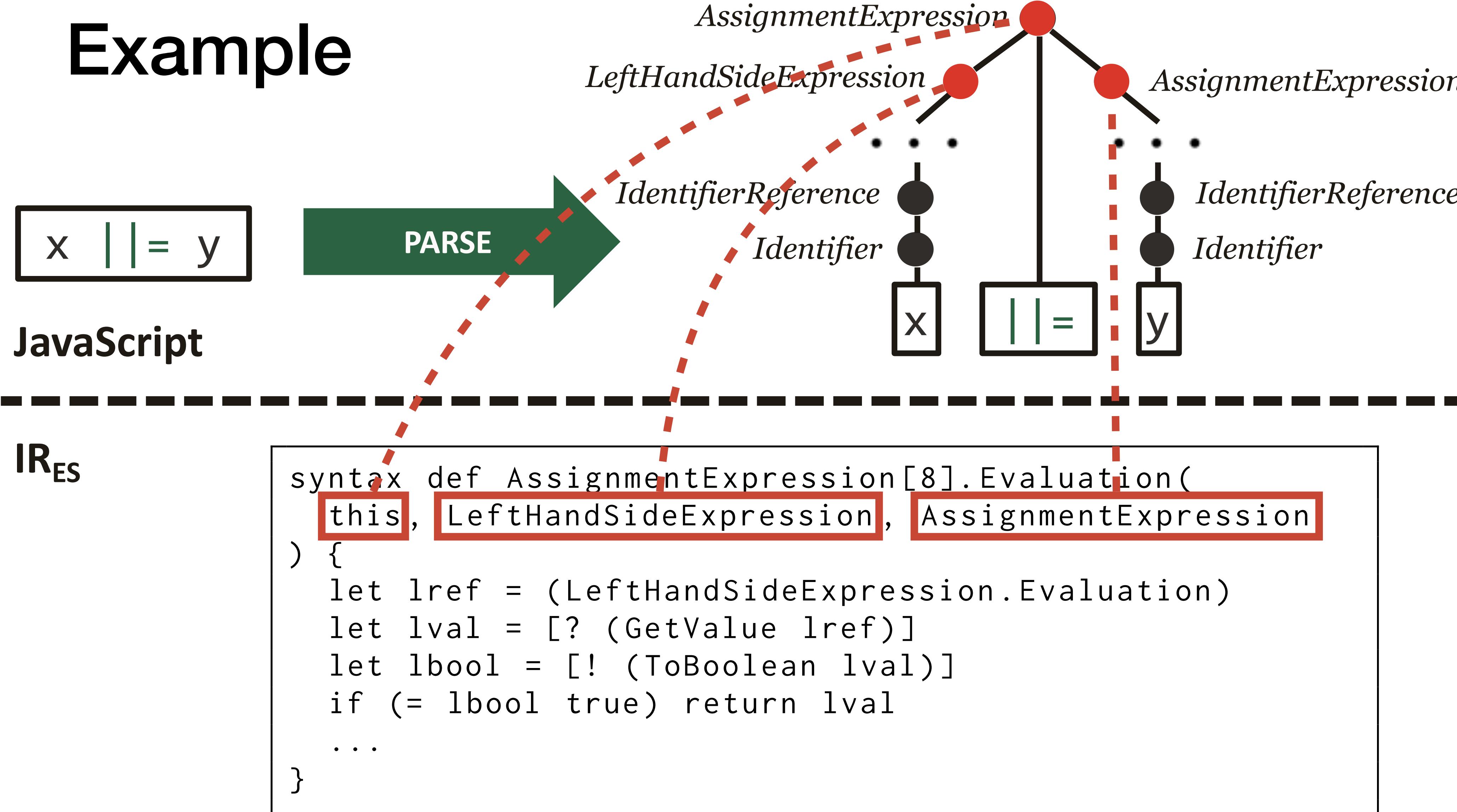
x ||= y
JavaScript



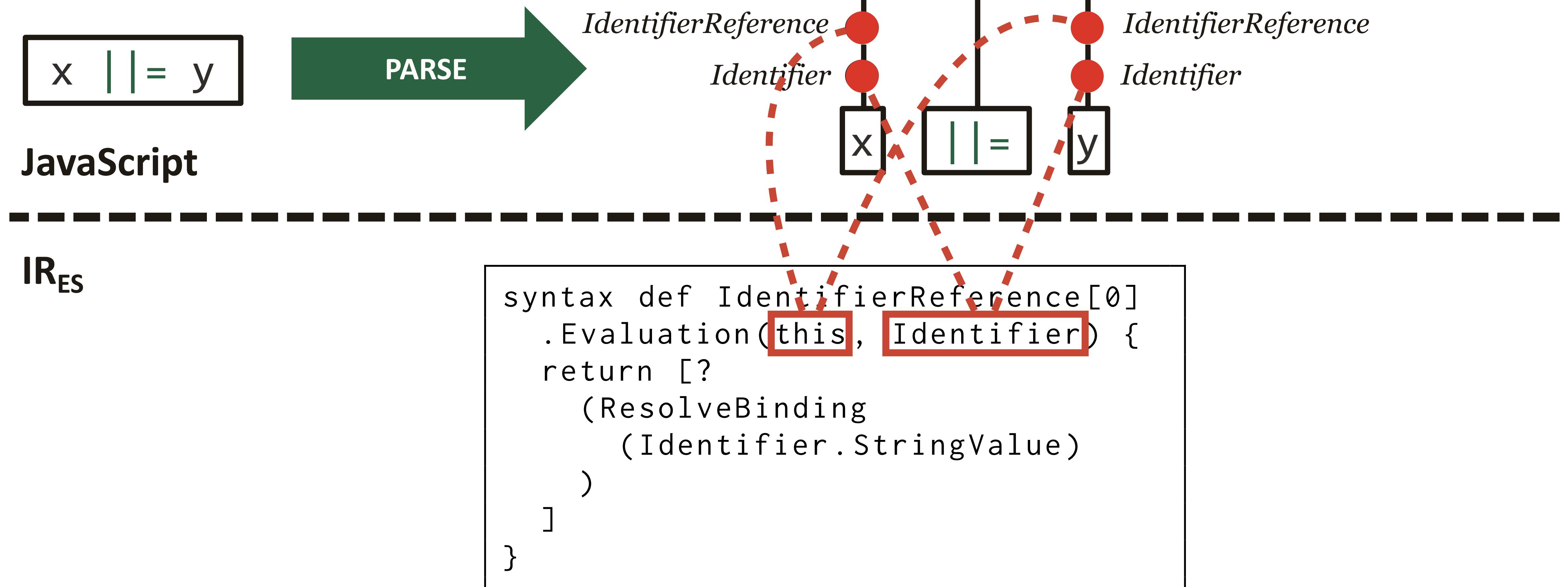
IR_{ES}

```
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
    ...
}
```

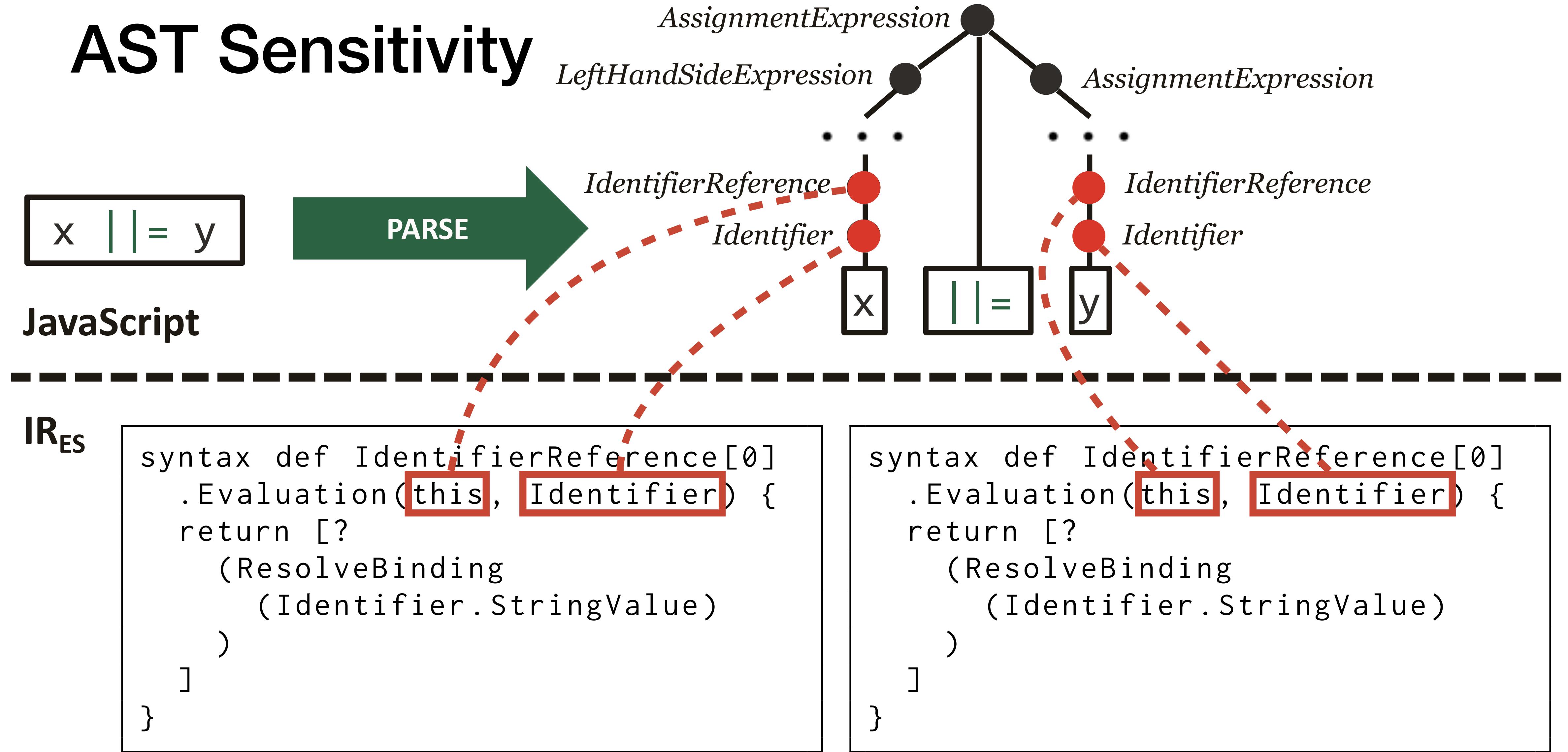
Example



AST Sensitivity



AST Sensitivity

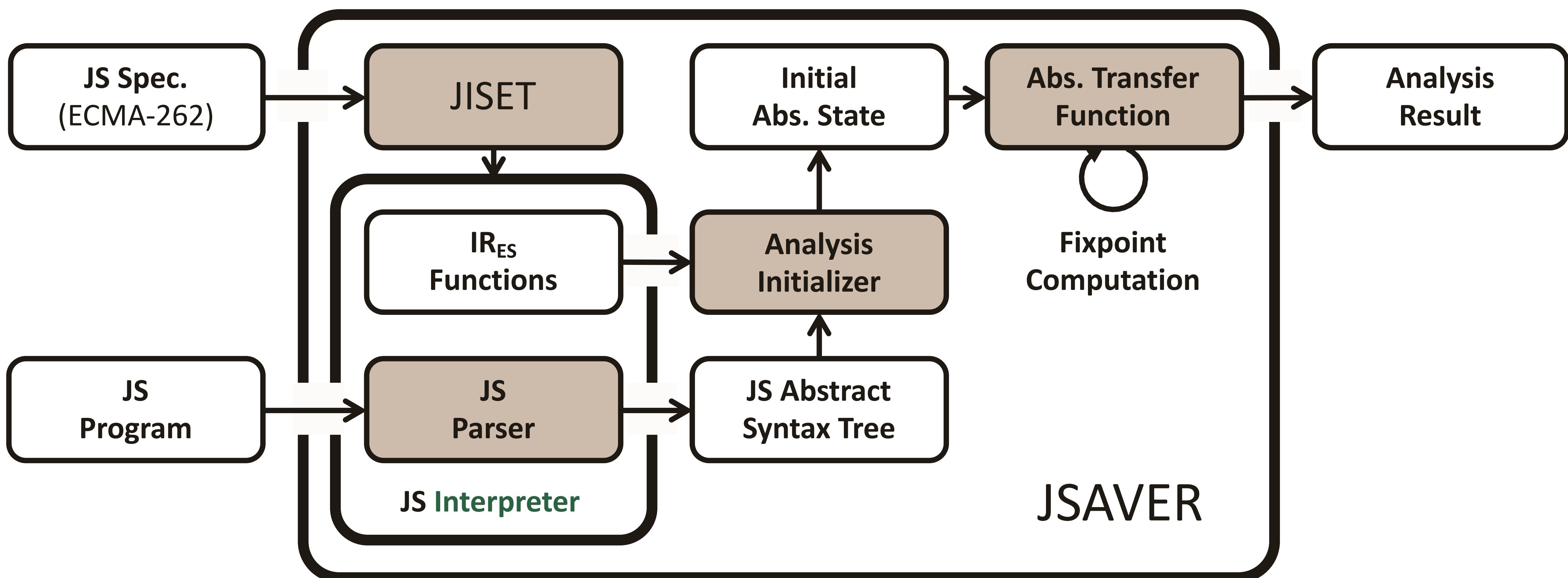


AST Sensitivity

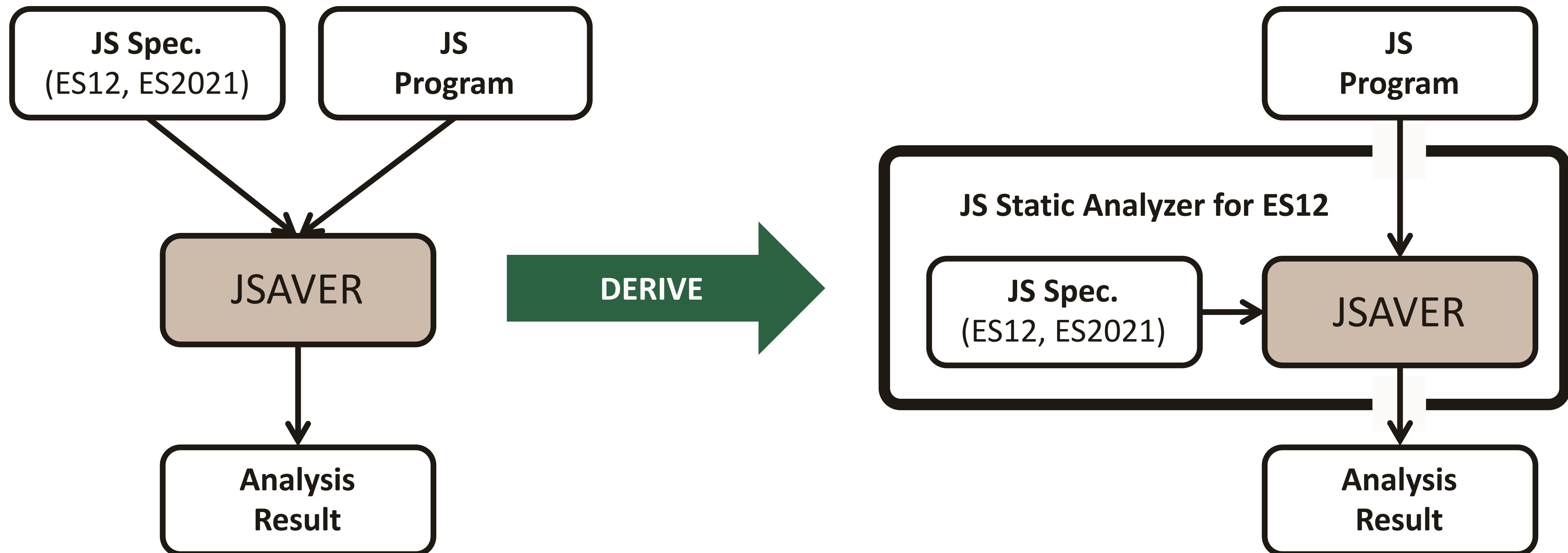
JavaScript	AST Sensitivity in IR_{ES}
Flow-Sensitivity	$\delta^{\text{js-flow}}(t_{\perp}) = \{\sigma = (_, _, \bar{c}, _) \in \mathbb{S} \mid \text{ast}(\bar{c}) = t_{\perp}\}$
k -Callsite-Sensitivity	$\delta^{\text{js-}k\text{-cfa}}([t_1, \dots, t_n]) = \{\sigma = (_, _, \bar{c}, _) \in \mathbb{S} \mid n \leq k \wedge (n = k \vee \text{js-ctxt}^{n+1}(\bar{c}) = \perp) \wedge \forall 1 \leq i \leq n. \text{ast} \circ \text{js-ctxt}^i(\bar{c}) = t_i\}$

Our Tool - JSAVER

- JavaScript Static Analyzer via ECMAScript Representation



JS Static Analyzer Derivation via JSAVER

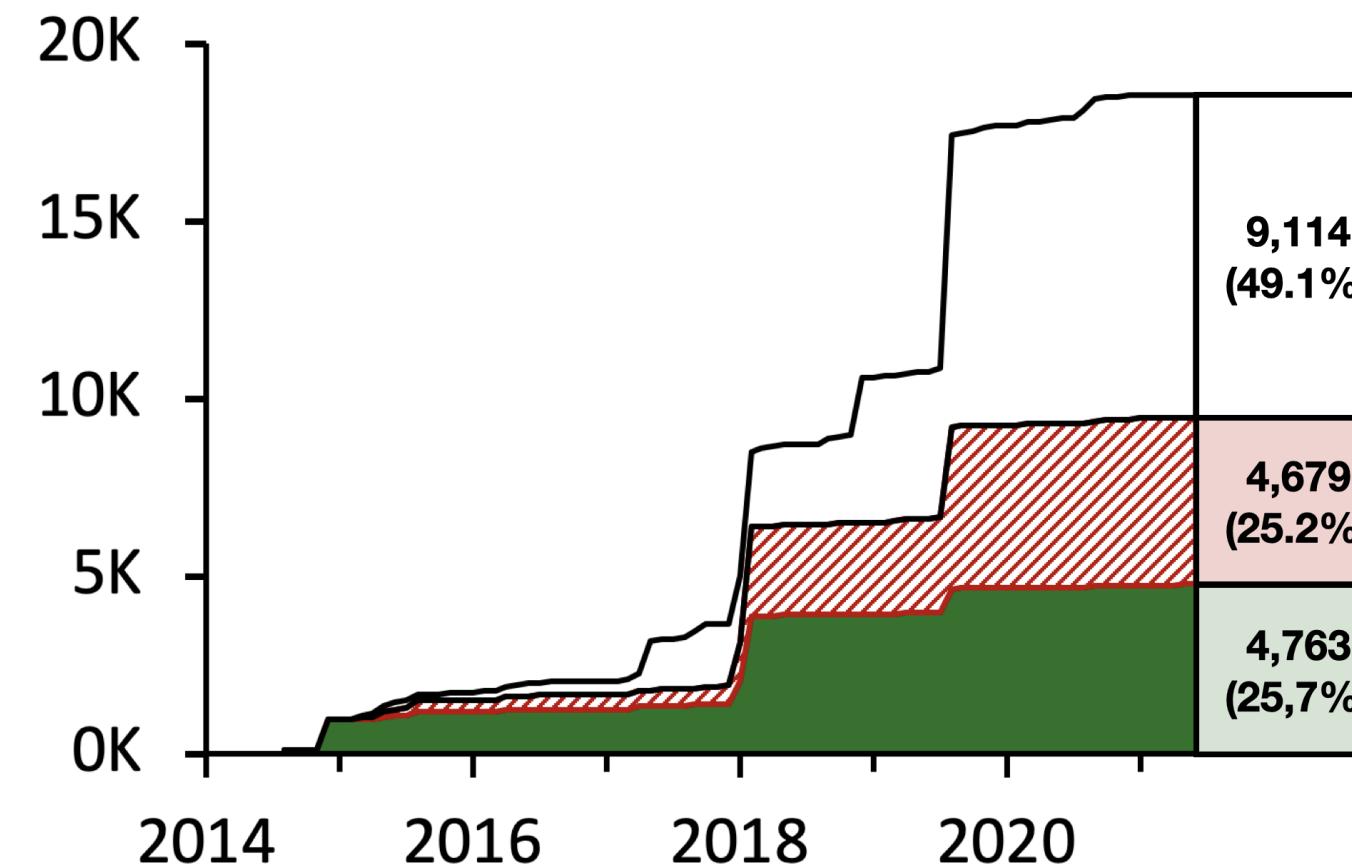


Evaluation Setting

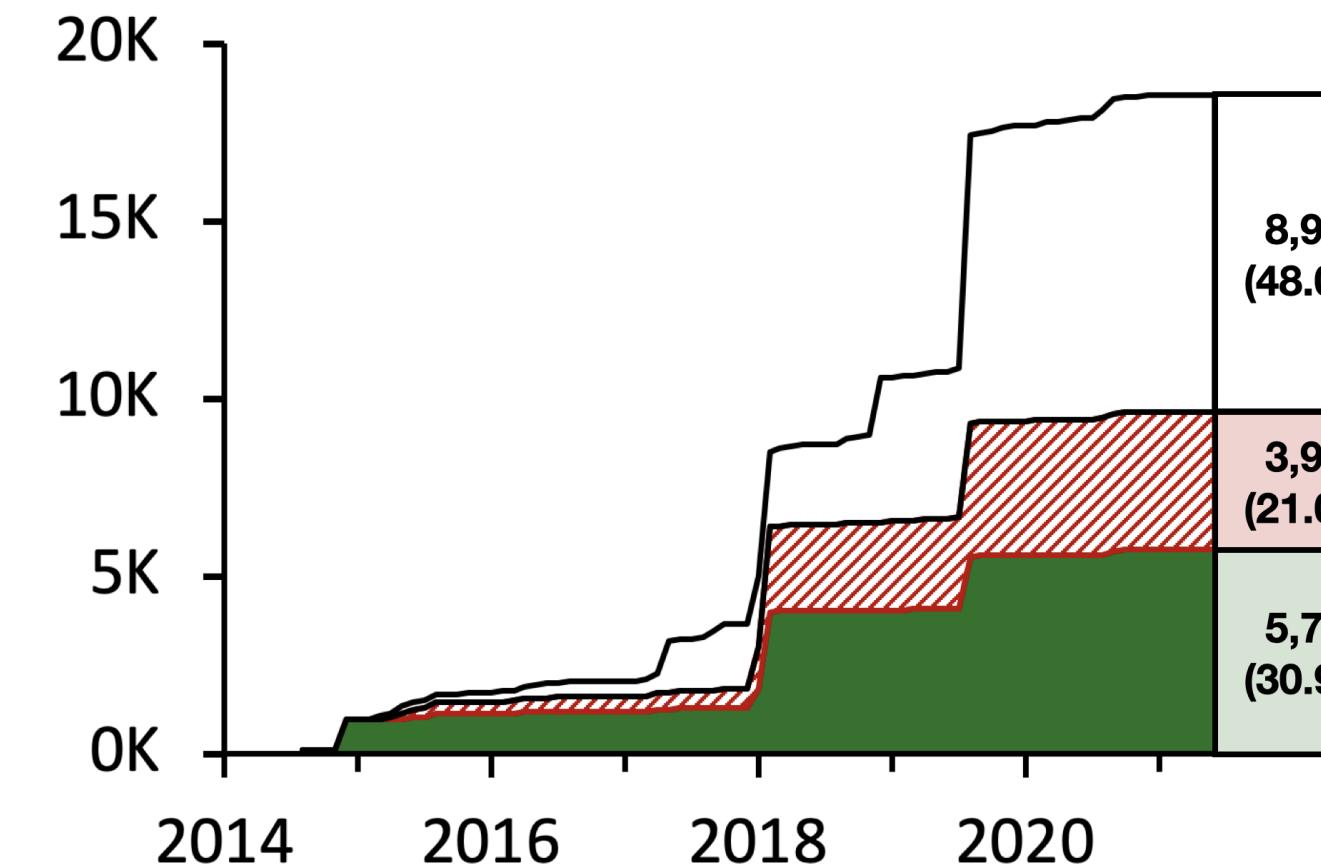
- **Derived Analyzer - JSA_{ES12}**
 - JavaScript Static Analyzer derived from **ES12** (ES2021) via **JSAVER**
- **Comparison Targets**
 - State-of-the-art JavaScript Static Analyzers + JavaScript Transpiler
 - TAJS / SAFE + Babel
- **Analysis Targets**
 - **Test262** (Official Conformance Test Suite) 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

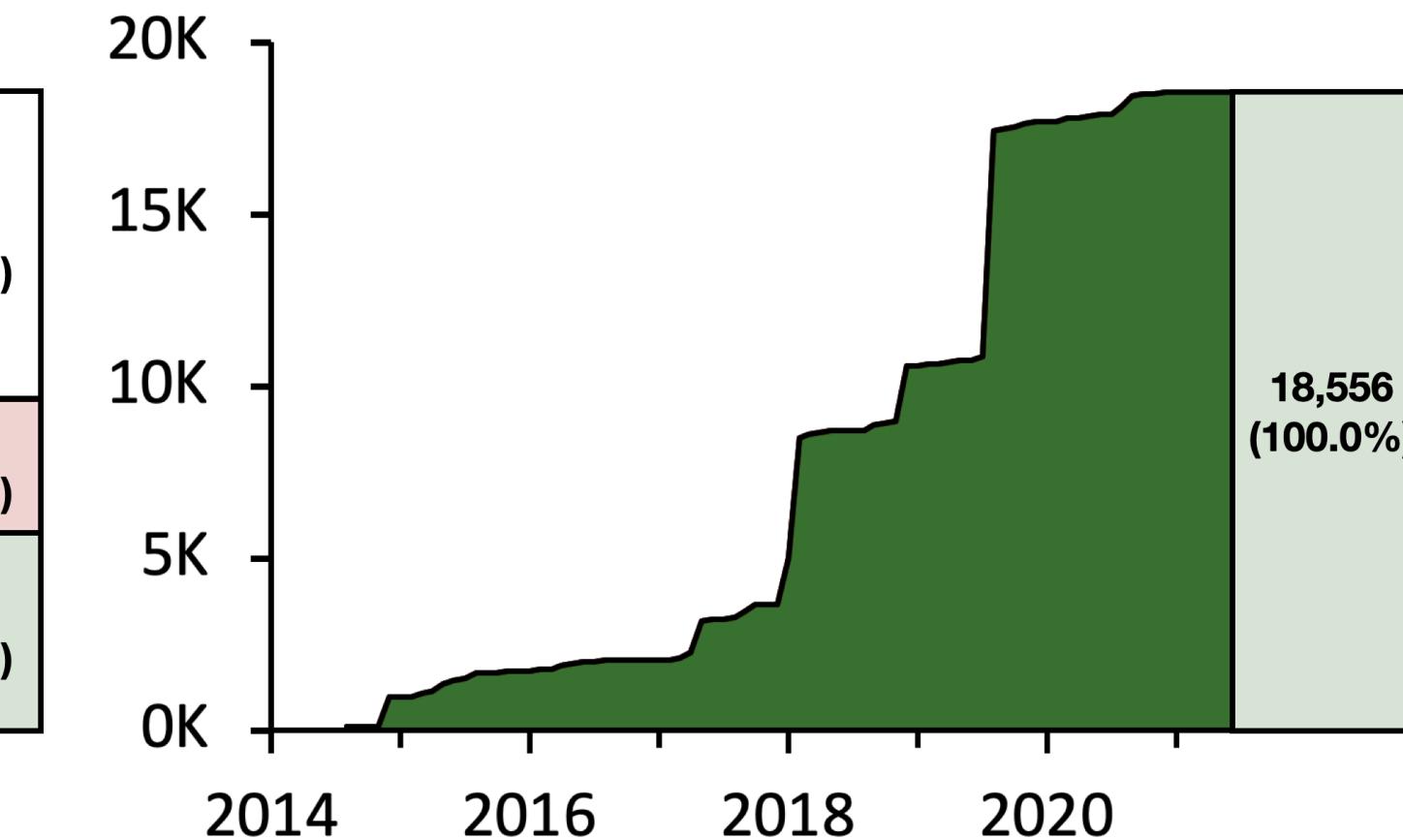
- Can JSA_{ES12} analyze JavaScript programs using new language features in a sound way?



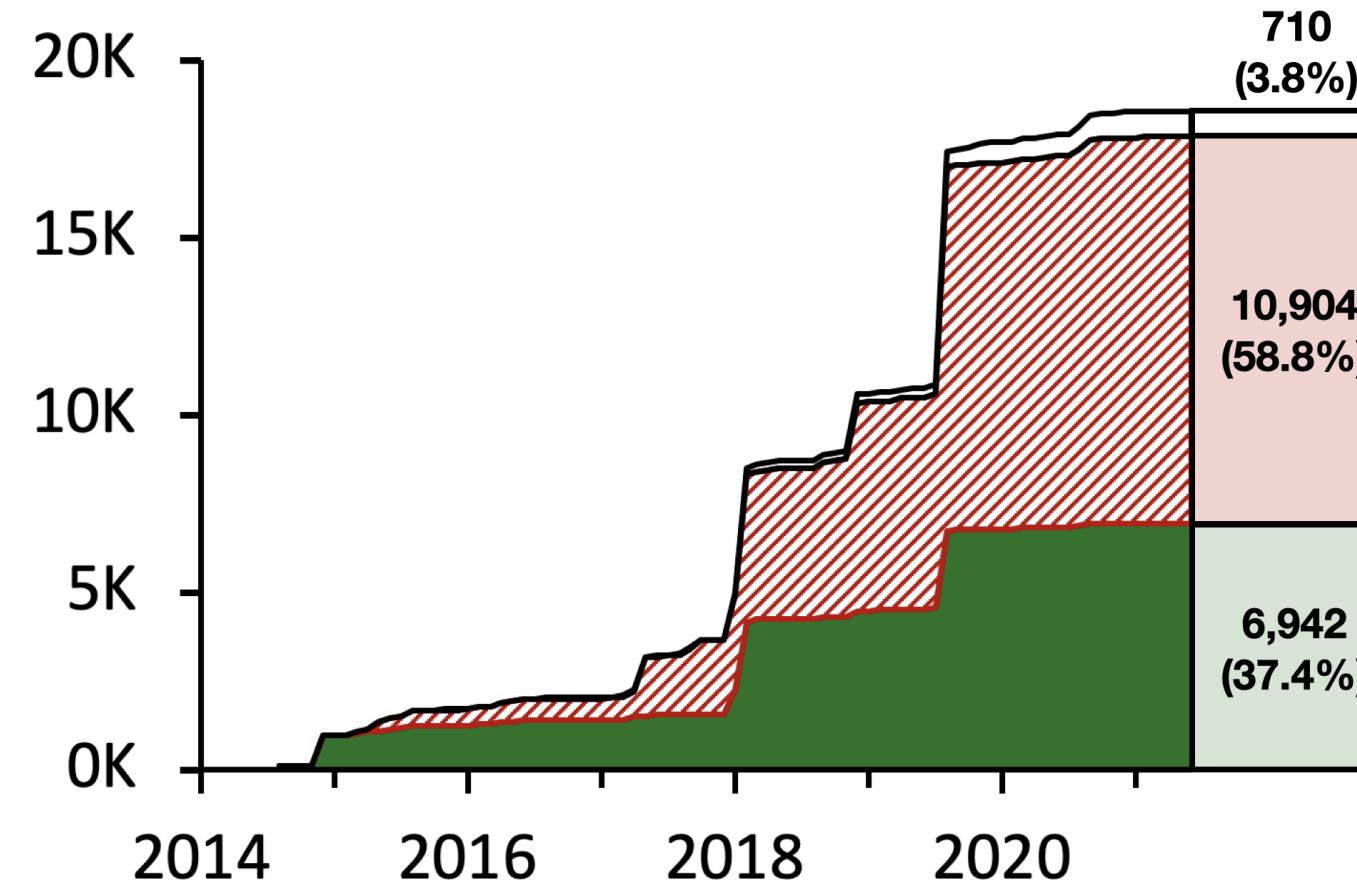
(a) Analysis results of TAJJS



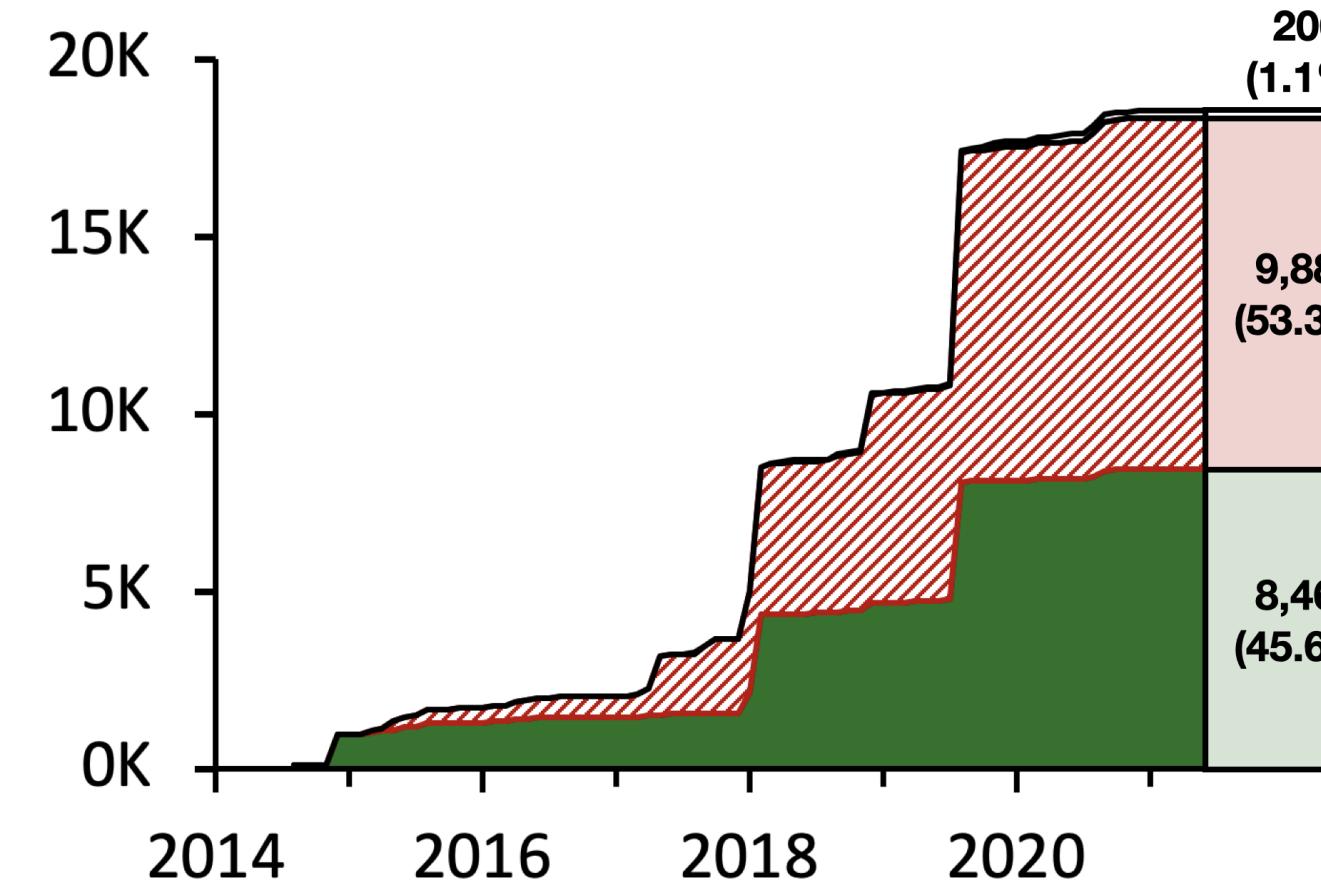
(b) Analysis results of SAFE



(c) Analysis results of JSA_{ES12}



(d) Analysis results of TAJJS with Babel



(e) Analysis results of SAFE with Babel

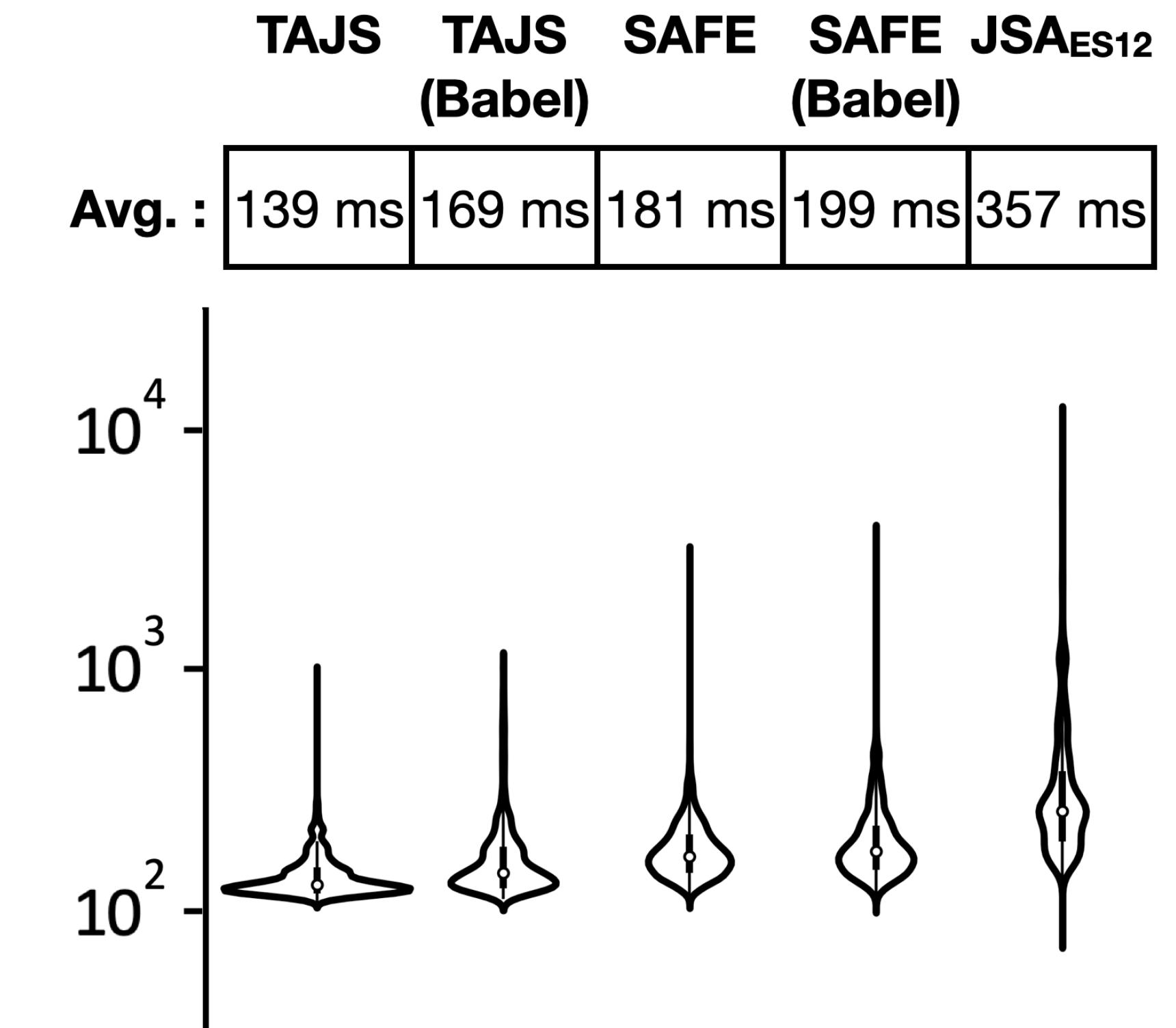
legend :
□ error
▨ unsound
■ sound
x-axis : creation time (year)
y-axis : # tests

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

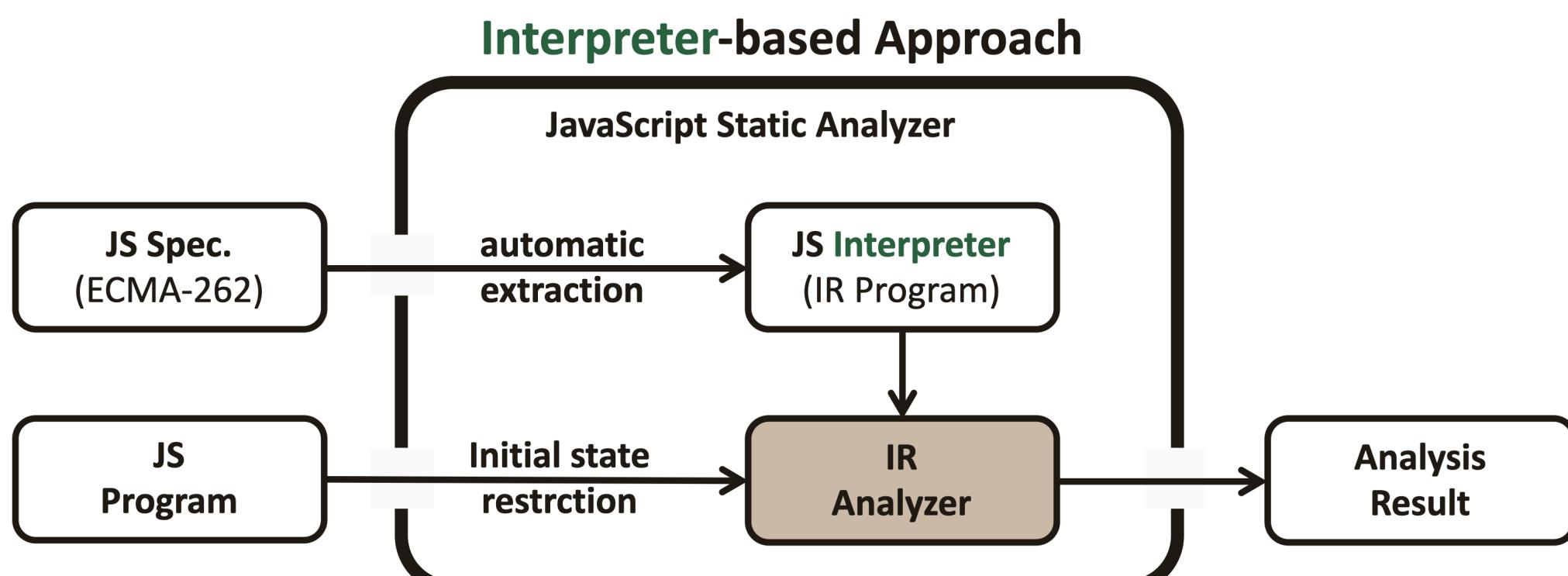


(a) The analysis precision



(b) The analysis performance

Core Idea - Meta-level Static Analysis



KCSE 2023 우수 국제학회/학술지 초청 논문 세미나 - ESEC/FSE 2022 - JSAVER

8 / 19

AST Sensitivity

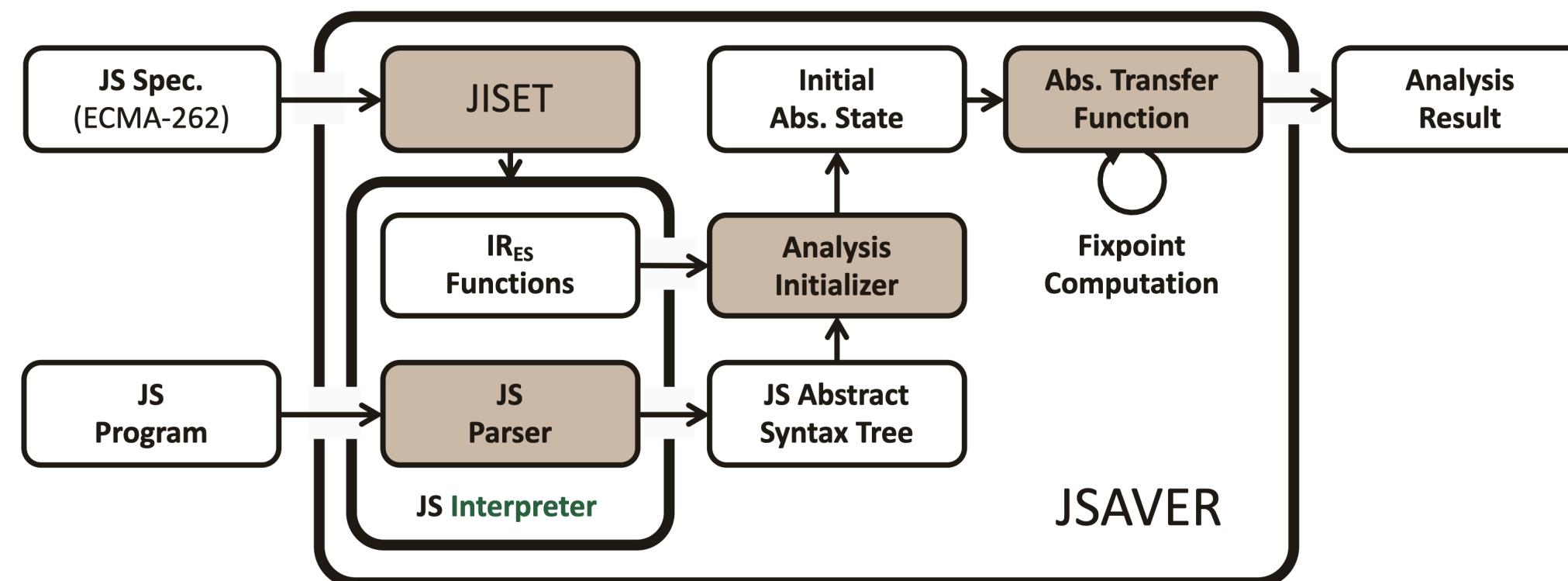
JavaScript	AST Sensitivity in IR_{ES}
Flow-Sensitivity	$\delta^{\text{js-flow}}(t_{\perp}) = \{\sigma = (_, _, \bar{c}, _) \in \mathbb{S} \mid \text{ast}(\bar{c}) = t_{\perp}\}$
k-Callsite-Sensitivity	$\delta^{\text{js-}k\text{-cfa}}([t_1, \dots, t_n]) = \{\sigma = (_, _, \bar{c}, _) \in \mathbb{S} \mid n \leq k \wedge (n = k \vee \text{js-ctxt}^{n+1}(\bar{c}) = \perp) \wedge \forall 1 \leq i \leq n. \text{ast} \circ \text{js-ctxt}^i(\bar{c}) = t_i\}$

KCSE 2023 우수 국제학회/학술지 초청 논문 세미나 - ESEC/FSE 2022 - JSAVER

13 / 19

Our Tool - JSAVER

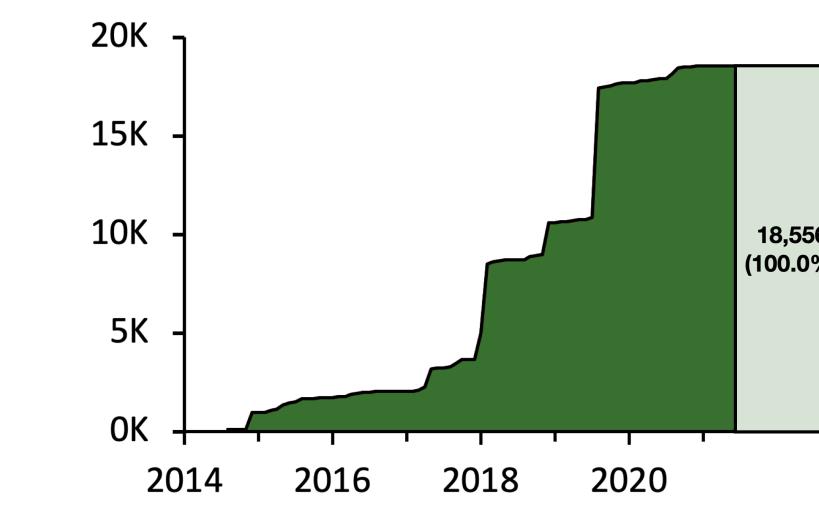
- JavaScript Static Analyzer via ECMAScript Representation



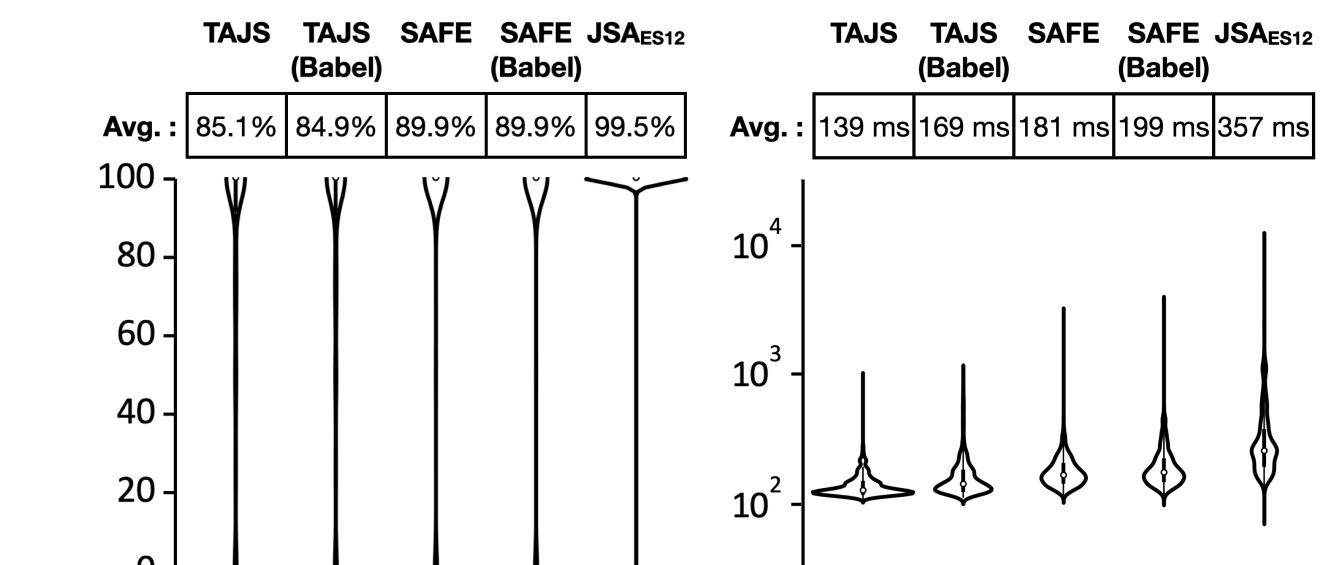
KCSE 2023 우수 국제학회/학술지 초청 논문 세미나 - ESEC/FSE 2022 - JSAVER

14 / 19

RQ1) Soundness



RQ2) Precision & Performance



(a) The analysis precision

(b) The analysis performance