[OPLSS: Static Analysis]

Automatic Derivation of Static Analysis

Sukyoung Ryu

with PLRG@KAIST and friends

July 5, 2023

OPLSS: Static Analysis

(1) **Concepts in Static Analysis**

(2) **Operational / Denotational Semantics**

(3) **Abstract Interpretation**

(4) **Automatic Derivation of Static Analysis**

[OPLSS: Static Analysis]

How to design and implement programming languages

Sukyoung Ryu

with PLRG@KAIST and friends

July 5, 2023

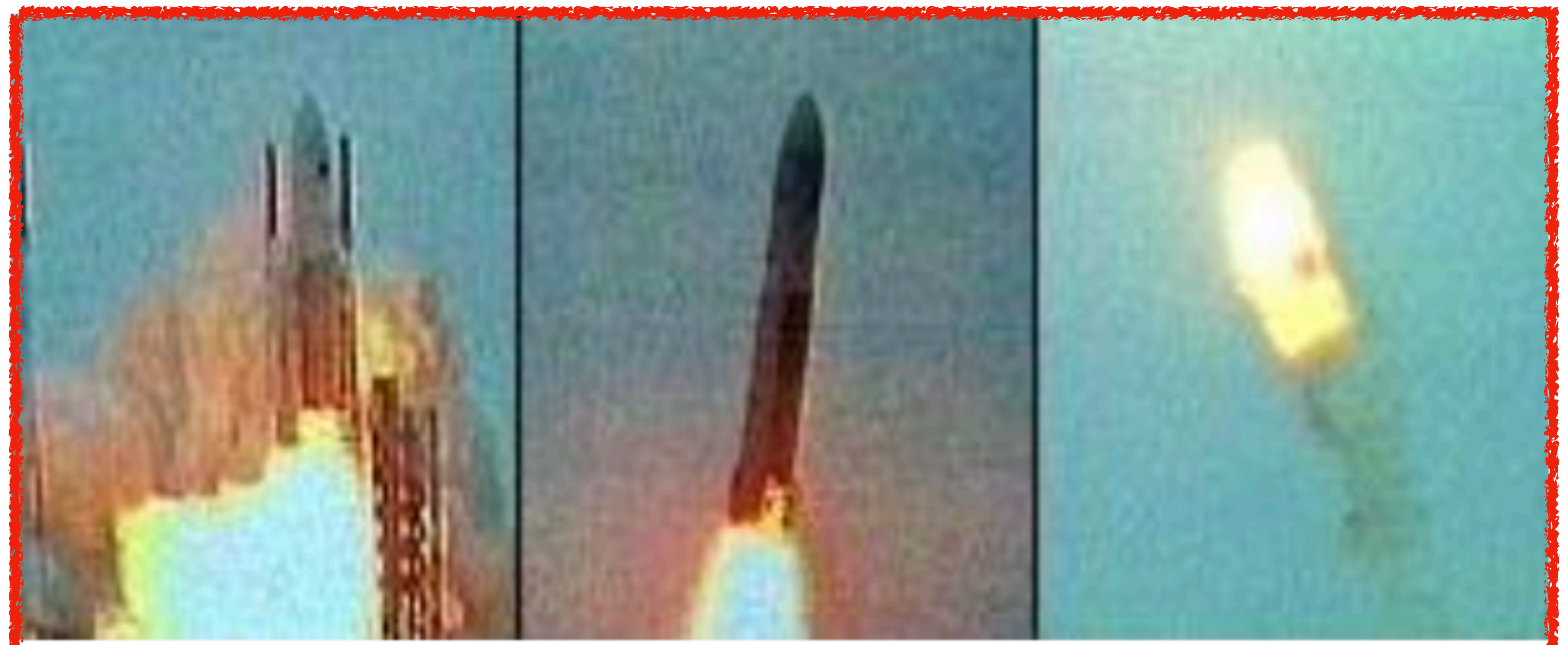
KAIST: Exception Analyzer

Links to other SML resources

Please feel free to send additional URLs that should be on this list to smlnj-dev-list@mailman.cs.uchicago.edu

SML Programming Resources

- Concurrent ML
- sml_tk, a library for using the TK graphical interface
- Martin Erwig's Functional Graph Library
- Yi and Ryu's SML/NJ exception analyzer



The Ariane-5 Rocket (1996)
Integer Overflow
$100M

Harvard: Debugging Everywhere

Debugging Everywhere

The goal of the *Debugging Everywhere* project is to make debugging a cheap, ubiquitous service. We intend to begin by getting compilers to emit *Active Debugging Information*, which we expect will support multi-language, multi-platform debugging much more readily than older approaches like Dwarf or dbx ``stabs.''

```
ldb Fib (stopped) > t
  0 <_print:2> (Mips/mjr.c:25,2)  void _print(char *s = (0x1000008c) " ")
* 1 <fib:51+0x24> (Fib.java:23,32)
         void fib(Fib this = {int buffer = 10,
                              int[] a = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
                                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
                             }, int n = 10)
  2 <main:3+0x18> (Fib.java:5,23)  void main(String[] argv = {})
  3 <mAiN:end+0x1c> (mininub.c:7,9)
         int mAiN(int argc = 2, char **argv = 0x7fff7b24,
                  char **envp = 0x7fff7b34)
```

Sun Microsystems: Fortress

Fortress is a discontinued experimental programming language for high-performance computing, created by Sun Microsystems with funding from DARPA's High Productivity Computing Systems project. One of the language designers was Guy L. Steele Jr., whose previous work includes Scheme, Common Lisp, and Java.

$$
\begin{array}{l}
z\colon \mathrm{Vec} := 0 \\
r\colon \mathrm{Vec} := x \\
p\colon \mathrm{Vec} := r \\
\rho\colon \mathrm{Elt} := r^T\, r \\
\texttt{for } j \leftarrow seq(1 : cgit_{\max})\ \texttt{do} \\
\quad q = A\, p \\
\quad \alpha = \dfrac{\rho}{p^T\, q} \\
\quad z := z + \alpha\, p \\
\quad r := r - \alpha\, q \\
\quad \rho_0 = \rho \\
\quad \rho := r^T\, r \\
\quad \beta = \dfrac{\rho}{\rho_0} \\
\quad p := r + \beta\, p \\
\texttt{end} \\
(z, \|x - A\, z\|)
\end{array}
$$

KAIST: JavaScript Analyzer

`module` M $\{s\cdots\}$	module definition
`module` $M = M\cdots.M$`;`	module alias
`import` $M\cdots.x$`;`	qualified import
`import` $M\cdots.x$`:` x`;`	aliased import
`import` $M\cdots.*$`;`	import all
`export var` x $[= e]$`;`	exported variable
`export function` x`(`$x\cdots$`)`$\{s\cdots\}$	exported function
`export module` M $\{s\cdots\}$	exported module
`export module` $M = M\cdots.M$`;`	exported module alias
`export` x`;`	exported local
`export` x`:` x`;`	exported local alias
`export` x`:` $M\cdots.x$`;`	exported qualified alias

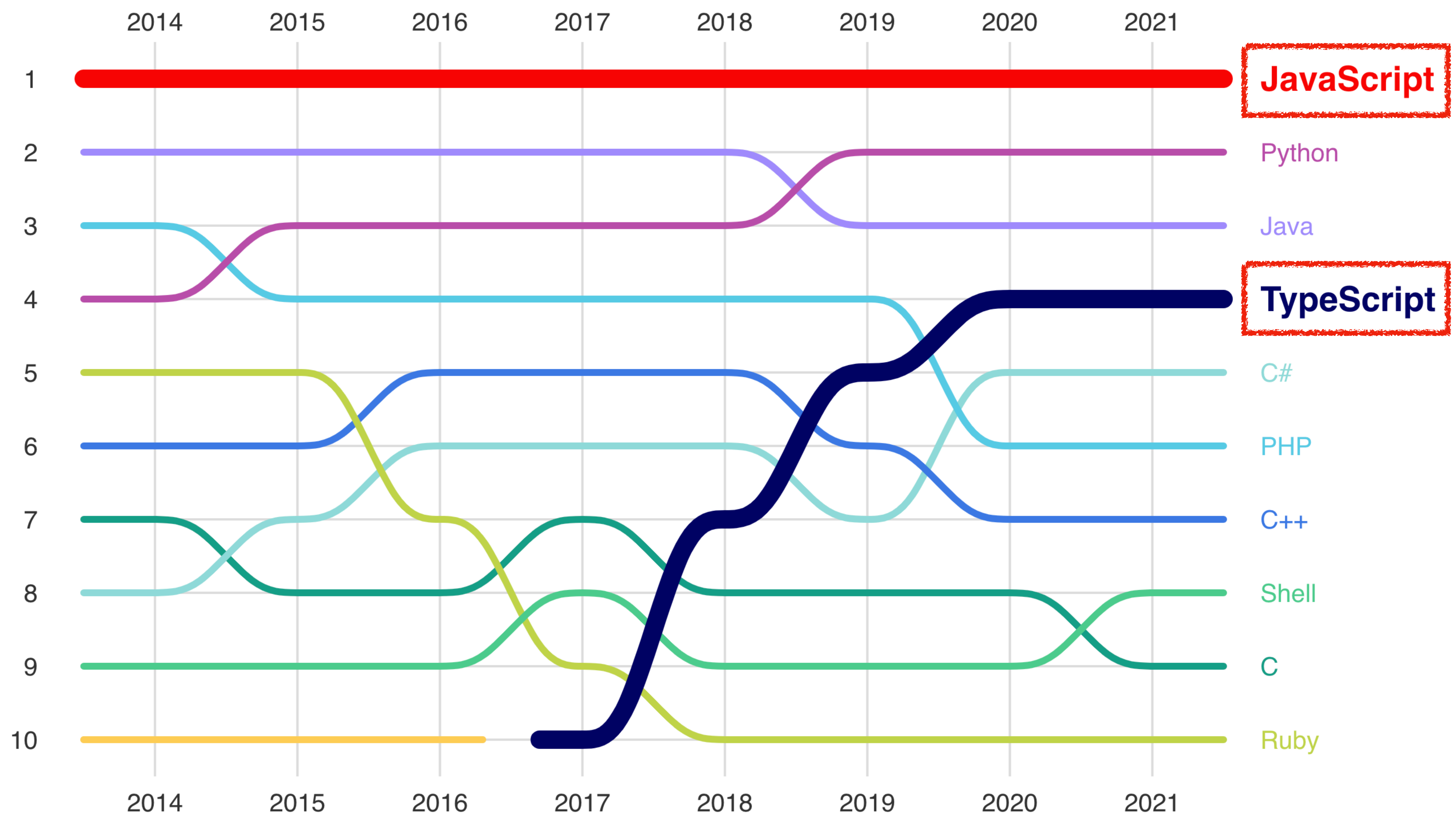
Figure 9. Extended syntax for JavaScript modules

$v ::= \cdots \mid \alpha$	value
$\alpha ::= \langle\!\langle v_g \rangle\!\rangle$	accessor
$v_g ::= \mathsf{func}()\{\ \mathsf{return}\ e\ \}$	getter

Figure 10. Extended syntax for λ_{JS} to allow accessors

$\phi ::= x\cdots$	path
$\varphi_i ::= \phi.(x)$	internal qualified name
$\varphi_e ::= \phi.x$	external qualified name
$\varphi ::= \varphi_i \mid \varphi_e$	qualified name
$\bar{\varphi} ::= \phi.*$	expanded qualified name
$\tau ::= \mathsf{var} \mid \mathsf{module}$	desugaring type
$\varsigma ::= \epsilon \mid \mathsf{local} \mid \mathsf{export}\ \varphi_e$	desugaring scope
$\rho ::= \bot \mid \tau\,\varphi$	desugaring binding
$\bar{\rho} ::= \bot \mid \top$	expanded desugaring binding
$\Sigma ::= \{(\varphi, \rho\,\varsigma)\cdots\} \cup \{(\bar{\varphi}, \bar{\rho})\cdots\}$	desugaring environment
$\Sigma^* ::= \epsilon \mid \Sigma^*\ \Sigma \mid \Sigma^*\ x$	desugaring environment chain

Figure 12. Desugaring environment for the modified λ_{JS}



https://octoverse.github.com/

JavaScript Complex Semantics

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

Always return `false`?

JavaScript Complex Semantics

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

Always return `false`?

NO!!

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

JavaScript Analyzer

SAFE @ KAIST

TAJS @ Aarhus University

WALA @ IBM T.J. Watson

ECMA-262: JavaScript Language Specification

ecma INTERNATIONAL maintained by TC 39

Syntax

ArrayLiteral$_{\texttt{[Yield, Await]}}$:

  [*Elision*$_{\texttt{opt}}$]

  [*ElementList*$_{\texttt{[?Yield, ?Await]}}$]

  [*ElementList*$_{\texttt{[?Yield, ?Await]}}$, *Elision*$_{\texttt{opt}}$]

The production of *ArrayLiteral* in ES12

ECMA-262: JavaScript Language Specification

ecma INTERNATIONAL maintained by TC 39

Syntax

ArrayLiteral`[Yield, Await]` **:**

  [*Elision*`opt` **]**

  [*ElementList*`[?Yield, ?Await]` **]**

  [*ElementList*`[?Yield, ?Await]` **,** *Elision*`opt` **]**

The production of *ArrayLiteral* in ES12

Semantics

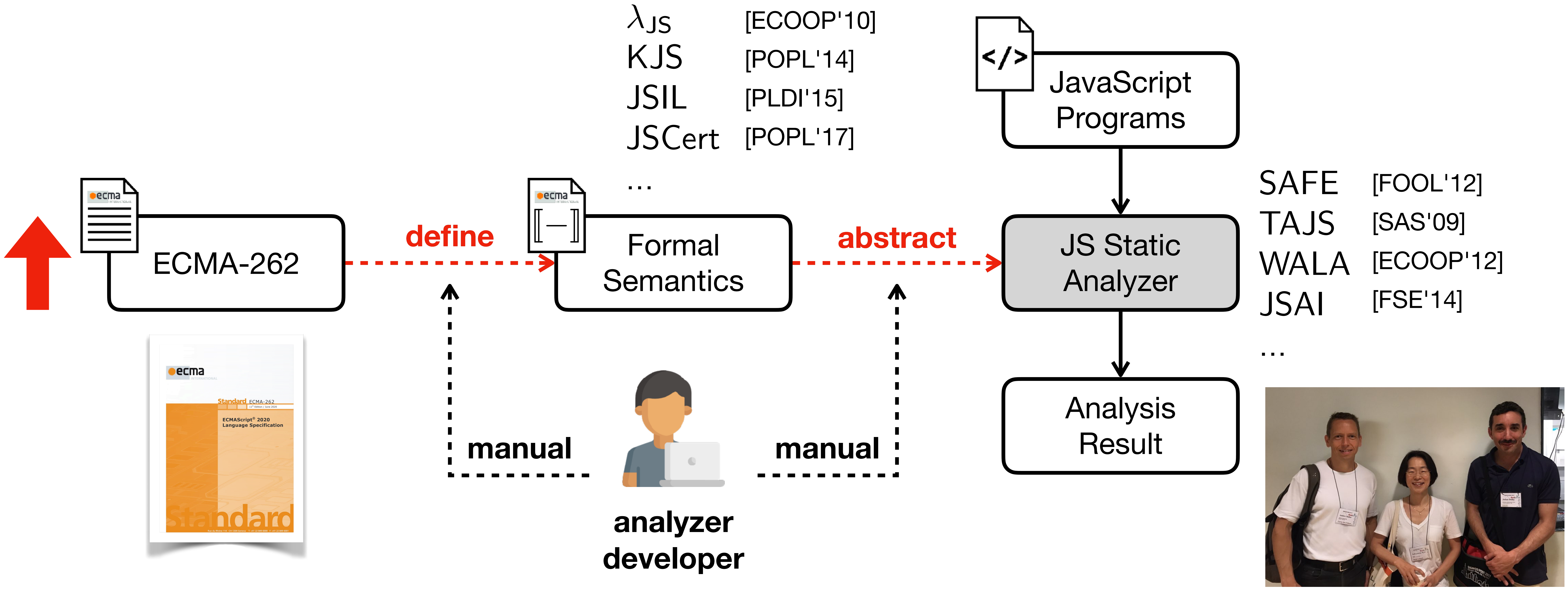
13.2.5.2 Runtime Semantics: Evaluation

ArrayLiteral **: [** *ElementList* **,** *Elision*`opt` **]**

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
   a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments *array* and *nextIndex*.
   b. ReturnIfAbrupt(*len*).
5. Return *array*.

The Evaluation **algorithm for**
the third alternative of *ArrayLiteral* in ES12

Problem: Hand-Written JavaScript Static Analyzer



SAFE: Formal Specification and Implementation of a Scalable Analysis Framework for ECMAScript

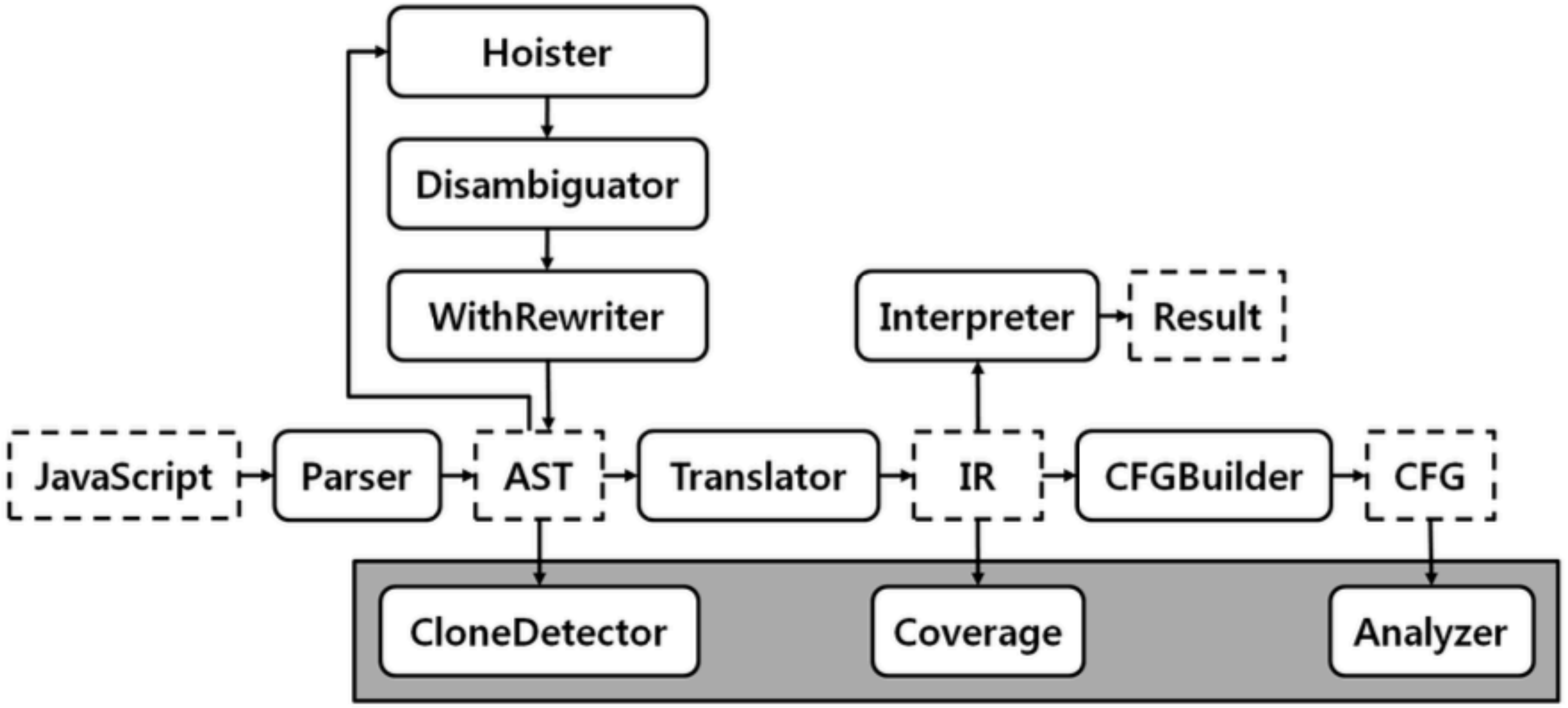


Figure 2. SAFE flow graph

$$
\begin{array}{rcccl}
(H, A, tb) & \in & \multicolumn{3}{l}{\mathit{Heap} \times \mathit{Env} \times \mathit{ThisBinding}} \\
H & \in & \mathit{Heap} & = & \mathit{Loc} \xrightarrow{\mathit{fin}} \mathit{Object} \\
A & \in & \mathit{Env} & ::= & \#\texttt{Global} \\
 & & & \mid & er :: A \\
er & \in & \mathit{EnvRec} & = & \mathit{DeclEnvRec} \cup \mathit{ObjEnvRec} \\
\sigma & \in & \mathit{DeclEnvRec} & = & \mathit{Var} \xrightarrow{\mathit{fin}} \mathit{StoreValue} \\
l & \in & \mathit{ObjEnvRec} & = & \mathit{Loc} \\
tb & \in & \mathit{ThisBinding} & = & \mathit{Loc}
\end{array}
$$

Figure 5. Execution contexts and other domains

$$
\begin{array}{rcccl}
ct & \in & \mathit{Completion} & ::= & nc \\
 & & & \mid & ac \\
nc & \in & \mathit{NormalCompletion} & ::= & \texttt{Normal}(vt) \\
ac & \in & \mathit{AbruptCompletion} & ::= & \texttt{Break}(vt, x) \\
 & & & \mid & \texttt{Return}(v) \\
 & & & \mid & \texttt{Throw}(ve) \\
vt & \in & & & \mathit{Val} \cup \{\texttt{empty}\} \\
ve & \in & \mathit{ValError} & = & \mathit{Val} \cup \mathit{Error}
\end{array}
$$

Figure 6. Completion specification type

Static Analysis of JavaScript Web Applications in the Wild via Practical DOM Modeling

different levels of representations. Its default static analyzer on CFGs supports flow-sensitive and context-sensitive analyses of JavaScript programs by faithfully modeling the semantics of ECMAScript 5 [18]. A SAFE analysis computes the following summary map for a program:

$$\hat{s} \in \widehat{\mathsf{S}} = \mathsf{Node} \times \mathsf{Context} \rightarrow \widehat{\mathsf{Heap}}$$

which maps a program point represented by a pair of a CFG node and a context to an over-approximate abstract heap information. A variable or an object property in an abstract heap, $\hat{h} \in \widehat{\mathsf{Heap}}$, maps to an abstract value $\hat{v}$ represented by a 6-tuple of lattice elements as follows:

$$\hat{v} \in \widehat{\mathsf{V}} = \widehat{\mathsf{Undef}} \times \widehat{\mathsf{Null}} \times \widehat{\mathsf{Bool}} \times \widehat{\mathsf{Number}} \times \widehat{\mathsf{String}} \times \wp(\widehat{\mathsf{Loc}})$$

where $\wp(\widehat{\mathsf{Loc}})$ is a finite set of abstract locations that map to abstract objects in abstract heaps and the others are simple abstract domains for primitive types, `Undefined`, `Null`, `Bool`, `Number`, and `String`. Their definitions are available in the SAFE manual [17]. For example, an abstract value $\hat{v}$ that may be `true` or `null` is represented as follows:

$$\hat{v} = \langle \bot_{\mathrm{Undef}}, \widehat{\mathsf{null}}, \widehat{\mathsf{true}}, \bot_{\mathrm{Number}}, \bot_{\mathrm{String}}, \varnothing \rangle.$$

With the domains, the default static analyzer performs sound and elaborate analyses on CFGs of JavaScript programs with the transfer function $\hat{F} \in \widehat{\mathsf{S}} \rightarrow \widehat{\mathsf{S}}$ to compute a final summary map $\hat{s}_{\mathsf{final}}$ from the following least fixpoint computation:

$$\hat{s}_{\mathsf{final}} = \mathsf{leastFix}\ \lambda\hat{s}.(\hat{s}_{\mathrm{I}} \sqcup_{\widehat{\mathsf{S}}} \hat{F}(\hat{s}))$$

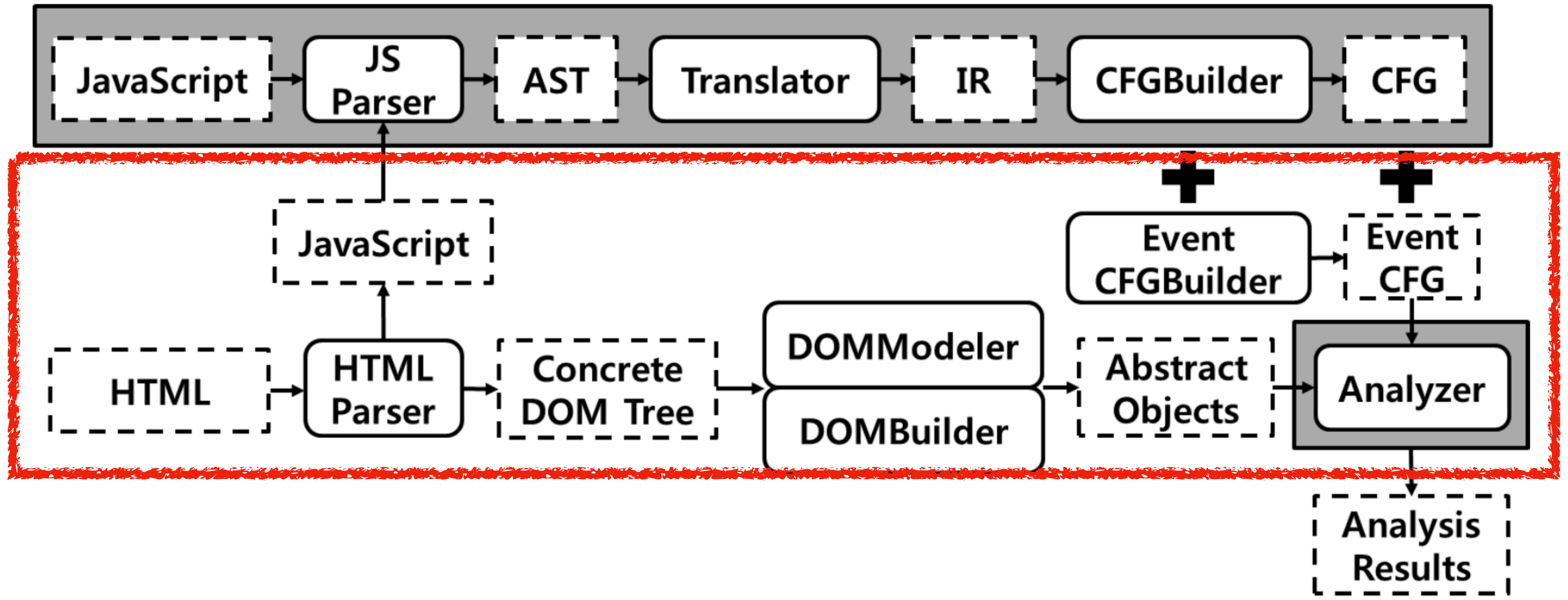
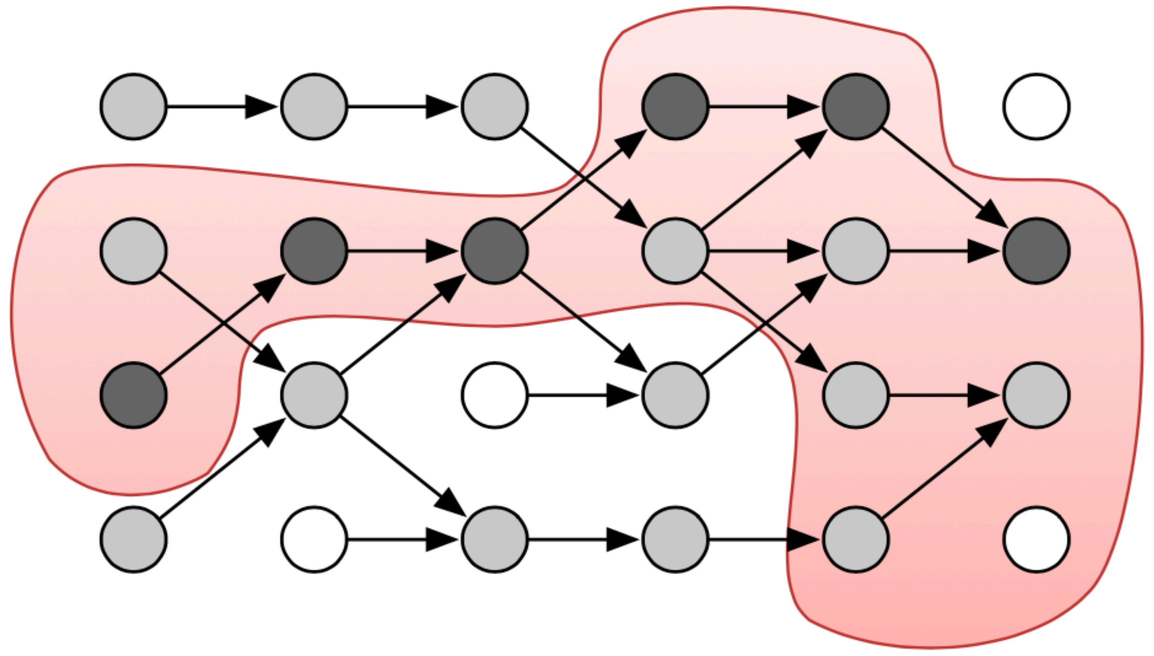


Fig. 3: Overall structure of the $\mathsf{SAFE}_{\mathsf{WApp}}$ framework

Practically Tunable Static Analysis Framework for Large-Scale JavaScript Applications



Collecting Semantics. We denote a program $\mathcal{P}$ as a directed graph $\langle \mathbb{C}, \mathbb{E} \rangle$ where $\mathbb{C}$ and $\mathbb{E}$ represent a set of nodes and a set of edges, respectively. A node $c \in \mathbb{C}$ denotes a control point in the program and an edge $(c, c') \in \mathbb{E} \subseteq \mathbb{C} \times \mathbb{C}$ denotes a control flow from c to c'.

Given a program $\mathcal{P}$, we define its collecting semantics using $\langle \mathbb{C}, \wp(\mathbb{S}), f, \rightarrow_\phi \rangle$ where:

- $\mathbb{C}$: a finite set of control points;
- $\sigma \in \wp(\mathbb{S})$: a powerset of concrete states;
- $f(c) \in \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$: a set of local semantic functions at a given control point c;
- $\phi \in \mathbb{C} \rightarrow \wp(\mathbb{S})$: a set of program states mapping control points to sets of concrete states; and
- $\rightarrow_\phi$: a set of control flows for a given program state ϕ

$$
\begin{tikzcd}
\mathbb{C} \to \wp(\mathbb{S}) \arrow[r, shift right, "\alpha_m"'] \arrow[dr, shift right, "\alpha_s"'] & \mathbb{C} \to \hat{\mathbb{S}} \arrow[l, shift right, "\gamma_m"'] \\
& \dot{\mathbb{X}} \arrow[u, "\mathcal{I}"'] \arrow[ul, shift right, "\gamma_s"']
\end{tikzcd}
$$

C. *Tuned Static Analysis*

Finally, using the baseline analysis $\hat{F}$ and the contour $\dot{\mathcal{C}}$, our framework derives a tuned static analysis which over-approximates the selected set of program executions approximated by $\dot{\mathcal{C}}$. Using the sound abstract semantic function $\hat{f}$ from the baseline analysis and the interpretation of the contour $\mathcal{I}(\dot{\mathcal{C}})$, we can define a tuned analysis which over-approximates selected executions as the least fixpoint of the following abstract semantic function:

$$\hat{F}_s(\hat{\phi}) = \lambda c \in \mathbb{C}.\ \hat{f}(c)\left(\bigsqcup_{c' \overset{s}{\hookrightarrow}_{\hat{\phi}} c} \hat{\phi}(c') \right) \sqcap \mathcal{I}(\dot{\mathcal{C}}),$$

Theorem 1 (Correctness of $\hat{F}_s$): $\alpha_m(\mathbf{lfp} F_s) \sqsubseteq \mathbf{lfp} \hat{F}_s$

Also, the interpretation of the contour $\mathcal{I}(\dot{\mathcal{C}})$ is an over-approximation of the derived tuned analysis of the selected executions:

Theorem 2: $\mathbf{lfp} \hat{F}_s \sqsubseteq \mathcal{I}(\dot{\mathcal{C}})$

SAFE$_{WAPI}$: Web API Misuse Detector for Web Applications

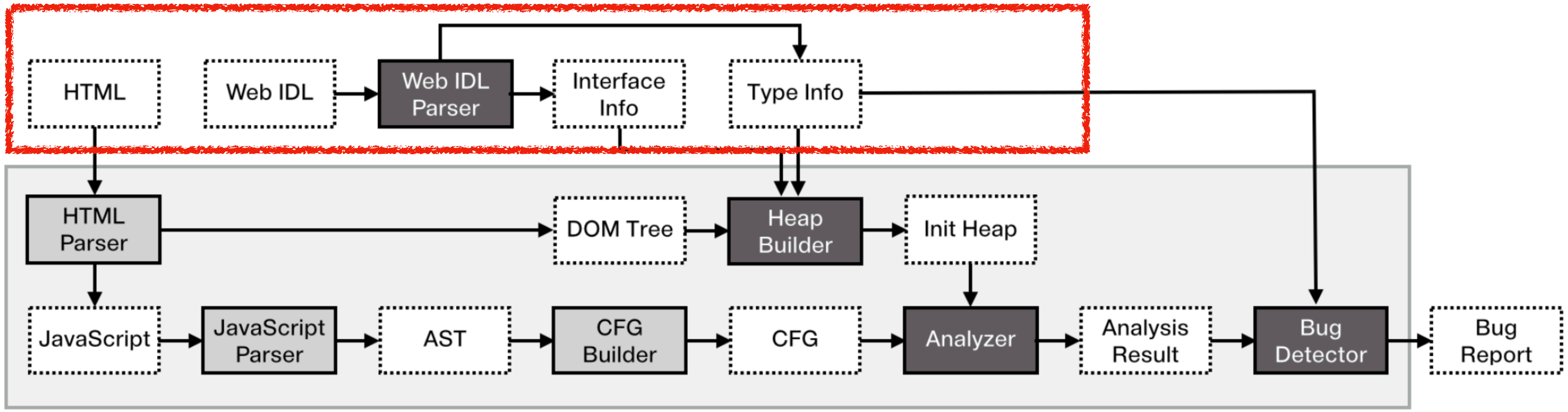


Figure 4: Architecture of SAFE$_{WAPI}$

```
[Callback=FunctionOnly, NoInterfaceObject]
interface CalendarArraySuccessCallback {
  void onsuccess(Calendar[] calendars);
};
```

```
[NoInterfaceObject] interface Calendar {

  readonly attribute CalendarId id;

  readonly attribute DOMString name;

  CalendarItem get(CalendarItemId id);

  void add(CalendarItem item);
```

Figure 1: Web API specification written in Web IDL

1. Accesses to absent properties of platform objects (AbsProp)
2. Wrong number of arguments to API function calls (ArgNum)
3. Missing error callback functions (ErrorCB)
4. Unhandled API calls that may throw exceptions (ExnHnd)
5. Wrong types of arguments to API function calls (ArgTyp)
6. Accesses to absent attributes of dictionary objects (AbsAttr)

Automatic Modeling of Opaque Code for JavaScript Static Analysis

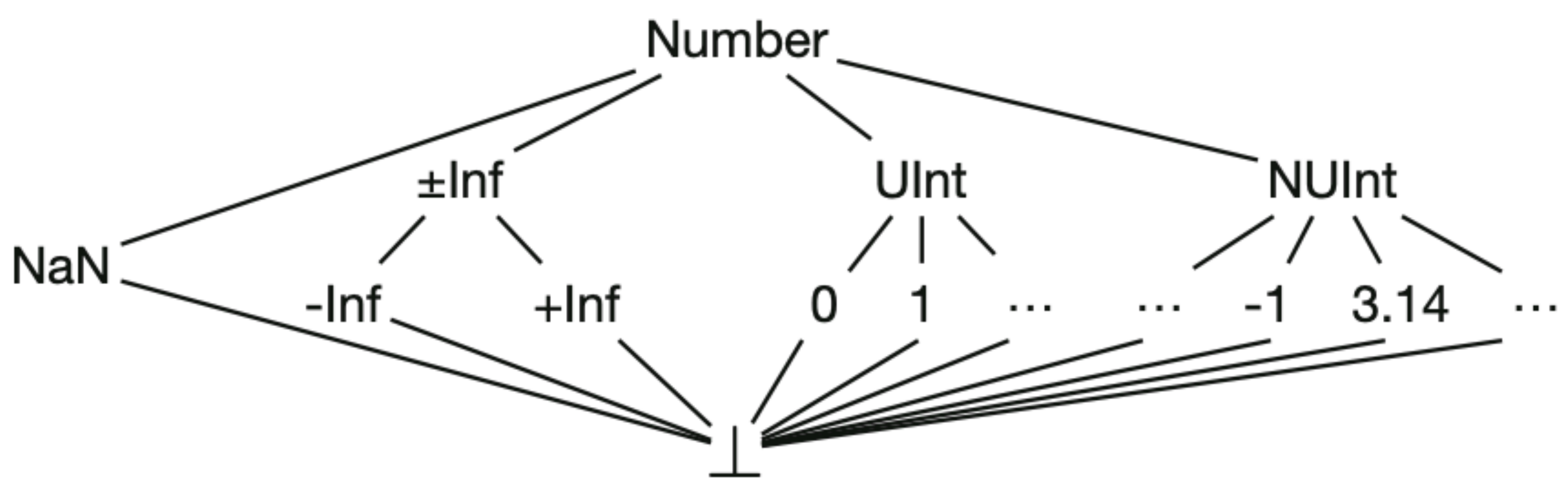


Fig. 2. The SAFE number domain for JavaScript

We write the SRA model as $\Downarrow_{SRA}: C \times \widehat{S} \rightarrow \widehat{S}$ and define it as follows:

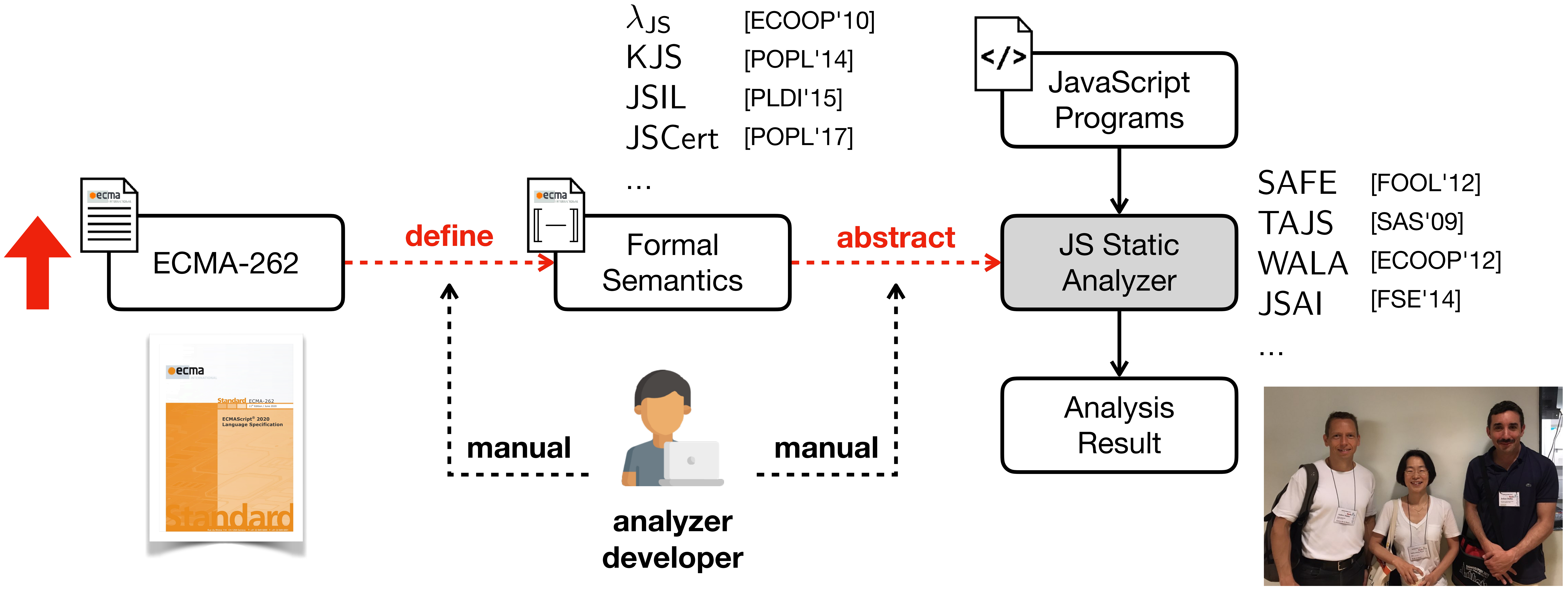
$$\begin{aligned}\Downarrow_{SRA}(c, \widehat{s}) &= \quad Abstract(\{Run(c,s) \quad \mid s \in Sample(\widehat{s})\}) \\ &= Broaden(\bigsqcup\{\alpha(\{Run(c,s)\}) \mid s \in Sample(\widehat{s})\})\end{aligned}$$

We now describe how $\Downarrow_{SRA}$ works using an example abstract domain for even and odd integers as shown in Fig. 1. Let us consider the code snippet `x := abs(x)` at a program point c where the library function `abs` is opaque. We use maps from variables to their concrete values for concrete states, maps from variables to their abstract values for abstract states, and the identity function for *Broaden* in this example.

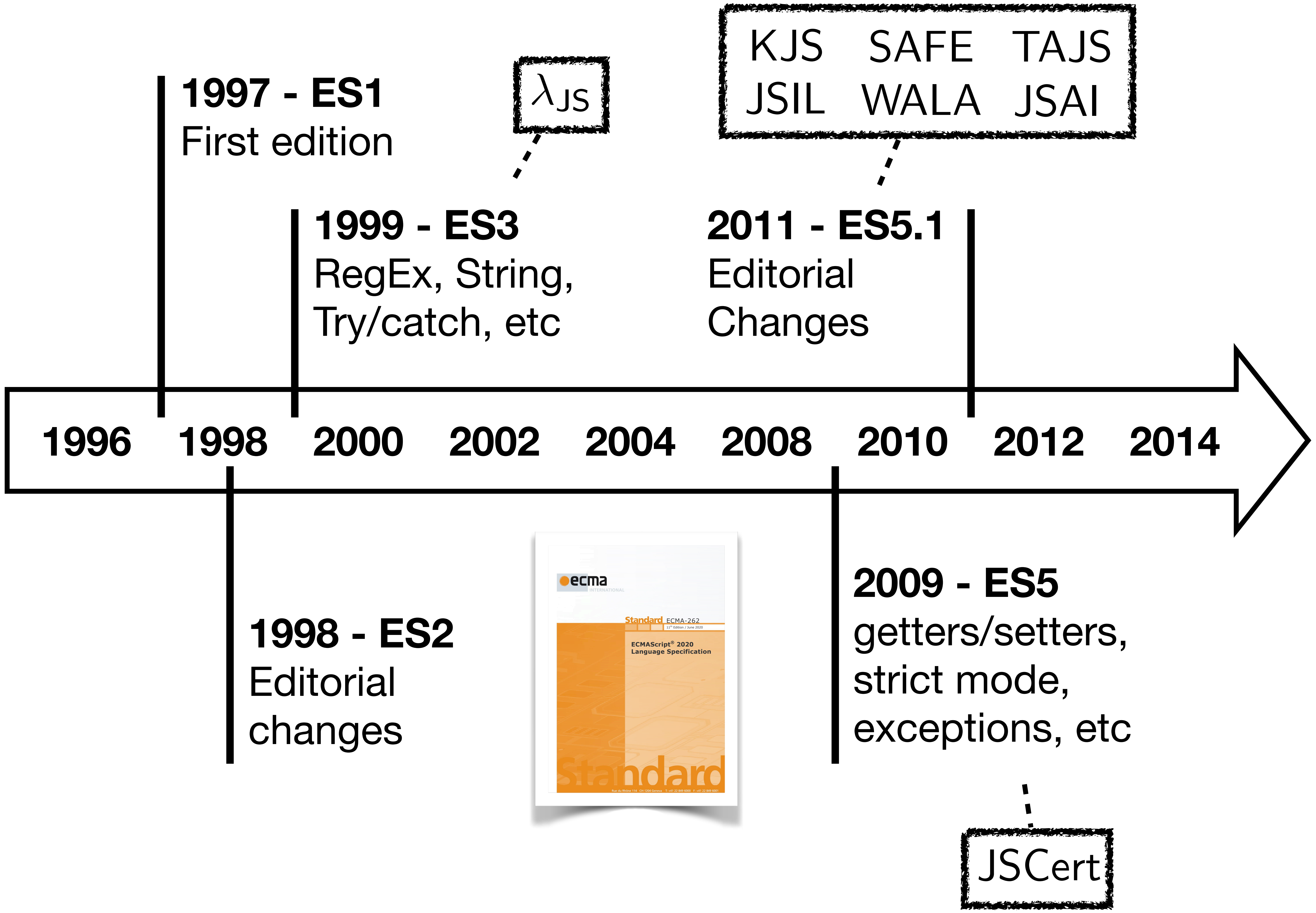
Case $\widehat{s}_1 \equiv [\mathtt{x} : n]$ *where* n *is a constant integer:*

$$\begin{aligned}\Downarrow_{SRA}(c, \widehat{s}_1) &= \bigsqcup\{\alpha(\{Run(c,s)\}) \mid s \in Sample(\widehat{s}_1)\} \\ &= \bigsqcup\{\alpha(\{Run(c,s)\}) \mid s \in \{[\mathtt{x} : n]\}\} \\ &= \bigsqcup\{\alpha(\{Run(c,[\mathtt{x} : n])\})\} \\ &= \bigsqcup\{\alpha(\{[\mathtt{x} : |n|]\})\} \\ &= [\mathtt{x} : |n|]\end{aligned}$$

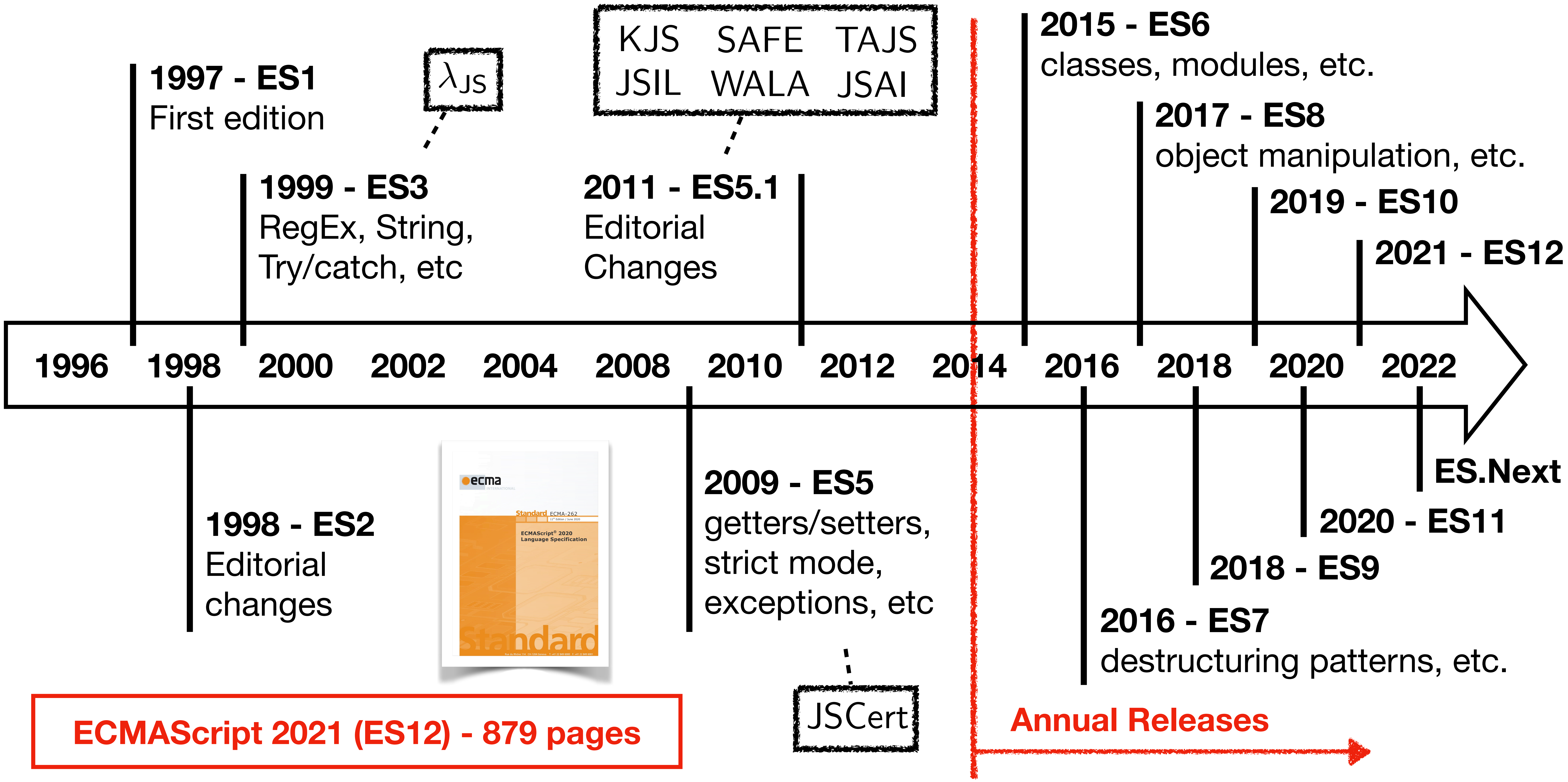
Problem: Hand-Written JavaScript Static Analyzer



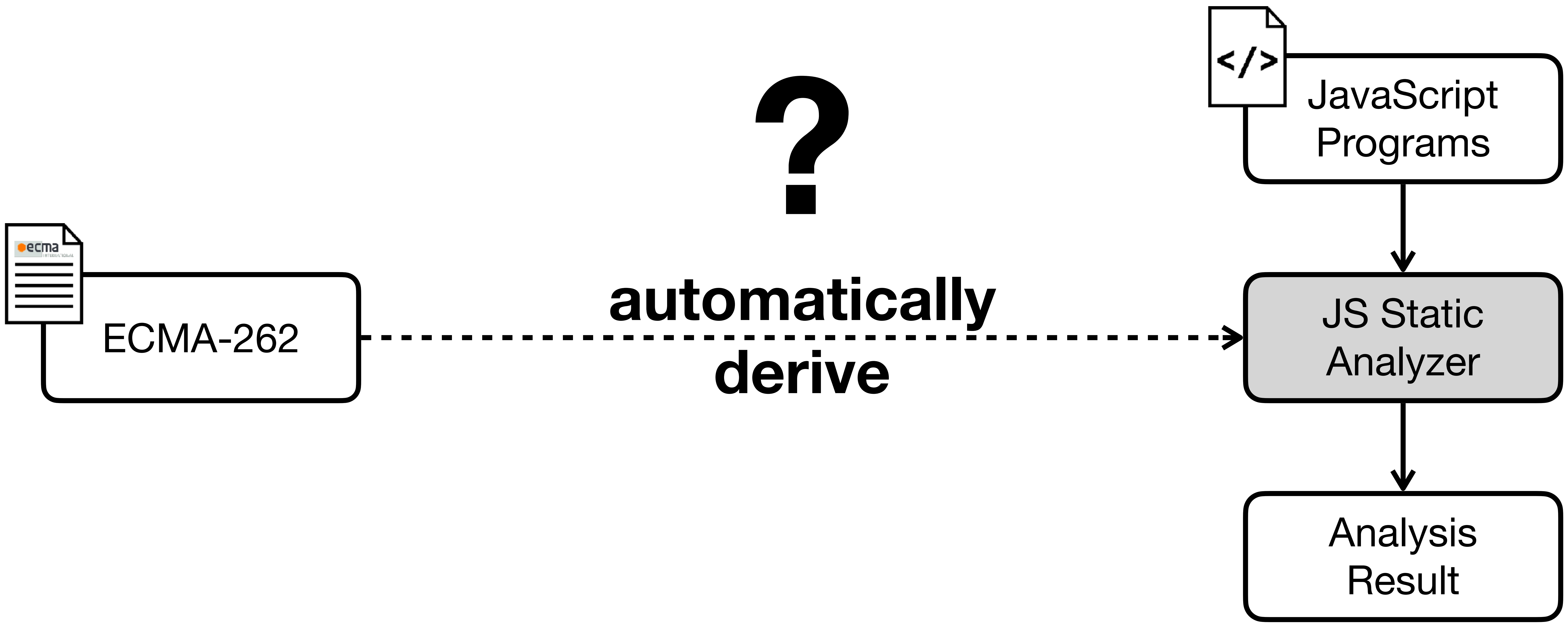
Problem: Fast Evolving JavaScript



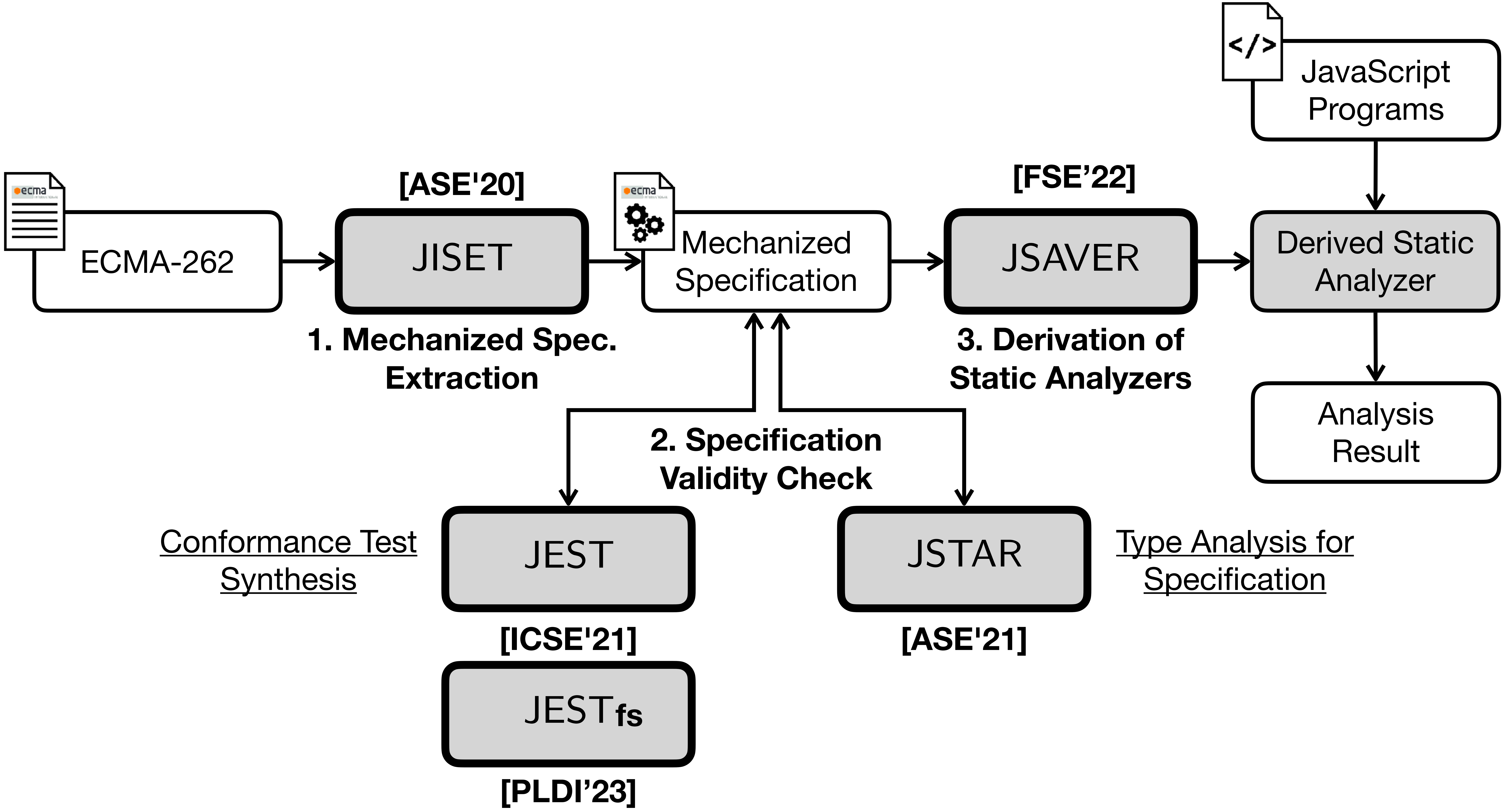
Problem: Fast Evolving JavaScript



Main Idea: Deriving Static Analyzer from Spec.

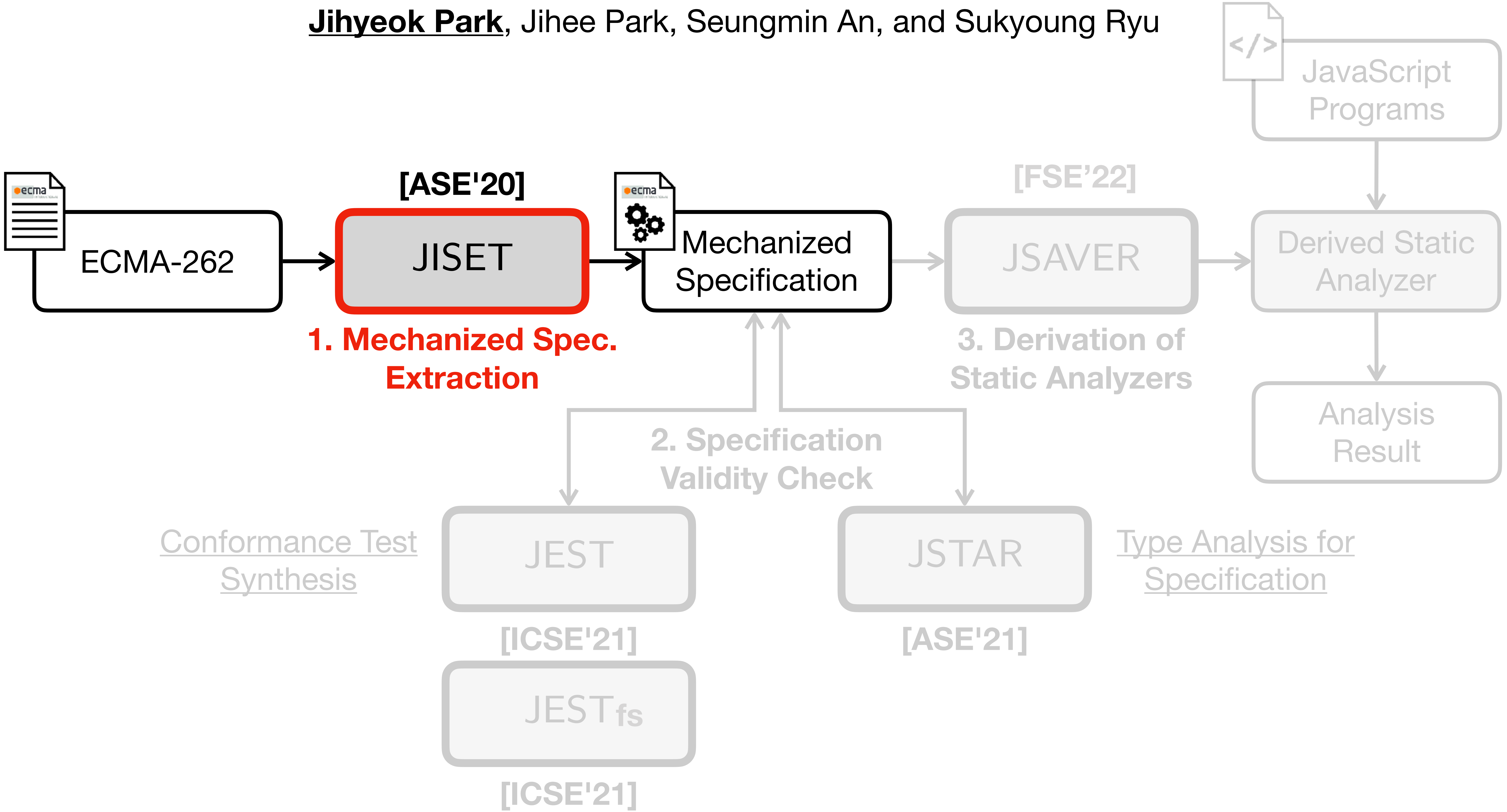


Overall Structure



JISET: JavaScript IR-based Semantics Extraction Toolchain

Jihyeok Park, Jihee Park, Seungmin An, and Sukyoung Ryu



Motivation: Patterns in Writing Style of ECMA-262

13.2.5.2 Runtime Semantics: Evaluation

ArrayLiteral : **[** *ElementList* **,** *Elision*$_{opt}$ **]**

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
    a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments *array* and *nextIndex*.
    b. ReturnIfAbrupt(*len*).
5. Return *array*.

The Evaluation **algorithm for**
the third alternative of *ArrayLiteral* **in ES12**

Key Idea: Metalanguage for ECMA-262

- **IR**$_{\textbf{ES}}$ - **I**ntermediate **R**epresentation for **E**CMA**S**cript

Programs	$\mathfrak{P} \ni P ::= f^*$
Functions	$\mathcal{F} \ni f ::= \texttt{syntax}^? \ \texttt{def} \ \texttt{x}(\texttt{x}^*) \ \{[\ell : i]^*\}$
Variables	$\mathcal{X} \ni \texttt{x}$
Labels	$\mathcal{L} \ni \ell$
Instructions	$\mathcal{I} \ni i ::= r := e \mid \texttt{x} := \{\} \mid \texttt{x} := e(e^*)$ $\mid \ \texttt{if} \ e \ \ell \ \ell \mid \texttt{return} \ e$
Expressions	$\mathcal{E} \ni e ::= v^{\text{p}} \mid \texttt{op}(e^*) \mid r$
References	$\mathcal{R} \ni r ::= \texttt{x} \mid e[e] \mid e[e]_{\text{js}}$

States	$\sigma \in \mathbb{S}$	$= \mathcal{L} \times \mathbb{E} \times \mathbb{C}^* \times \mathbb{H}$
Environments	$\rho \in \mathbb{E}$	$= \mathcal{X} \xrightarrow{\text{fin}} \mathbb{V}$
Calling Contexts	$c \in \mathbb{C}$	$= \mathcal{L} \times \mathbb{E}$
Heaps	$h \in \mathbb{H}$	$= \mathbb{A} \xrightarrow{\text{fin}} \mathcal{L} \times \mathbb{M} \times \mathbb{M}_{\text{js}}$
Internal Field Maps	$m \in \mathbb{M}$	$= \mathbb{V}_{\text{str}} \xrightarrow{\text{fin}} \mathbb{V}$
External Field Maps	$m_{\text{js}} \in \mathbb{M}_{\text{js}}$	$= \mathbb{V}_{\text{str}} \xrightarrow{\text{fin}} \mathbb{V}$
Values	$v \in \mathbb{V}$	$= \mathbb{A} \uplus \mathbb{V}^{\text{p}} \uplus \mathbb{T} \uplus \mathcal{F}$
Primitive Values	$v^{\text{p}} \in \mathbb{V}^{\text{p}}$	$= \mathbb{V}_{\text{bool}} \uplus \mathbb{V}_{\text{int}} \uplus \mathbb{V}_{\text{str}} \uplus \cdots$
$\boxed{\text{JS ASTs}}$	$\boxed{t \in \mathbb{T}}$	

Key Idea: Metalanguage for ECMA-262

13.2.5.2 Runtime Semantics: Evaluation

ArrayLiteral **:** **[** *ElementList* **,** *Elision*$_{opt}$ **]**

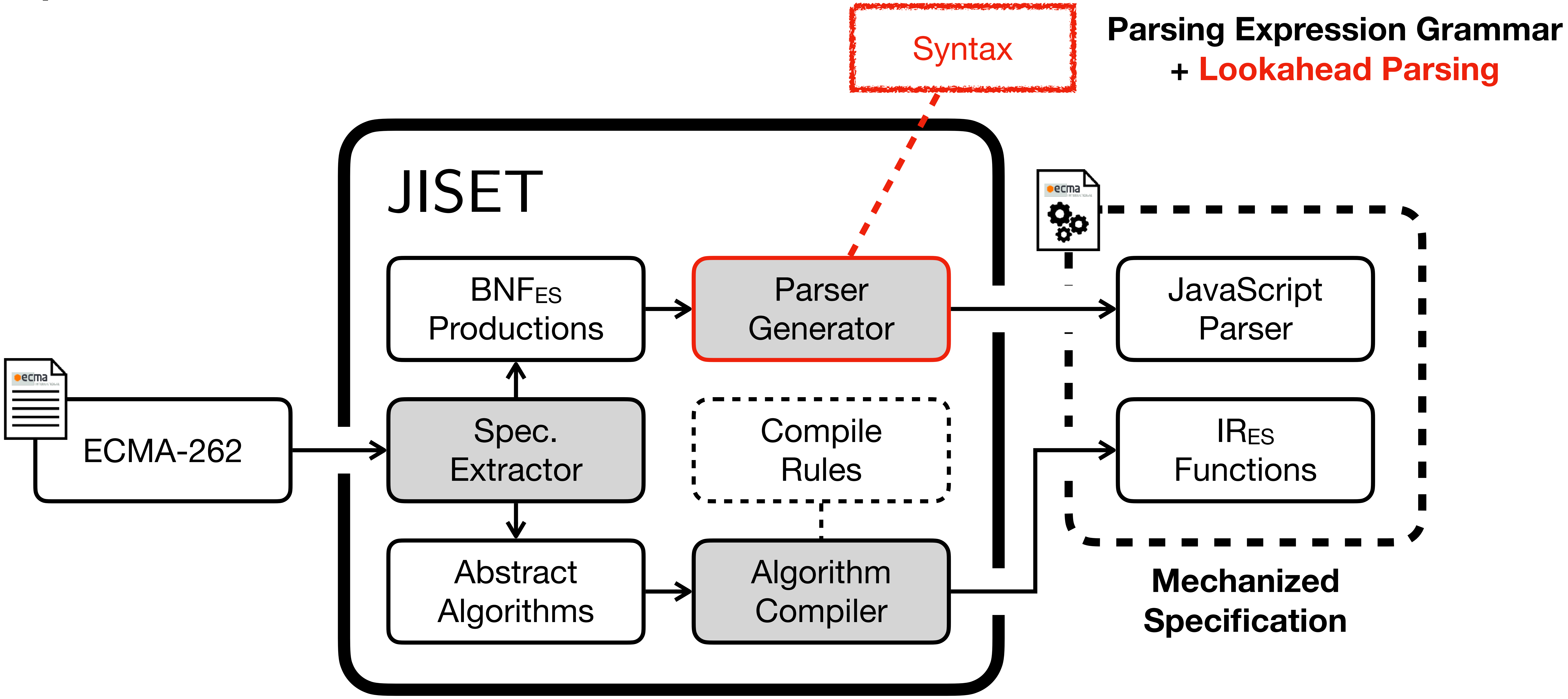
1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
   a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments *array* and *nextIndex*.
   b. ReturnIfAbrupt(*len*).
5. Return *array*.

compile

```
syntax def ArrayLiteral[2].Evaluation(
  this, ElementList, Elision
) {
  let array = [! (ArrayCreate 0)]
  let nextIndex = (ElementList.ArrayAccumulation array 0)
  [? nextIndex]
  if (! (= Elision absent)) {
    let len = (Elision.ArrayAccumulation array nextIndex)
    [? len]
  }
  return array
}
```

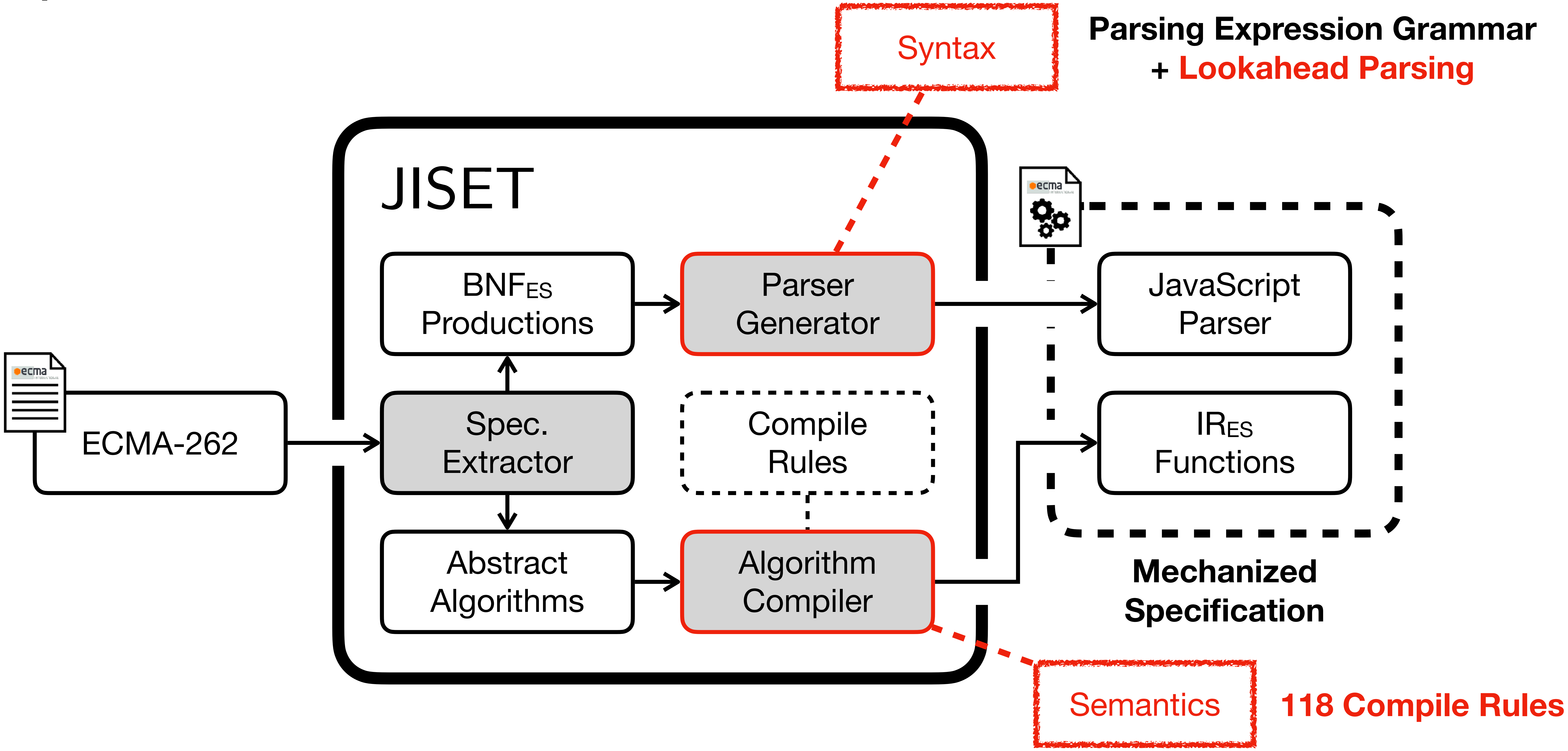
JISET [ASE'20]

<u>J</u>avaScript <u>I</u>R-based <u>S</u>emantics <u>E</u>xtraction <u>T</u>oolchain



JISET [ASE'20]

JavaScript IR-based Semantics Extraction Toolchain



JISET - Evaluation

≈ 96% Compiled

auto / manual

T: Total **L**: Core Language Semantics **B**: Built-in Libraries

Version	# Algo.		
ES7	2,105	T	**10,471 / 10,982** (95.35%)
		L	**8,041 / 8,415** (95.56%)
		B	**2,430 / 2,567** (94.66%)
ES8	2,238	T	**11,181 / 11,732** (95.30%)
		L	**8,453 / 8,811** (95.94%)
		B	**2,728 / 2,921** (93.39%)
ES9	2,370	T	**11,849 / 12,393** (95.61%)
		L	**8,932 / 9,311** (95.93%)
		B	**2,917 / 3,082** (94.65%)
ES10	2,396	T	**12,022 / 12,569** (95.65%)
		L	**9,073 / 9,456** (94.95%)
		B	**2,949 / 3,113** (94.73%)
ES11	2,521	T	**12,505 / 13,047** (94.85%)
		L	**9,495 / 9,881** (96.09%)
		B	**3,010 / 3,166** (95.07%)
ES12	2,640	T	**12,975 / 13,544** (95.80%)
		L	**9,717 / 10,136** (95.87%)
		B	**3,258 / 3,408** (95.60%)
Average	2,378	T	**11,834 / 12,378** (95.61%)
		L	**8,952 / 9,335** (95.90%)
		B	**2,882 / 3,043** (94.71%)

JISET - Evaluation

≈ 96% Compiled

Passed All Tests

auto manual

T: Total L: Core Language Semantics B: Built-in Libraries

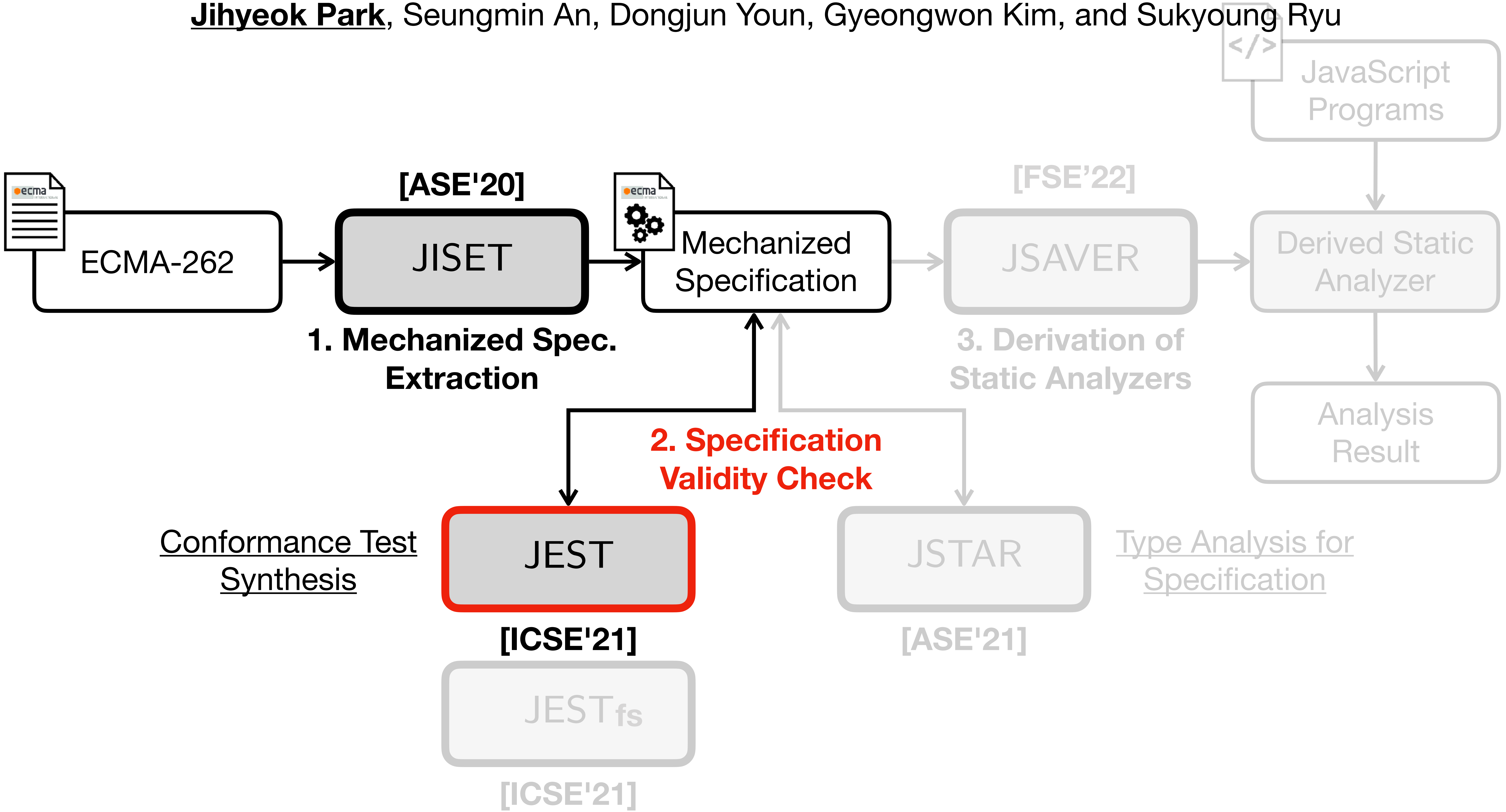
Version	# Algo.		
ES7	2,105	T	**10,471 / 10,982** (95.35%)
		L	**8,041 / 8,415** (95.56%)
		B	**2,430 / 2,567** (94.66%)
ES8	2,238	T	**11,181 / 11,732** (95.30%)
		L	**8,453 / 8,811** (95.94%)
		B	**2,728 / 2,921** (93.39%)
ES9	2,370	T	**11,849 / 12,393** (95.61%)
		L	**8,932 / 9,311** (95.93%)
		B	**2,917 / 3,082** (94.65%)
ES10	2,396	T	**12,022 / 12,569** (95.65%)
		L	**9,073 / 9,456** (94.95%)
		B	**2,949 / 3,113** (94.73%)
ES11	2,521	T	**12,505 / 13,047** (94.85%)
		L	**9,495 / 9,881** (96.09%)
		B	**3,010 / 3,166** (95.07%)
ES12	2,640	T	**12,975 / 13,544** (95.80%)
		L	**9,717 / 10,136** (95.87%)
		B	**3,258 / 3,408** (95.60%)
Average	2,378	T	**11,834 / 12,378** (95.61%)
		L	**8,952 / 9,335** (95.90%)
		B	**2,882 / 3,043** (94.71%)

Complete Missing Parts

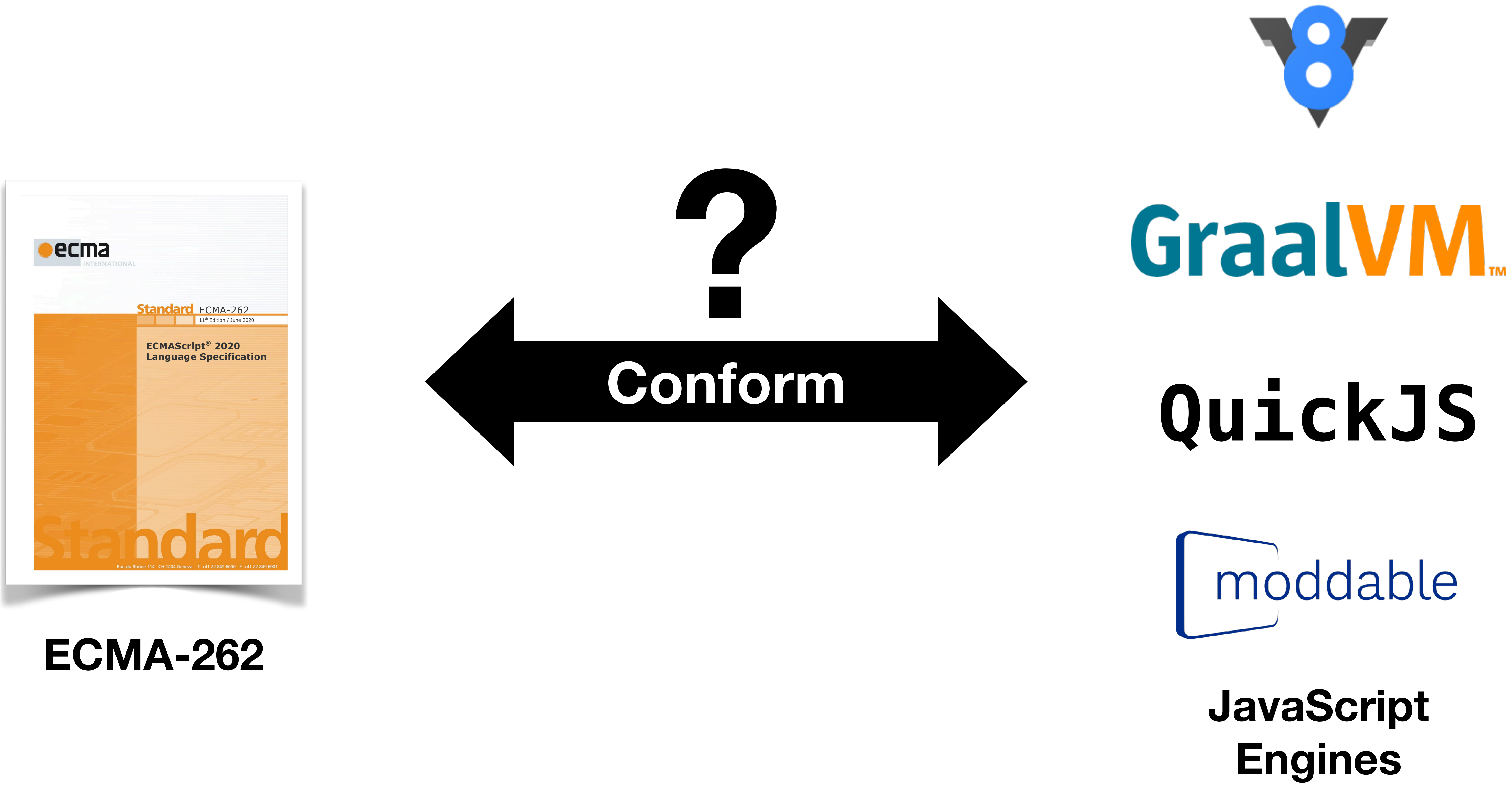
- **Test262** (Official Conformance Tests)
  - 18,556 applicable tests
- **Parsing tests**
  - Passed all 18,556 tests
- **Evaluation Tests**
  - Passed all 18,556 tests

JEST: N+1-version Differential Testing of Both JavaScript Engines

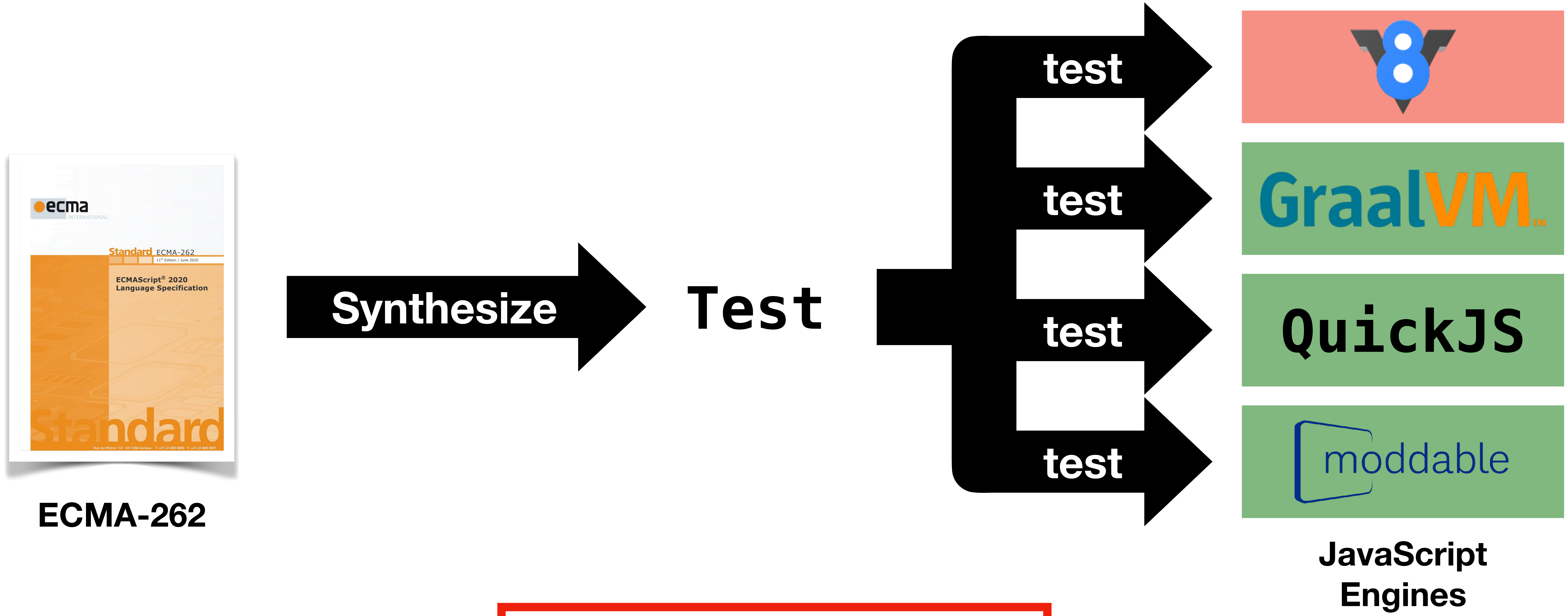
Jihyeok Park, Seungmin An, Dongjun Youn, Gyeongwon Kim, and Sukyoung Ryu



JEST - Conformance with Engines

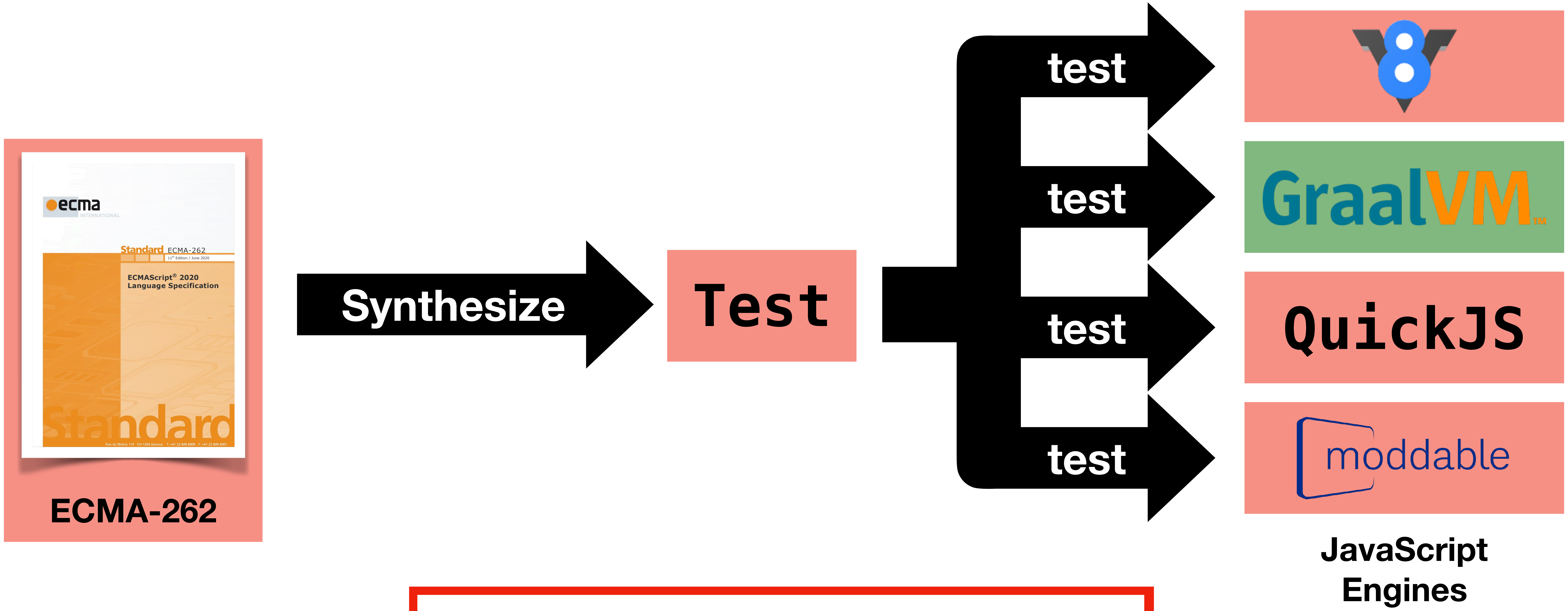


JEST - N+1-version Differential Testing



An engine bug in

JEST - N+1-version Differential Testing

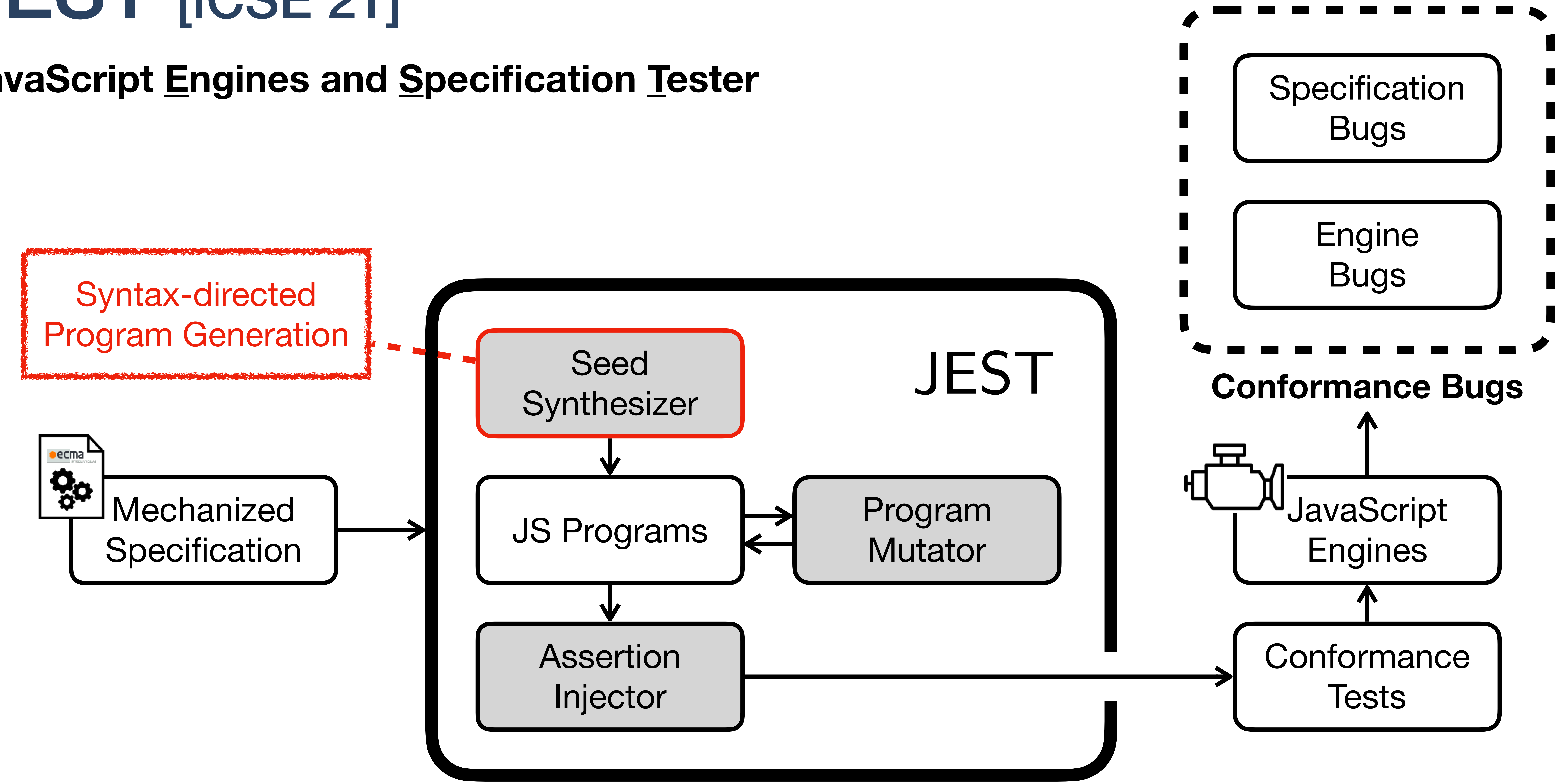


A <u>specification</u> bug in ECMA-262

An <u>engine</u> bug in GraalVM

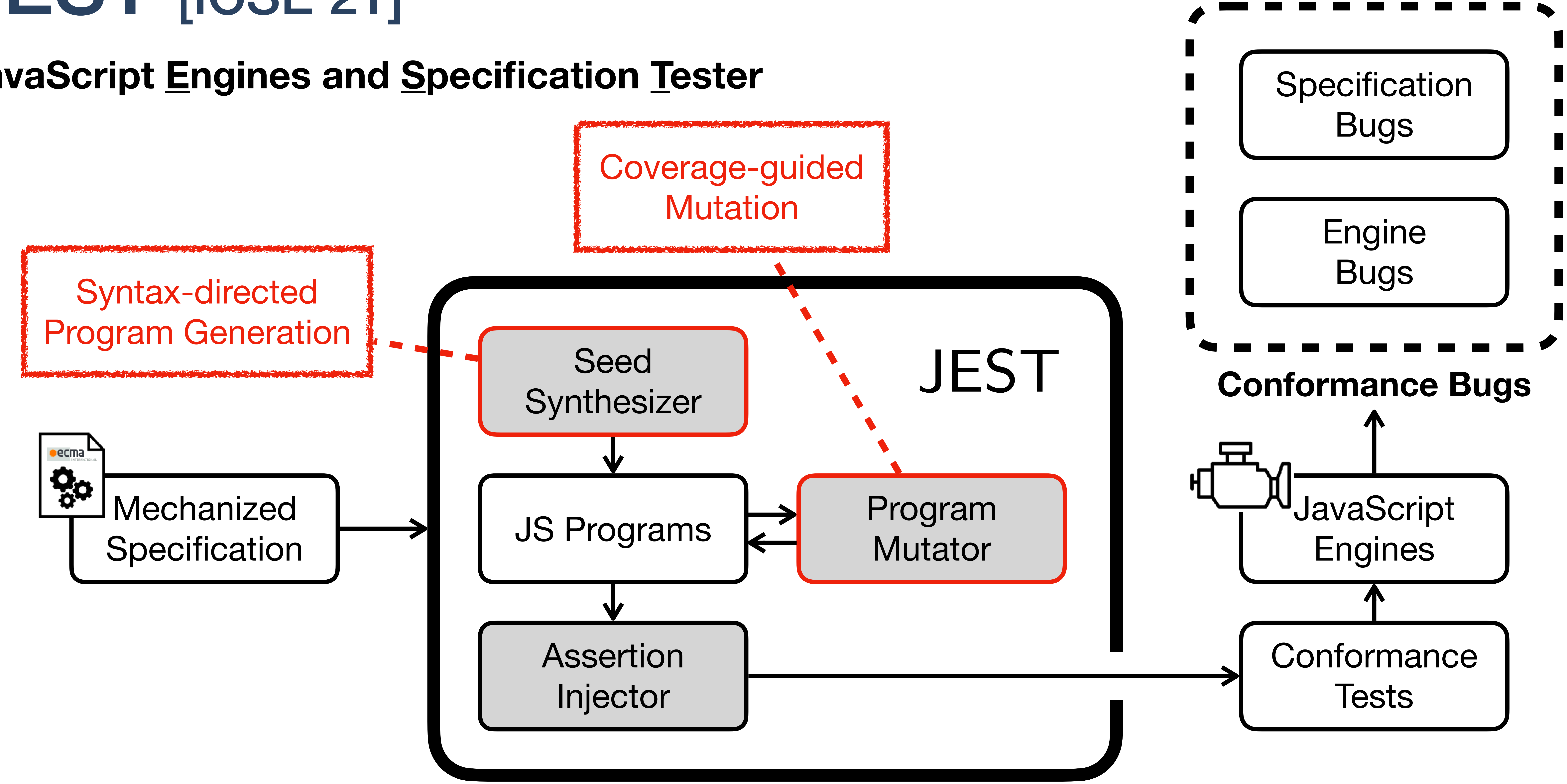
JEST [ICSE'21]

JavaScript Engines and Specification Tester



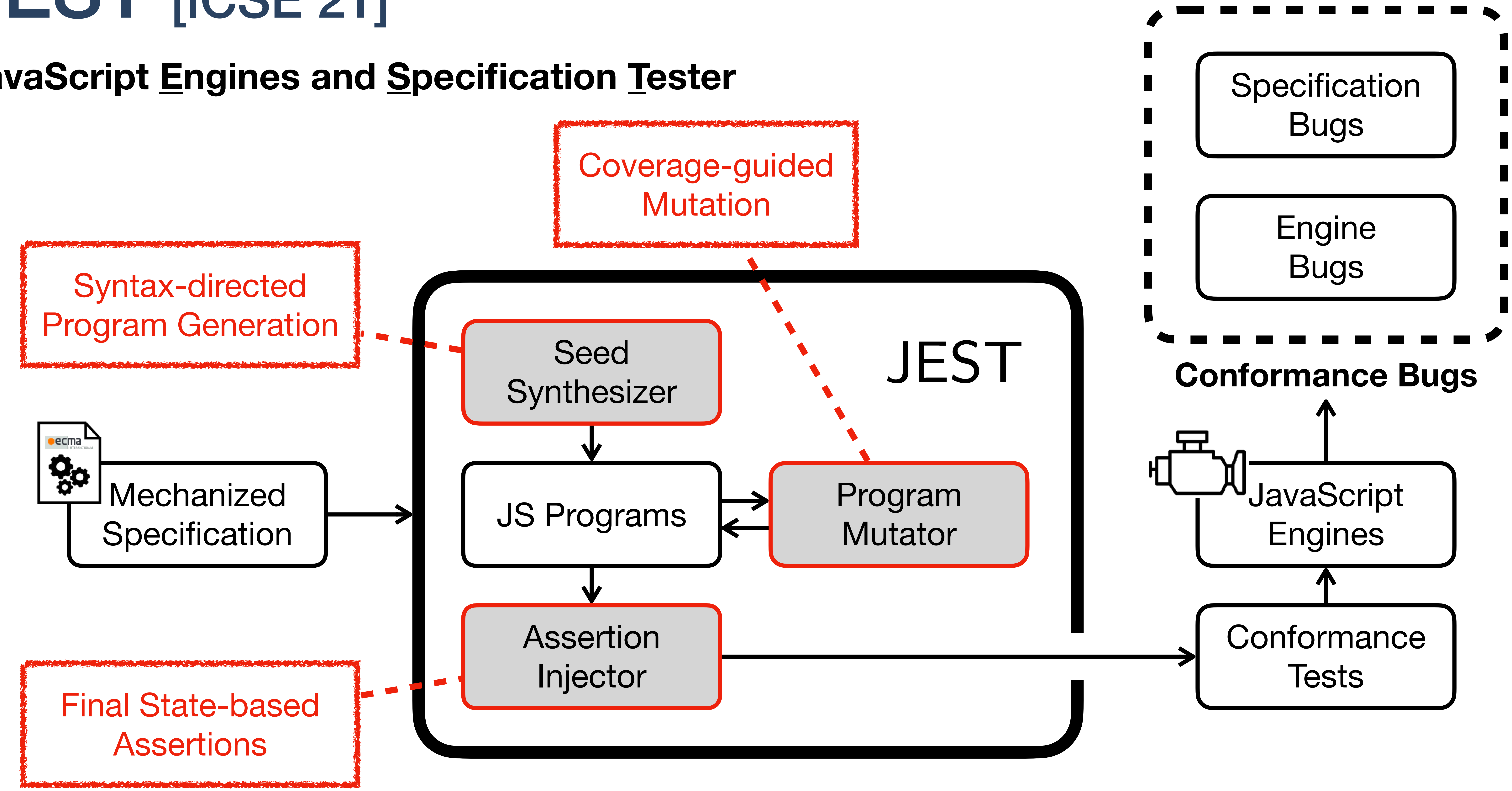
JEST [ICSE'21]

JavaScript Engines and Specification Tester



JEST [ICSE'21]

JavaScript Engines and Specification Tester



JEST - Assertion Injector (7 Kinds)

```
  var x = 1 + 2;

+ $assert.sameValue(x, 3);
```

JEST - Assertion Injector (7 Kinds)

1. **Exceptions (**`Exc`**)**

```
+ // Throw
  let x = 42;
  function x() {};
```

---

2. **Aborts (**`Abort`**)**

```
+ // Abort
  var x = 42; x++;
```

---

3. **Variable Values (**`Var`**)**

```
  var x = 1 + 2;
+ $assert.sameValue(x, 3);
```

---

4. **Object Values (**`Obj`**)**

```
  var x = {}, y = {}, z = { p: x, q: y };
+ $assert.sameValue(z.p, x);
+ $assert.sameValue(z.q, y);
```

JEST - Assertion Injector (7 Kinds)

5. **Object Properties** (`Desc`)

```
  var x = { p: 42 };
+ $verifyProperty(x, "p", {
+   value: 42.0, writable: true,
+   enumerable: true, configurable: true
+ });
```

6. **Property Keys** (`Key`)

```
  var x = {[Symbol.match]: 0, p: 0, 3: 0, q: 0, 1: 0}
+ $assert.compareArray(
+   Reflect.ownKeys(x),
+   ["1", "3", "p", "q", Symbol.match]
+ );
```

7. **Internal Methods and Slots** (`In`)

```
  function f() {}
+ $assert.sameValue(Object.getPrototypeOf(f),
+                   Function.prototype);
+ $assert.sameValue(Object.isExtensible(x), true);
+ $assert.callable(f);
+ $assert.constructable(f);
```

JEST - Evaluation

- **JEST successfully synthesized 1,700 conformance tests from ES11**

44 Bugs in Engines

TABLE II: The number of engine bugs detected by JEST

Engines	Exc	Abort	Var	Obj	Desc	Key	In	**Total**
V8	0	0	0	0	0	2	0	2
GraalVM	6	0	0	0	2	8	0	16
QuickJS	3	0	1	0	0	2	0	6
Moddable XS	12	0	0	0	3	5	0	20
Total	21	0	1	0	5	17	0	44

JEST - Evaluation

- **JEST successfully synthesized 1,700 conformance tests from ES11**

44 Bugs in Engines

TABLE II: The number of engine bugs detected by JEST

Engines	Exc	Abort	Var	Obj	Desc	Key	In	Total
V8	0	0	0	0	0	2	0	2
GraalVM	6	0	0	0	2	8	0	16
QuickJS	3	0	1	0	0	2	0	6
Moddable XS	12	0	0	0	3	5	0	20
Total	21	0	1	0	5	17	0	44

27 Bugs in Spec.

TABLE III: Specification bugs in ECMAScript 2020 (ES11) detected by JEST

Name	Feature	#	Assertion	Known	Created	Resolved	Existed
ES11-1	Function	12	Key	O	2019-02-07	2020-04-11	429 days
ES11-2	Function	8	Key	O	2015-06-01	2020-04-11	1,776 days
ES11-3	Loop	1	Exc	O	2017-10-17	2020-04-30	926 days
ES11-4	Expression	4	Abort	O	2019-09-27	2020-04-23	209 days
ES11-5	Expression	1	Exc	O	2015-06-01	2020-04-28	1,793 days
ES11-6	Object	1	Exc	X	2019-02-07	2020-11-05	637 days

JEST - Example in GraalVM

Crash

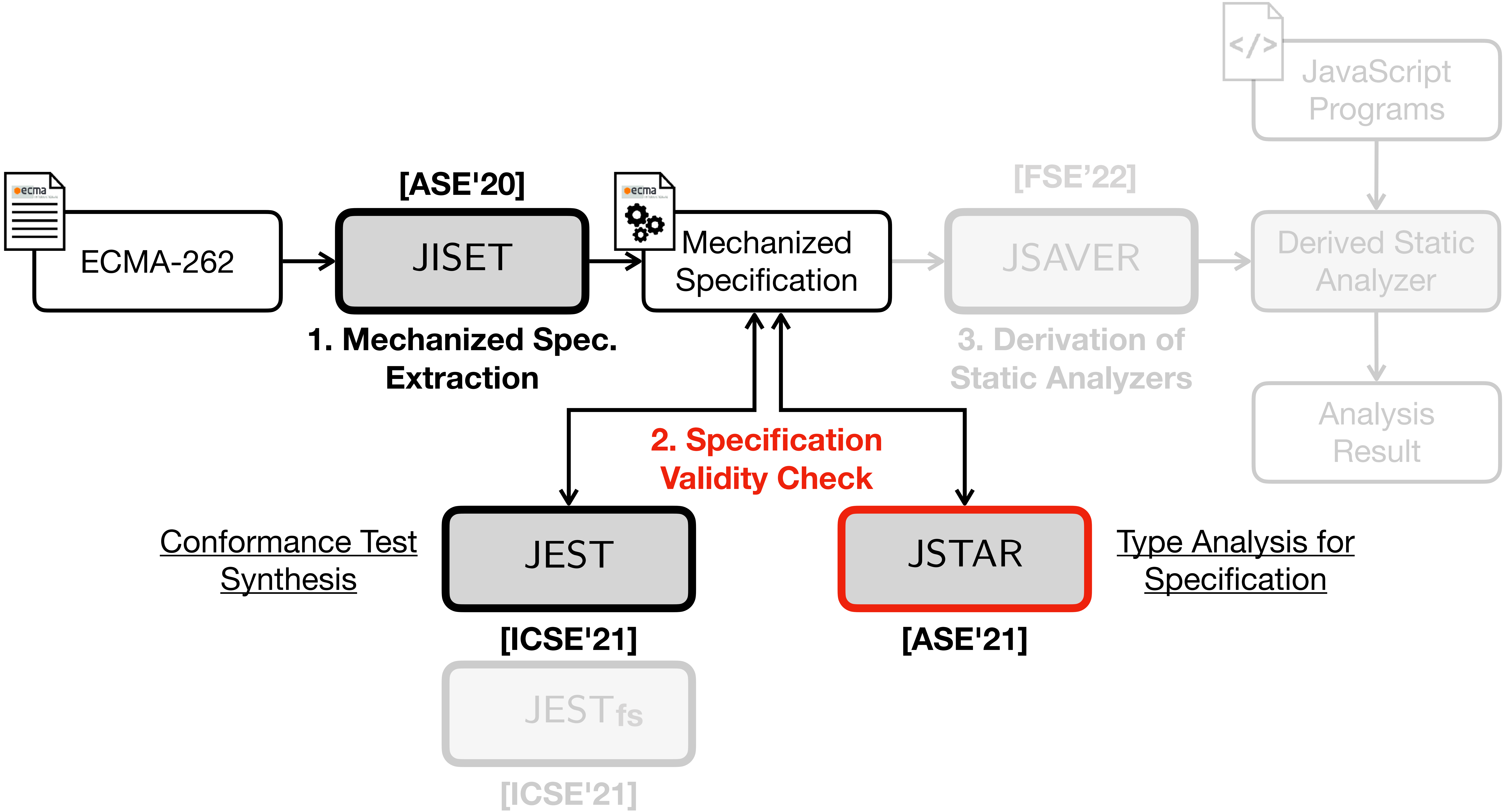
```
try { ++undefined; } catch(e) { }
```

"Right now, we are running Test262 and the V8 and Nashorn unit test suites in our CI for every change, it might make sense to add your suite as well."

- A Developer of GraalVM

JSTAR: JavaScript Specification Type Analyzer using Refinement

Jihyeok Park, Seungmin An, Wonho Shin, Yusung Sim, and Sukyoung Ryu



JSTAR - Types in Specification

20.3.2.28 Math.round (x)

1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

…

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Types in Specification

20.3.2.28 Math.round ($\boxed{x}$) `x: (String v Boolean v Number v Object v ...)`

1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Types in Specification

20.3.2.28 Math.round ($\boxed{x}$) `x: (String ∨ Boolean ∨ Number ∨ Object ∨ ...)`

1. Let $\boxed{n}$ be ? ToNumber(x). `n: (Number) ∧ ToNumber(x): (Number ∨ Exception)`
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

…

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Types in Specification

20.3.2.28 Math.round ($\boxed{x}$) `x: (String ∨ Boolean ∨ Number ∨ Object ∨ ...)`

1. Let $\boxed{n}$ be ? ToNumber(x). `n: (Number) ∧ ToNumber(x): (Number ∨ Exception)`
2. If n is an integral Number, return n.
3. If $\boxed{x < 0.5}$ and $\boxed{x > 0}$, return **+0**.
4. If $\boxed{x < 0}$ and $\boxed{x \geq -0.5}$, return **-0**.

...

Type Mismatch for numeric operator \`>\`

```
Math.round(true)  = ???
Math.round(false) = ???
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Types in Specification

20.3.2.28 Math.round ($\boxed{x}$) `x: (String ∨ Boolean ∨ Number ∨ Object ∨ ...)`

1. Let $\boxed{n}$ be ? ToNumber(x). `n: (Number) ∧ ToNumber(x): (Number ∨ Exception)`
2. If n is an integral Number, return n.
3. If $\boxed{x < 0.5}$ and $\boxed{x > 0}$, return **+0**.
4. If $\boxed{x < 0}$ and $\boxed{x \geq \text{-}0.5}$, return **-0**.

...

Type Mismatch for numeric operator `>`

```
Math.round(true)  = ???
Math.round(false) = ???
```

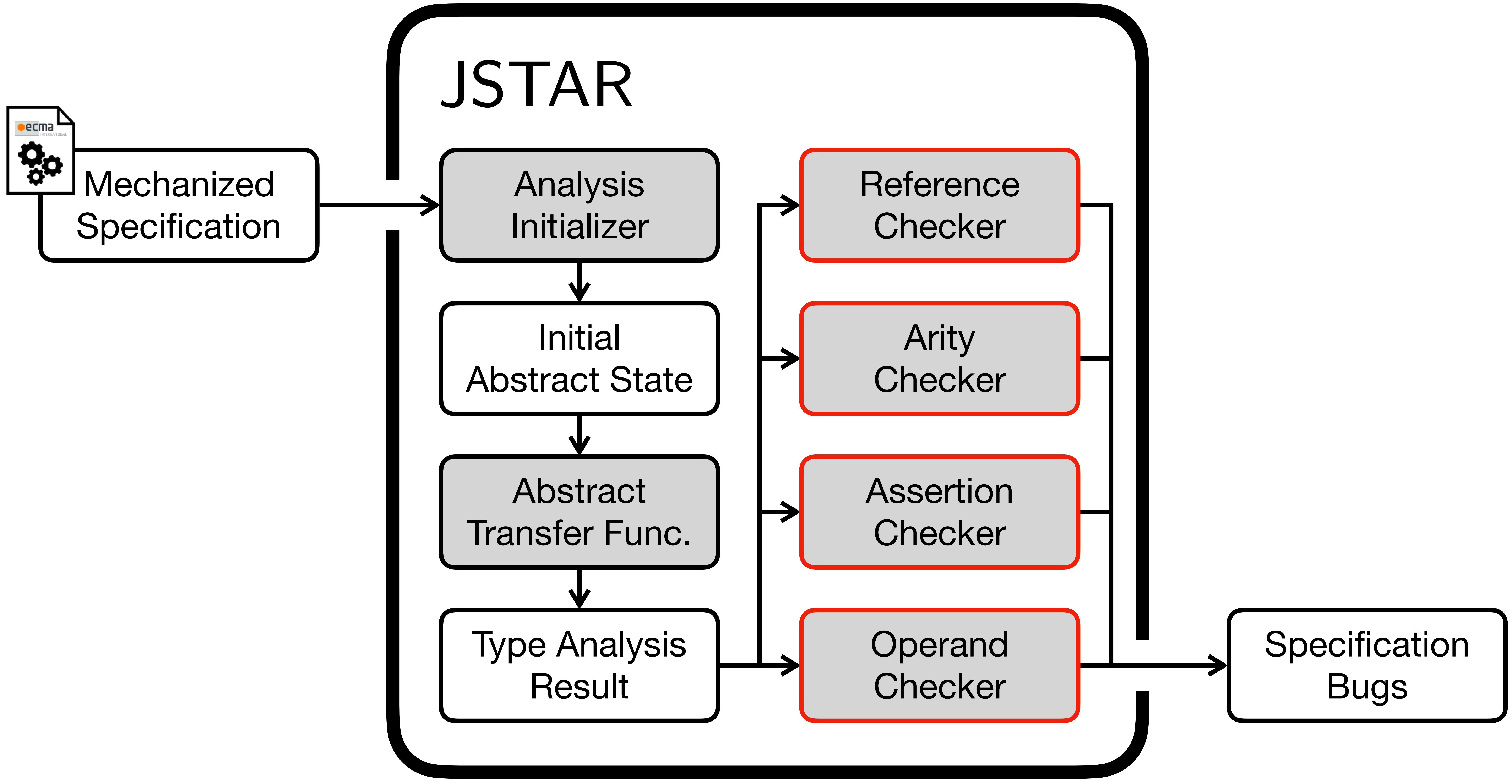
3. If $n < 0.5$ and $n > 0$, return **+0**.
4. If $n < 0$ and $n \geq \text{-}0.5$, return **-0**.

```
Math.round(true)  = 1
Math.round(false) = 0
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

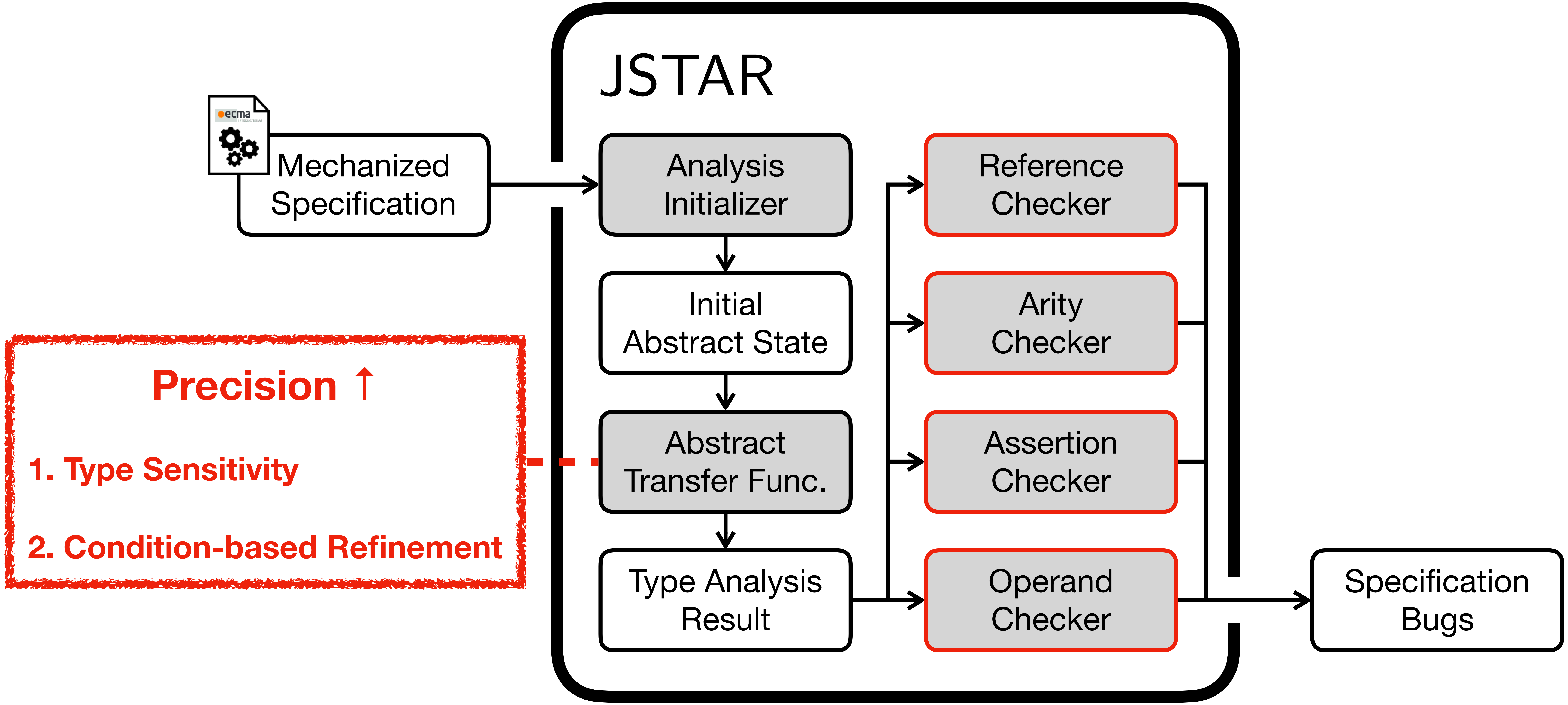
JSTAR [ASE'21]

JavaScript Specification Type Analyzer using Refinement



JSTAR [ASE'21]

<u>J</u>avaScript <u>S</u>pecification <u>T</u>ype <u>A</u>nalyzer using <u>R</u>efinement



Intermediate Representation for ECMAScript

$$
\begin{array}{llcl}
\text{Functions} & \mathbb{F} \ni f & ::= & \texttt{def}\ \texttt{x}\,\texttt{(}\texttt{x}^*,\ \texttt{[}\texttt{x}^*\texttt{]}\texttt{)}\ \ell \\
\text{Instructions} & \mathbb{I} \ni i & ::= & \texttt{let}\ \texttt{x} = e \mid \texttt{x} = \texttt{(}e\ e^*\texttt{)} \mid \texttt{assert}\ e \\
 & & & \mid \texttt{if}\ e\ \ell\ \ell \mid \texttt{return}\ e \mid r = e \\
\text{References} & r & ::= & \texttt{x} \mid r\,\texttt{[}e\texttt{]} \\
\text{Expressions} & e & ::= & t\ \texttt{\{}[\texttt{x} : e]^*\texttt{\}} \mid \texttt{[}e^*\texttt{]} \mid e : \tau \mid r\texttt{?} \\
 & & & \mid e \oplus e \mid \ominus\, e \mid r \mid c \mid p \\
\text{Primitives} & \mathbb{P} \ni p & ::= & \texttt{undefined} \mid \texttt{null} \mid b \mid n \mid j \mid s \mid \texttt{@}s \\
\text{Types} & \mathbb{T} \ni \tau & ::= & t \mid \texttt{[]} \mid \texttt{[}\tau\texttt{]} \mid \texttt{js} \mid \texttt{prim} \\
 & & & \mid \texttt{undefined} \mid \texttt{null} \mid \texttt{bool} \mid \texttt{numeric} \\
 & & & \mid \texttt{num} \mid \texttt{bigint} \mid \texttt{str} \mid \texttt{symbol}
\end{array}
$$

Semantic Domains

States	$d \in \mathbb{S}$	$= \mathbb{L} \times \mathbb{C}^* \times \mathbb{H} \times \mathbb{E}$
Contexts	$\kappa \in \mathbb{C}$	$= \mathbb{L} \times \mathbb{E} \times \mathbb{X}$
Heaps	$h \in \mathbb{H}$	$= \mathbb{A} \to \mathbb{O}$
Addresses	$a \in \mathbb{A}$	
Objects	$o \in \mathbb{O}$	$= (\mathbb{T}_t \times (\mathbb{V}_s \to \mathbb{V})) \uplus \mathbb{V}^*$
Nominal Types	$t \in \mathbb{T}_t$	
Environments	$\sigma \in \mathbb{E}$	$= \mathbb{X} \times \mathbb{V}$
Values	$v \in \mathbb{V}$	$= \mathbb{F} \uplus \mathbb{A} \uplus \mathbb{V}_c \uplus \mathbb{P}$
Constants	$c \in \mathbb{V}_c$	
Strings	$s \in \mathbb{V}_s$	

Semantics of Instructions

B. Instructions: $[\![i]\!]_i : \mathbb{S} \to \mathbb{S}$

- Variable Declarations:

$$[\![\texttt{let x} = e]\!]_i(d) = (\texttt{next}(\ell), \overline{\kappa}, h, \sigma[\texttt{x} \mapsto v])$$

where

$$[\![e]\!]_e(d) = ((\ell, \overline{\kappa}, h, \sigma), v)$$

- Function Calls:

$$[\![\texttt{x} = \texttt{(}e_0\ e_1 \cdots e_n\texttt{)}]\!]_i(d) = (\ell_{\texttt{f}}, \kappa :: \overline{\kappa}, h, \sigma')$$

where

$$\begin{aligned}
&[\![e_0]\!]_e(d) = (d_0, \texttt{def f(p}_1, \cdots, \texttt{p}_m\texttt{)}\ \ell_{\texttt{f}} \wedge \\
&[\![e_1]\!]_e(d_0) = (d_1, v_1) \wedge \cdots \wedge [\![e_n]\!]_e(d_{n-1}) = (d_n, v_n) \wedge \\
&d_n = (\ell, \overline{\kappa}, h, \sigma) \wedge k = \min(n, m) \wedge \\
&\sigma' = [\texttt{p}_1 \mapsto v_1, \cdots, \texttt{p}_k \mapsto v_k] \wedge \kappa = (\texttt{next}(\ell), \sigma, \texttt{x})
\end{aligned}$$

- Assertions:

$$[\![\texttt{assert}\ e]\!]_i(d) = d' \quad \text{if } [\![e]\!]_e(d) = (d', \texttt{\#t})$$

- Branches:

$$[\![\texttt{if}\ e\ \ell_{\texttt{t}}\ \ell_{\texttt{f}}]\!]_i(d) = \begin{cases} (\ell_{\texttt{t}}, \overline{\kappa}, h, \sigma) & \text{if } v = \texttt{\#t} \\ (\ell_{\texttt{f}}, \overline{\kappa}, h, \sigma) & \text{if } v = \texttt{\#f} \end{cases}$$

where

$$[\![e]\!]_e(d) = ((\ell_{\texttt{t}}, \overline{\kappa}, h, \sigma), v)$$

- Returns:

$$[\![\texttt{return}\ e]\!]_i(d) = (\ell, \overline{\kappa}, h, \sigma[\texttt{x} \mapsto v])$$

where

$$[\![e]\!]_e(d) = ((\_, (\ell, \sigma, \texttt{x}) :: \overline{\kappa}, h, \_), v)$$

- Variable Updates:

$$[\![\texttt{x} = e]\!]_i(d) = (\texttt{next}(\ell), \overline{\kappa}, h, \sigma[\texttt{x} \mapsto v])$$

where

$$[\![e]\!]_e(d) = ((\ell, \overline{\kappa}, h, \sigma), v)$$

- Field Updates:

$$[\![r\texttt{[}e_0\texttt{]} = e_1]\!]_i(d) = (\texttt{next}(\ell), \overline{\kappa}, h[a \mapsto o'], \sigma)$$

where

$$\begin{aligned}
&[\![r]\!]_e(d) = (d', a) \wedge [\![e_0]\!]_e(d') = (d_0, v_0) \wedge \\
&[\![e_1]\!]_e(d_0) = ((\ell, \overline{\kappa}, h, \sigma), v_1) \wedge o = h(a) \wedge \\
&o' = \begin{cases} o_r & \text{if } o = (t, \texttt{fs}) \wedge v_0 = s \\ o_l & \text{if } o = [v'_1, \cdots, v'_m] \wedge v_0 = n \end{cases} \wedge \\
&o_r = (t, \texttt{fs}[s \mapsto v_1]) \wedge o_l = [\cdots, v'_{n-1}, v_1, v'_{n+1}, \cdots]
\end{aligned}$$

Semantics of Expressions

D. Expressions: $[\![e]\!]_e : \mathbb{S} \to \mathbb{S} \times \mathbb{V}$

- Records:

$$[\![t \ \{\mathtt{x}_1 : e_1, \cdots, \mathtt{x}_n : e_n\}]\!]_e(d) = (d', a)$$

where

$$\begin{aligned}&[\![e_1]\!]_e(d) = (d_1, v_1) \wedge \cdots \wedge [\![e_n]\!]_e(d_{n-1}) = (d_n, v_n) \wedge \\ &d_n = (\ell, \overline{\kappa}, h, \sigma) \wedge \mathtt{fs} = [\mathtt{x}_1 \mapsto v_1, \cdots, \mathtt{x}_n \mapsto v_n] \\ &a \notin \mathrm{Domain}(h) \wedge d' = (\ell, \overline{\kappa}, h[a \mapsto (t, \mathtt{fs})], \sigma)\end{aligned}$$

- Lists:

$$[\![\mathtt{[}e_1, \cdots, e_n\mathtt{]}]\!]_e(d) = (d', a)$$

where

$$\begin{aligned}&[\![e_1]\!]_e(d) = (d_1, v_1) \wedge \cdots \wedge [\![e_n]\!]_e(d_{n-1}) = (d_n, v_n) \wedge \\ &d_n = (\ell, \overline{\kappa}, h, \sigma) \wedge a \notin \mathrm{Domain}(h) \wedge \\ &d' = (\ell, \overline{\kappa}, h[a \mapsto [v_1, \cdots, v_n]], \sigma)\end{aligned}$$

- Type Checks:

$$[\![e : \tau]\!]_e(d) = (d', b)$$

where

$$[\![e]\!]_e(d) = (d', v) \wedge b = \begin{cases} \mathtt{\#t} & \text{if } v \text{ is a value of } \tau \\ \mathtt{\#f} & \text{otherwise} \end{cases}$$

- Variable Existence Checks:

$$[\![\mathtt{x?}]\!]_e(d) = (d, b)$$

where

$$d = (\_, \_, \_, \sigma) \wedge b = \begin{cases} \mathtt{\#t} & \text{if } \mathtt{x} \in \mathrm{Domain}(\sigma) \\ \mathtt{\#f} & \text{otherwise} \end{cases}$$

- Field Existence Checks:

$$[\![r\mathtt{[}e\mathtt{]?}]\!]_e(d) = (d'', b)$$

where

$$\begin{aligned}&[\![r]\!]_e(d) = (d', a) \wedge [\![e]\!]_e(d') = (d'', v) \wedge \\ &d'' = (\ell, \overline{\kappa}, h, \sigma) \wedge o = h(a) \wedge \\ &b = \begin{cases} \mathtt{\#t} & \text{if } o = (t, \mathtt{fs}) \wedge v = s \wedge s \in \mathrm{Domain}(\mathtt{fs}) \\ \mathtt{\#t} & \text{if } o = [v'_1, \cdots, v'_m] \wedge v = n \wedge 1 \le n \le m \\ \mathtt{\#f} & \text{otherwise} \end{cases}\end{aligned}$$

- Binary Operations:

$$[\![e \oplus e]\!]_e(d) = (d'', v_0 \oplus v_1)$$

where

$$[\![e_0]\!]_e(d) = (d', v_0) \wedge [\![e_1]\!]_e(d') = (d'', v_1)$$

- Unary Operations:

$$[\![\ominus\ e]\!]_e(d) = (d', \ominus\ v)$$

where

$$[\![e]\!]_e(d) = (d', v)$$

- References:

$$[\![r]\!]_e(d) = [\![r]\!]_r(d)$$

- Constants:

$$[\![c]\!]_e(d) = (d, c)$$

- Primitives:

$$[\![p]\!]_e(d) = (d, p)$$

Abstract Semantic Domains

Abstract States	$d^\sharp \in \mathbb{S}^\sharp = \mathbb{M} \times \mathbb{R}$
Result Maps	$m \in \mathbb{M} = \mathbb{L} \times \mathbb{T}^* \to \mathbb{E}^\sharp$
Return Point Maps	$r \in \mathbb{R} = \mathbb{F} \times \mathbb{T}^* \to \mathcal{P}(\mathbb{L} \times \mathbb{T}^* \times \mathbb{X})$
Abstract Environments	$\sigma^\sharp \in \mathbb{E}^\sharp = \mathbb{X} \to \mathbb{T}^\sharp$
Abstract Types	$\tau^\sharp \in \mathbb{T}^\sharp = \mathcal{P}(\mathbb{T})$

Abstract Semantic Functions

Then, we define the abstract semantics $[\![P]\!]^\sharp$ of a program P as the least fixpoint of the abstract transfer $F^\sharp : \mathbb{S}^\sharp \to \mathbb{S}^\sharp$:

$$[\![P]\!]^\sharp = \lim_{n\to\infty}(F^\sharp)^n(d^\sharp_\iota)$$
$$F^\sharp(d^\sharp) = d^\sharp \sqcup \left(\bigsqcup_{(\ell,\overline{\tau})\in \mathrm{Domain}(m)} [\![\mathtt{inst}(\ell)]\!]^\sharp_i(\ell,\overline{\tau})(d^\sharp)\right)$$

where $d^\sharp = (m, \_)$ and $d^\sharp_\iota$ denotes the initial abstract state.

B. Instructions: $[\![i]\!]^\sharp_i : (\mathbb{L} \times \mathbb{T}^*) \to \mathbb{S}^\sharp \to \mathbb{S}^\sharp$

- Variable Declarations:

$$[\![\mathtt{let\ x} = e]\!]^\sharp_i(\ell,\overline{\tau})(d^\sharp) = (\{(\mathtt{next}(\ell),\overline{\tau}) \mapsto \sigma^\sharp_{\mathrm{x}}\}, \varnothing)$$

where

$$d^\sharp = (m, \_) \wedge \sigma^\sharp = m(\ell,\overline{\tau}) \wedge$$
$$\sigma^\sharp_{\mathrm{x}} = \sigma^\sharp[\mathrm{x} \mapsto [\![e]\!]^\sharp_e(\sigma^\sharp)]$$

JSTAR - Evaluation

59.2% Precision

93 Bugs Detected

- **Type Analysis for 864 versions of ECMA-262 in 3 years**

Checker	**Bug Kind**	**Precision = (# True Bugs) / (# Detected Bugs)**					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25	/ -25
Operand	NoNumber	22 / 113	2 / 65	22 / 44	2 / 6	/ -69	/ -59
	Abrupt		20 / 48		20 / 38		/ -10
Total		92 / 279 (33.0%)		93 / 157 (59.2%)		+1 / -122 (+26.3%)	

JSTAR - Evaluation

59.2% Precision

93 Bugs Detected

- **Type Analysis for 864 versions of ECMA-262 in 3 years**

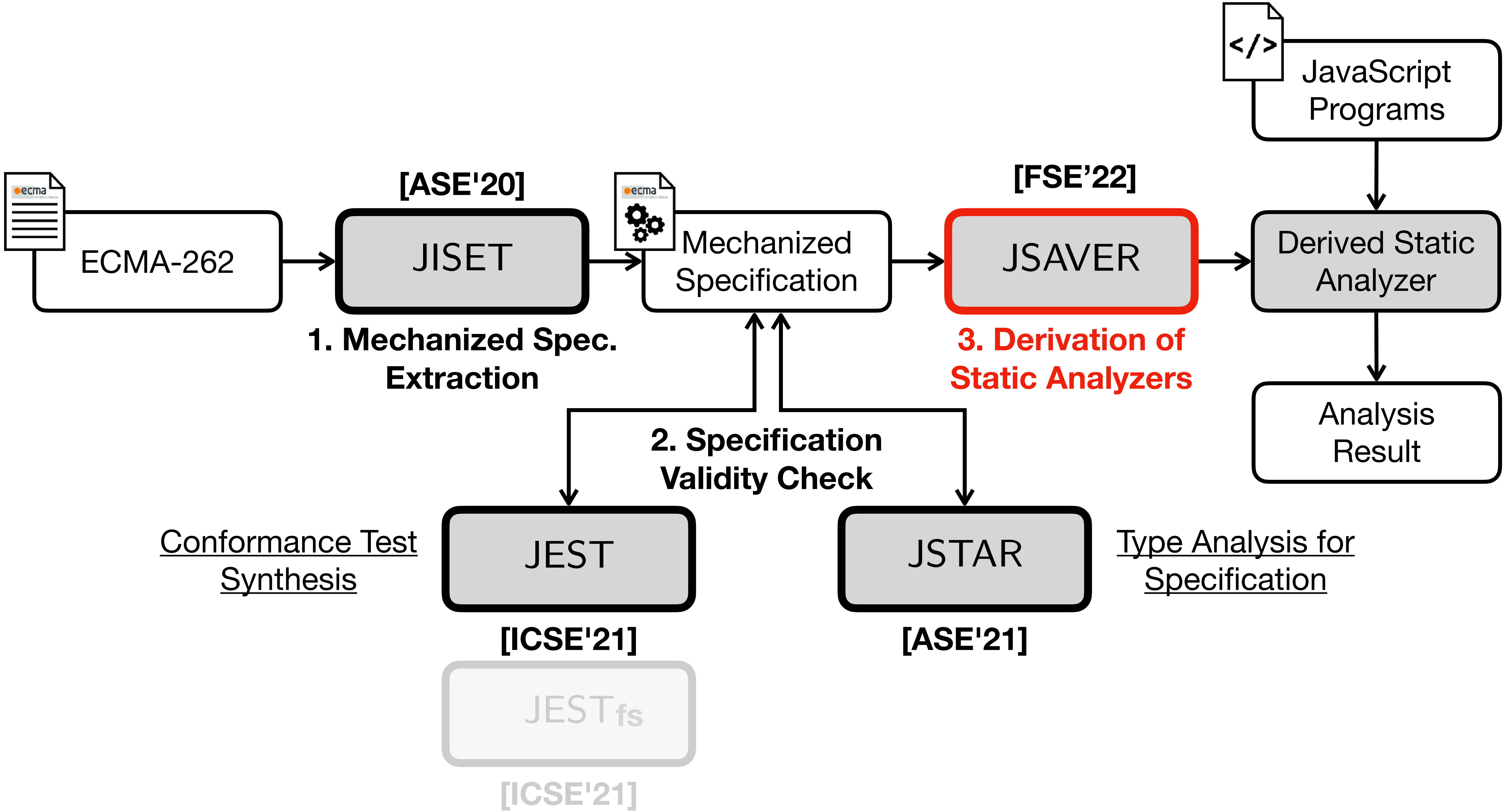
Checker	Bug Kind	Precision = (# True Bugs) / (# Detected Bugs)					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25	/ -25
Operand	NoNumber	22 / 113	2 / 65	22 / 44	2 / 6	/ -69	/ -59
	Abrupt		20 / 48		20 / 38		/ -10
Total		92 / 279 (33.0%)		93 / 157 (59.2%)		+1 / -122 (+26.3%)	

Name	Feature	#	Checker	Created	Life Span
ES12-1	Switch	3	Reference	2015-09-22	1,996 days
ES12-2	Try	3	Reference	2015-09-22	1,996 days
ES12-3	Arguments	1	Reference	2015-09-22	1,996 days
ES12-4	Array	2	Reference	2015-09-22	1,996 days
ES12-5	Async	1	Reference	2015-09-22	1,996 days
ES12-6	Class	1	Reference	2015-09-22	1,996 days
ES12-7	Branch	1	Reference	2015-09-22	1,996 days
ES12-8	Arguments	2	Operand	2015-12-16	1,910 days

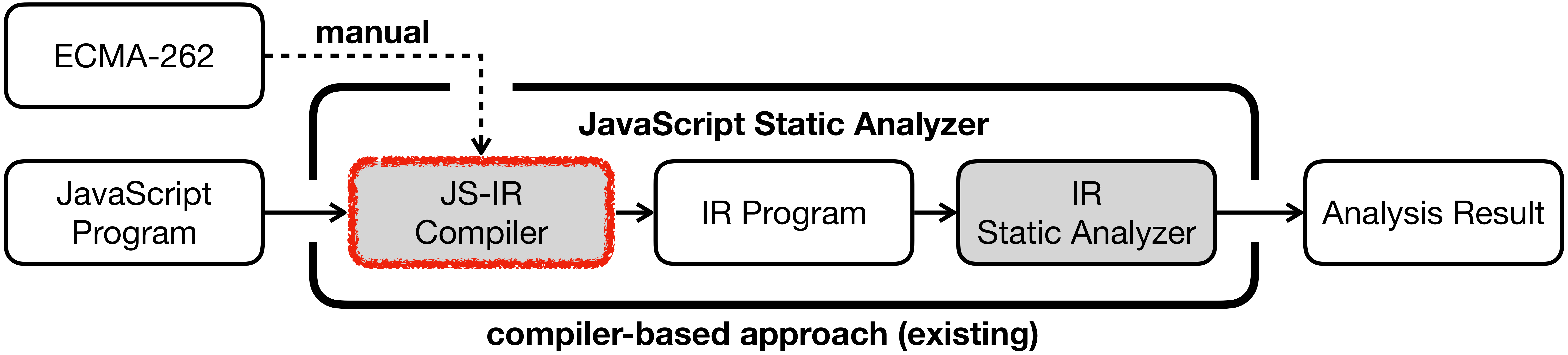
14 Bugs in ES12

Automatically Deriving JavaScript Static Analyzers from Language Specifications

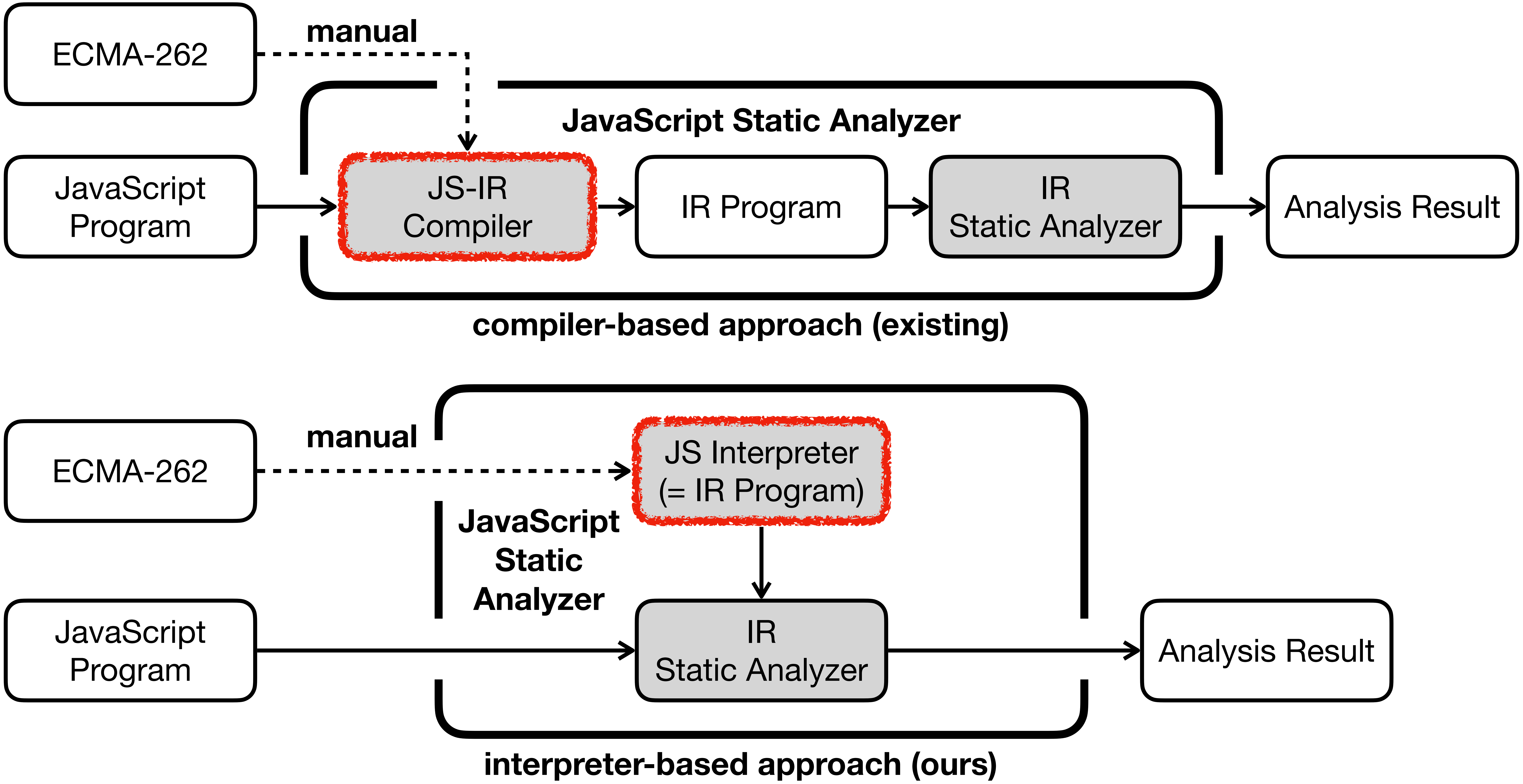
Jihyeok Park, Seungmin An, and Sukyoung Ryu



JSAVER - Meta-Level Static Analysis

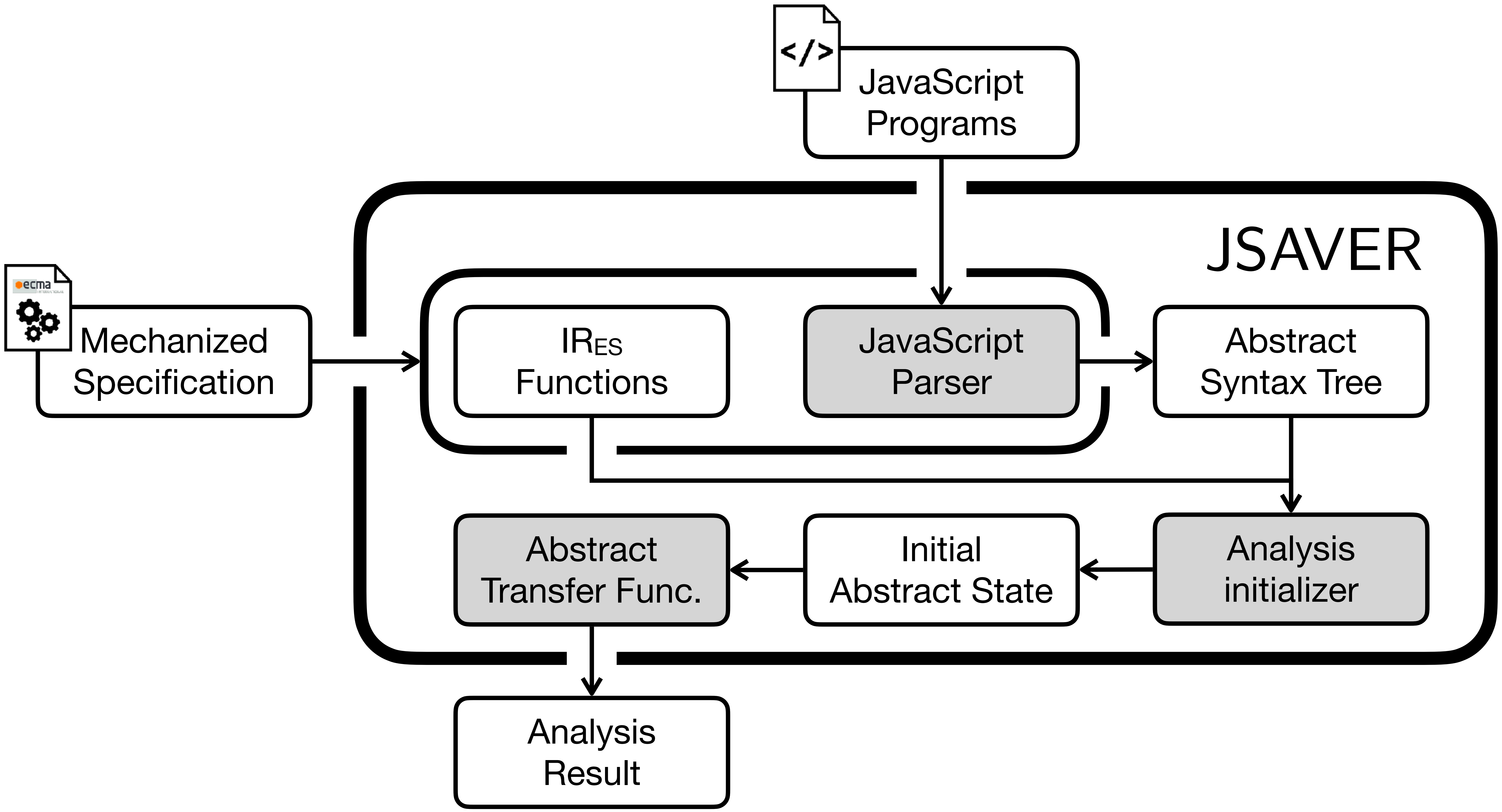


JSAVER - Meta-Level Static Analysis



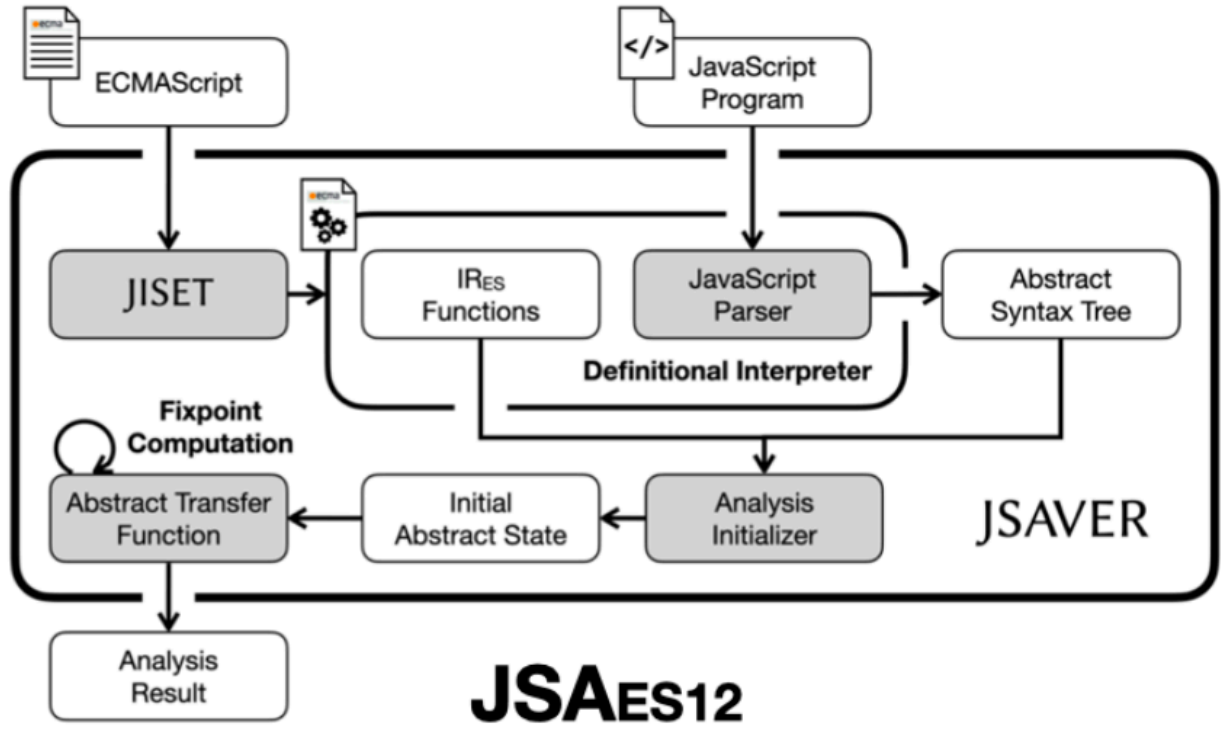
JSAVER [FSE'22]

JavaScript Static Analyzer via ECMAScript Representation

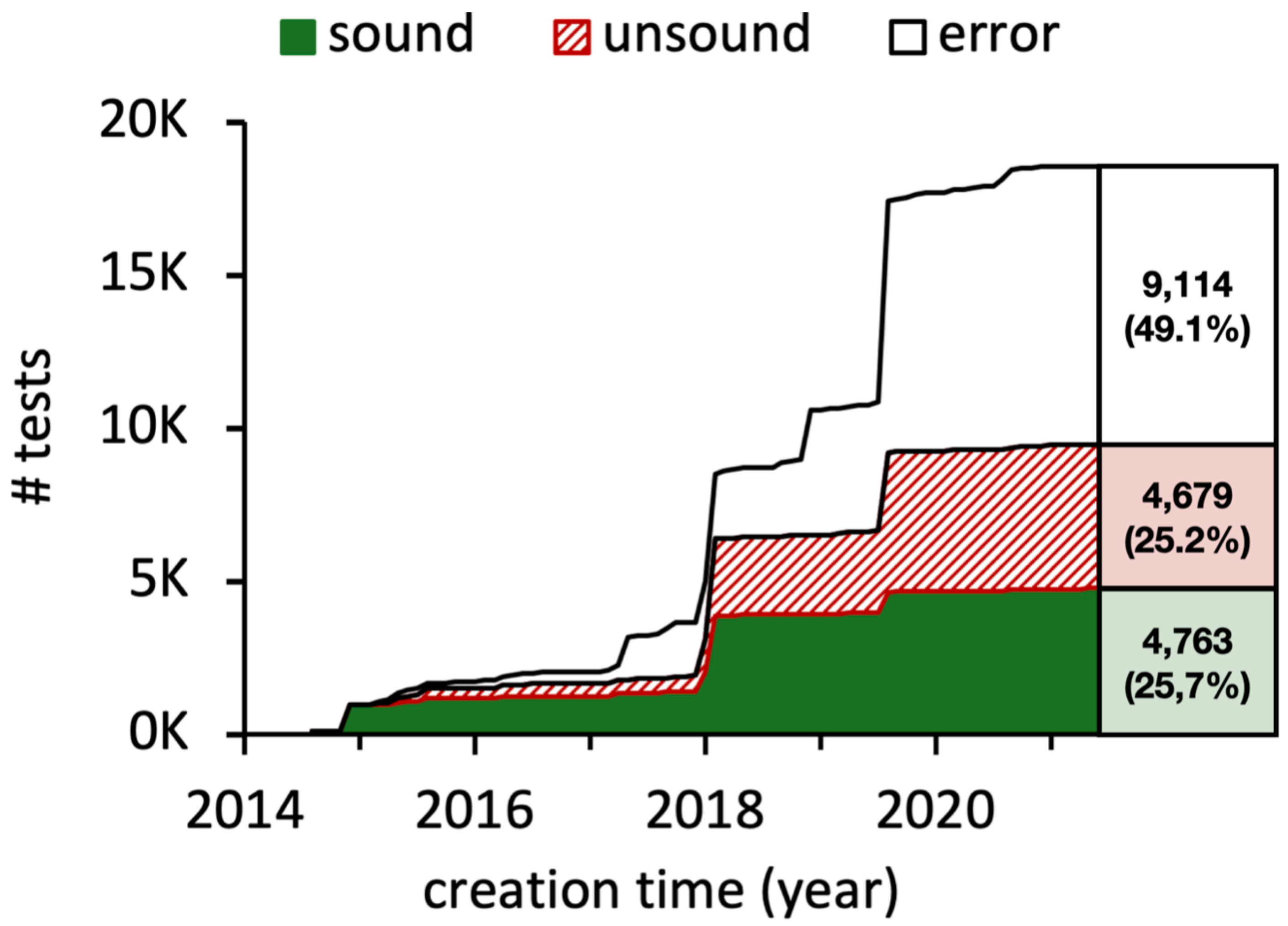


JSAVER - Evaluation

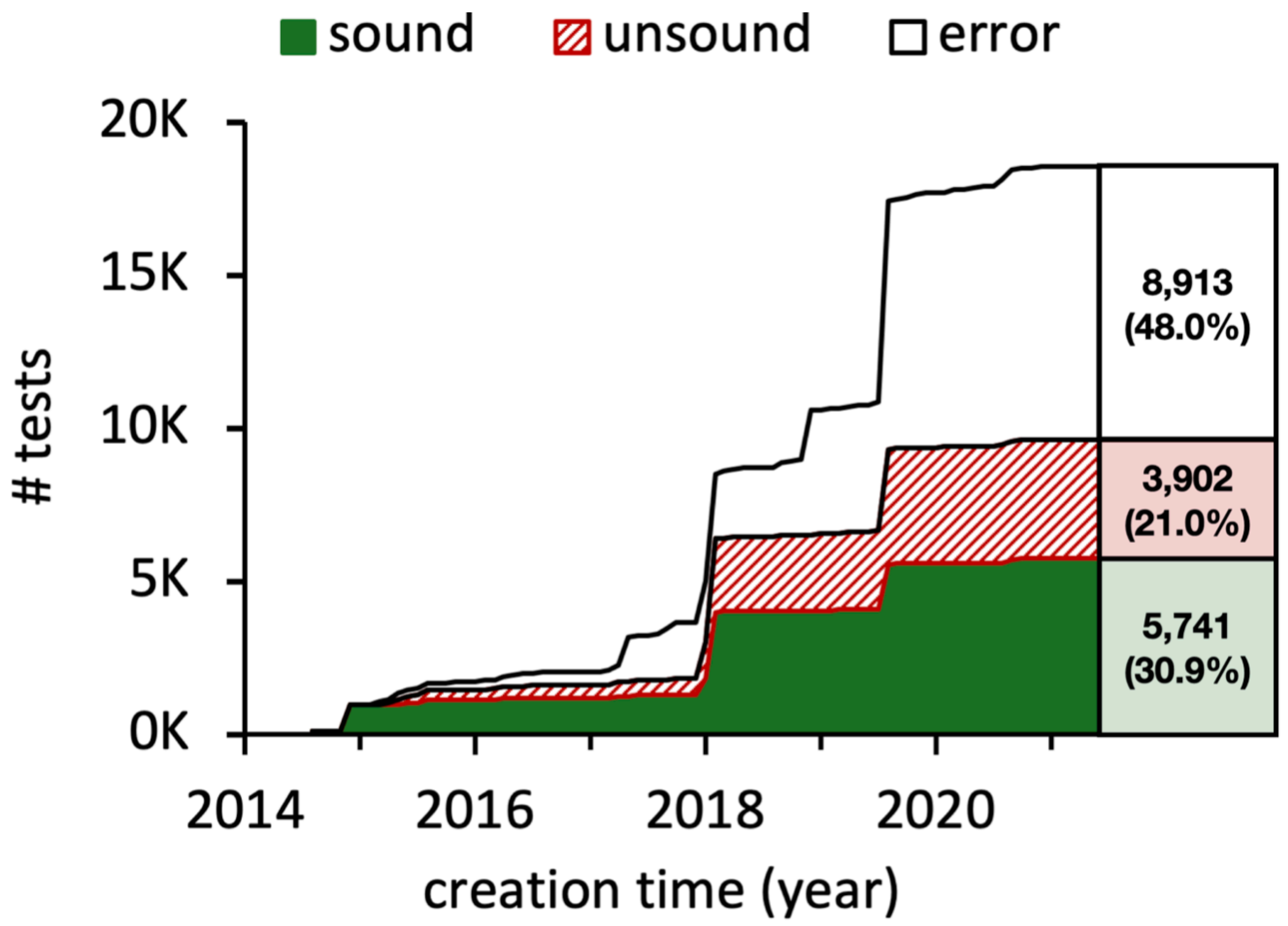
- **Soundness / Precision / Performance**
  - 18,556 applicable tests in Test262
  - 3,903 tests analyzable by all the three analyzers



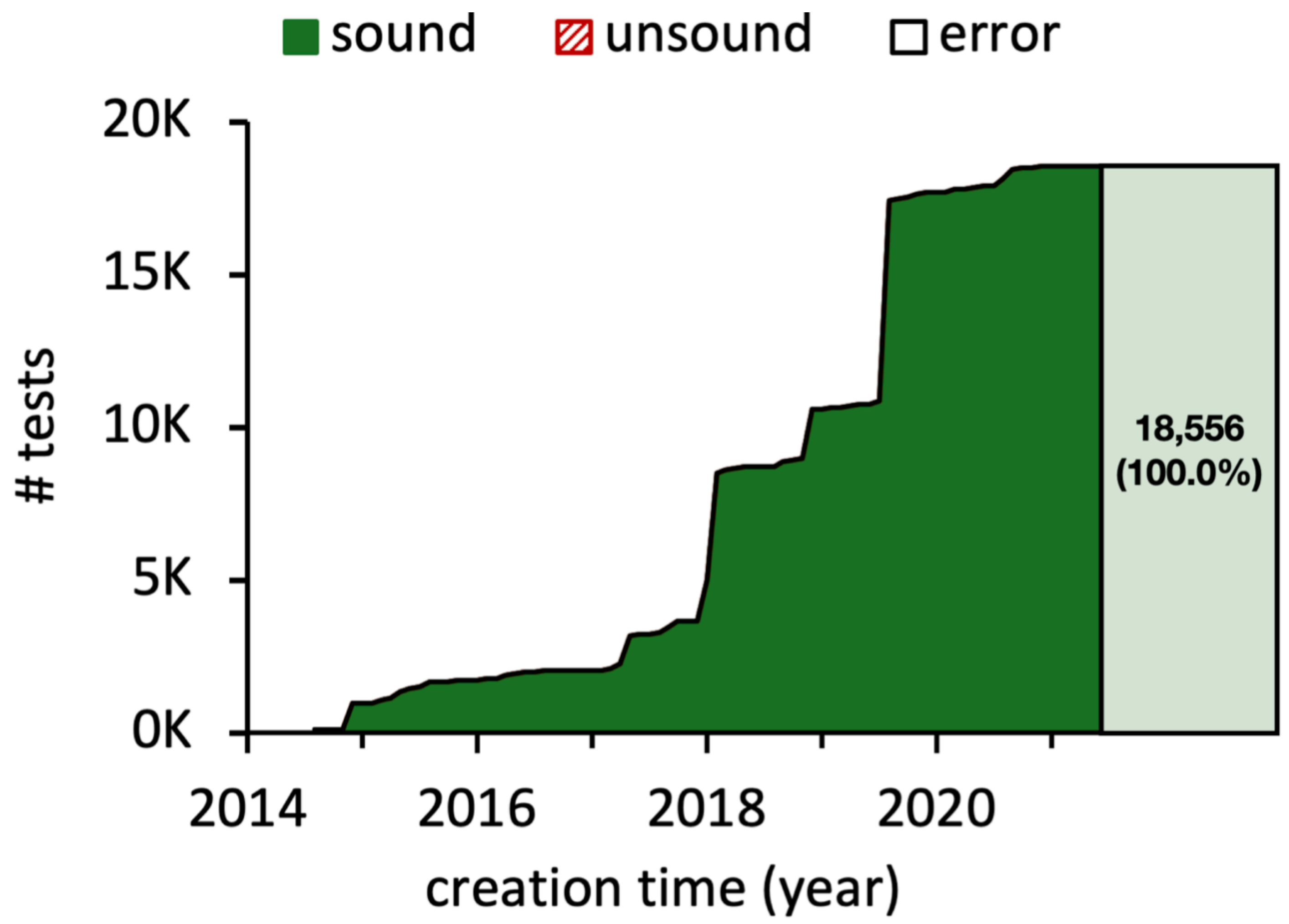
TAJS SAFE JSA$_{ES12}$



(a) Analysis results of TAJS



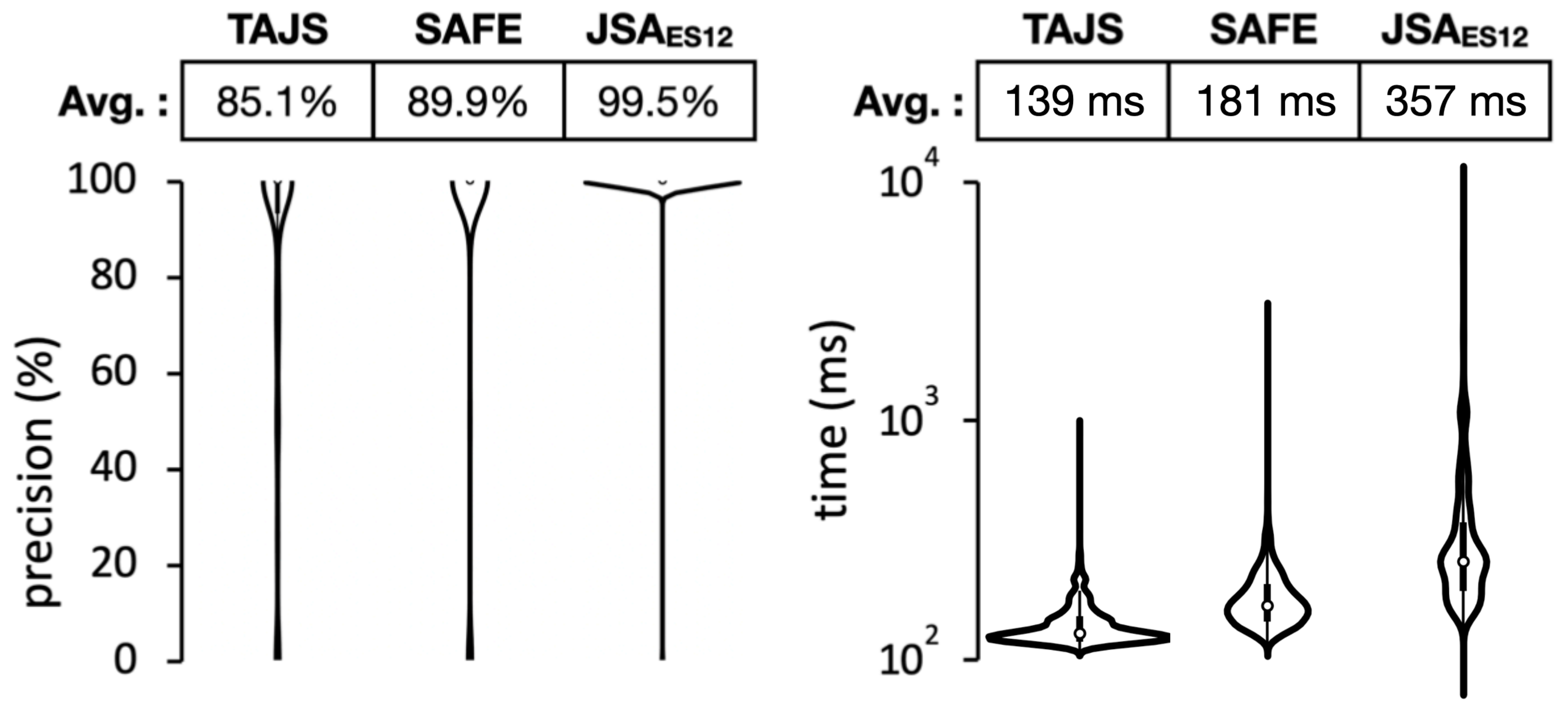
(b) Analysis results of SAFE



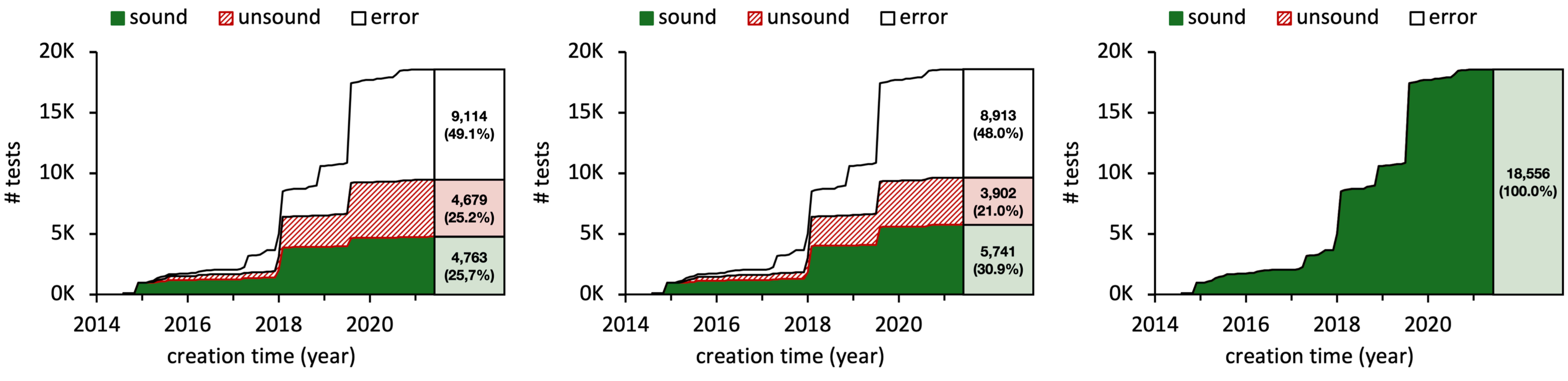
(c) Analysis results of JSA$_{ES12}$

JSAVER - Evaluation

- **Soundness / Precision / Performance**
  - 18,556 applicable tests in Test262
  - 3,903 tests analyzable by all the three analyzers



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



(a) Analysis results of TAJS **(b)** Analysis results of SAFE **(c)** Analysis results of JSA_{ES12}

moddable

My colleagues and I at Moddable appreciate your work to report so many issues. … If you don't mind me asking, my team is curious about the methodology you are using to find these issues. What can you share about that? As background, our primary testing is done with test262. It is an excellent resource but it is not truly comprehensive, as your results reinforce.

ECMA-262, 13th edition, June 2022
ECMAScript® 2022
Language Specification

Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute to the development of this specification:

GitHub Repository: https://github.com/tc39/ecma262
Issues: All Issues, File a New Issue
Pull Requests: All Pull Requests, Create a New Pull Request
Test Suite: Test262
Editors:

- Shu-yu Guo (@_shu)
- Michael Ficarra (@smooshMap)
- Kevin Gibbons (@bakkoting)

During my time as editor of ECMA-262, one of my primary goals has been to make the specification easier to consume for automated analysis tools such as yours. So I'm very happy to see you've had success. I would like to know what difficulties you had in automatically consuming the specification, and what we can do to make it easier for tools like yours in the future. Have you looked at the changes we've made to the 2021 edition or the 2022 draft yet? There have been some fairly substantial changes. I would also like to know how we could integrate your tooling into our development and editing processes so they can co-evolve.

TC
39

"Yeah, first of all, I want to, I can hardly express how amazing this work is, this is really impressive. I sat through the presentation with my mouth open the whole time. So thank you very much."

"First, this is truly amazing work. My mind is blown. I tried to get screenshots, just to remember the slides and then was just taking screenshots of every slide. So I stopped."

"I think this was an excellent presentation. In terms of committee feedback, what you're hearing here, this is the committee in ecstatic mode. This is, this is the maximum that I've heard in terms of positive feedback for a presentation. So, so thank you very much."

tc39 / notes Public

Code | Issues 2 | Pull requests 2 | Actions | Security | Insights

main | notes / meetings / 2022-01 / jan-26.md

js-choi 2022.01.26: Add links to ad-hoc overflow meeting

3 contributors

Presentation from KAIST research group

Presenter: KAIST research group (SAN, JHP, and SRU) & TC39 editor group (MF, KG, and SYG)

- slides
- JavaScript Static Analysis for Evolving Language Specifications: https://www.youtube.com/watch?v=3Jlu_jnHB8g (only for the people with the link because the last part of the video is under paper review)
- ECMAScript Debugger: https://www.youtube.com/watch?v=syfZ3v6JNg8 (publicly available)

SRU: Hi everyone. I am SRU. I am a faculty at the School of Computing at KAIST. Today two of my students are going to present our recent work on automatically generating various tools about JavaScript programs from ES, the language specification written in English. So, first JHP is going to present a very brief version of his PhD dissertation, and then SAN is going to show you a pretty cool tool, a debugger for ES.

JHP: I'm JHP, a PhD candidate in the Programming Language Research Group at KAIST. I will briefly introduce our recent work with these slides. I will first explain why we started this work, and which tools we have developed so far.

Sukyoung Ryu
November 30, 2022 ·

We are thrilled to share that ESMeta is integrated into the CI of both ECMA-262 and Test262:

https://github.com/tc39/ecma262/pull/2926
https://github.com/tc39/test262/pull/3730
https://github.com/es-meta/esmeta

Now, each ECMA-262 PR will execute the ESMeta type checker, and any new or changed tests in a Test262 PR will be executed using the ESMeta interpreter.

Thank you all for making this happen especially 박지혁!

GITHUB.COM

GitHub - es-meta/esmeta: ECMAScript Specification (ECMA-262) Metalanguage

Sukyoung Ryu
November 30, 2022

We are thrilled to share that ESMeta is integrated into the CI of both ECMA-262 and Test262:

https://github.com/tc39/ecma262/pull/2926
https://github.com/tc39/test262/pull/3730
https://github.com/es-meta/esmeta

Now, each ECMA-262 PR will execute the ESMeta type checker, and any new or changed tests in a Test262 PR will be executed using the ESMeta interpreter.

Thank you all for making this happen especially 박지혁!

es-meta/esmeta

ECMAScript Specification (ECMA-262) Metalanguage

ESMETA

10 Contributors · 7 Issues · 110 Stars · 9 Forks

GITHUB.COM

GitHub - es-meta/esmeta: ECMAScript Specification (ECMA-262) Metalanguage

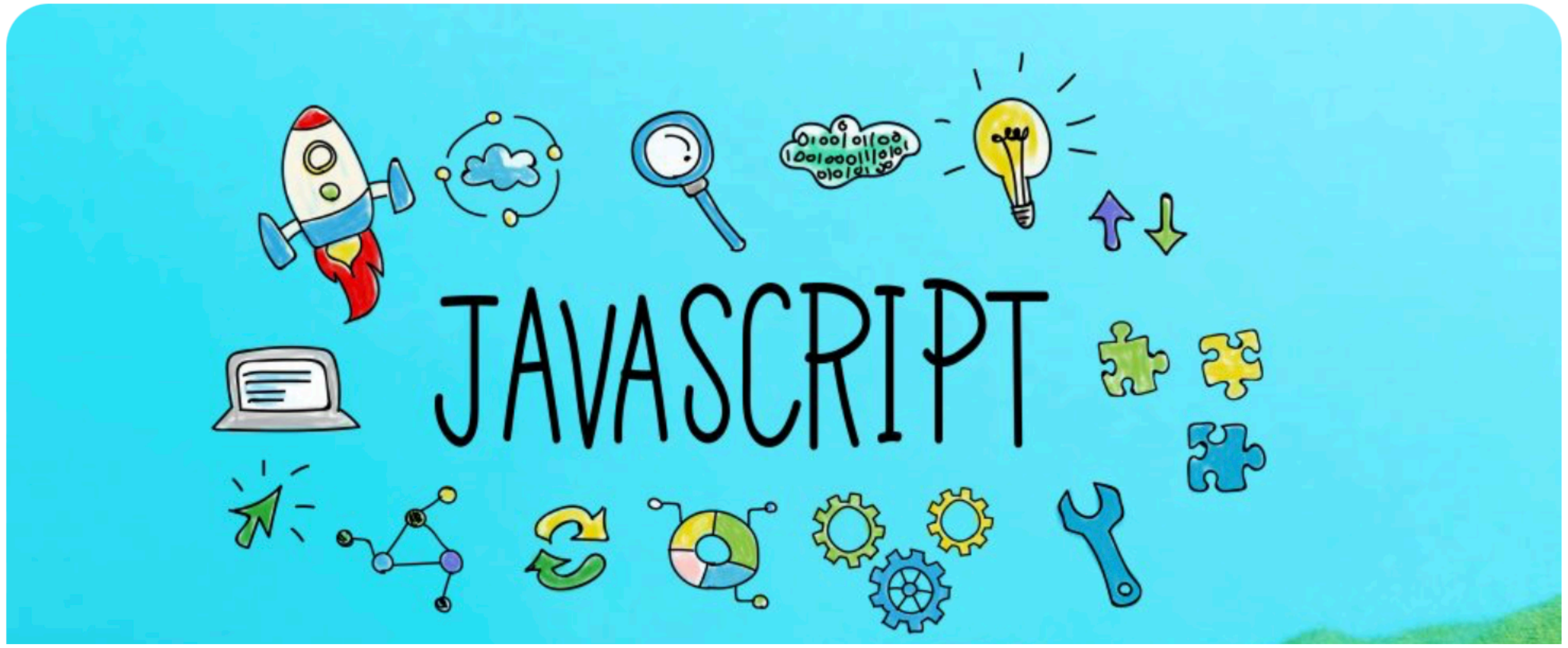
SIGPLAN

PL Perspectives

Perspectives on computing and technology from and for those with an interest in programming languages.

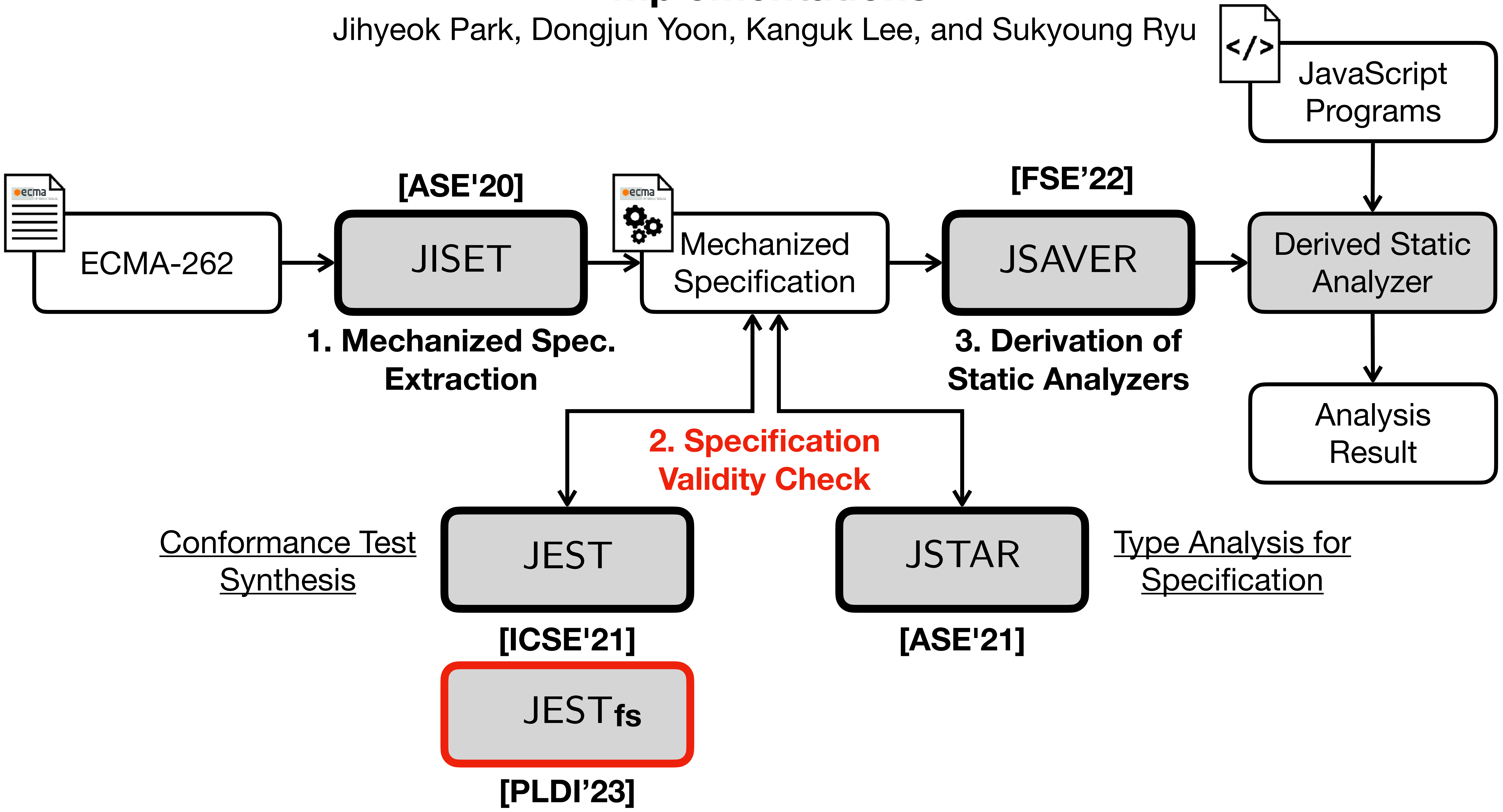
From Research Prototypes to Continuous Integration: Guiding the Design and Implementation of JavaScript

by Sukyoung Ryu and Jihyeok Park on Jan 12, 2023 | Tags: continuous integration, javascript, language design, mechanized specification



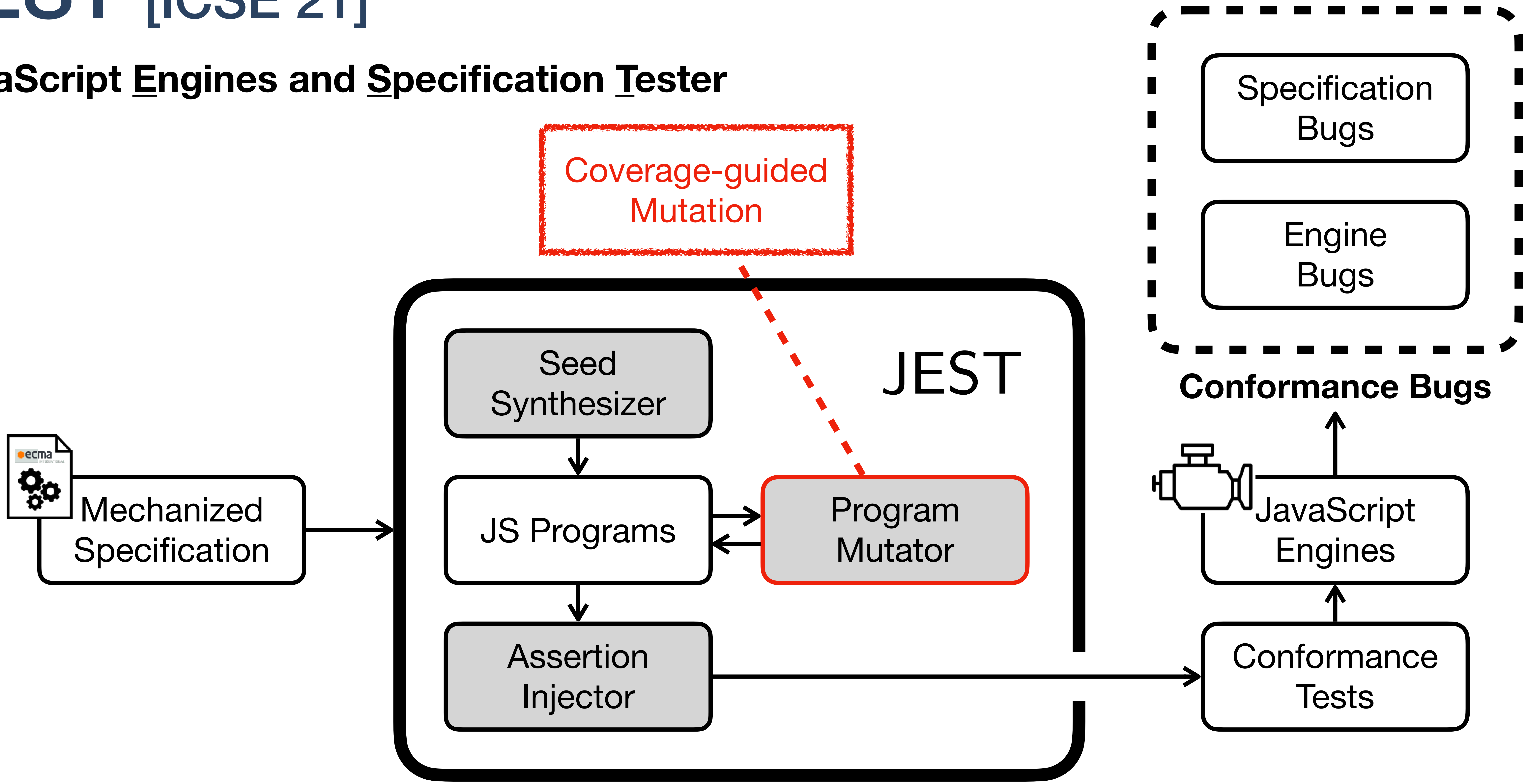
Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations

Jihyeok Park, Dongjun Yoon, Kanguk Lee, and Sukyoung Ryu

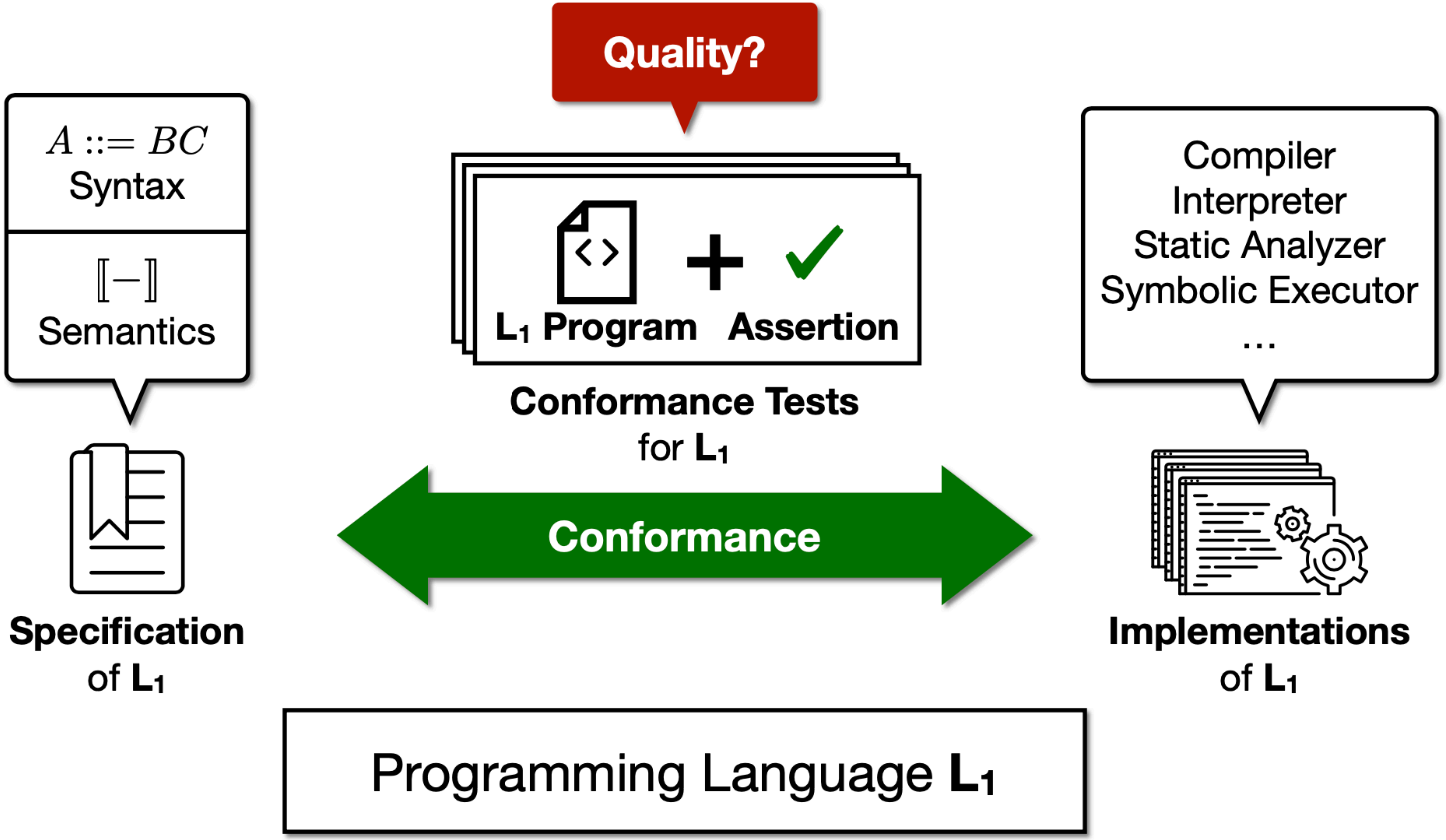


JEST [ICSE'21]

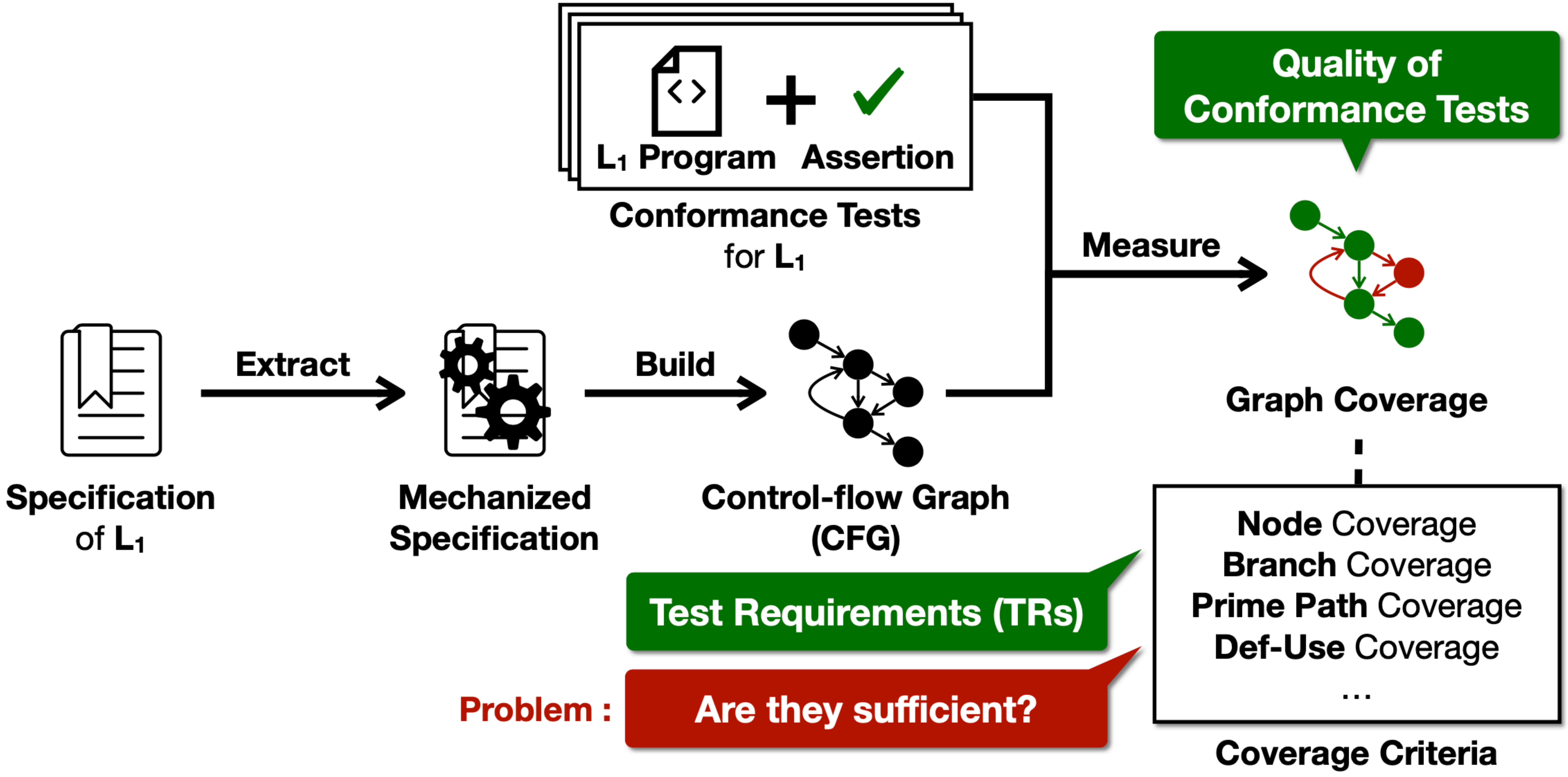
JavaScript Engines and Specification Tester



Conformance Testing of PL Implementations



Graph Coverage for Language Specification



Normal algorithms

EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

[7]...

5. Return ?[9] ApplyStringOrNumericBinaryOperator[8](*lval*, *opText*, *rval*).[10]

ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

[11]...

3. Let *lnum* be ?[13] ToNumeric[12](*lval*).
4. Let *rnum* be ?[15] ToNumeric[14](*rval*).
5. If Type(*lnum*) is different from Type(*rnum*)[16], throw a **TypeError** exception.[17]

...[18]

ToNumeric (*value*)

[19]...

2. If Type(*primValue*) is BigInt[20], return *primValue*.[21]

...[22]

Fig. 3. Three normal algorithms transitively used in the semantics of *AdditiveExpression* in ES13

Built-in methods

Number (*value*)

1. If *value* is present[23], then
   a. Let *prim* be ?[25] ToNumeric[24](*value*).

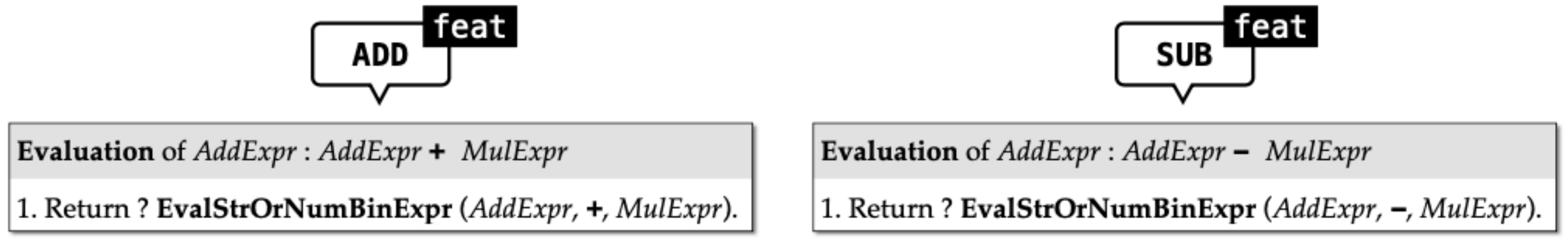
...[26]

Fig. 4. Built-in method **Number** in ES13

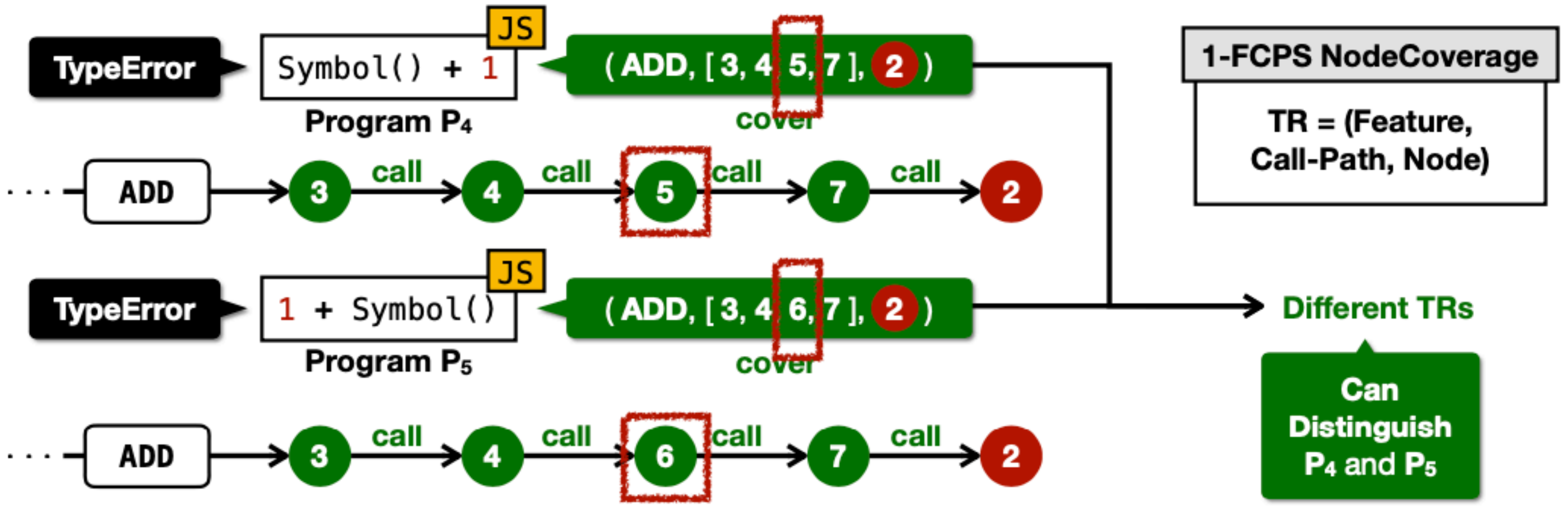
k-Feature-Sensitive (k-FS) Coverage

- k-Feature-Sensitive (k-FS) coverage criterion **divides** the given TRs with the at most k-**innermost enclosing** language **features**

FS Coverage
TR = (Feature$^{\le k}$, given TR)



k-Feature-Call-Path-Sensitive (k-FCPS) Coverage

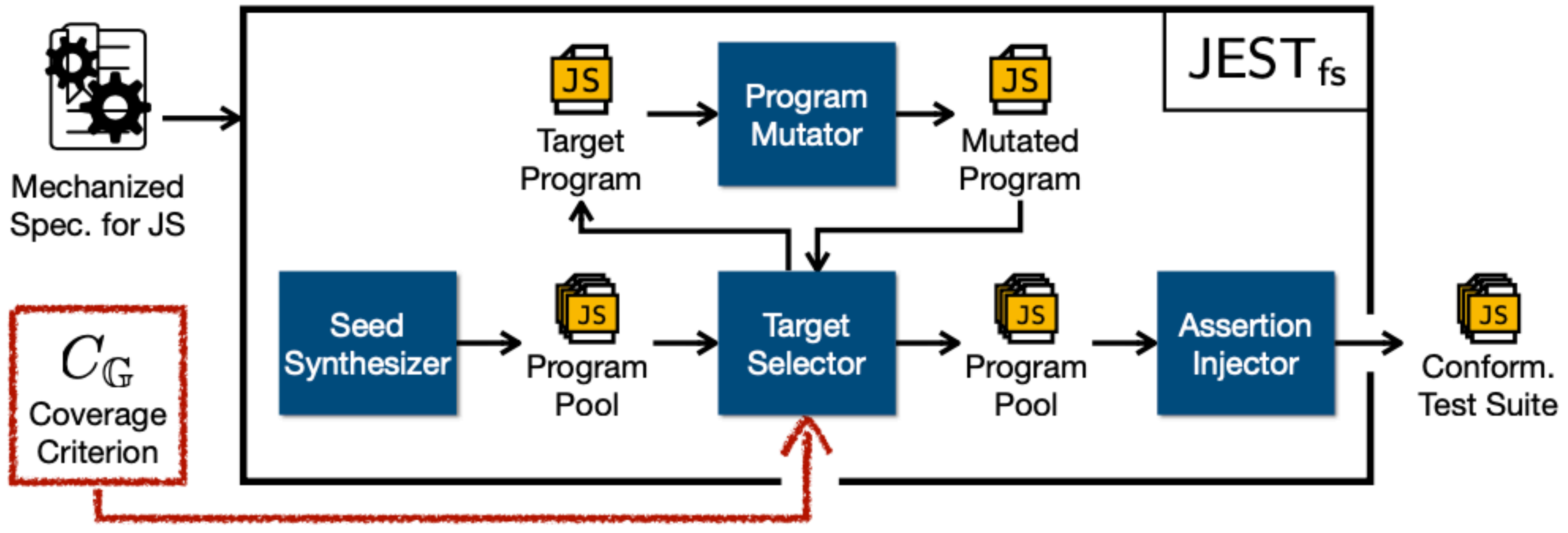


- k-Feature-Call-Path-Sensitive (k-FCPS) coverage criterion **divides** the k-FS TRs with the **call-paths from** the innermost enclosing language feature

FS Coverage
TR = (Feature$^{\le k}$, Call-Path, given TR)

Implementation — JEST$_{fs}$

- **JEST** [1] is a **JavaScript conformance test generator** using **Coverage-Guided Fuzzing**
- We implemented **JEST$_{fs}$** as an extension of **JEST** with k-**FS** and k-**FCPS** coverage criteria



[1] Park et al., "JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification", ICSE 2021

RQ2) Effectiveness of k-FS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	# Covered k-F(CP)S-TR (k)			# Syn. Test	# Bug
	# Node	# Branch	# Total		
0-FS node-or-branch (0-fs)	10.0	5.6	15.6	2,111	55
1-FS node-or-branch (1-fs)	79.3	45.7	125.0	6,766	83
1-FCPS node-or-branch (1-fcps)	179.7	97.6	277.3	9,092	87
2-FS node-or-branch (2-fs)	1,199.8	696.3	1,896.1	97,423	102
2-FCPS node-or-branch (2-fcps)	2,323.1	1,297.6	3,620.7	122,589	111

+28
+19

RQ3) Effectiveness of k-FCPS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	# Covered k-F(CP)S-TR (k)			# Syn. Test	# Bug
	# Node	# Branch	# Total		
0-FS node-or-branch (0-fs)	10.0	5.6	15.6	2,111	55
1-FS node-or-branch (1-fs)	79.3	45.7	125.0	6,766	83
1-FCPS node-or-branch (1-fcps)	179.7	97.6	277.3	9,092	87
2-FS node-or-branch (2-fs)	1,199.8	696.3	1,896.1	97,423	102
2-FCPS node-or-branch (2-fcps)	2,323.1	1,297.6	3,620.7	122,589	111

+4
+9

RQ1) Conformance Bug Detection

Kind	Name	Version	Release	# Detected Unique Bugs		
				# New	# Confirmed	# Reported
Engine	V8	v10.8.121	2022.10.06	0	0	4
	JSC	v615.1.10	2022.10.26	15	15	24
	GraalJS	v22.2.0	2022.07.26	9	9	10
	SpiderMonkey	v107.0b4	2022.10.24	1	3	4
	Total			**25**	**27**	**42**
Transpiler	Babel	v7.19.1	2022.09.15	30	30	35
	SWC	v1.3.10	2022.10.21	27	27	41
	Terser	v5.15.1	2022.10.05	1	1	18
	Obfuscator	v4.0.0	2022.02.15	0	0	7
	Total			**58**	**58**	**101**
Total				**83**	**85**	**143**

Rust?

rust-lang / rust Public

Watch 1.5k · Fork 10.3k · Star 76.9k

Code · Issues 5k+ · Pull requests 731 · Actions · Projects 1 · Security 3 · Insights

Coherence can be bypassed by an indirect impl for a trait object #57893

New issue

Open **arielb1** opened this issue on Jan 25, 2019 · 50 comments

arielb1 commented on Jan 25, 2019 • edited by lcnr

Contributor

Comments

The check for manual `impl Object for Object` only makes sure there is no *direct* `impl Object for dyn Object` - it does not consider such indirect impls. Therefore, you can write a blanket `impl<T: ?Sized> Object for T` that conflicts with the builtin `impl Object for dyn Object`.

Assignees

nikomatsakis

Labels

A-dst · A-traits · C-bug · I-unsound · P-high · T-compiler · T-lang · T-types

Verse?

The Verse Calculus: a Core Calculus for Functional Logic Programming

LENNART AUGUSTSSON, Epic Games, Sweden
JOACHIM BREITNER
KOEN CLAESSEN, Epic Games, Sweden
RANJIT JHALA, Epic Games, USA
SIMON PEYTON JONES, Epic Games, United Kingdom
OLIN SHIVERS, Epic Games, USA
TIM SWEENEY, Epic Games, USA

Functional logic languages have a rich literature, but it is tricky to give them a satisfying semantics. In this paper we describe the Verse calculus, $\mathcal{VC}$, a new core calculus for functional logical programming. Our main contribution is to equip $\mathcal{VC}$ with a small-step rewrite semantics, so that we can reason about a $\mathcal{VC}$ program in the same way as one does with lambda calculus; that is, by applying successive rewrites to it.

This draft paper describes our current thinking about Verse. It is very much a work in progress, not a finished product. The broad outlines of the design are stable. However, the details of the rewrite rules may well change; we think that the current rules are not confluent, in tiresome ways. (If you are knowledgeable about confluence proofs, please talk to us!)

We are eager to enagage in a dialogue with the community. Please do write to us.

WebAssembly?

$t.binop$

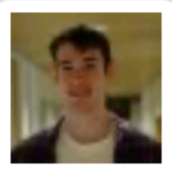
1. Assert: due to validation, two values of value type t are on the top of the stack.
2. Pop the value $t.\mathsf{const}\ c_2$ from the stack.
3. Pop the value $t.\mathsf{const}\ c_1$ from the stack.
4. If $binop_t(c_1, c_2)$ is defined, then:
   a. Let c be a possible result of computing $binop_t(c_1, c_2)$.
   b. Push the value $t.\mathsf{const}\ c$ to the stack.
5. Else:
   a. Trap.

$$
\begin{array}{lcll}
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & (t.\mathsf{const}\ c) & (\text{if } c \in binop_t(c_1, c_2)) \\
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & \mathsf{trap} & (\text{if } binop_t(c_1, c_2) = \{\})
\end{array}
$$

☆ WebAssembly: sequential and concurrent semantics

joint work with Guillaume Barbier, Martin Bodin, Sunjay Cauligi, Craig Disselkoen, Stephen Dolan, Shaked Flur, Philippa Gardner, Tal Garfinkel, Shu-yu Guo, Neelakantan R. Krishnaswami, Amit Levy, Petar Maksimović, Jean Pichon-Pharabod, Anton Podkopaev, Natalie Popescu, Christopher Pulte, John Renner, Andreas Rossberg, Deian Stefan, Rao Xiaoj

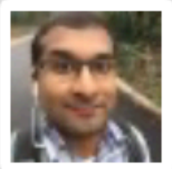
WebAssembly is the first new programming language to be natively supported on the Web platform since JavaScript. WebAssembly was designed from the ground up using formal semantics, and each new feature must be fully formally specified before it can be adopted into the standard. This talk will describe recent work in mechanising WebAssembly's semantics, and specifying the language's relaxed memory model. We will also discuss future challenges in verifying security-related properties of the language.


Conrad Watt — Presenter
University of Cambridge, UK

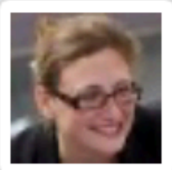

Andreas Rossberg
Dfinity Stiftung
Germany

Rao Xiaoj
Imperial College
United Kingdom

Guillaume Barbier
ENS Rennes, France


Sunjay Cauligi
University of California at San Diego, USA

Stephen Dolan
University of Cambridge, UK


Philippa Gardner
Imperial College London, UK

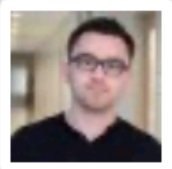

Shu-yu Guo
Bloomberg, USA

Amit Levy

Jean Pichon-Pharabod
University of Cambridge, UK

Natalie Popescu
University of California San Diego


John Renner
University of California at San Diego, USA


Deian Stefan
University of California at San Diego, USA

Dagstuhl Seminar 23101

Foundations of WebAssembly

(Mar 05 – Mar 10, 2023)

Permalink

Please use the following short url to reference this page: https://www.dagstuhl.de/23101

Organizers

- Karthikeyan Bhargavan (INRIA - Paris, FR)
- Jonathan Protzenko (Microsoft - Redmond, US)
- Andreas Rossberg (München, DE)
- Deian Stefan (University of California - San Diego, US)

Contact

- Andreas Dolzmann (for scientific matters)
- Christina Schwarz (for administrative matters)

timeline

2018 Wasm 1.0

focus on low-level languages to get off the ground quickly

2022 Wasm 2.0

multiple values, vector types, reference types, table ops, bulk ops, multiple tables

2023 Wasm 2.1

tail calls, relaxed vector ops, deterministic profile

2024 Wasm 2.2+

atomics, exceptions, typed references, garbage collection, 64-bit addresses, multiple memories

202X Wasm 2.X

stack switching, threading, memory control, type imports, ...

https://github.com/WebAssembly/proposals

timeline

2018 Wasm 1.0

focus on low-level languages to get off the ground quickly

2022 Wasm 2.0

multiple values, vector types, reference types, tabl

2023 Wasm 2.1

tail calls, relaxed vector ops, deterministic profile

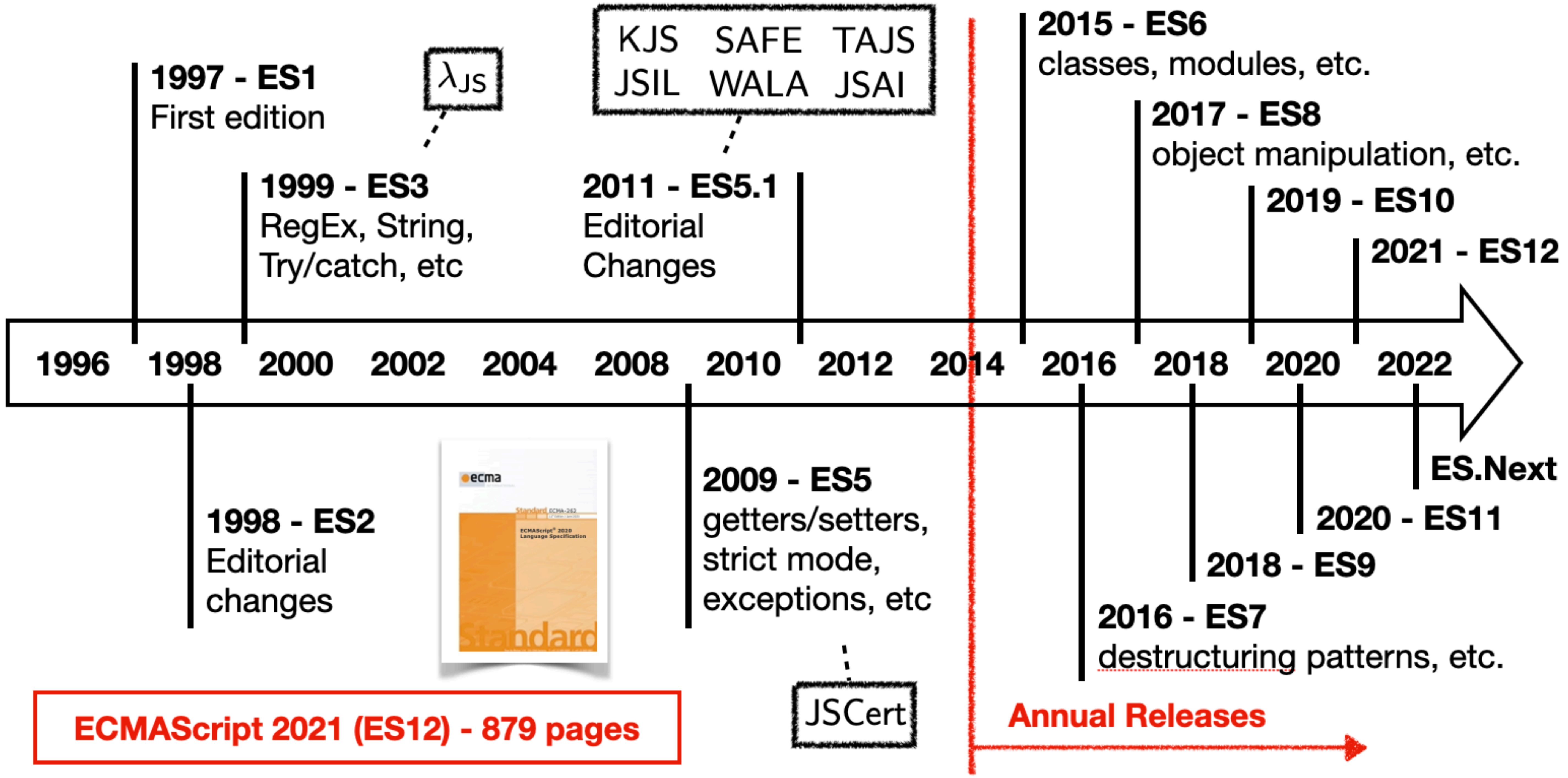
2024 Wasm 2.2+

atomics, exceptions, typed references, garbage c

202X Wasm 2.X

stack switching, threading, memory control, type i

Problem: Fast Evolving JavaScript



ECMAScript 2021 (ES12) - 879 pages

$t.\mathit{binop}$

1. Assert: due to validation, two values of value type t are on the top of the stack.
2. Pop the value $t.\mathsf{const}\ c_2$ from the stack.
3. Pop the value $t.\mathsf{const}\ c_1$ from the stack.
4. If $\mathit{binop}_t(c_1, c_2)$ is defined, then:
   a. Let c be a possible result of computing $\mathit{binop}_t(c_1, c_2)$.
   b. Push the value $t.\mathsf{const}\ c$ to the stack.
5. Else:
   a. Trap.

$$
\begin{array}{lcll}
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.\mathit{binop} & \hookrightarrow & (t.\mathsf{const}\ c) & (\text{if } c \in \mathit{binop}_t(c_1, c_2)) \\
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.\mathit{binop} & \hookrightarrow & \mathsf{trap} & (\text{if } \mathit{binop}_t(c_1, c_2) = \{\})
\end{array}
$$

```
rule Step_pure/binop-val:
  (CONST nt c_1) (CONST nt c_2) (BINOP nt binop)  ~>  (CONST nt c)
  -- if $binop(binop, nt, c_1, c_2) = c  ;; TODO

rule Step_pure/binop-trap:
  (CONST nt c_1) (CONST nt c_2) (BINOP nt binop)  ~>  TRAP
  -- if $binop(binop, nt, c_1, c_2) = epsilon  ;; TODO
```

t.binop

1. Assert: due to validation, two values of value type t are on the top of the stack.
2. Pop the value $t.\mathsf{const}\ c_2$ from the stack.
3. Pop the value $t.\mathsf{const}\ c_1$ from the stack.
4. If $binop_t(c_1, c_2)$ is defined, then:
   a. Let c be a possible result of computing $binop_t(c_1, c_2)$.
   b. Push the value $t.\mathsf{const}\ c$ to the stack.
5. Else:
   a. Trap.

$$
\begin{array}{lcll}
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & (t.\mathsf{const}\ c) & (\text{if } c \in binop_t(c_1, c_2)) \\
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & \mathsf{trap} & (\text{if } binop_t(c_1, c_2) = \{\})
\end{array}
$$

```
$ (cd ../spec && dune exec ../src/exe-watsup/main.exe -- *.watsup -v -l --sideconditions --animate --prose)
watsup 0.3 generator
== Parsing...
== Elaboration...
== IL Validation...
```

```
binop
1. Assert: Due to validation, a value is on the top of the stack.
2. Pop YetE (CONST_admininstr(nt, c_2)) from the stack.
3. Assert: Due to validation, a value is on the top of the stack.
4. Pop YetE (CONST_admininstr(nt, c_1)) from the stack.
5. If YetC (), then:
  a. Push YetE (CONST_admininstr(nt, c)) to the stack.
6. If YetC (), then:
  a. Trap.
```

1.1.2 binop nt $binop$

1. Assert: Due to validation, a value of value type nt is on the top of the stack.
2. Pop const nt c_2 from the stack.
3. Assert: Due to validation, a value of value type nt is on the top of the stack.
4. Pop const nt c_1 from the stack.
5. Let r_0 be the result of computing $binop(binop, nt, c_1, c_2)$.
6. If the length of r_0 is 1, then:
   a. Let $[c]$ be the result of computing $binop(binop, nt, c_1, c_2)$.
   b. Push const nt c to the stack.
7. Let r_1 be the result of computing $binop(binop, nt, c_1, c_2)$.
8. If r_1 is [], then:
   a. Trap.

$$
\begin{array}{lllll}
[\textsc{E-binop-val}] & (nt.\mathsf{const}\ c_1)\ (nt.\mathsf{const}\ c_2)\ (nt.binop) & \hookrightarrow & (nt.\mathsf{const}\ c) & \text{if } binop_{nt}(c_1,\ c_2) = c \\
[\textsc{E-binop-trap}] & (nt.\mathsf{const}\ c_1)\ (nt.\mathsf{const}\ c_2)\ (nt.binop) & \hookrightarrow & \mathsf{trap} & \text{if } binop_{nt}(c_1,\ c_2) = \epsilon
\end{array}
$$

P4?

P4_{16} Language Specification

version 1.2.4

The P4 Language Consortium

2023-05-15

Abstract

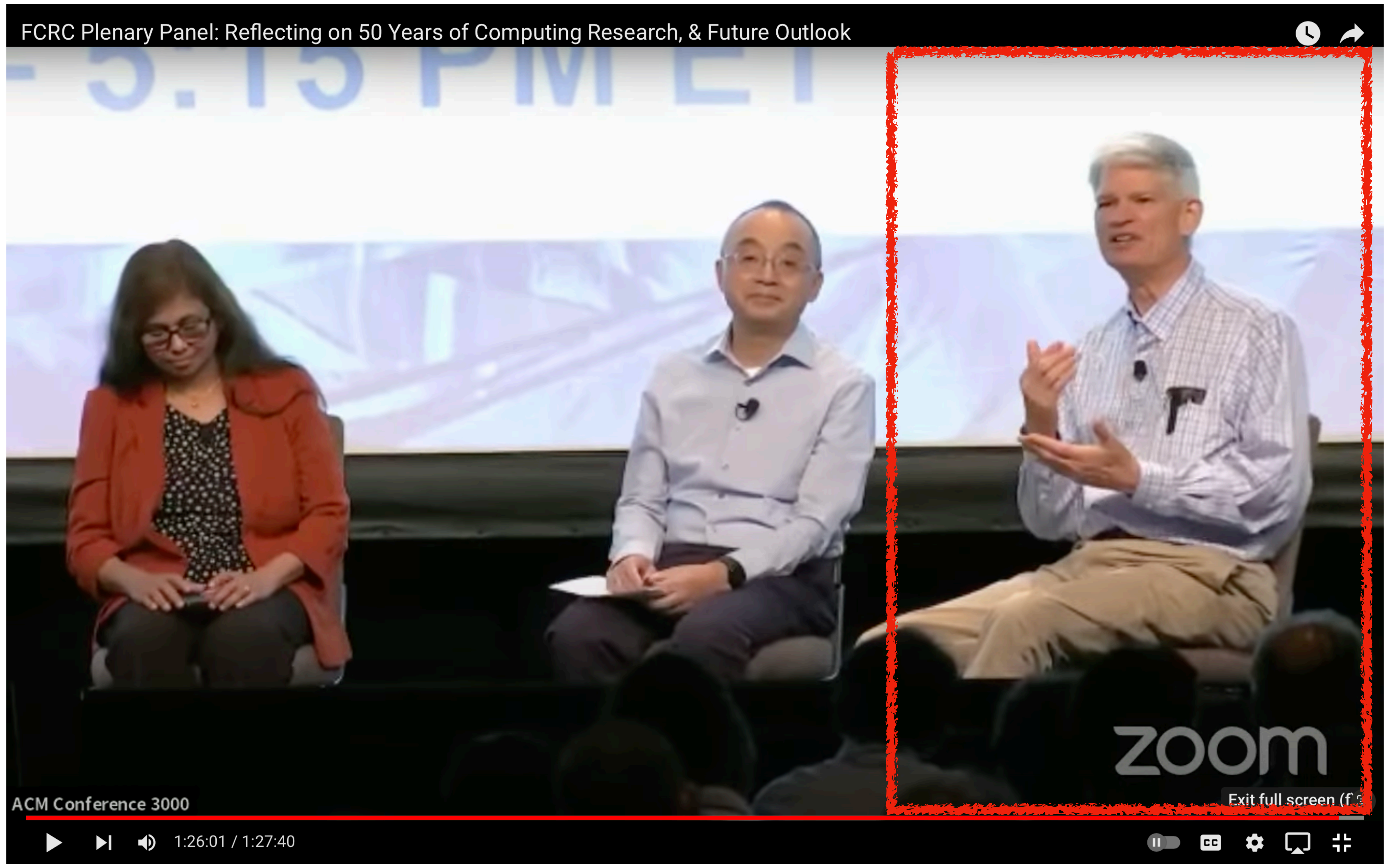
P4 is a language for programming the data plane of network devices. This document provides a precise definition of the P4_{16} language, which is the 2016 revision of the P4 language (`http://p4.org`). The target audience for this document includes developers who want to write compilers, simulators, IDEs, and debuggers for P4 programs. This document may also be of interest to P4 programmers who are interested in understanding the syntax and semantics of the language at a deeper level.

Reviewer #2

This is the right order to design and document languages:

first the semantics,
then the implementation
and documentation, ideally
generated from the semantics.

Automatic Derivation of Static Analysis



- "If you go and talk to an old person and he or she says, "We tried that 25 years ago and it didn't work," don't take that for an answer, ask why it didn't work and what's different today."
- "Keep the big picture in mind; look for opportunities to bridge disciplines rather than rat holding on just one; don't settle for just a 5% improvement, aim big; and finally read read read voraciously, read papers, read code, read books, absorb as much information as you can, you never can tell what you learned five years ago might suddenly be relevant now."