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

EIRIC Invite Talk

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

PLRG @ KAIST

October 12, 2021

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

Jihyeok Park	Seungmin An	Dongjun Youn	Gyeongwon Kim	Sukyoung Ryu
School of Computing	*School of Computing*	*School of Computing*	*School of Computing*	*School of Computing*
KAIST	*KAIST*	*KAIST*	*KAIST*	*KAIST*
Daejeon, South Korea	Daejeon, South Korea	Daejeon, South Korea	Daejeon, South Korea	Daejeon, South Korea
jhpark0223@kaist.ac.kr	h2oche@kaist.ac.kr	f52985@kaist.ac.kr	gyeongwon.kim@kaist.ac.kr	sryu.cs@kaist.ac.kr

Abstract—**Modern programming follows the continuous integration (CI) and continuous deployment (CD) approach rather than the traditional waterfall model. Even the development of modern programming languages uses the CI/CD approach to swiftly provide new language features and to adapt to new development environments. Unlike in the conventional approach, in the modern CI/CD approach, a language specification is no more the oracle of the language semantics because both the specification and its implementations (interpreters or compilers) can co-evolve. In this setting, both the specification and implementations may have bugs, and guaranteeing their correctness is non-trivial.**

In this paper, we propose a novel ***N+1-version differential testing*** **to resolve the problem. Unlike the traditional differential testing, our approach consists of three steps: 1) to automatically synthesize programs guided by the syntax and semantics from a given language specification, 2) to generate conformance tests by injecting assertions to the synthesized programs to check their final program states, 3) to detect bugs in the specification and implementations via executing the conformance tests on multiple implementations, and 4) to localize bugs on the specification using statistical information. We actualize our approach for the JavaScript programming language via JEST, which performs N+1-version differential testing for modern JavaScript engines and ECMAScript, the language specification describing the syntax and semantics of JavaScript in a natural language. We evaluated JEST with four JavaScript engines that support all modern JavaScript language features and the latest version of ECMAScript (ES11, 2020). JEST automatically synthesized 1,700 programs that covered 97.78% of syntax and 87.70% of semantics from ES11. Using the assertion-injected JavaScript programs, it detected 44 engine bugs in four different engines and 27 specification bugs in ES11.**

Index Terms—**JavaScript, conformance test generation, mechanized specification, differential testing**

I. Introduction

In Peter O'Hearn's keynote speech in ICSE 2020, he quoted the following from Mark Zuckerberg's Letter to Investors [1]:

> The Hacker Way is an approach to building that involves continuous improvement and iteration. Hackers believe that somethings can always be better, and that nothing is ever complete.

Indeed, modern programming follows the continuous integration (CI) and continuous deployment (CD) approach [2] rather than the traditional waterfall model. Instead of a sequential model that divides software development into several phases, each of which takes time, CI/CD amounts to a cycle of quick software development, deployment, and back to development with feedback. Even the development of programming languages uses the CI/CD approach.

Consider JavaScript, one of the most widely used programming languages for client-side and server-side programming [3] and embedded systems [4]–[6]. Various JavaScript engines provide diverse extensions to adapt to fast-changing user demands. At the same time, ECMAScript, the official specification that describes the syntax and semantics of JavaScript, is annually updated since ECMAScript 6 (ES6, 2015) [7] to support new features in response to user demands. Such updates in both the specification and implementations in tandem make it difficult for them to be in sync.

Another example is Solidity [8], the standard smart contract programming language for the Ethereum blockchain. The Solidity language specification is continuously updated, and the Solidity compiler is also frequently released. According to Hwang and Ryu [9], the average number of days between consecutive releases from Solidity 0.1.2 to 0.5.7 is 27. In most cases, the Solidity compiler reflects updates in the specification, but even the specification is revised according to the semantics implemented in the compiler. As in JavaScript, bidirectional effects in the specification and the implementation make it hard to guarantee their correspondence.

In this approach, both the specification and the implementation may contain bugs, and guaranteeing their correctness is a challenging task. The conventional approach to build a programming language is uni-directional from a language specification to its implementation. The specification is believed to be correct and the conformance of an implementation to the specification is checked by dynamic testing. Unlike in the conventional approach, in the modern CI/CD approach, the specification may not be the oracle, because both the specification and the implementation can co-evolve.

In this paper, we propose a novel *N+1-version differential testing*, which enables testing of co-evolving specifications and their implementations. The differential testing [10] is a testing technique, which executes N implementations of a specification concurrently for each input, and detects a problem when the outputs are in disagreement. In addition to N implementations, our approach tests the specification as well using a mechanized specification. Recently, several approaches

ICSE 2021

43rd INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING

ACM SIGSOFT Distinguished Paper Award ICSE 2021

43rd ACM/IEEE International Conference on Software Engineering

May 23 – 29, 2021 Virtual (originally Madrid, Spain)

Presented to

Jihyeok Park
KAIST, South Korea

Seungmin An
KAIST, South Korea

Dongjun Youn
KAIST, South Korea

Gyeongwon Kim
KAIST, South Korea

Sukyoung Ryu
KAIST, South Korea

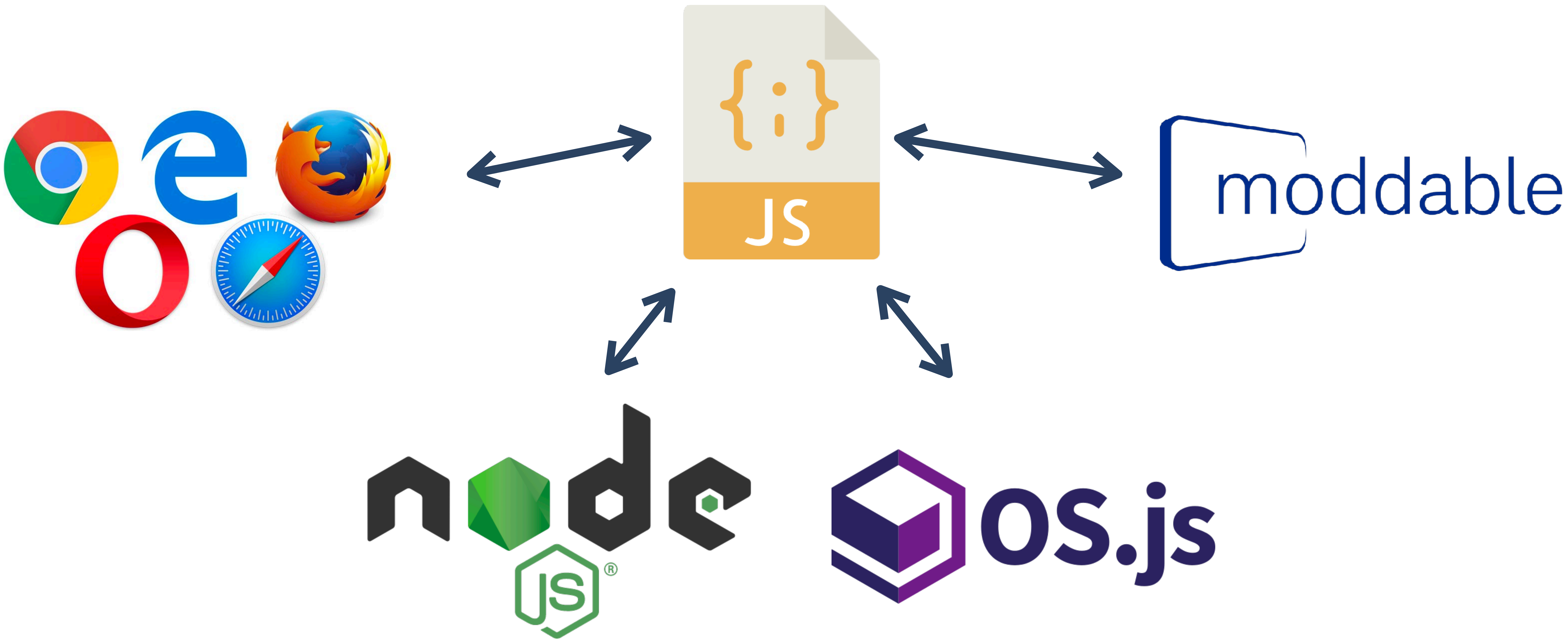
For

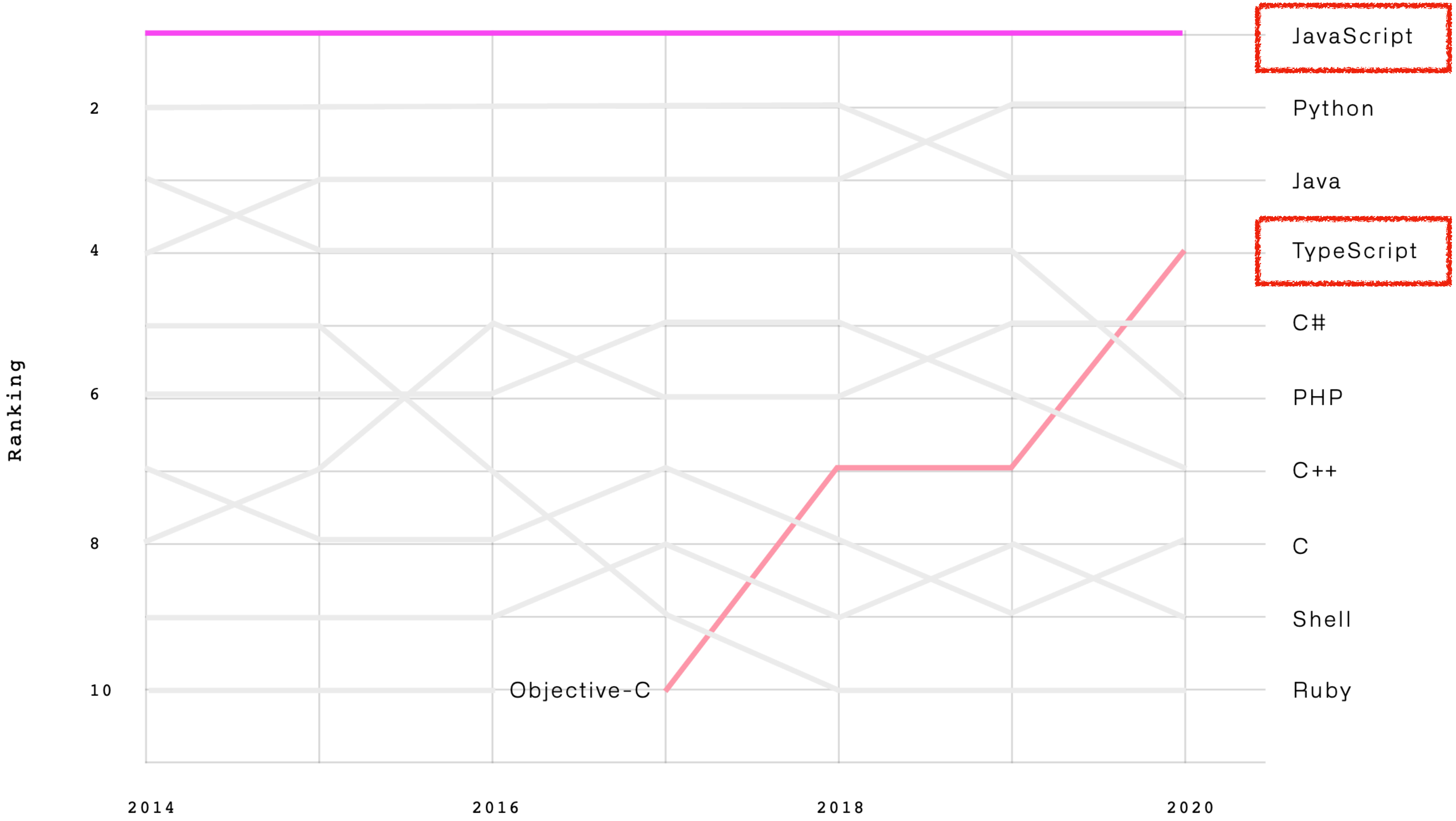
JEST: N+1-version Differential Testing of Both JavaScript Engines and Specification

Natalia Juristo	Arie van Deursen	Tao Xie
General Chair	Technical Track Program Co-Chairs	

75 YEARS IEEE COMPUTER SOCIETY · Association for Computing Machinery · tcse Technical Council on Software Engineering · SIGSOFT

JavaScript Is Everywhere





https://octoverse.github.com/

JavaScript Complex Semantics

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

Always return `false`?

NO!!

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

JavaScript Complex Semantics

```
(+(+!+[]+[+[]]+[+!+[]]))[(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+
[])[+[]]])[+!+[]+[+[]]]+([]+[])[([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!
+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+([]
[[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]
+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+
[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]][([][[]]+[])[+!+[]]+
(![]+[])[+!+[]]+((+[])[([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]
+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+([][[]]+[])
[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+(![]+
[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(!
[]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]+[])[+!+[]+[+!+[]]]+(!![]+
[])[!+[]+!+[]+!+[]]]](!+[]+!+[]+[+!+[]])[+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[!+[]
+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]
```

== "hello"

ECMAScript: JavaScript Specification

ecma INTERNATIONAL

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

Semantics

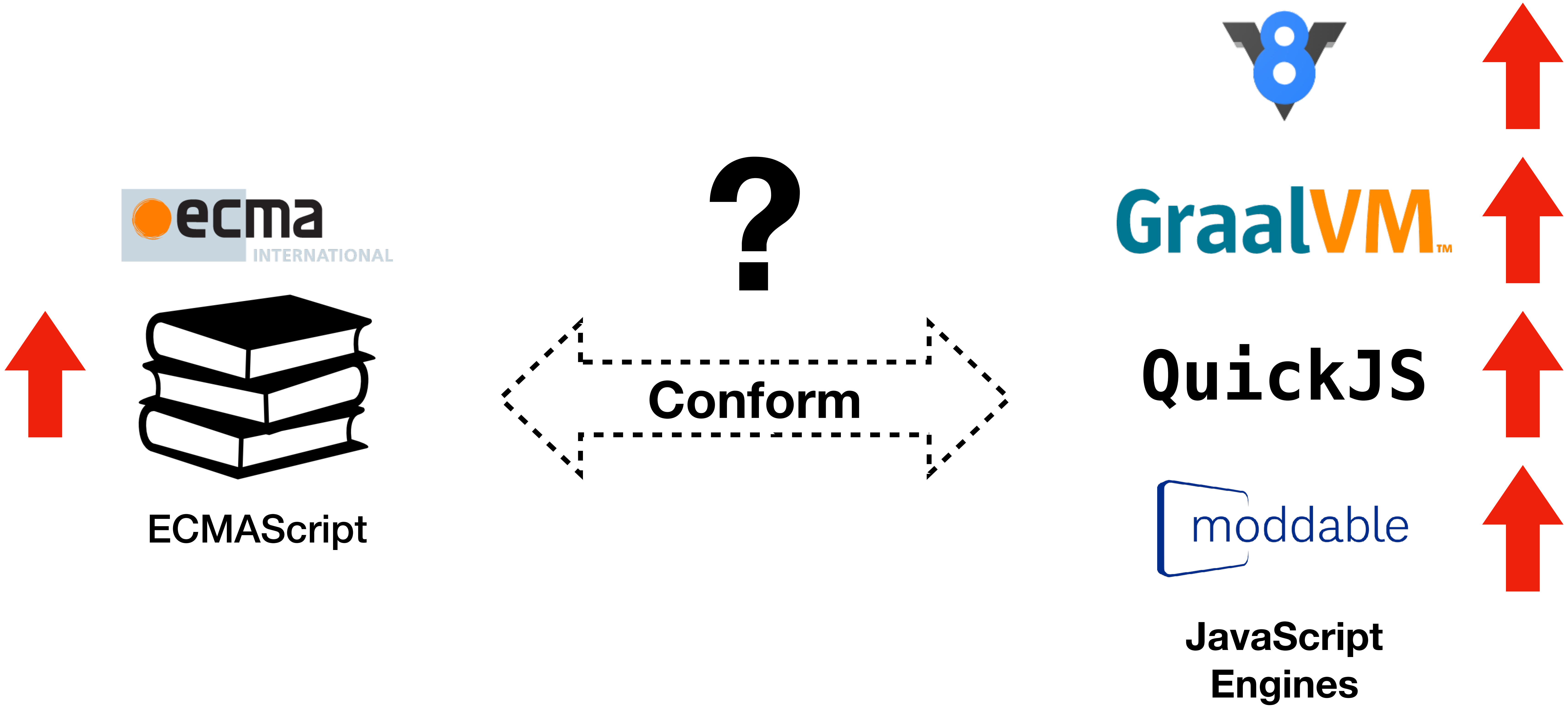
13.2.5.2 Runtime Semantics: Evaluation

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

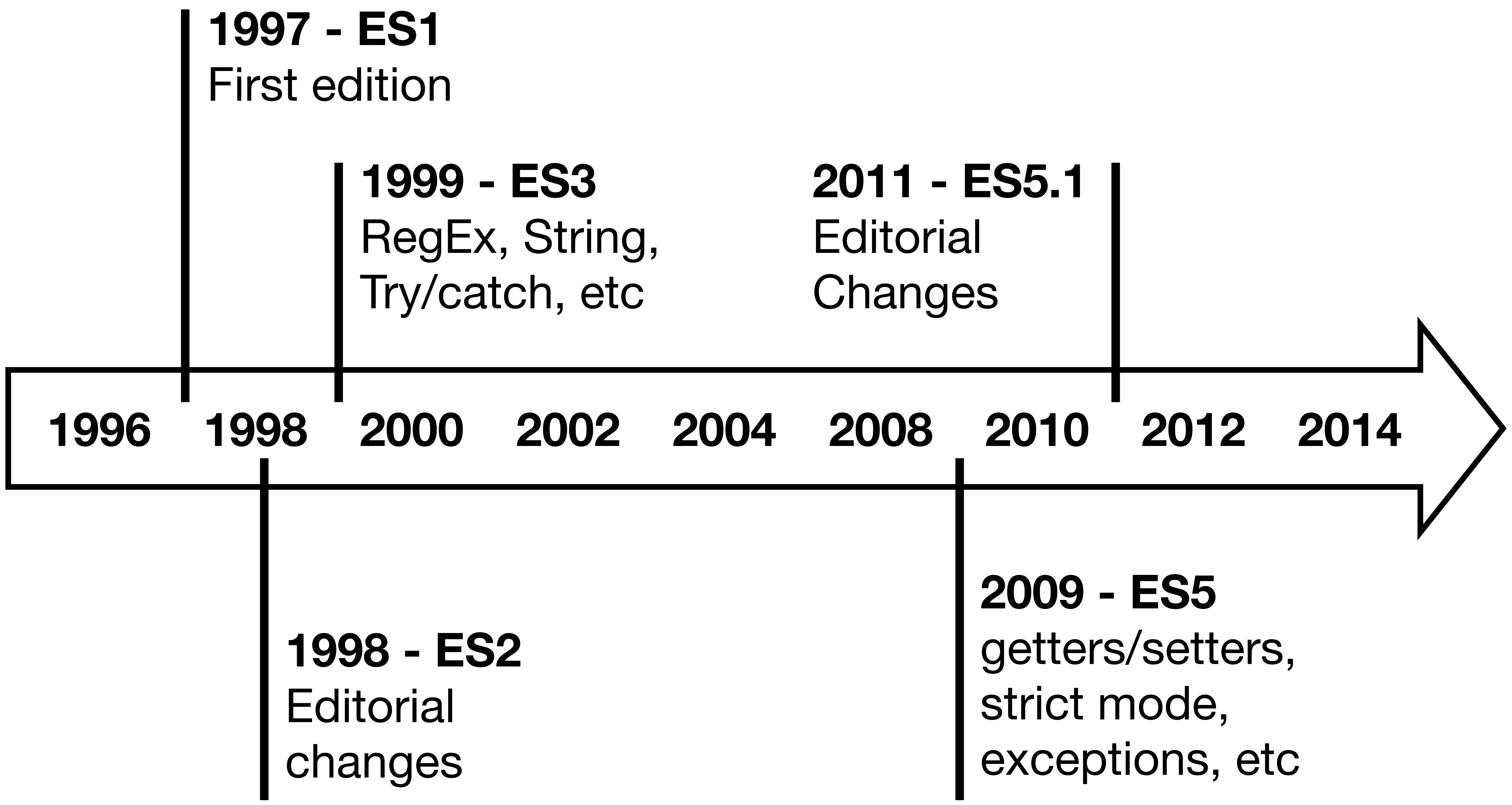
1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
   a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments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

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

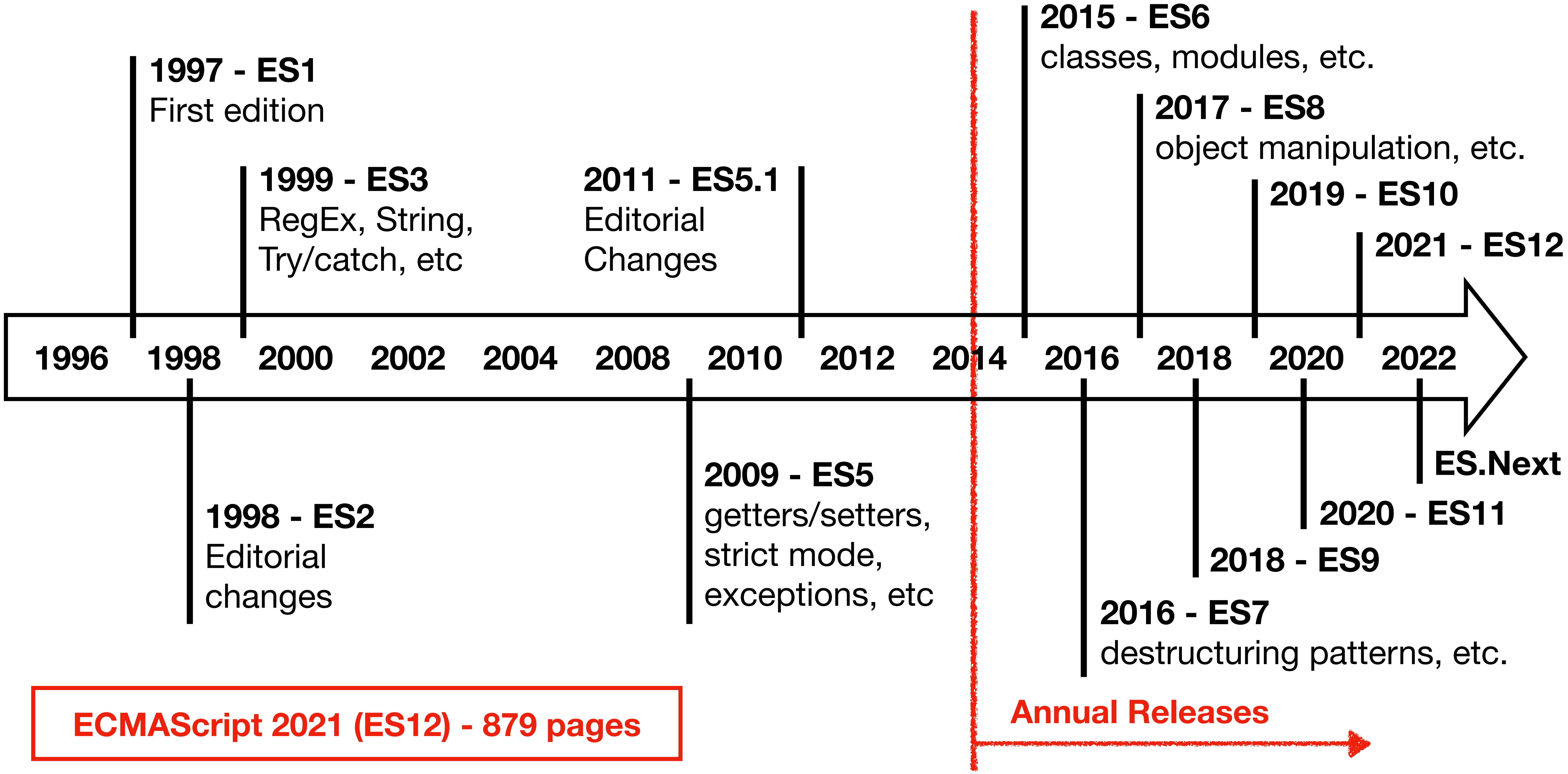
Problem: Manual Conformance Check



Problem: Fast Evolving JavaScript

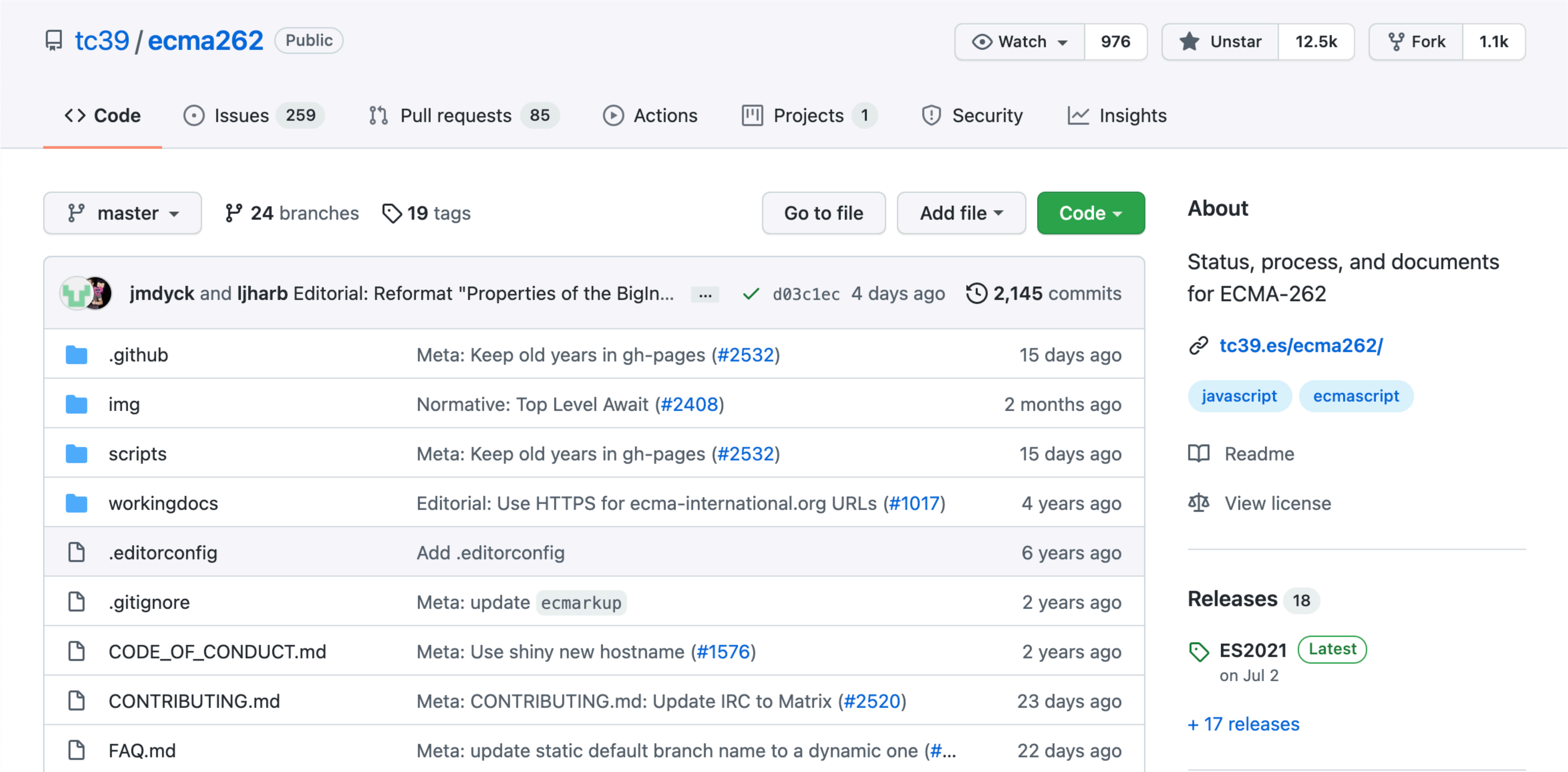


Problem: Fast Evolving JavaScript

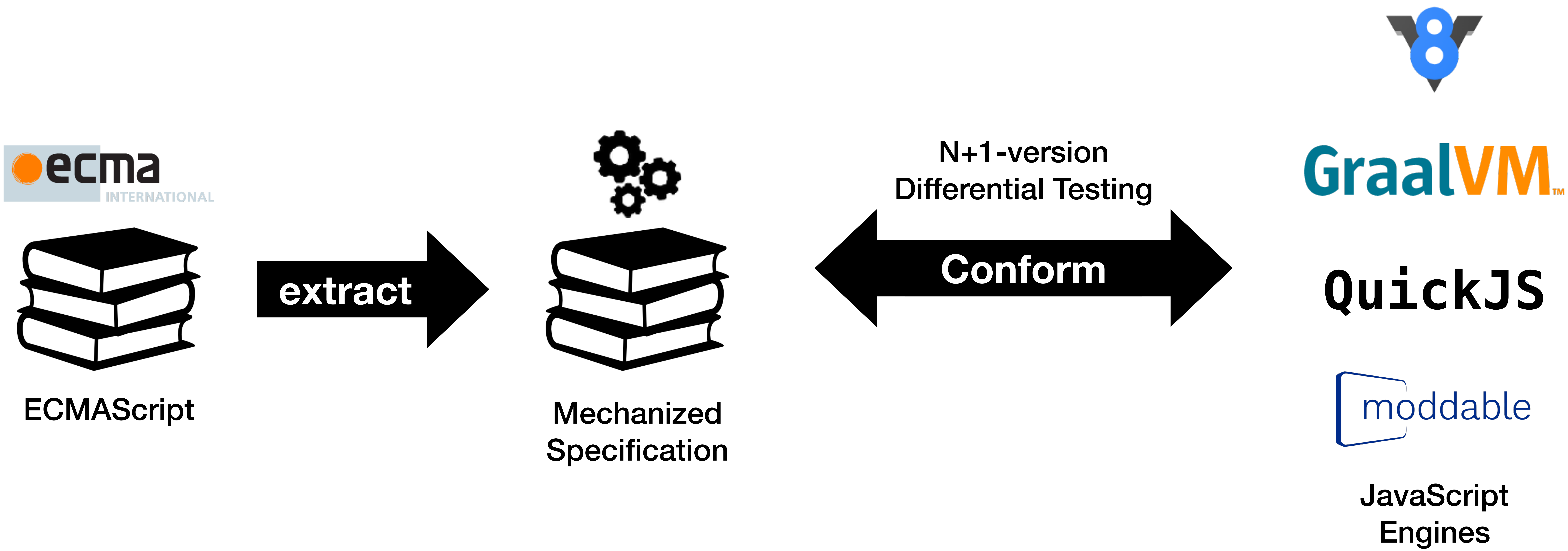


ECMAScript 2021 (ES12) - 879 pages

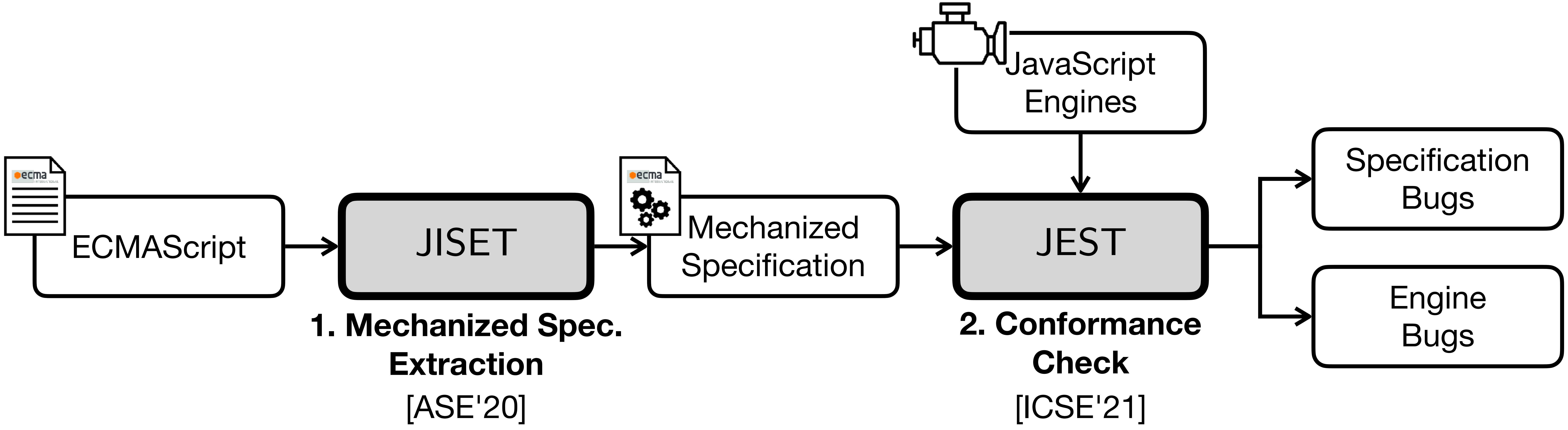
Problem: JavaScript is Open Source



Main Idea: Mechanized Specification

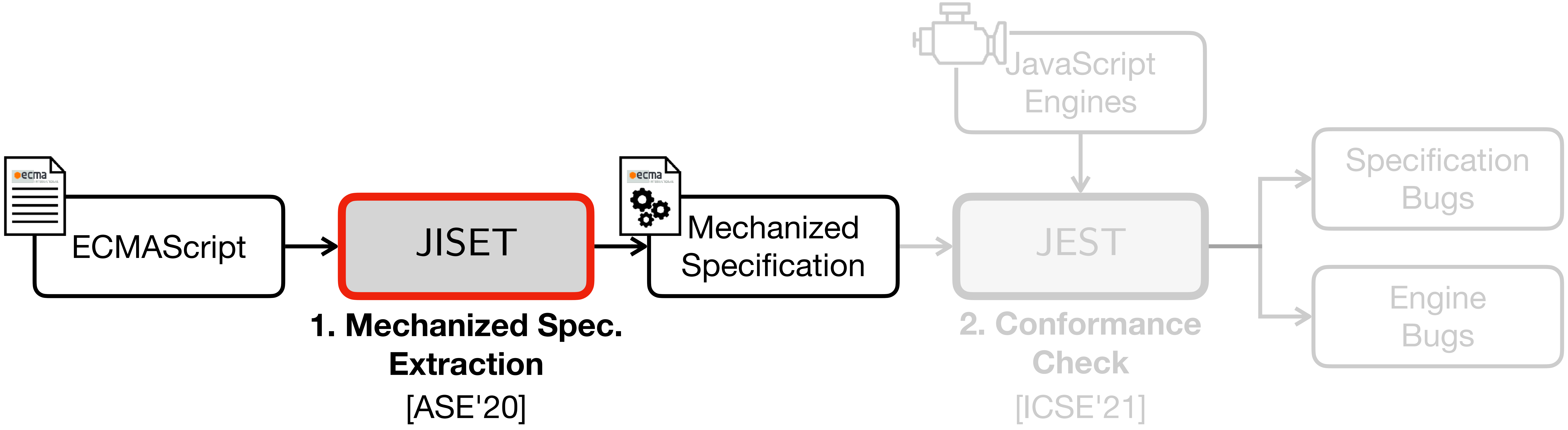


Overall Structure



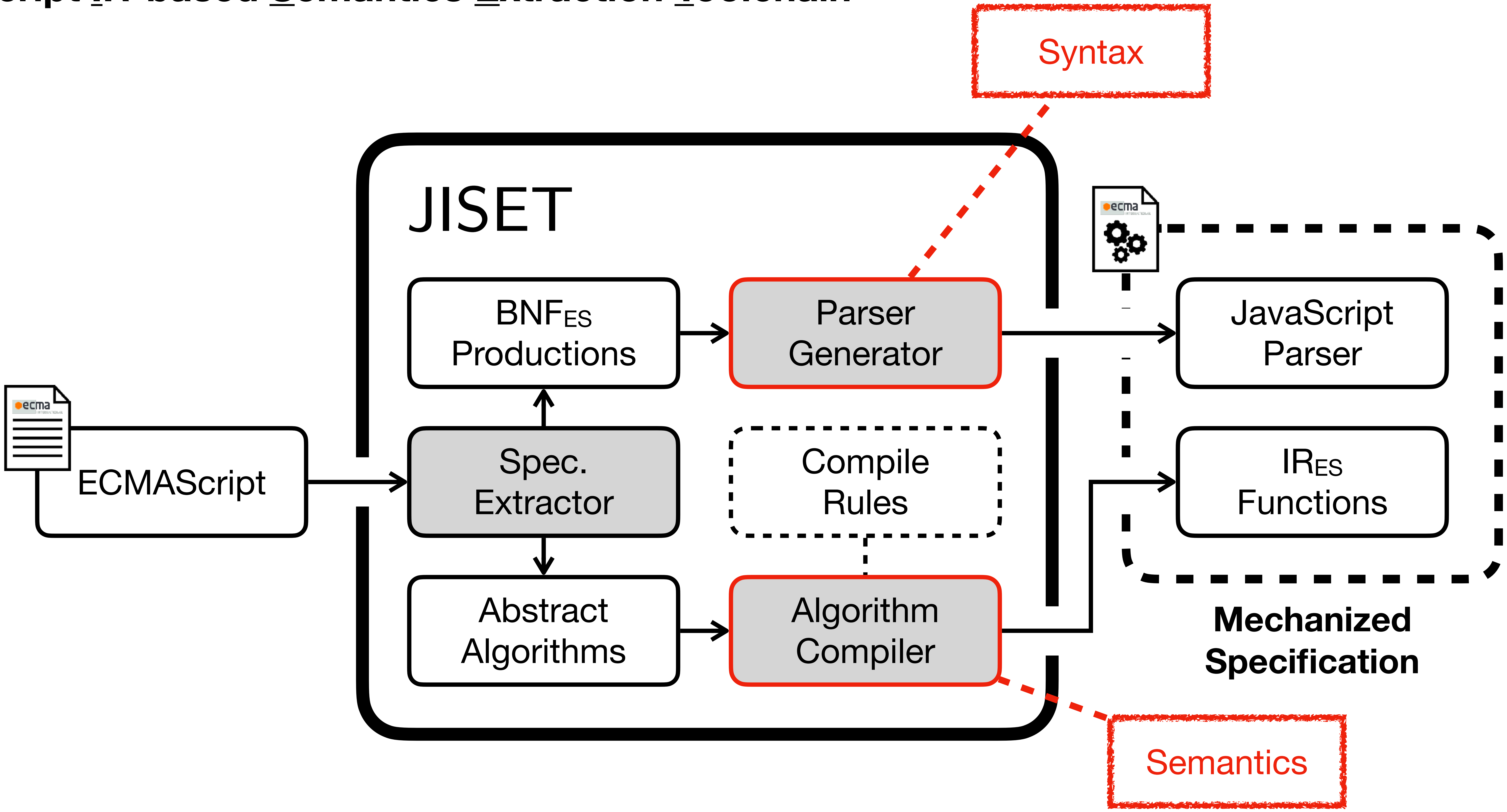
JISET: JavaScript IR-based Semantics Extraction Toolchain

Jihyeok Park, Jihee Park, Seungmin An, and Sukyoung Ryu
(Published in ASE'20)



JISET [ASE'20]

JavaScript IR-based Semantics Extraction Toolchain



JISET - Parser Generator (Syntax)

ArrayLiteral[Yield, Await] :

[*Elision*opt]

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

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

<u>Parsing Expression Grammar</u>
(+ Lookahead Parsing)

```
val ArrayLiteral: List[Boolean] => LAParser[T] = memo {
  case List(Yield, Await) =>
  "[" ~ opt(Elision) ~ "]"                        ^^ ArrayLiteral0 |
  "[" ~ ElementList(Yield,Await) ~ "]" ^^ ArrayLiteral1 |
  "[" ~ ElementList(Yield,Await) ~ ","
      ~ opt(Elision)~ "]"                         ^^ ArrayLiteral2
}
```

(POPL'04) Bryan Ford, "Parsing Expression Grammars: A Recognition-based Syntactic Foundation"

- **Context-Free Grammar (CFG)**
  - Unordered Choices

$$A ::= B \texttt{;} \mid B \texttt{+} B \texttt{;}$$
$$B ::= \texttt{x} \mid \texttt{xy}$$

`xy;` ✓

`x+x;` ✓

- **Context-Free Grammar (CFG)**
  - Unordered Choices

$$A ::= B\texttt{;} \mid B \texttt{+} B\texttt{;}$$
$$B ::= \texttt{x} \mid \texttt{xy}$$

`xy;` ✓

`x+x;` ✓

- **Parsing Expression Grammar (PEG)**
  - Ordered Choices

$$A ::= B\texttt{;} \ / \ B \texttt{+} B\texttt{;}$$
$$B ::= \texttt{x} \ / \ \boxed{\texttt{xy}}$$

always ignored

`xy;` ✗

`x+x;` ✓

- **Context-Free Grammar (CFG)**
  - Unordered Choices

$$A ::= B\,\texttt{;} \mid B \;\texttt{+}\; B\,\texttt{;}$$
$$B ::= \texttt{x} \mid \texttt{xy}$$

`xy;` ✓

`x+x;` ✓

- **Parsing Expression Grammar (PEG)**
  - Ordered Choices

$$A ::= B\,\texttt{;} \;/\; B \;\texttt{+}\; B\,\texttt{;}$$
$$B ::= \texttt{x} \;/\; \boxed{\texttt{xy}}$$

always ignored

`xy;` ✗

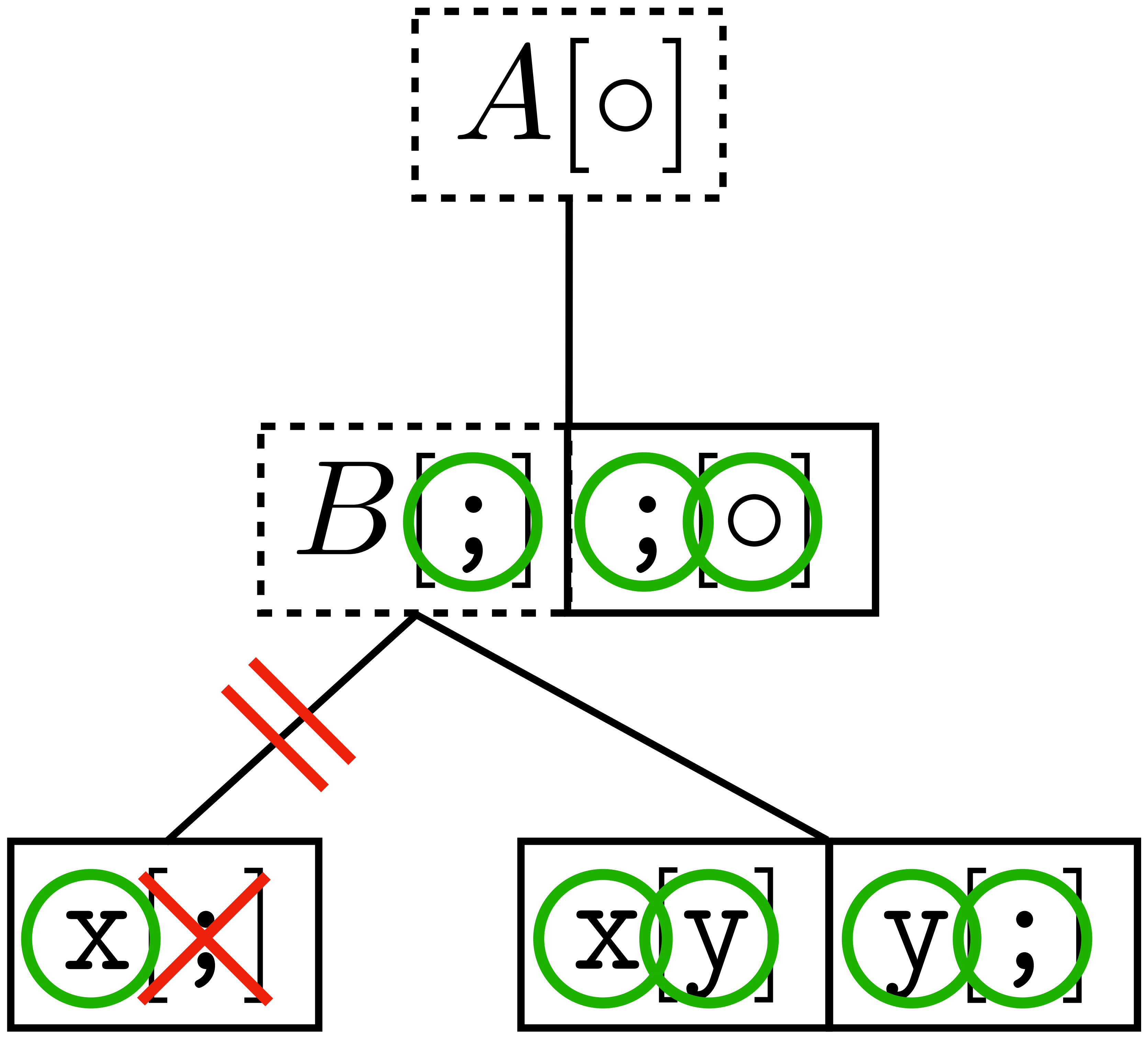
`x+x;` ✓

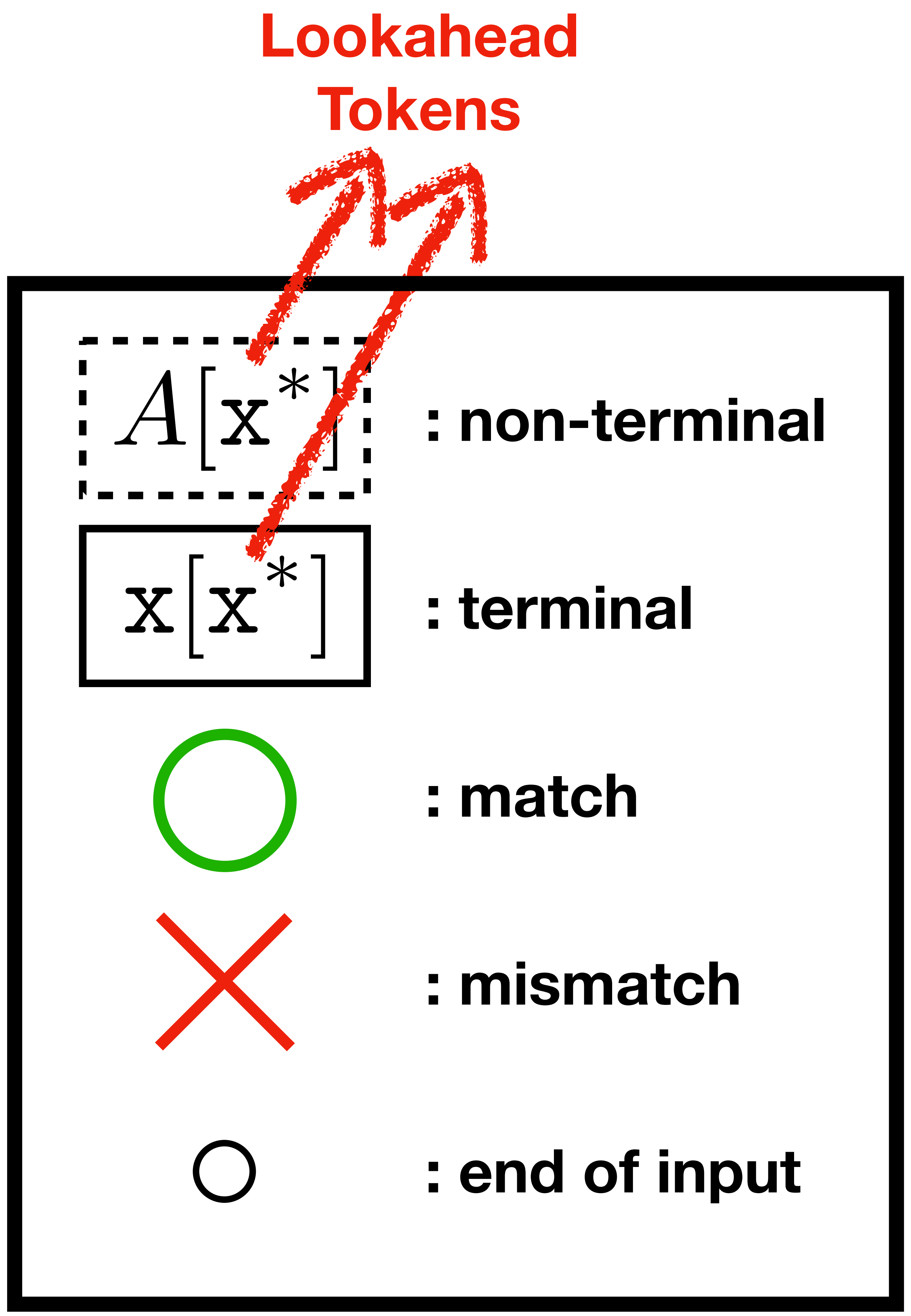
- **PEG with *Lookahead Parsing***
  - Ordered Choices with *Lookahead Tokens*

$$A ::= B\,\texttt{;} \;/\; B \;\texttt{+}\; B\,\texttt{;}$$
$$B ::= \texttt{x} \;/\; \texttt{xy}$$

`xy;` ✓

`x+x;` ✓





$$A ::= B\,; \;/\; B + B\,;$$
$$B ::= \mathrm{x} \;/\; \mathrm{xy}$$

input : x y ;

$$
\begin{aligned}
\mathbf{first}_\alpha(s_1 \cdots s_n) &= \mathbf{first}_s(s_1) :\!+ \mathbf{first}_s(s_2 \cdots s_n) \\
&\quad \text{where } x :\!+ y = \begin{cases} x \cup y & \text{if } \circ \in x \\ x & \text{otherwise} \end{cases} \\
\mathbf{first}_s(\epsilon) &= \{\circ\} \\
\mathbf{first}_s(\mathsf{a}) &= \{\mathsf{a}\} \\
\mathbf{first}_s(A(a_1, \cdots, a_k)) &= \mathbf{first}_\alpha(\alpha_1) \cup \cdots \cup \mathbf{first}_\alpha(\alpha_n) \\
&\quad \text{where } A(a_1, \cdots, a_k) = \alpha_1 \mid \cdots \mid \alpha_n \\
\mathbf{first}_s(s?) &= \mathbf{first}_s(s) \cup \{\circ\} \\
\mathbf{first}_s(+s) &= \mathbf{first}_s(s) \\
\mathbf{first}_s(-s) &= \{\circ\} \\
\mathbf{first}_s(s \setminus s') &= \mathbf{first}_s(s) \\
\mathbf{first}_s(\langle \neg \mathsf{LT} \rangle) &= \{\circ\}
\end{aligned}
$$

Algorithm for first tokens of BNF$_{\text{ES}}$

Algorithm for lookahead parsing

$$
\begin{aligned}
(s_1 \cdots s_n)[L] &= s_1[\mathbf{first}_s(s_2 \cdots s_n) :\!+ L]\ (s_1 \cdots s_n)[L] \\
\epsilon[L] &= +\mathbf{get}_s(L) \\
\mathsf{a}[L] &= \mathsf{a}\ + \mathbf{get}_s(L) \\
A(a_1, \cdots, a_k)[L] &= \alpha_1[L] \mid \cdots \mid \alpha_n[L] \\
&\quad \text{where } A(a_1, \cdots, a_k) = \alpha_1 \mid \cdots \mid \alpha_n \\
s?[L] &= s[L] \mid \epsilon[L] \\
(\pm s)[L] &= \pm(s[L]) \\
(s \setminus s')[L] &= s[L] \setminus s' \\
\langle \neg \mathsf{LT} \rangle &= \langle \neg \mathsf{LT} \rangle\ + \mathbf{get}_s(L)
\end{aligned}
$$

JISET - Algorithm Compiler (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*.

118 Compile Rules for
Steps in Abstract Algorithms

```
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
}
```

Parsing rules | Conversion Rules

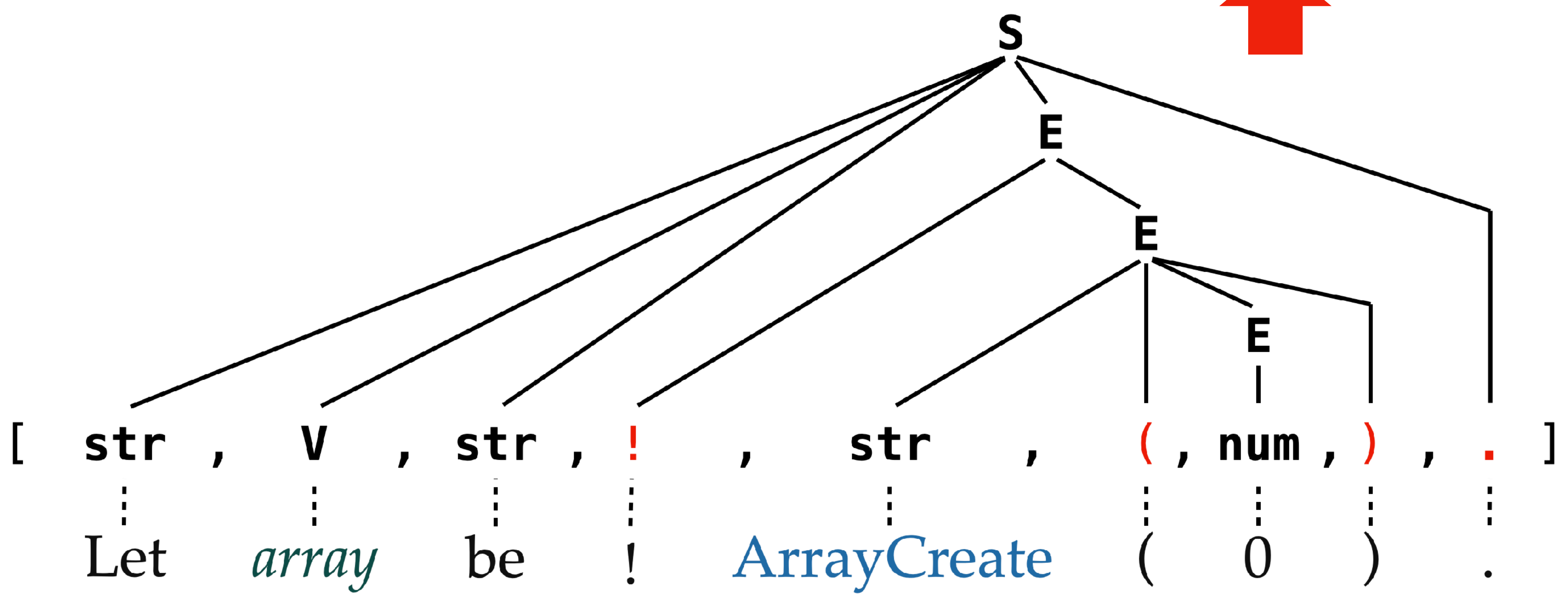
```
S = // statements
  Let ~ V ~ be ~ E ~ .  ^^ ILet

E = // expressions
  ! E                   ^^ EAbruptCheck |
  str ~ ( ~ E ~ )       ^^ ECall        |
  num                   ^^ _.toDouble
```

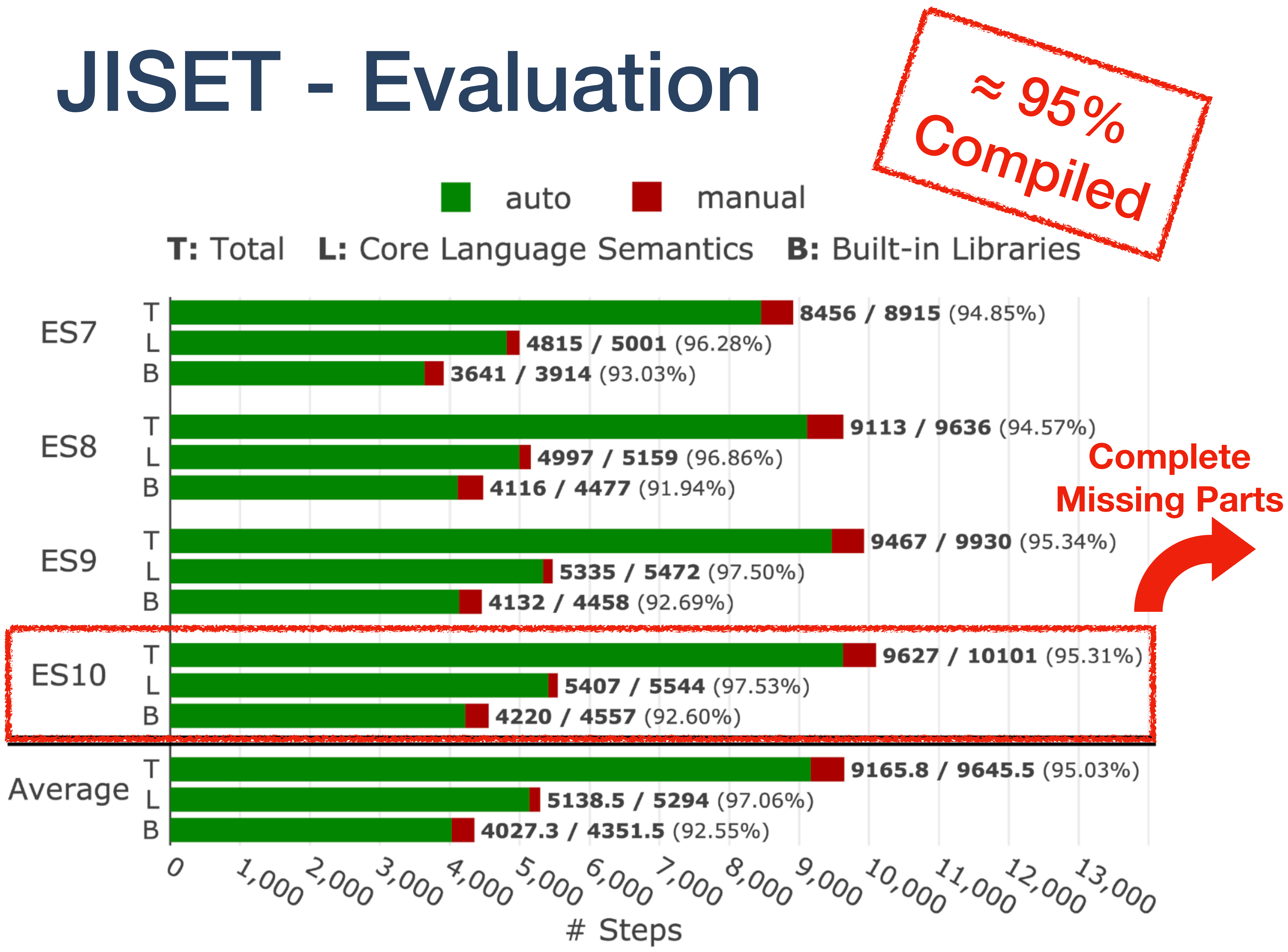
Simplified compile rules

```
let array = ! (ArrayCreate 0)
```

```
ILet(array, EAbruptCheck(
  ECall("ArrayCreate", 0)))
```



JISET - Evaluation



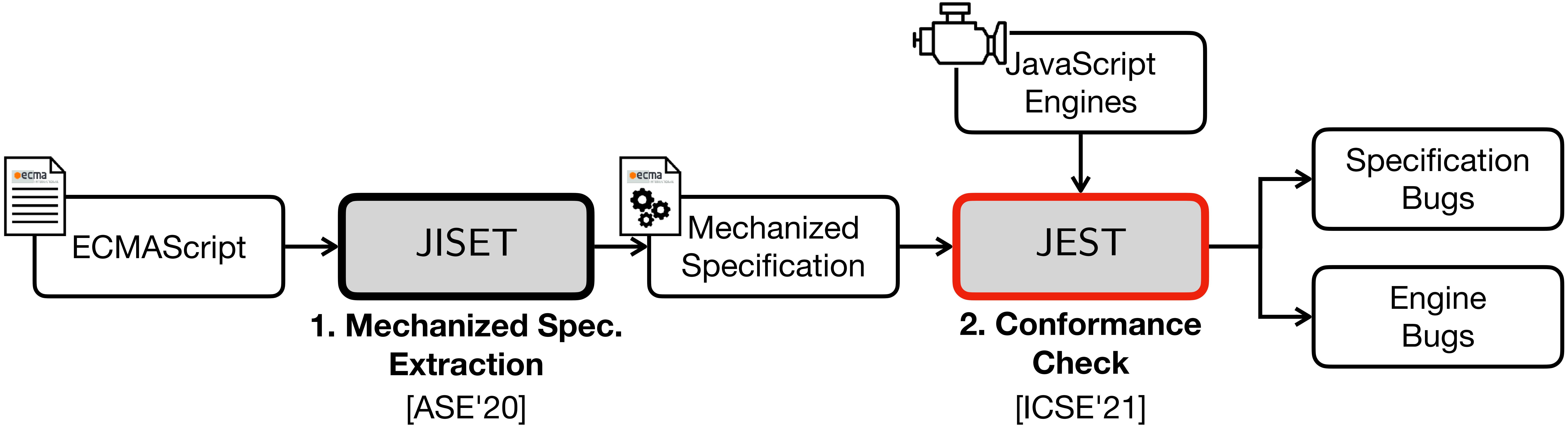
Passed All Tests

- **Test262**
  (Official Conformance Tests)
  - 18,064 applicable tests
- **Parsing tests**
  - Passed all 18,064 tests
- **Evaluation Tests**
  - Passed all 18,064 tests

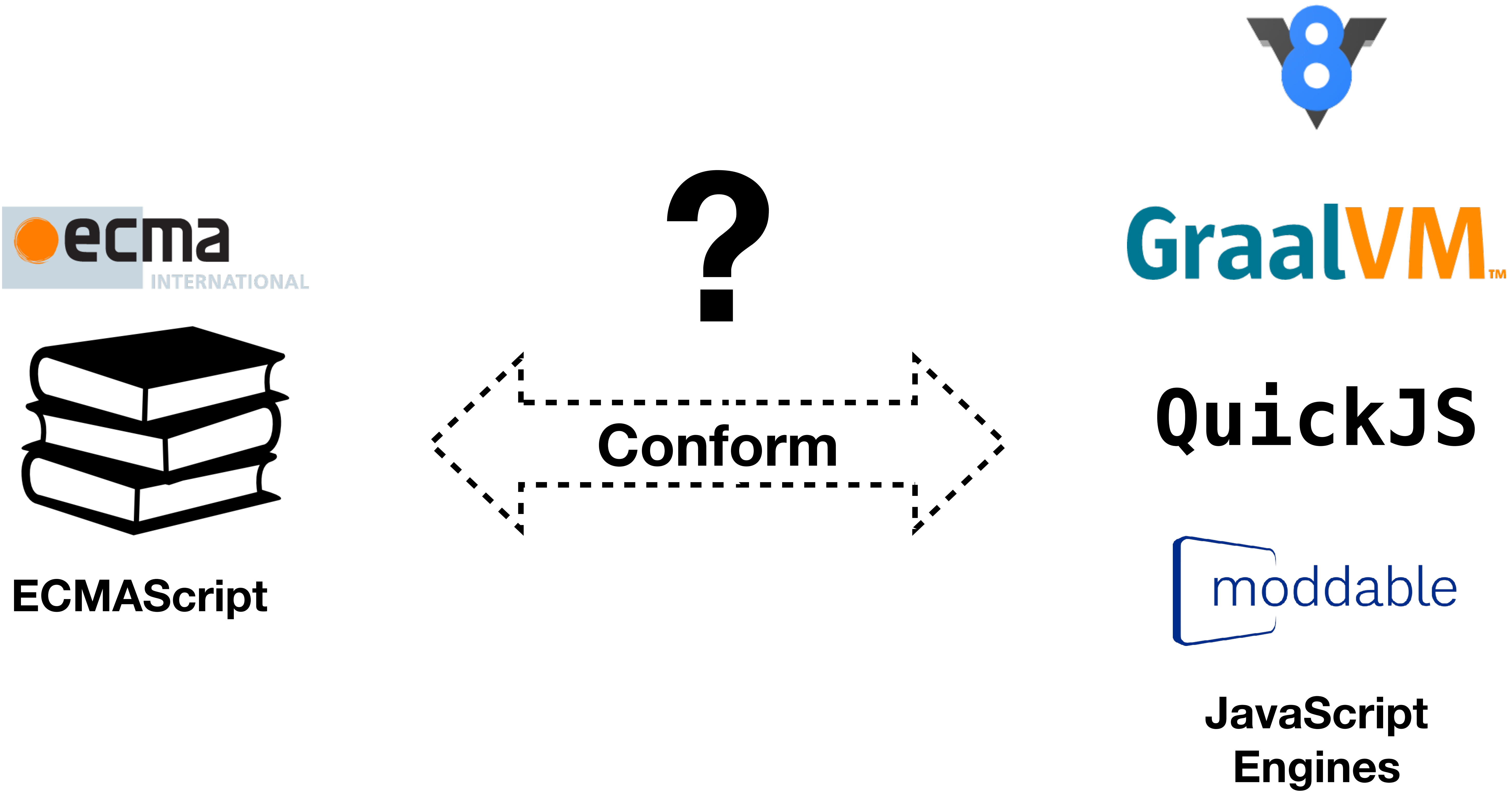
JEST: N+1-version Differential Testing of Both JavaScript Engines

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

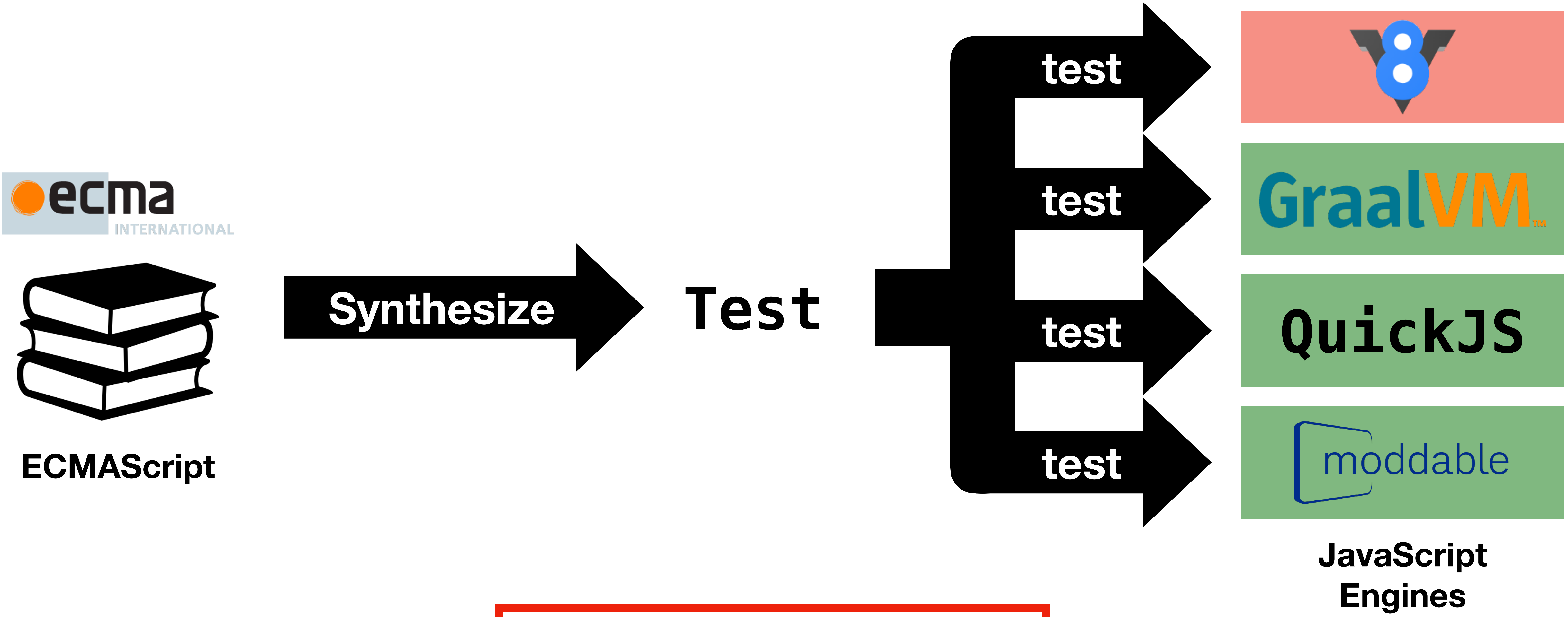
(Published in ICSE'21)



JEST - Conformance with Engines

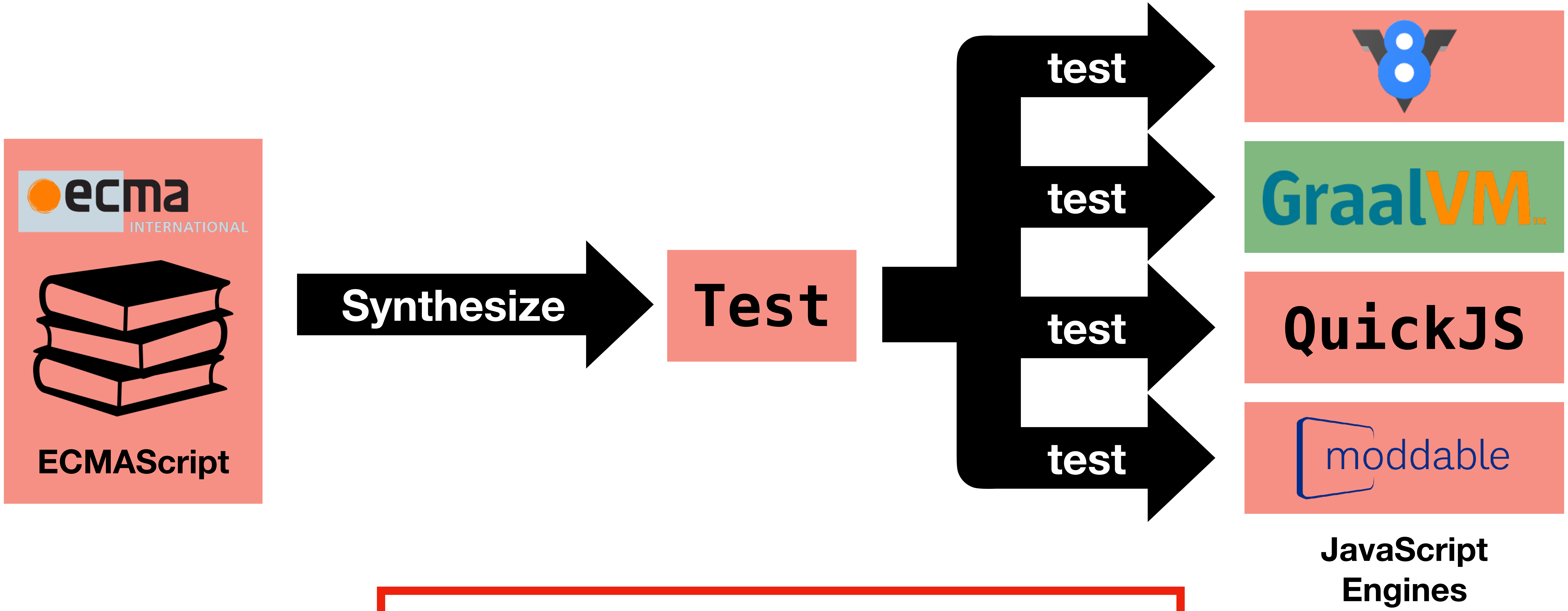


JEST - N+1-version Differential Testing



An engine bug in

JEST - N+1-version Differential Testing

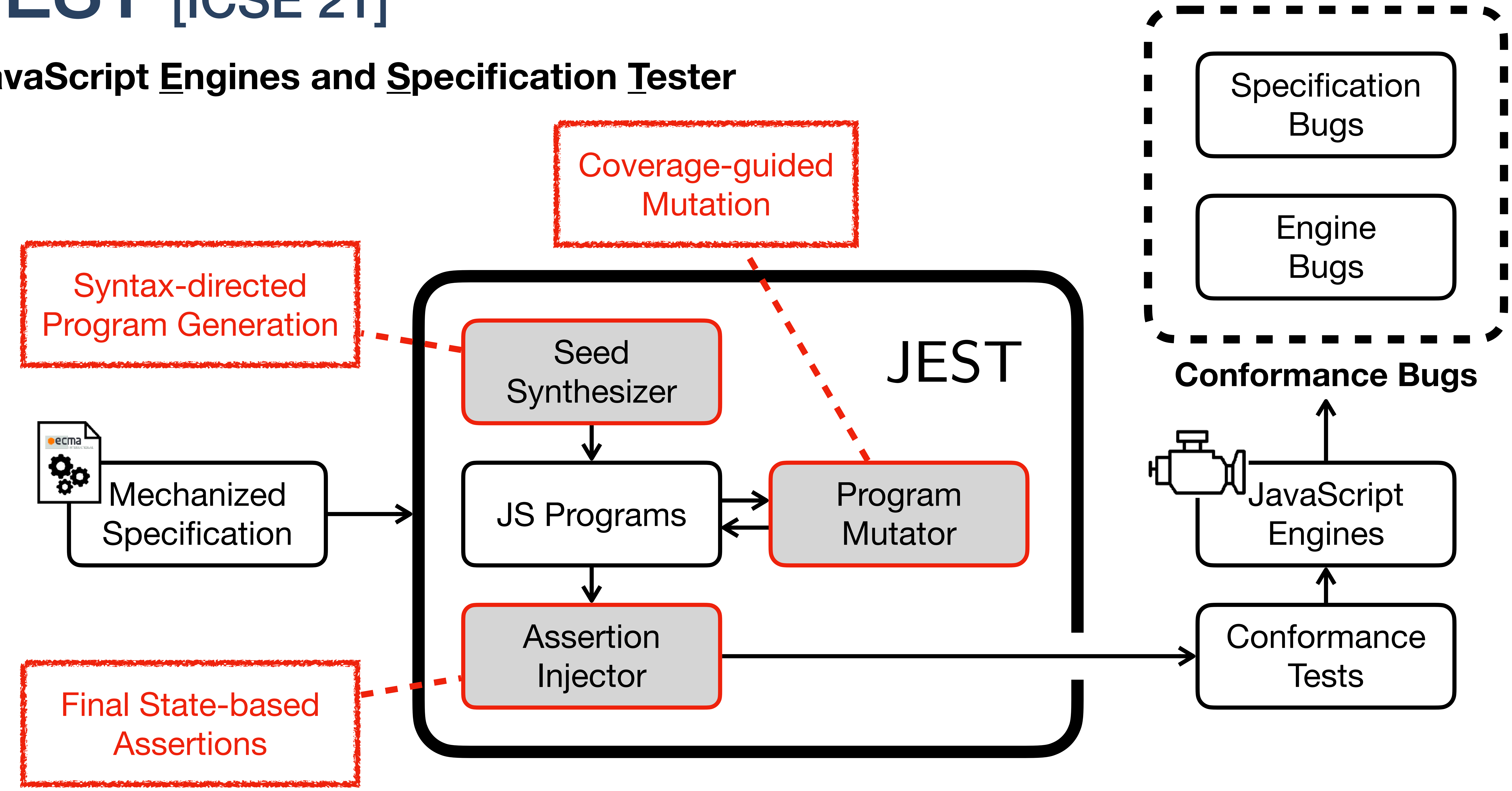


A specification bug in ECMAScript

An engine bug in GraalVM

JEST [ICSE'21]

JavaScript Engines and Specification Tester



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

44 Bugs in Engines

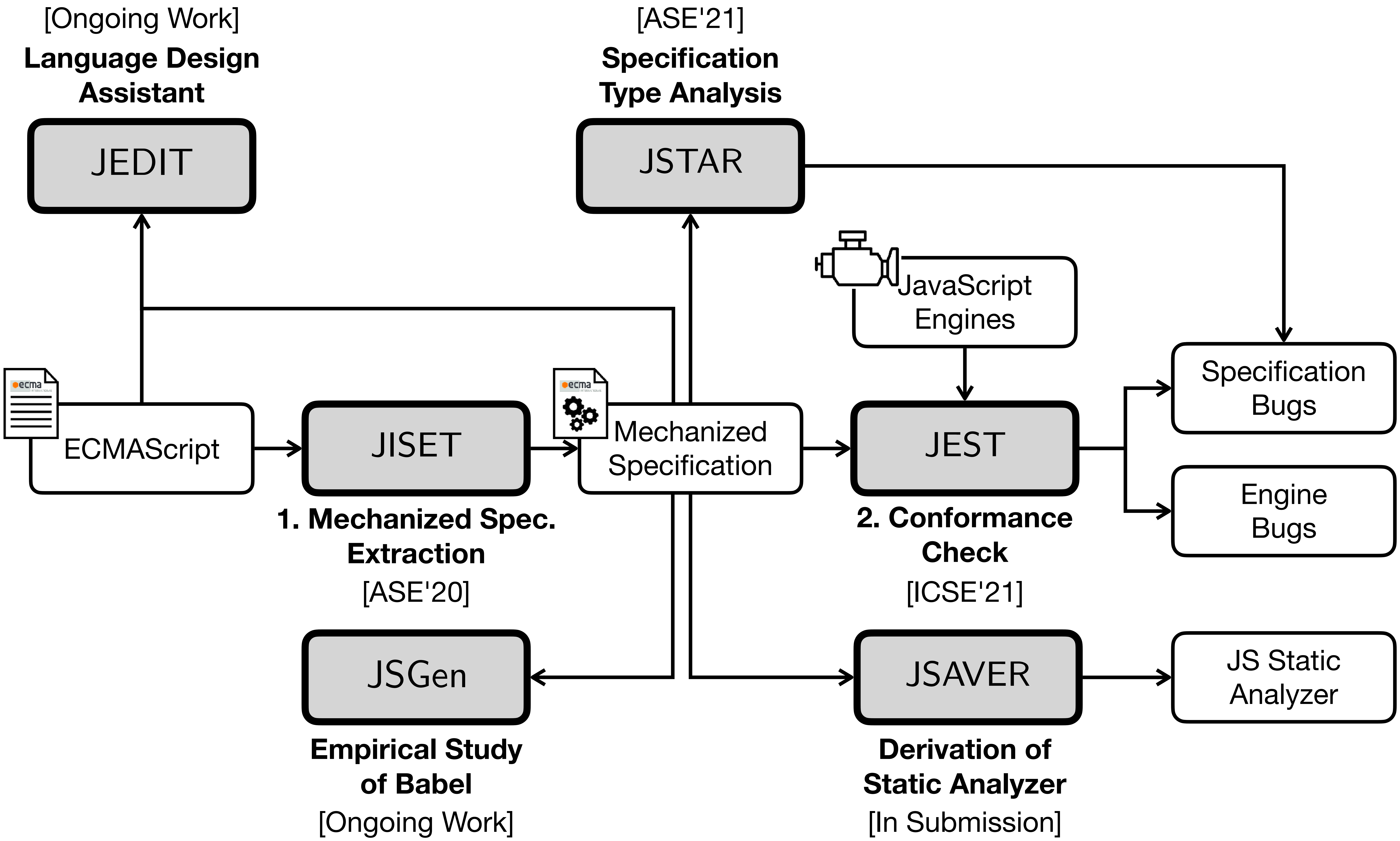
TABLE II: The number of engine bugs detected by JEST

Engines	Exc	Abort	Var	Obj	Desc	Key	In	Total
V8	0	0	0	0	0	2	0	2
GraalJS	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


[Ongoing Work]
Language Design Assistant
JEDIT
[ASE'21]
Specification Type Analysis
JSTAR
JavaScript Engines
Specification Bugs
ECMAScript
JISET
Mechanized Specification
JEST
Engine Bugs
1. Mechanized Spec. Extraction
[ASE'20]
2. Conformance Check
[ICSE'21]
JSGen
JSAVER
JS Static Analyzer
Empirical Study of Babel
[Ongoing Work]
Derivation of Static Analyzer
[In Submission]