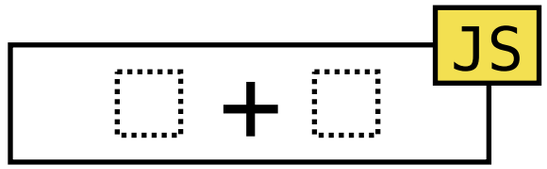
Trusted JavaScript Language Environments with ESMeta

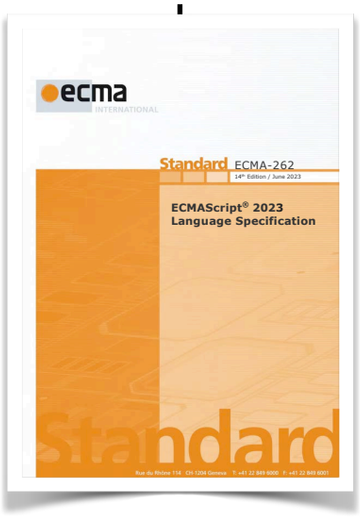
Jihyeok Park

2025.07.04 @ PLSS 2025

Language Specification (ECMA-262) of JavaScript



TC 39



ECMA-262
(JavaScript Spec.)

Syntax

AdditiveExpression`[Yield, Await]` :

MultiplicativeExpression`[?Yield, ?Await]`

AdditiveExpression`[?Yield, ?Await]` **+** *MultiplicativeExpression*`[?Yield, ?Await]`

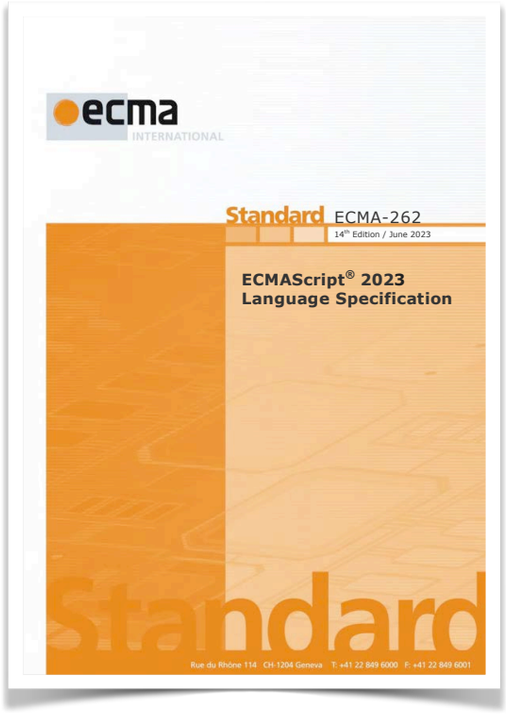
AdditiveExpression`[?Yield, ?Await]` **−** *MultiplicativeExpression*`[?Yield, ?Await]`

AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*

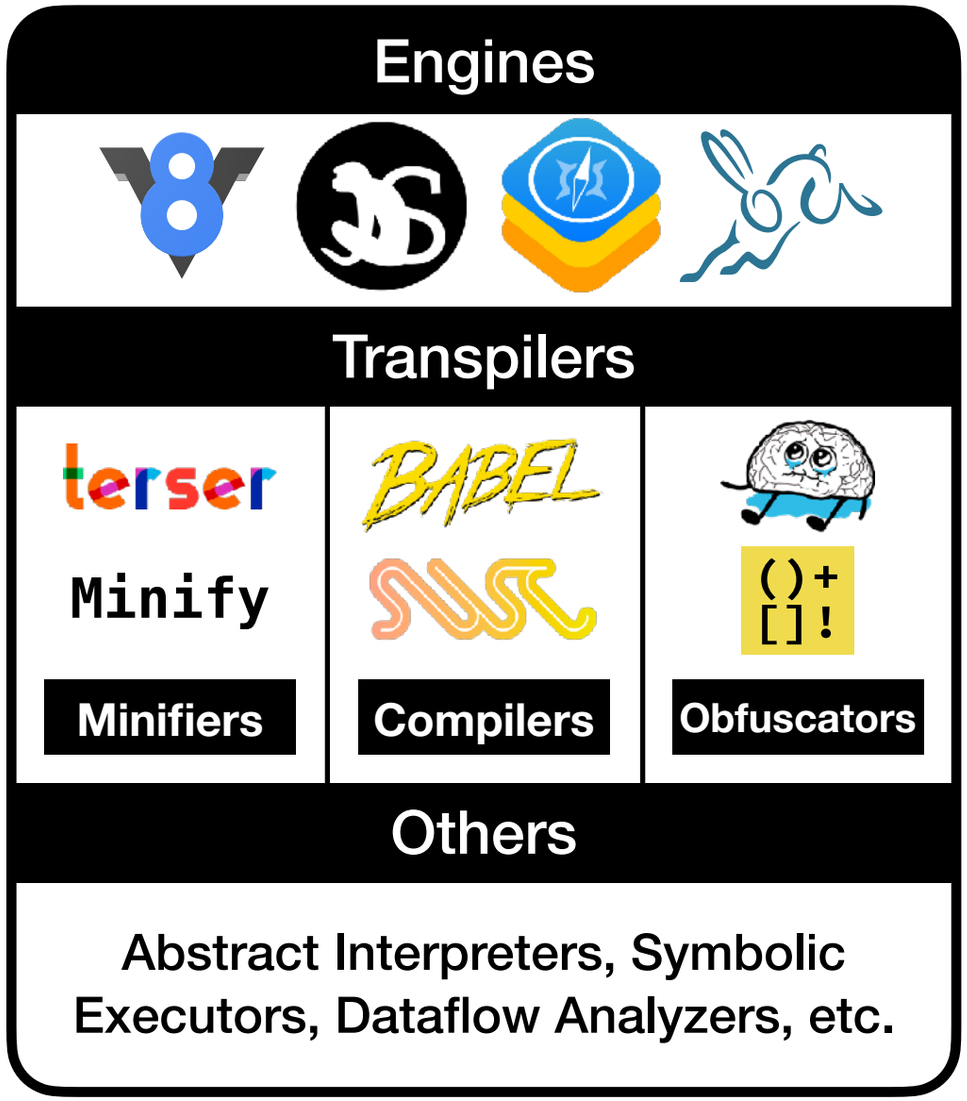
1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

Semantics

Design and Implementation of JavaScript



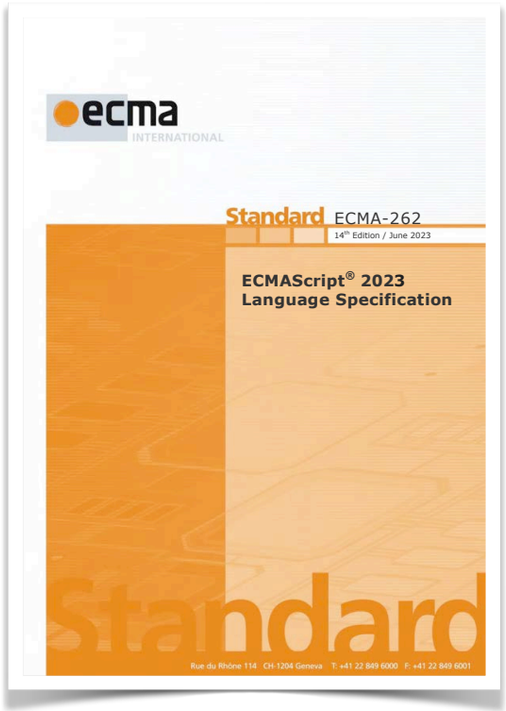
ECMA-262
(JavaScript Spec.)



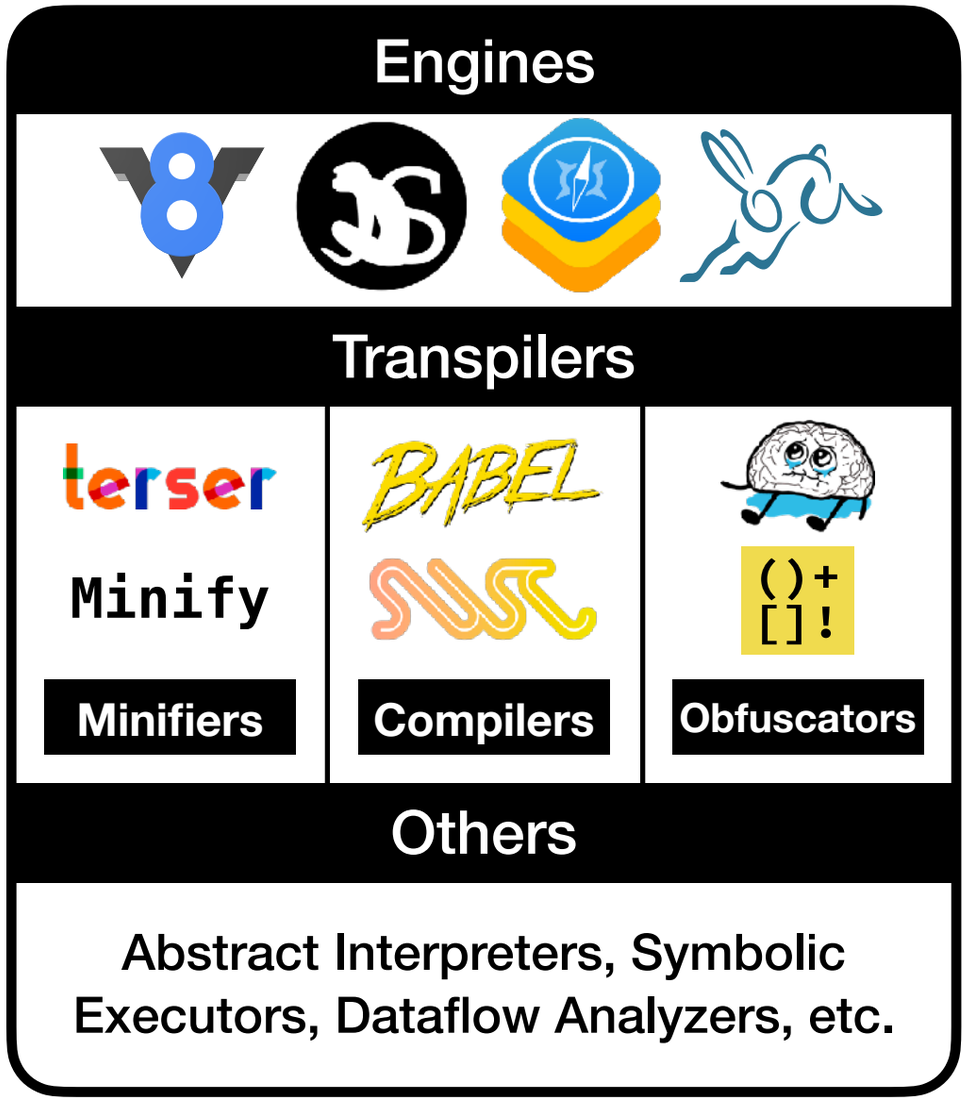
JavaScript Implementations

Design and Implementation of JavaScript

Update



ECMA-262
(JavaScript Spec.)

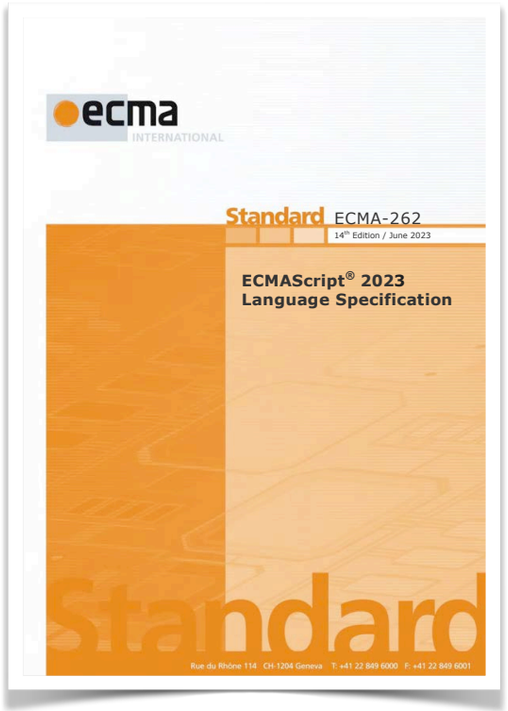


JavaScript Implementations

Update

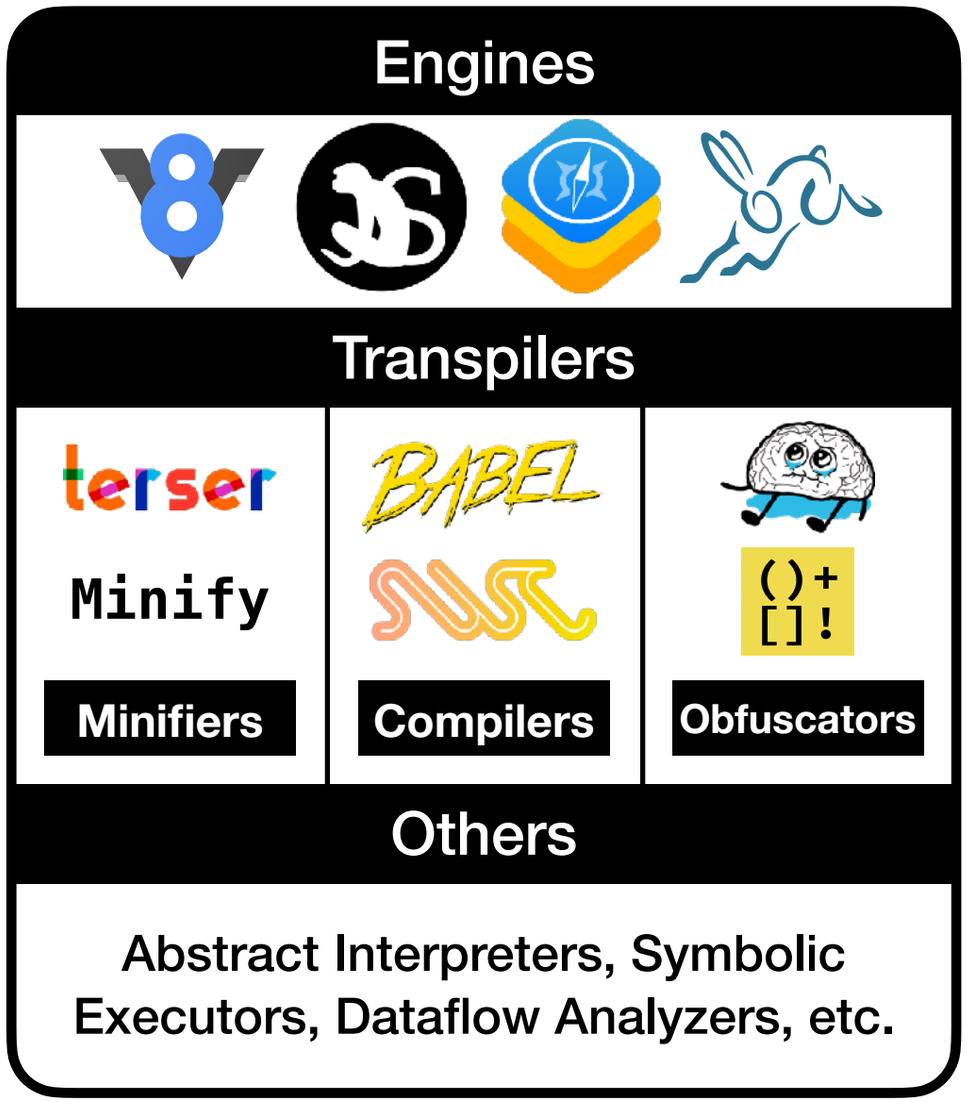
Design and Implementation of JavaScript

Update



ECMA-262
(JavaScript Spec.)

Conformance

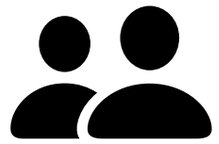


JavaScript Implementations

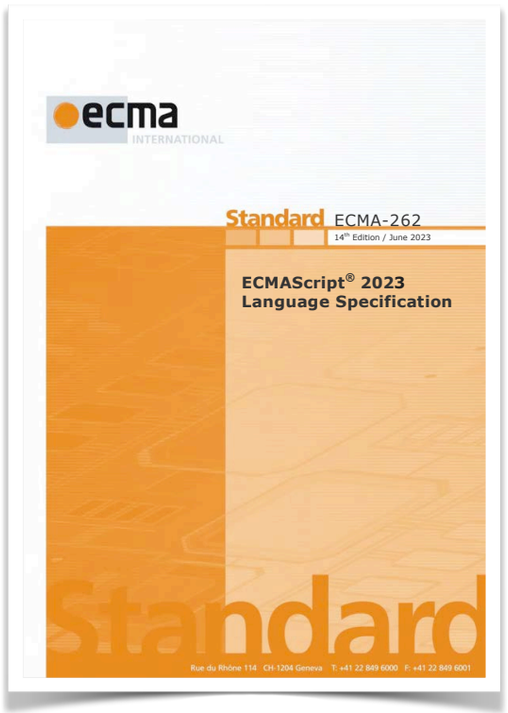
Update

Design and Implementation of JavaScript

MANUAL



Update



ECMA-262
(JavaScript Spec.)

Conformance

Engines

Transpilers

terser

Minify

Minifiers

BABEL

Compilers

()+
[]!

Obfuscators

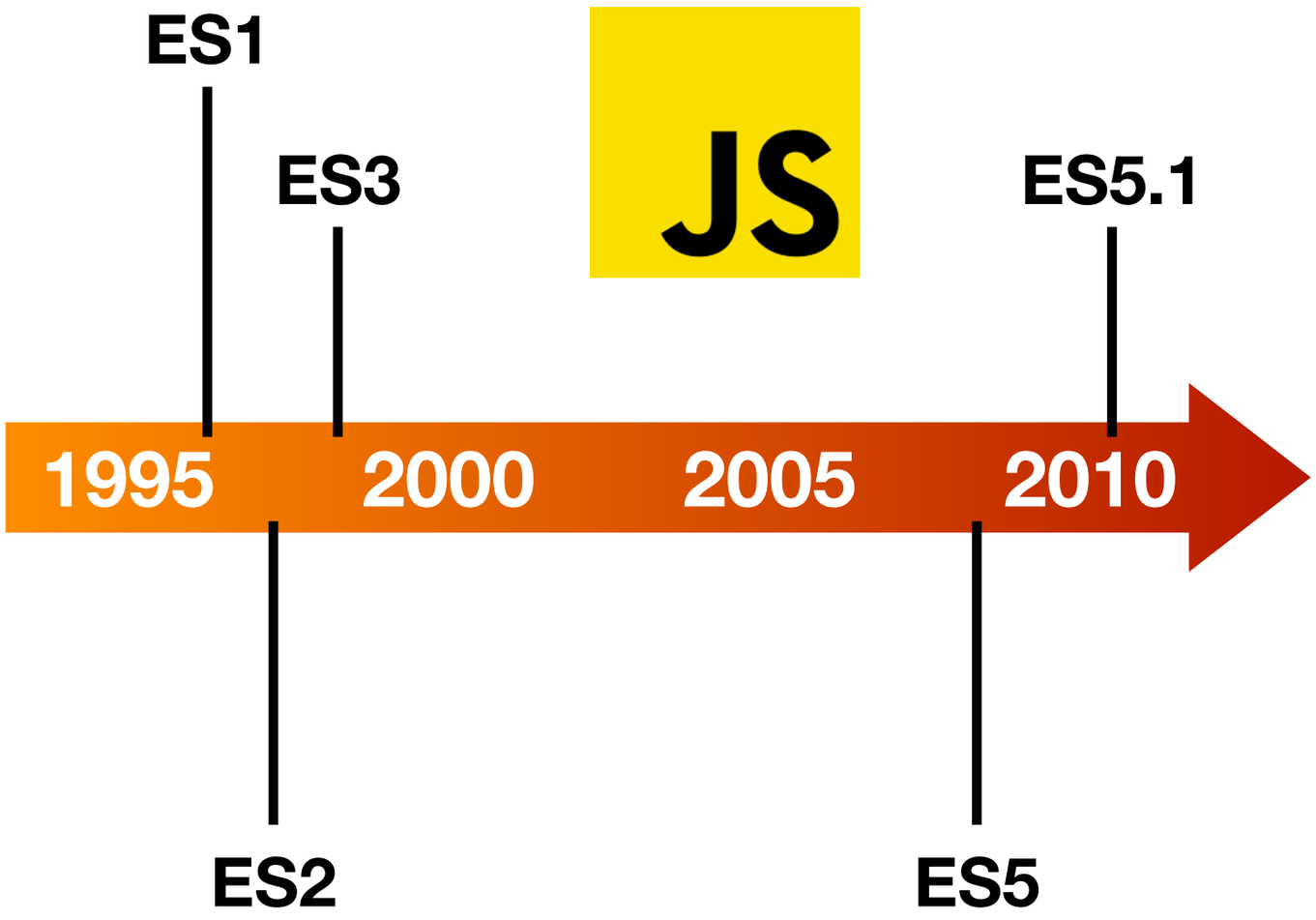
Others

Abstract Interpreters, Symbolic Executors, Dataflow Analyzers, etc.

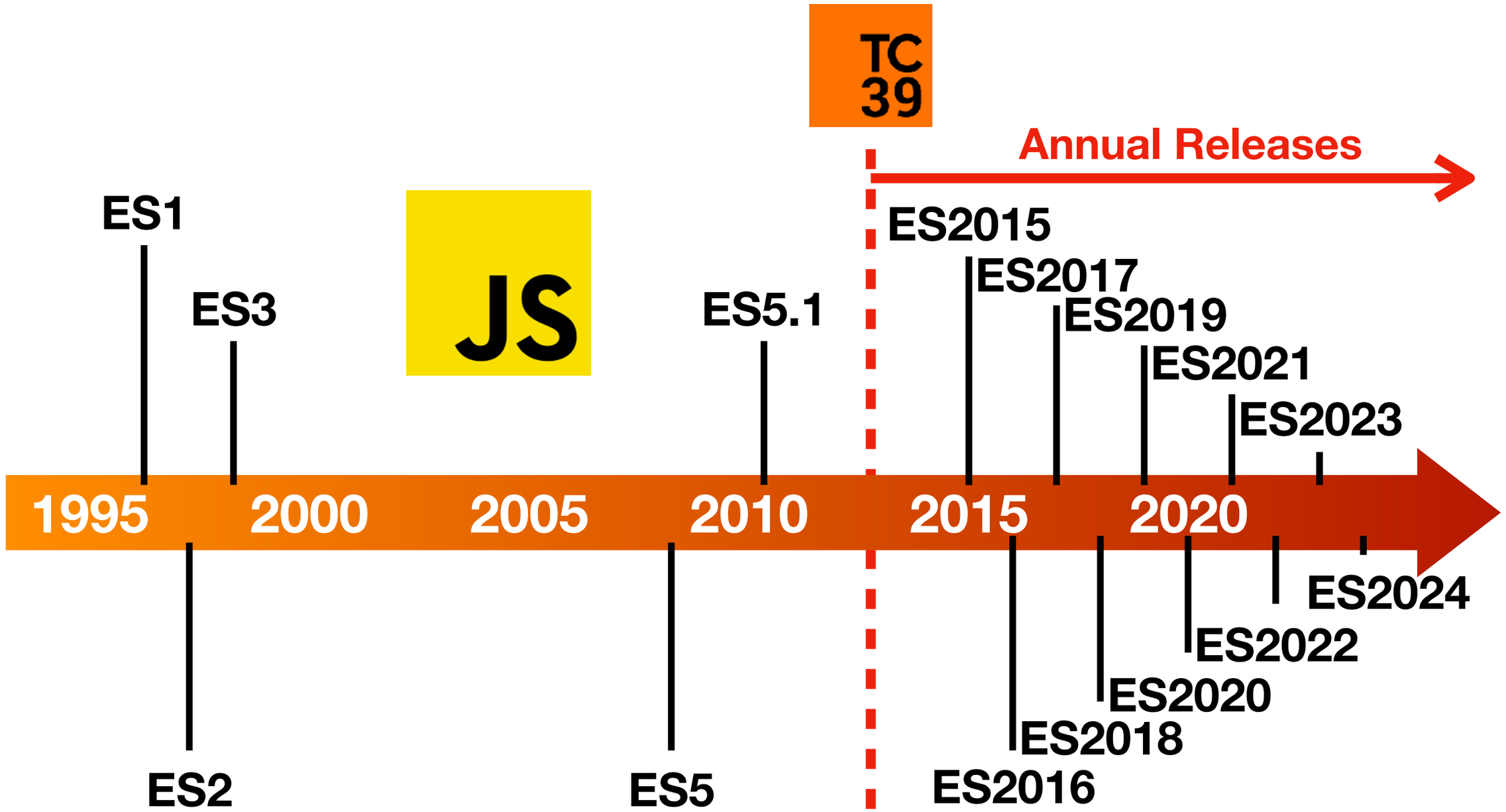
Update

JavaScript Implementations

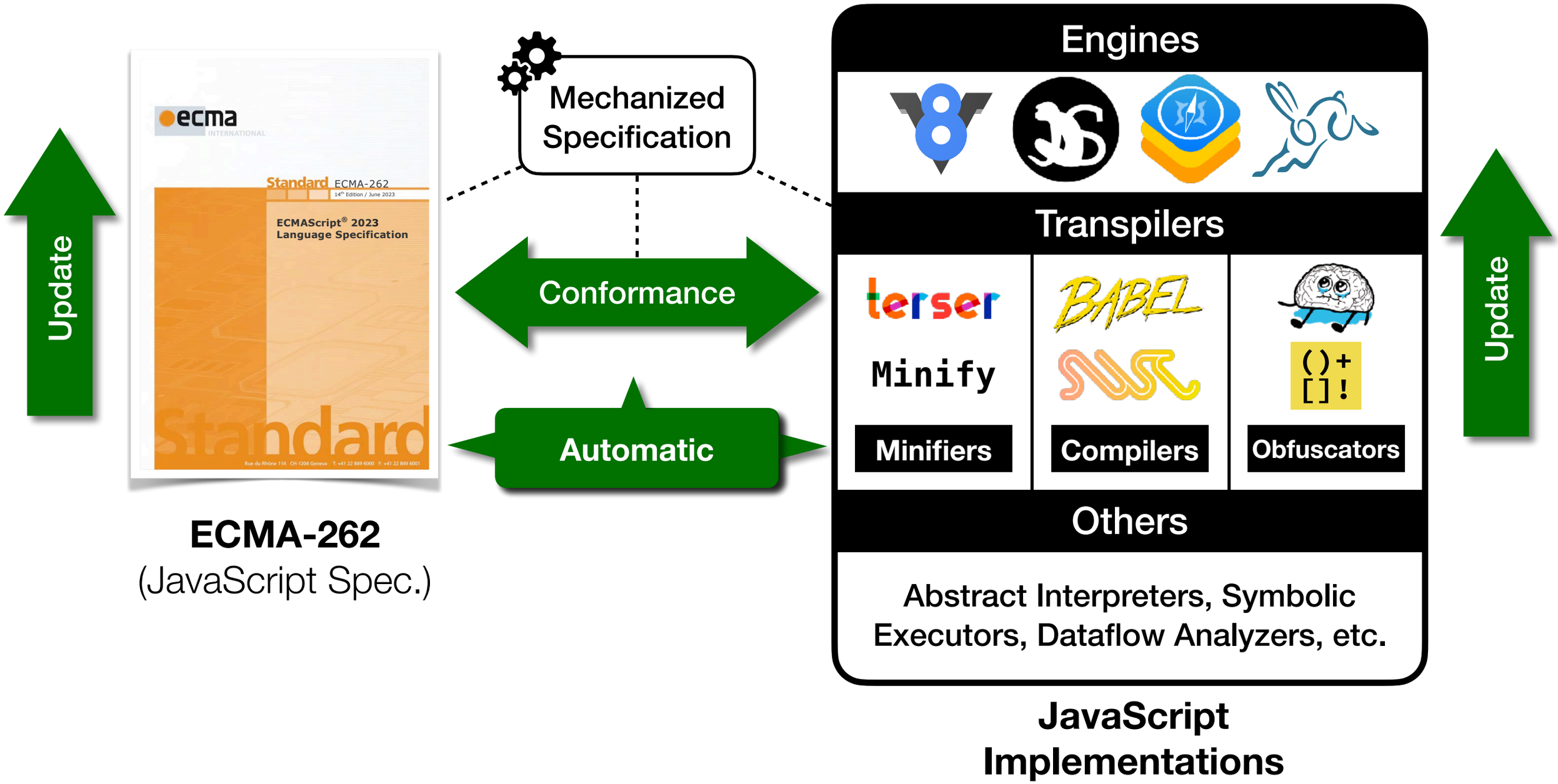
Problem - Fast Evolving JavaScript

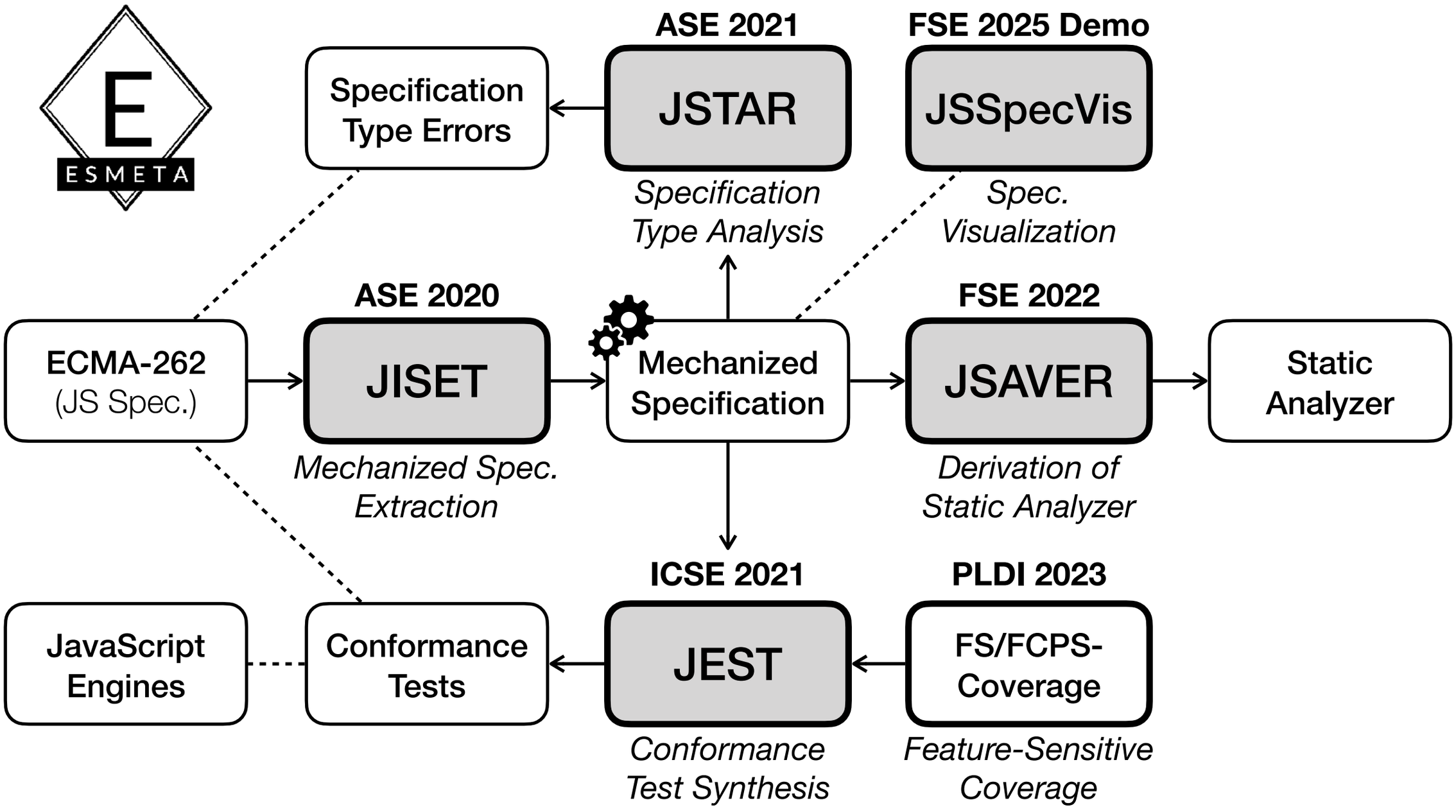


Problem - Fast Evolving JavaScript

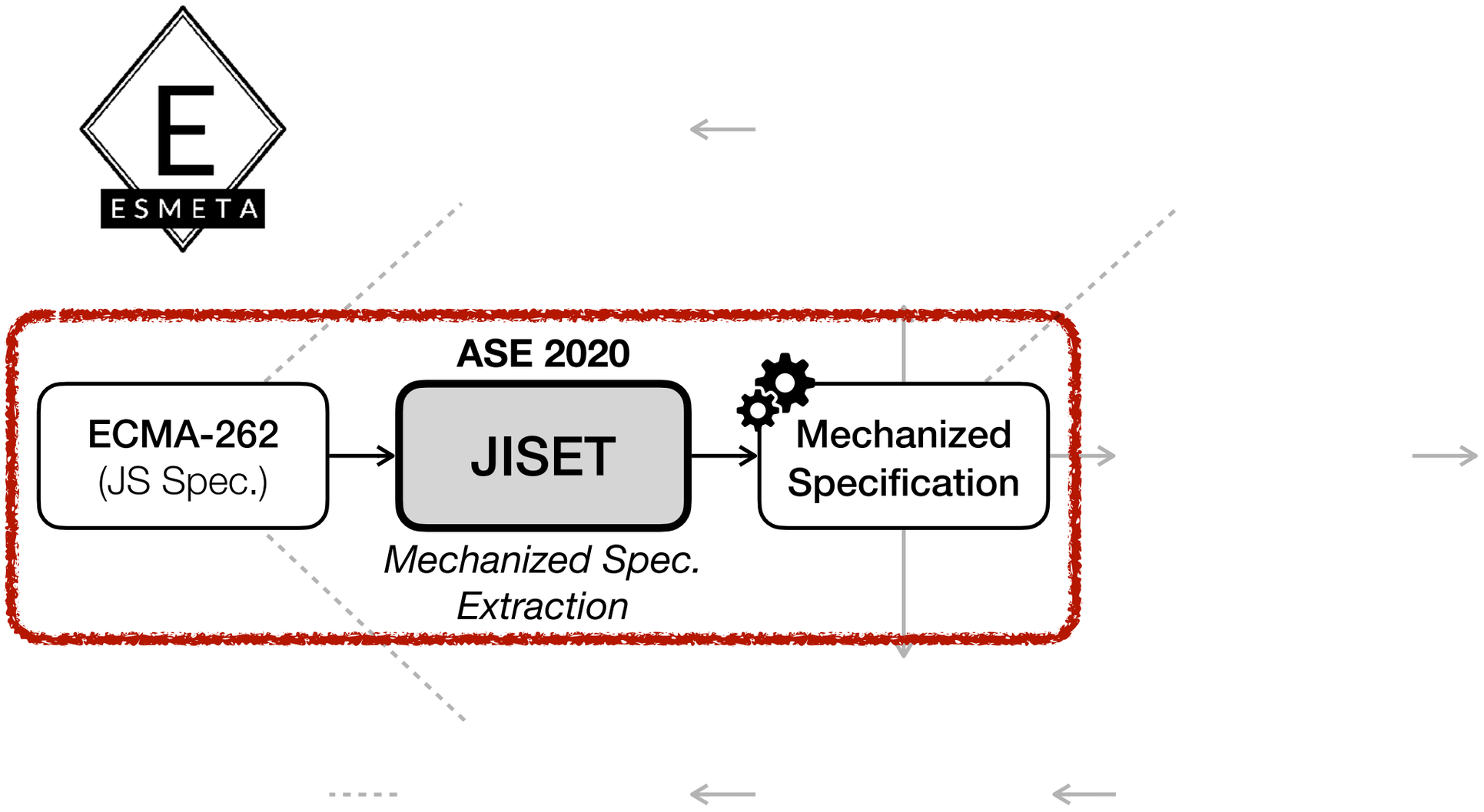


Solution - Mechanized Language Specification

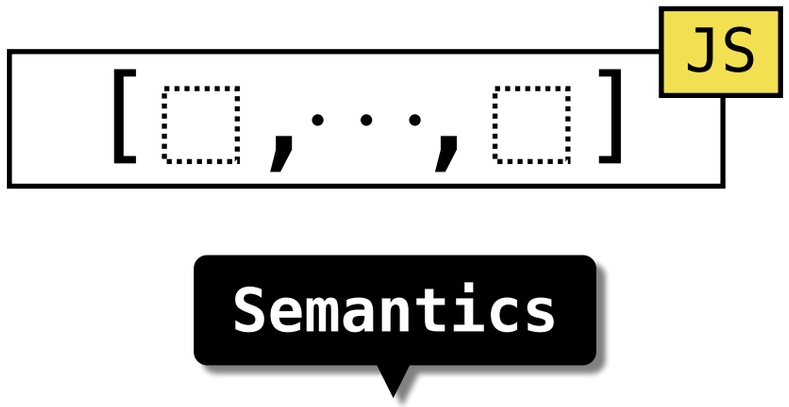



ESMETA
Specification Type Errors
ASE 2021
JSTAR
Specification Type Analysis
FSE 2025 Demo
JSSpecVis
Spec. Visualization
ECMA-262 (JS Spec.)
ASE 2020
JISET
Mechanized Spec. Extraction
Mechanized Specification
FSE 2022
JSAVER
Derivation of Static Analyzer
Static Analyzer
ICSE 2021
JEST
Conformance Test Synthesis
PLDI 2023
FS/FCPS-Coverage
Feature-Sensitive Coverage
JavaScript Engines
Conformance Tests

[ASE 2020] J. Park et al., “JISET: JavaScript IR-based Semantics Extraction Toolchain”



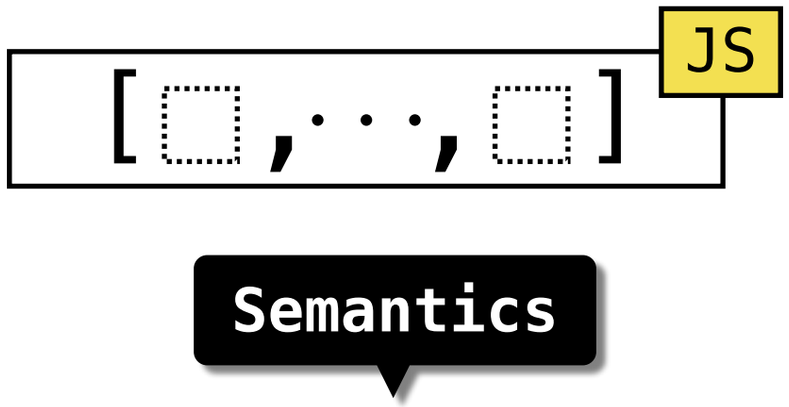
JISET - Patterns in Abstract Algorithms



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

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

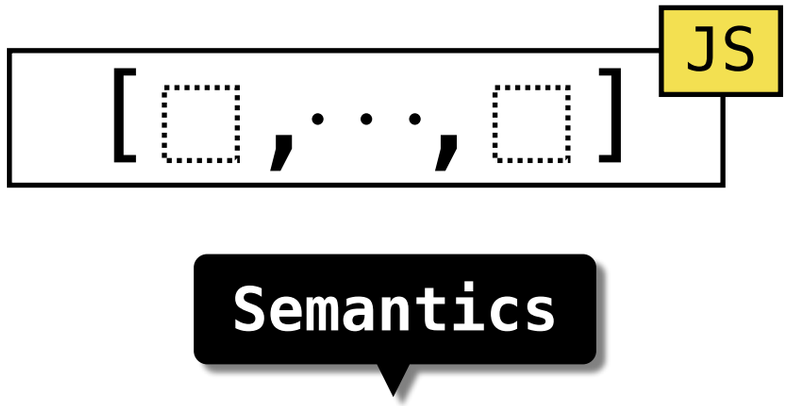
JISET - Patterns in Abstract Algorithms



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

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

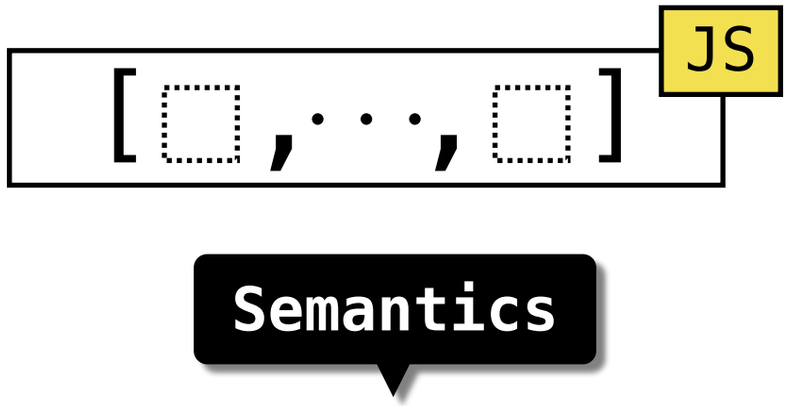
JISET - Patterns in Abstract Algorithms



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

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

JISET - Patterns in Abstract Algorithms



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

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

JISET - Metalanguage for ECMA-262

IR_{ES} - Intermediate Representation for ECMA-262

$$
\begin{array}{lll}
\text{Programs} & \mathfrak{P} \ni P ::= f^* & \\
\text{Functions} & \mathcal{F} \ni f ::= \texttt{syntax}^? \ \texttt{def x(x}^*\texttt{) \{}[\ell : i]^*\texttt{\}} & \\
\text{Variables} & \mathcal{X} \ni \texttt{x} & \\
\text{Labels} & \mathcal{L} \ni \ell & \\
\text{Instructions} & \mathcal{I} \ni i ::= r := e \mid \texttt{x} := \texttt{\{\}} \mid \texttt{x} := e(e^*) & \\
 & \quad \mid \texttt{if}\ e\ \ell\ \ell \mid \texttt{return}\ e & \\
\text{Expressions} & \mathcal{E} \ni e ::= v^{\text{p}} \mid \texttt{op}(e^*) \mid r & \\
\text{References} & \mathcal{R} \ni r ::= \texttt{x} \mid e\texttt{[}e\texttt{]} \mid e\texttt{[}e\texttt{]}_{\text{js}} & \\
 & \vdots & \\
\text{Values} & v \in \mathbb{V} & = \mathbb{A} \uplus \mathbb{V}^{\text{p}} \uplus \mathbb{T} \uplus \mathcal{F} \\
\text{Primitive Values} & v^{\text{p}} \in \mathbb{V}^{\text{p}} & = \mathbb{V}_{\text{bool}} \uplus \mathbb{V}_{\text{int}} \uplus \mathbb{V}_{\text{str}} \uplus \cdots \\
\text{JS ASTs} & t \in \mathbb{T} &
\end{array}
$$

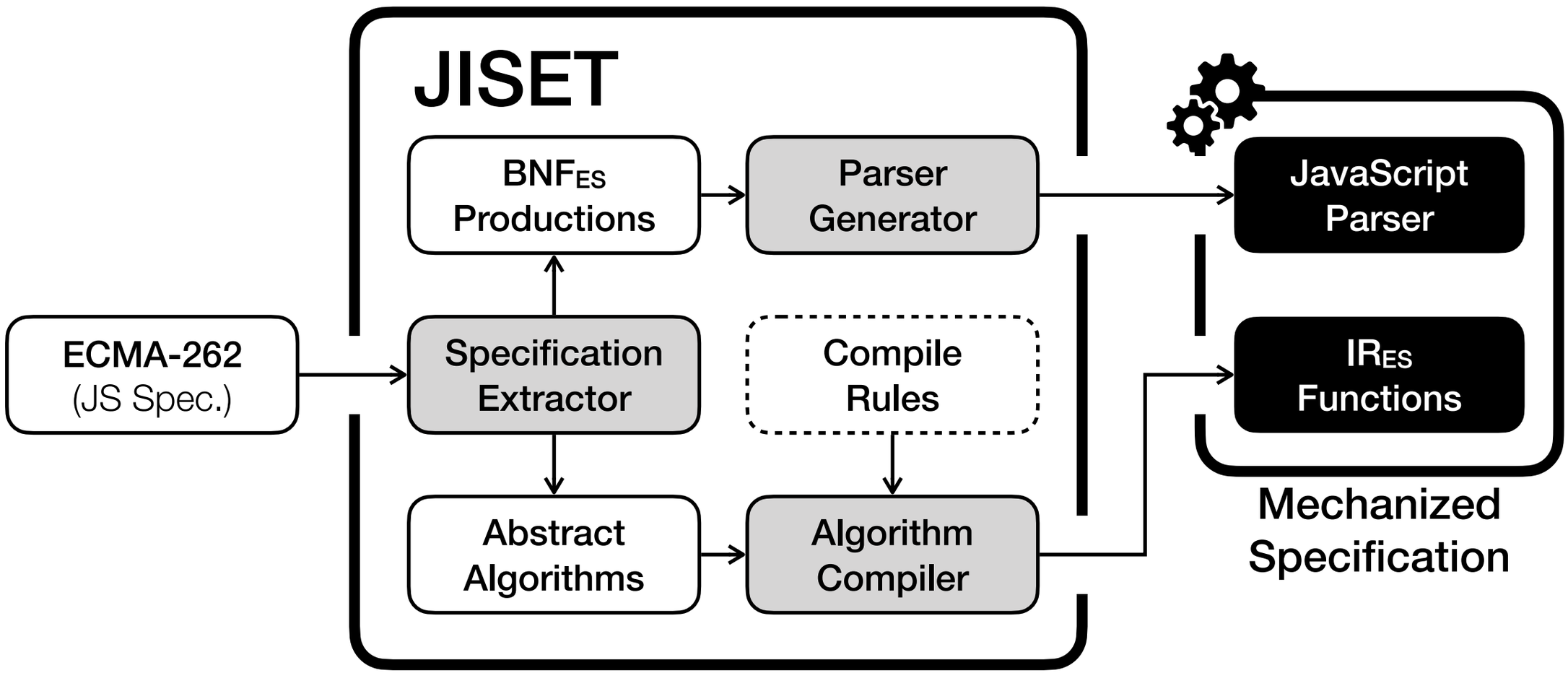
JISET - Metalanguage for ECMA-262

IR_{ES} - Intermediate Representation for ECMA-262

$$
\begin{array}{lll}
\text{Programs} & \mathfrak{P} \ni P ::= f^* & \\
\text{Functions} & \mathcal{F} \ni f ::= \texttt{syntax}^? \ \texttt{def x(x}^*\texttt{) \{}[\ell : i]^*\texttt{\}} & \\
\text{Variables} & \mathcal{X} \ni \texttt{x} & \\
\text{Labels} & \mathcal{L} \ni \ell & \\
\text{Instructions} & \mathcal{I} \ni i ::= r := e \mid \texttt{x} := \texttt{\{\}} \mid \texttt{x} := e(e^*) & \\
 & \quad \mid \texttt{if } e\ \ell\ \ell \mid \texttt{return } e & \\
\text{Expressions} & \mathcal{E} \ni e ::= v^{\text{p}} \mid \texttt{op}(e^*) \mid r & \\
\text{References} & \mathcal{R} \ni r ::= \texttt{x} \mid e[e] \mid e[e]_{\text{js}} & \\
 & \vdots & \\
\text{Values} & v \in \mathbb{V} & = \mathbb{A} \uplus \mathbb{V}^{\text{p}} \uplus \mathbb{T} \uplus \mathcal{F} \\
\text{Primitive Values} & v^{\text{p}} \in \mathbb{V}^{\text{p}} & = \mathbb{V}_{\text{bool}} \uplus \mathbb{V}_{\text{int}} \uplus \mathbb{V}_{\text{str}} \uplus \cdots \\
\text{JS ASTs} & t \in \mathbb{T} &
\end{array}
$$

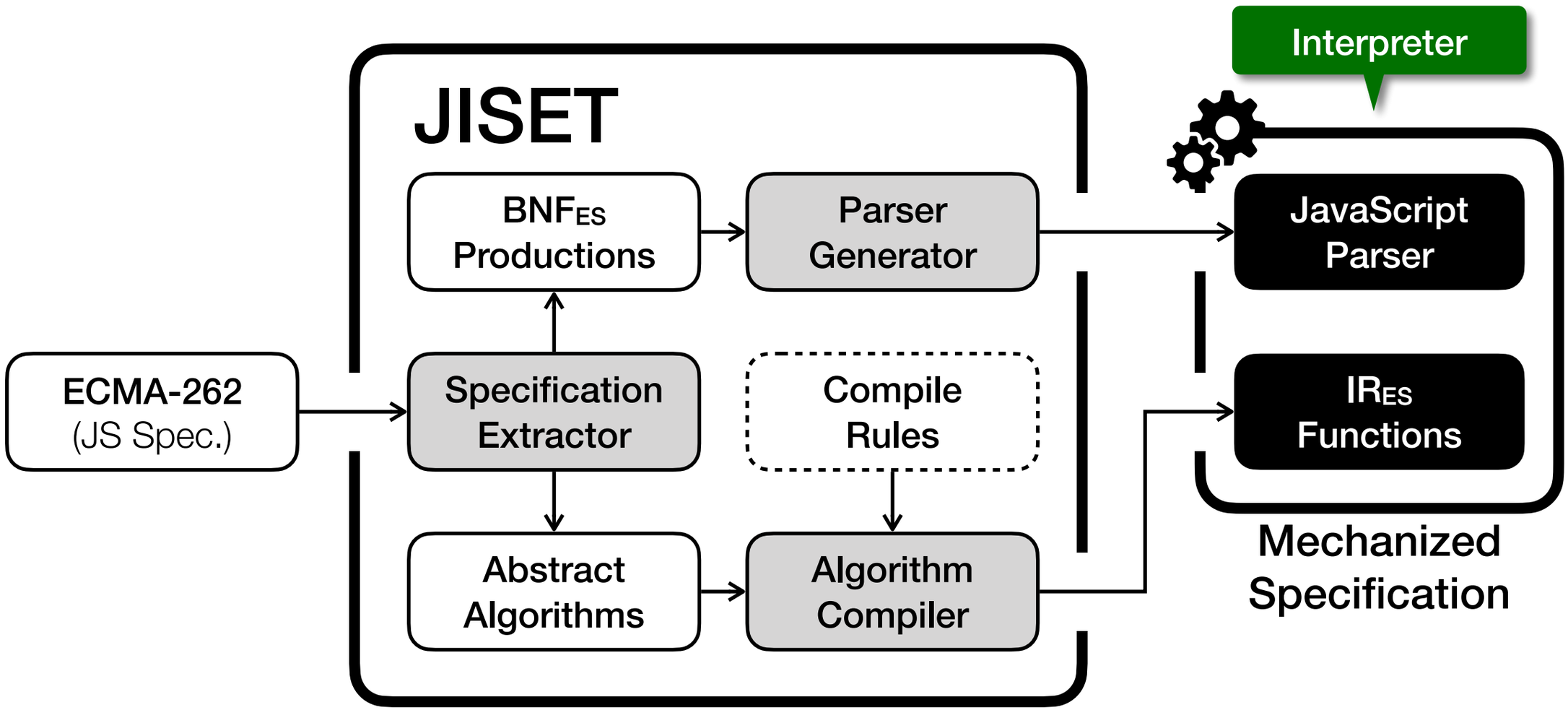
JISET

(**J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain)



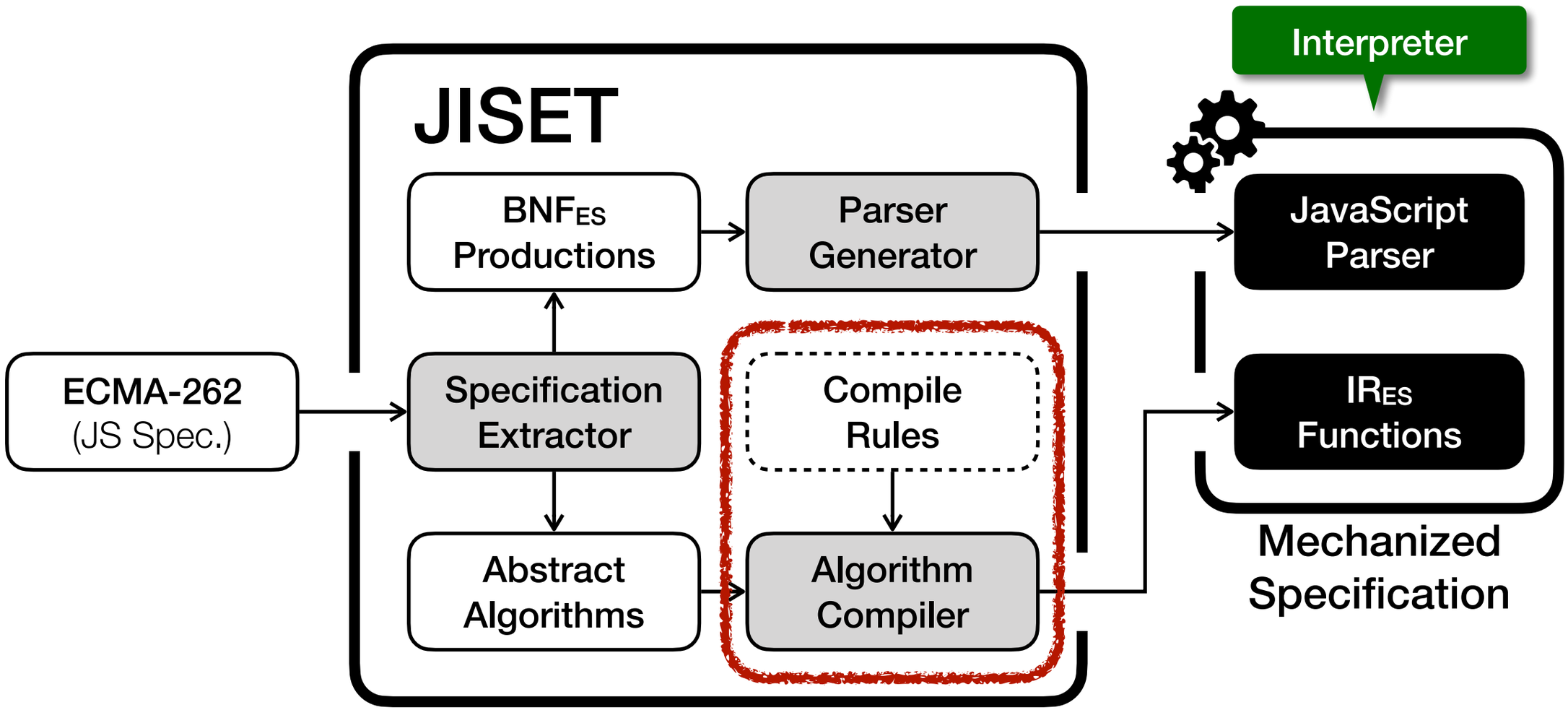
JISET

(**J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain)



JISET

(**J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain)

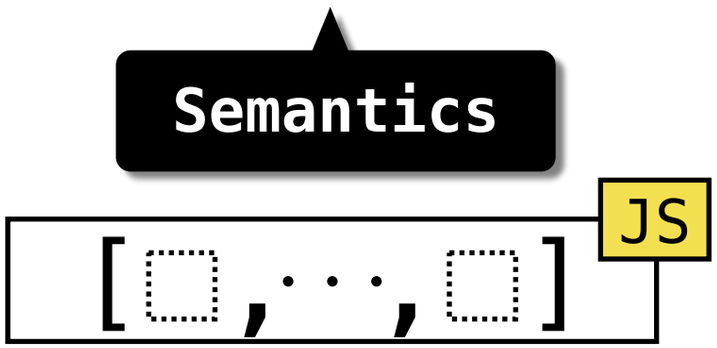


JISET - Algorithm Compilers

Abstract algorithm for *ArrayLiteral* **in ES13**

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

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.



JISET - Algorithm Compilers

Abstract algorithm for *ArrayLiteral* **in ES13**

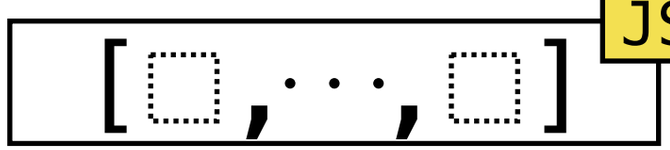
ArrayLiteral : `[` *ElementList* `,` *Elision*$_{opt}$ `]`

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

118 compile rules for steps in abstract algorithms

Semantics

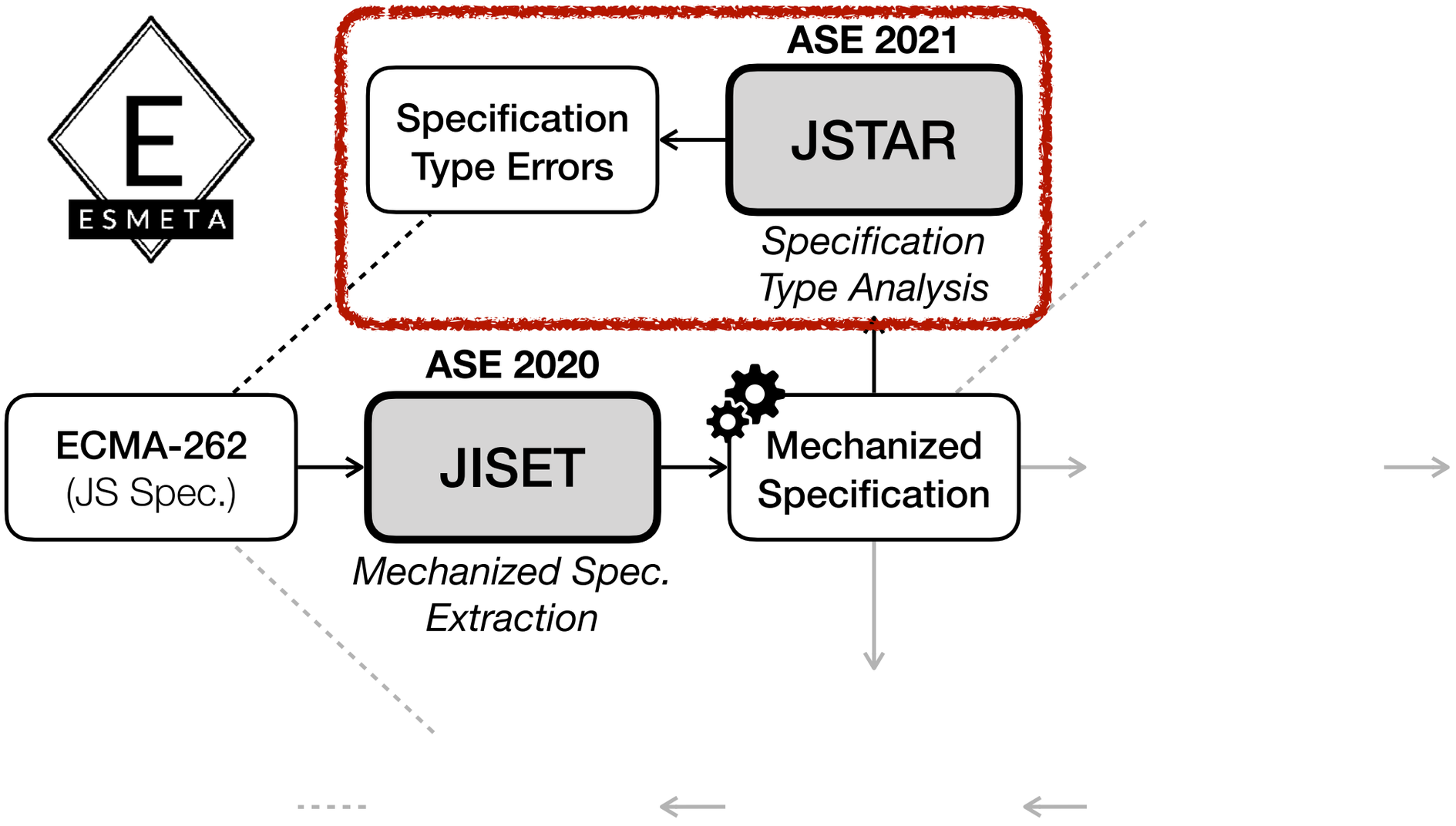
JS



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

IR$_{ES}$ function for *ArrayLiteral* **in ES13**

[ASE 2021] J. Park et al., “JSTAR: JavaScript Specification Type Analyzer using Refinement”



JSTAR - Specification Type Analysis

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 - Specification Type Analysis

20.3.2.28 Math.round (*x*)

x : String | Boolean | Number | Object | ...

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 - Specification Type Analysis

20.3.2.28 Math.round (x)

x : String | Boolean | Number | Object | ...

Number | Exception

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 - Specification Type Analysis

20.3.2.28 Math.round (*x*)

x : String | Boolean | Number | Object | ...

Filter Exception

Number | Exception

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 - Specification Type Analysis

20.3.2.28 Math.round (*x*)

x : String | Boolean | Number | Object | ...

n : Number

Filter Exception

Number | Exception

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 - Specification Type Analysis

20.3.2.28 Math.round (*x*)

x : String | Boolean | Number | Object | ...

n : Number

Filter Exception

Number | Exception

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.

Type Error:
`<`, `>`, and `>=`
are numeric operators

. . .

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

JSTAR - Specification Type Analysis

20.3.2.28 Math.round (x)

x : String | Boolean | Number | Object | ...

n : Number

Filter Exception

Number | Exception

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.

. . .

Type Error:
`<`, `>`, and `>=`
are numeric operators

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

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

JSTAR - Specification Type Analysis

20.3.2.28 Math.round (x)

x : String | Boolean | Number | Object | ...

n : Number

Filter Exception

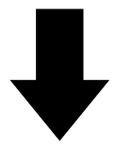
Number | Exception

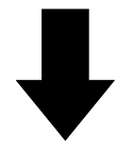
1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return +0.
4. If $x < 0$ and $x \geq -0.5$, return -0.

. . .

Type Error:
`<`, `>`, and `>=`
are numeric operators

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





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

Fixed

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

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

JSTAR - Evaluation

- Type analysis on **864 versions** of ECMA-262 in 3 years

59.2% Precision

93 Errors Detected

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 New Bugs In ES2021

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

2. ...
    b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then
        i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").
        ...
5. Else if *patternIsRegExp* is **true**, then
    a. Let *P* be ? Get(*pattern*, "**source**").

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

2. ...

    b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

        i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

        ...

5. Else if *patternIsRegExp* is **true**, then

    a. Let *P* be ? Get(*pattern*, "**source**").

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

patternIsRegExp: Boolean { True => *pattern*: Object }
pattern: ESValue

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

patternIsRegExp: Boolean { True => *pattern*: Object }
pattern: ESValue

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

patternIsRegExp: True { True => *pattern*: Object }
pattern: ESValue → *pattern*: Object

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

patternIsRegExp: Boolean { True => *pattern*: Object }
pattern: ESValue

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

patternIsRegExp: True { True => *pattern*: Object }
pattern: ESValue → *pattern*: Object

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

patternIsRegExp: True { True => *pattern*: Object }
pattern: ESValue → *pattern*: Object

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

JSTAR - Occurrence Typing (On Going)

22.2.4.1 RegExp (*pattern*, *flags*)

pattern: ESValue

1. Let *patternIsRegExp* be ? IsRegExp(*pattern*).

Normal[Boolean]
{ Normal[True] => *pattern*: Object}

RETURN

patternIsRegExp: Boolean { True => *pattern*: Object }
pattern: ESValue

2. ...

b. If *patternIsRegExp* is **true** and *flags* is **undefined**, then

patternIsRegExp: True { True => *pattern*: Object }
pattern: ESValue → *pattern*: Object

i. Let *patternConstructor* be ? Get(*pattern*, "**constructor**").

...

5. Else if *patternIsRegExp* is **true**, then

requires **Object**

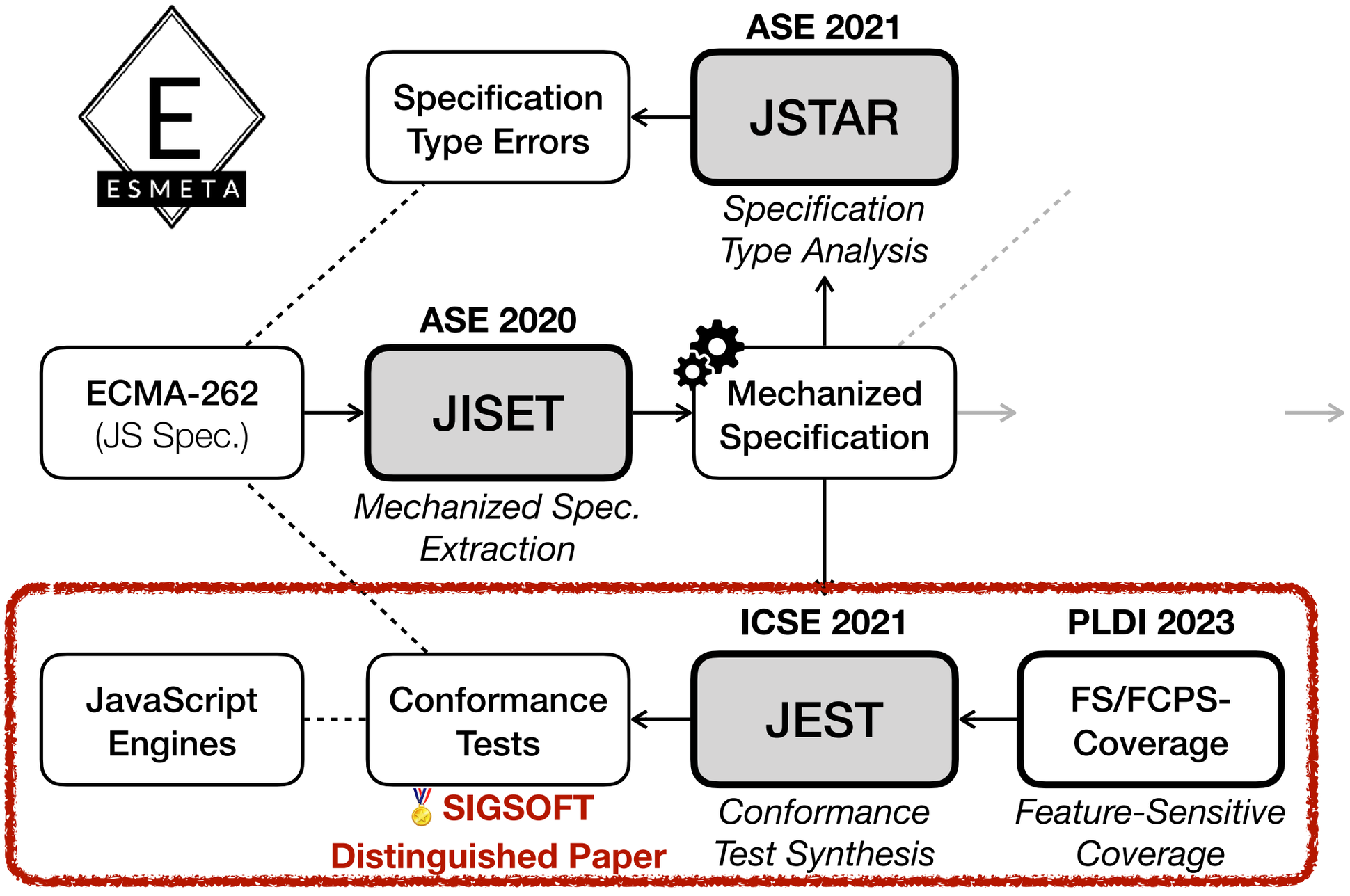
patternIsRegExp: True { True => *pattern*: Object }
pattern: ESValue → *pattern*: Object

a. Let *P* be ? Get(*pattern*, "**source**").

requires **Object**

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

[PLDI 2023] J. Park et al., “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”



Conformance of JavaScript Engines

ECMA-262
(JavaScript Spec.)

Conformance

GraalVM

QuickJS

moddable

JavaScript Engines

Conformance of JavaScript Engines

ECMA-262
(JavaScript Spec.)

How?

Conformance

GraalVM

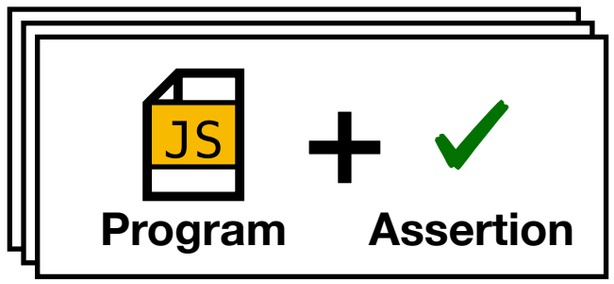
QuickJS

moddable

JavaScript Engines

Conformance of JavaScript Engines

ECMA-262
(JavaScript Spec.)



Conformance Tests

Conformance

Test262
(Official Test Suite)

GraalVM

QuickJS

moddable

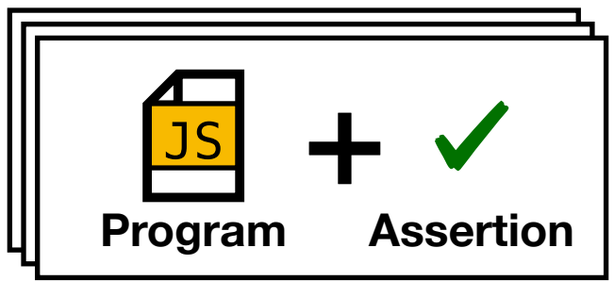
JavaScript Engines

Conformance of JavaScript Engines

ECMA-262
(JavaScript Spec.)

Example: `4 + 2n` JS

Assertion: `TypeError`



Conformance Tests

Conformance

Test262
(Official Test Suite)

GraalVM

QuickJS

moddable

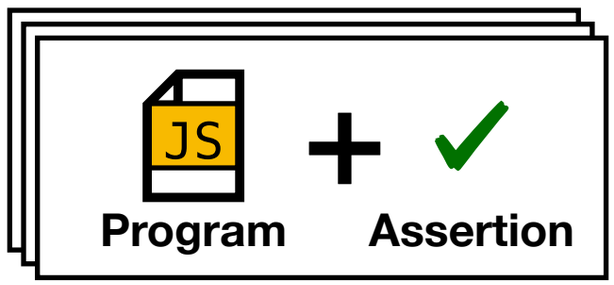
JavaScript Engines

Problem - Manual Approach

ECMA-262
(JavaScript Spec.)

Example: `4 + 2n` JS

Assertion: `TypeError`



Conformance Tests

Conformance

Test262
(Official Test Suite)

`Manual`

GraalVM

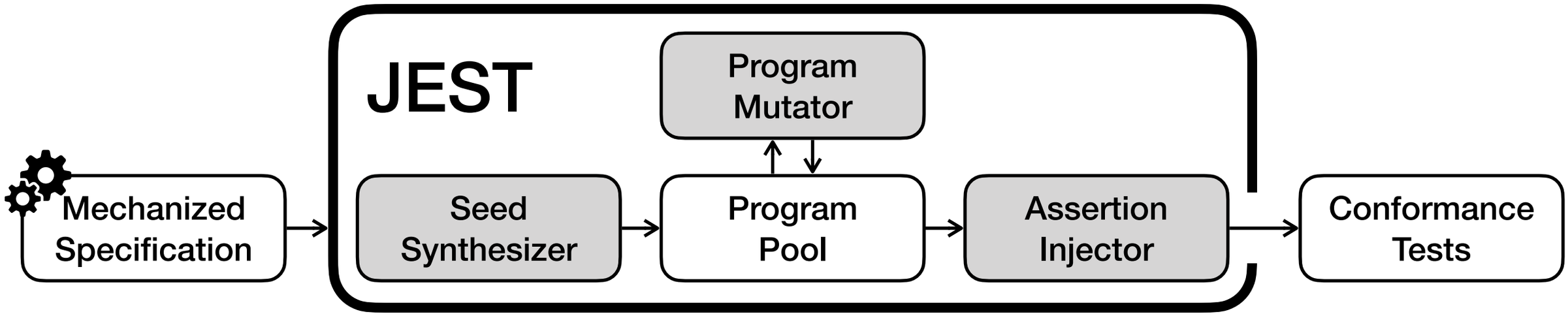
`QuickJS`

moddable

JavaScript Engines

JEST

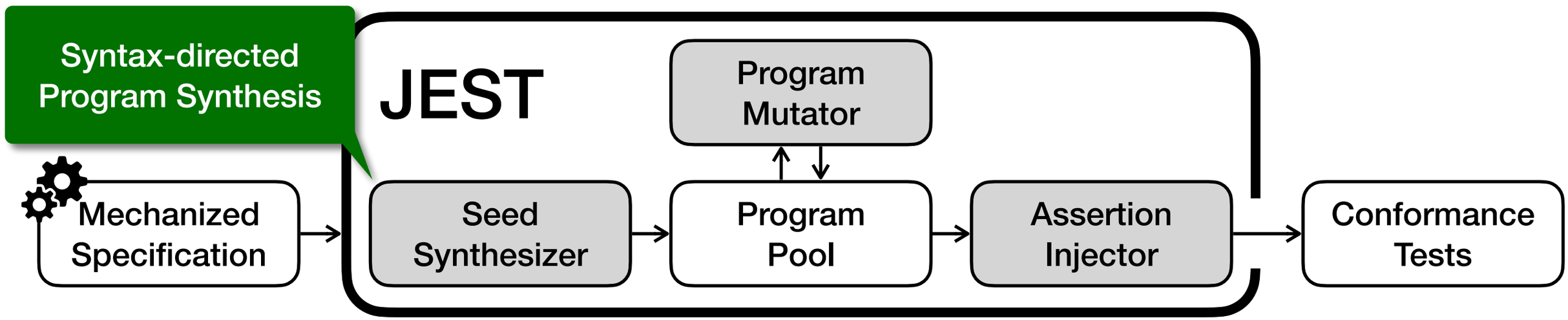
(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)

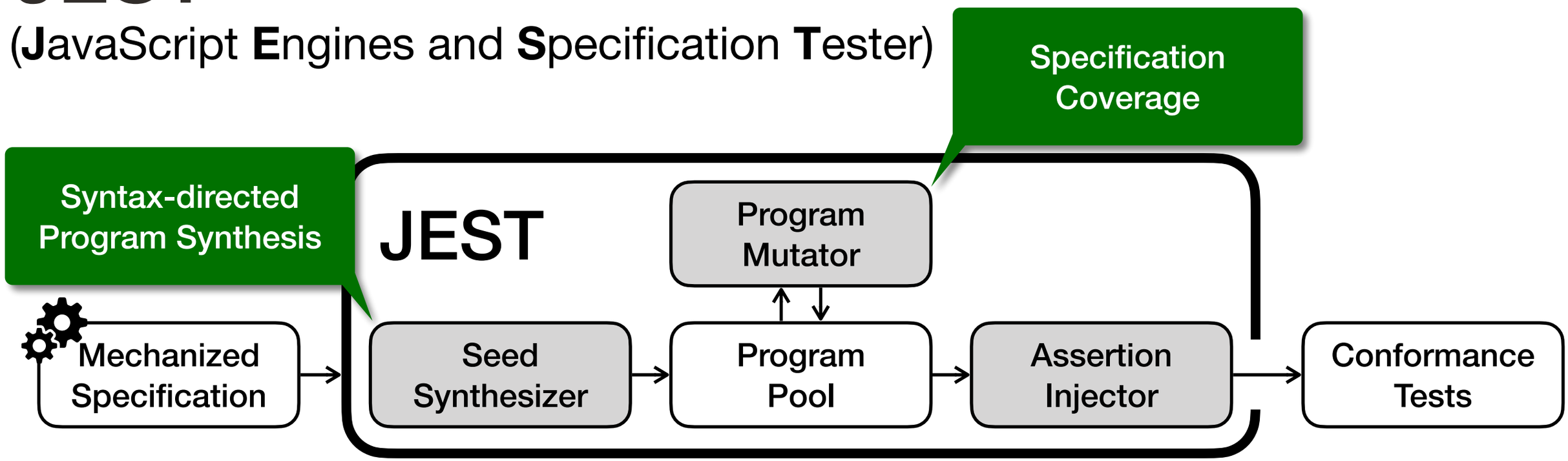


Program Pool

```
let x = 42;
```

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



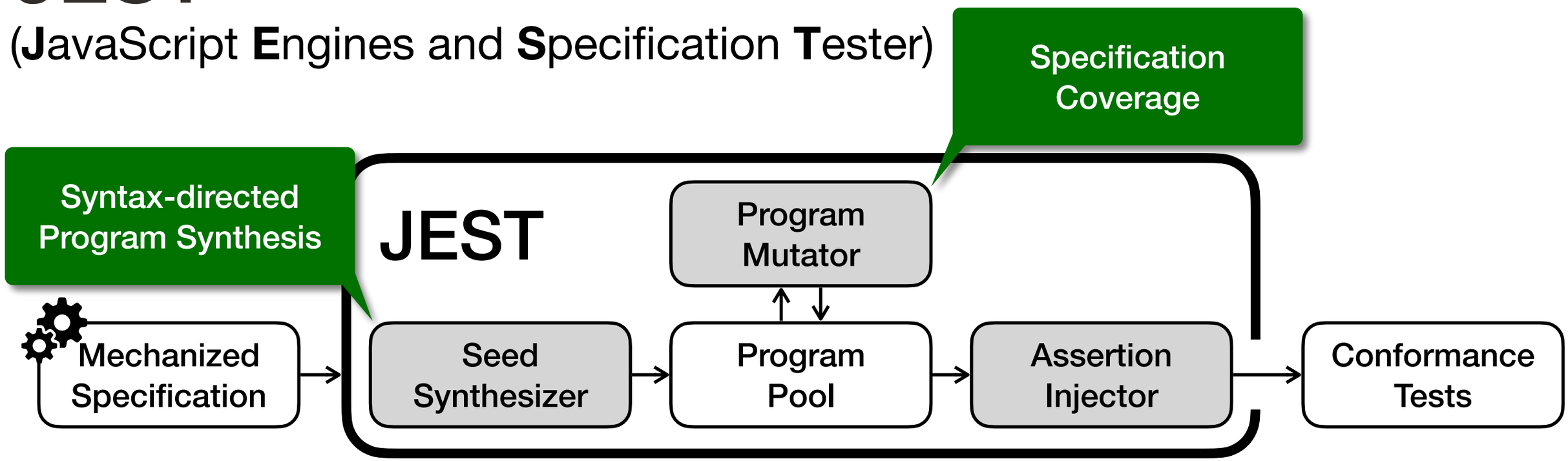
Program Pool

```
let x = 1 + 2;
```

```
let x = 42;
```

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

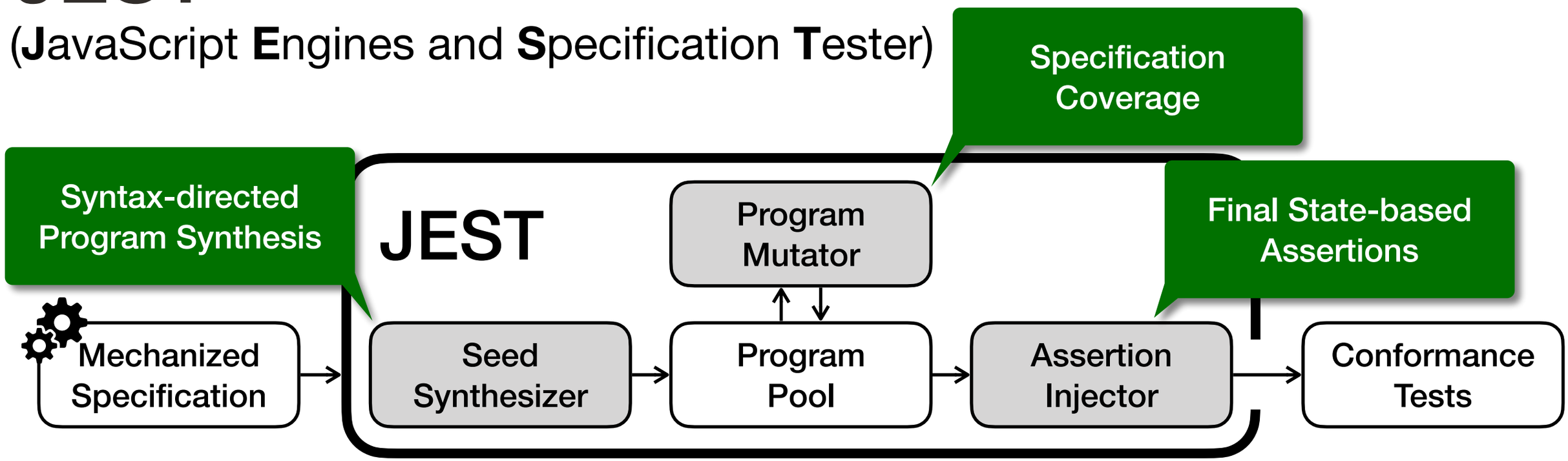
```
let x = 1 + 2;
```

```
let x = 42;
```

```
let x = ![];
```

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

```
let x = 1 + 2;
assert(x == 3);
```

```
let x = 42;
assert(x == 42);
```

```
let x = ![];
assert(x == false);
```

JEST - Specification Coverage

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

...

3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   ...
7. Else,
   ...

JEST - Specification Coverage

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

...

3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   ...
7. Else,
   ...

```
4 * 2n
```
JS

JEST - Specification Coverage

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

...

3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   ...
7. Else,
   ...

JS `1n - 2n`

JS `4 * 2n`

JEST - Specification Coverage

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

...

3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   - ...
7. Else,
   - ...

JS `3 + 2`

JS `1n - 2n`

JS `4 * 2n`

JEST - Feature-Sensitive Coverage

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

…

3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   …
7. Else,
   …

Language Features

	JS	JS	JS
Addition	`3 + 2`	`1n + 2n`	`17 + 2n`
Subtraction	`4 - 7`	`7n - 2n`	`1n - 23`
Multiplication	`2 * 9`	`9n * 3n`	`3n * 45`

⋮

[PLDI 2023] J. Park et al., “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”

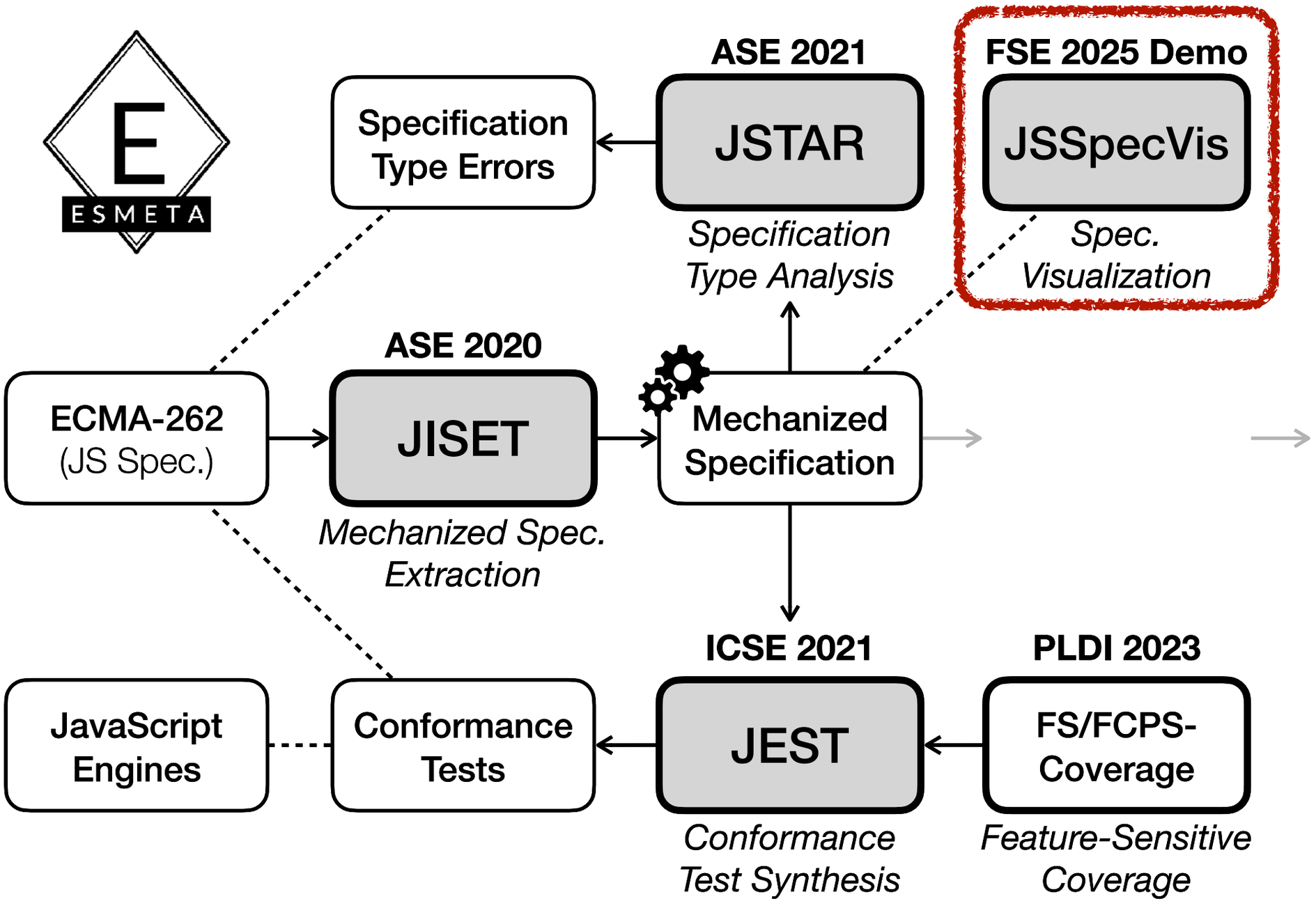
JEST - Evaluation

42 Bugs
In Engines

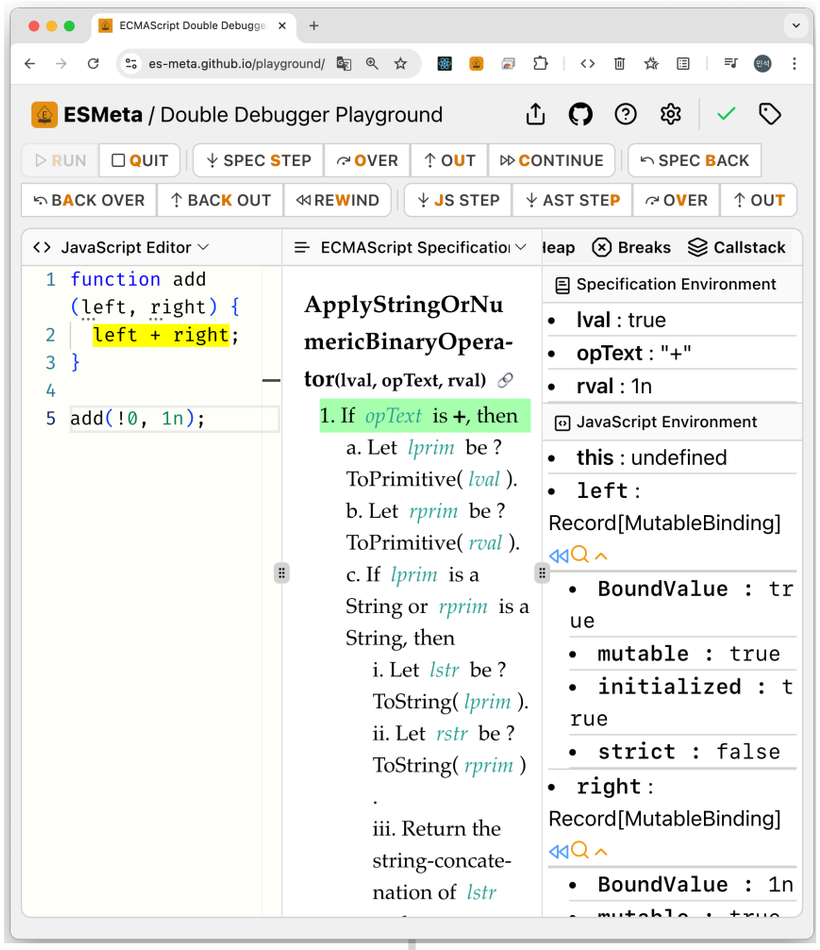
94 Bugs
In Transpilers

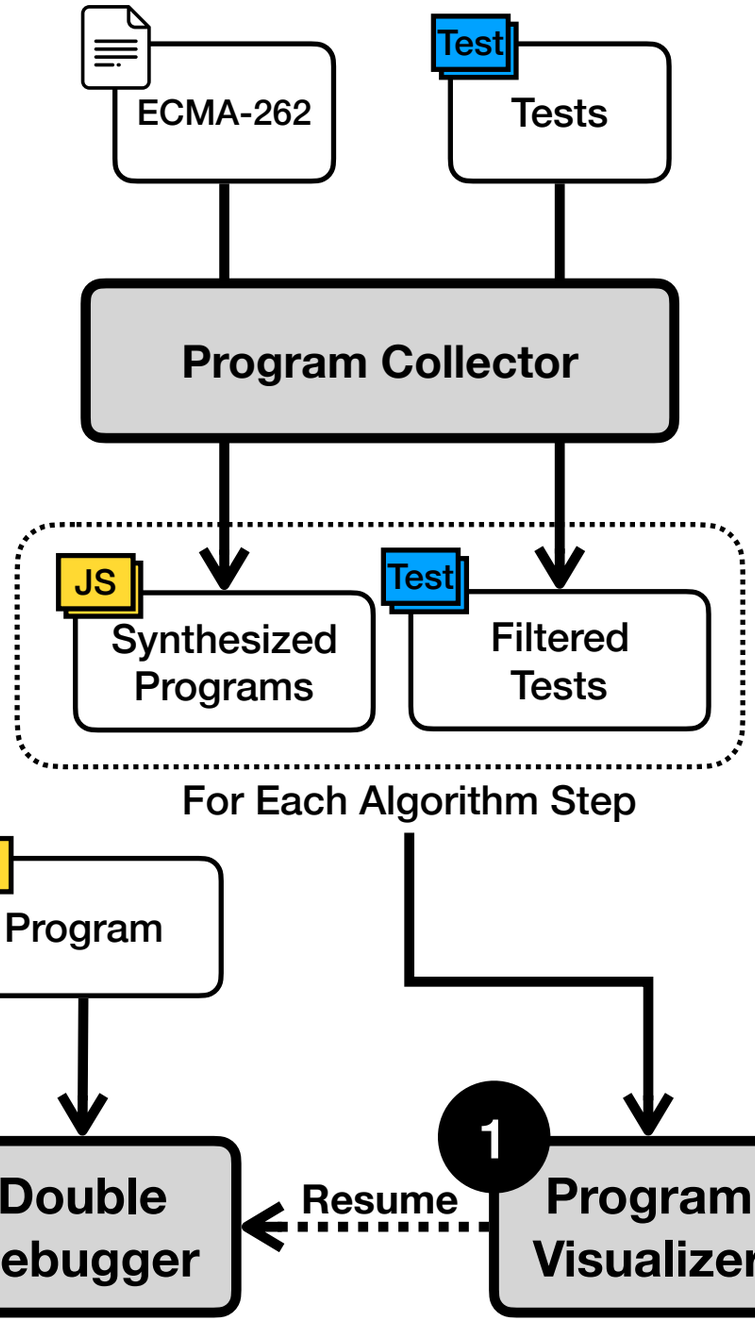
Kind	Name	Version	Release	# Detected Unique Bugs		
				# New	# Confirmed	# Reported
Engine	V8	v10.8.121	2022.10.06	3	3	4
	JSC	v615.1.10	2022.10.26	26	26	26
	GraalJS	v22.2.0	2022.07.26	11	11	11
	SpiderMonkey	v107.0b4	2022.10.24	2	4	4
	Total			**42**	**44**	**45**
Transpiler	Babel	v7.19.1	2022.09.15	37	37	39
	SWC	v1.3.10	2022.10.21	37	37	47
	Terser	v5.15.1	2022.10.05	19	19	19
	Obfuscator	v4.0.0	2022.02.15	1	2	7
	Total			**94**	**95**	**112**
Total				**136**	**139**	**157**

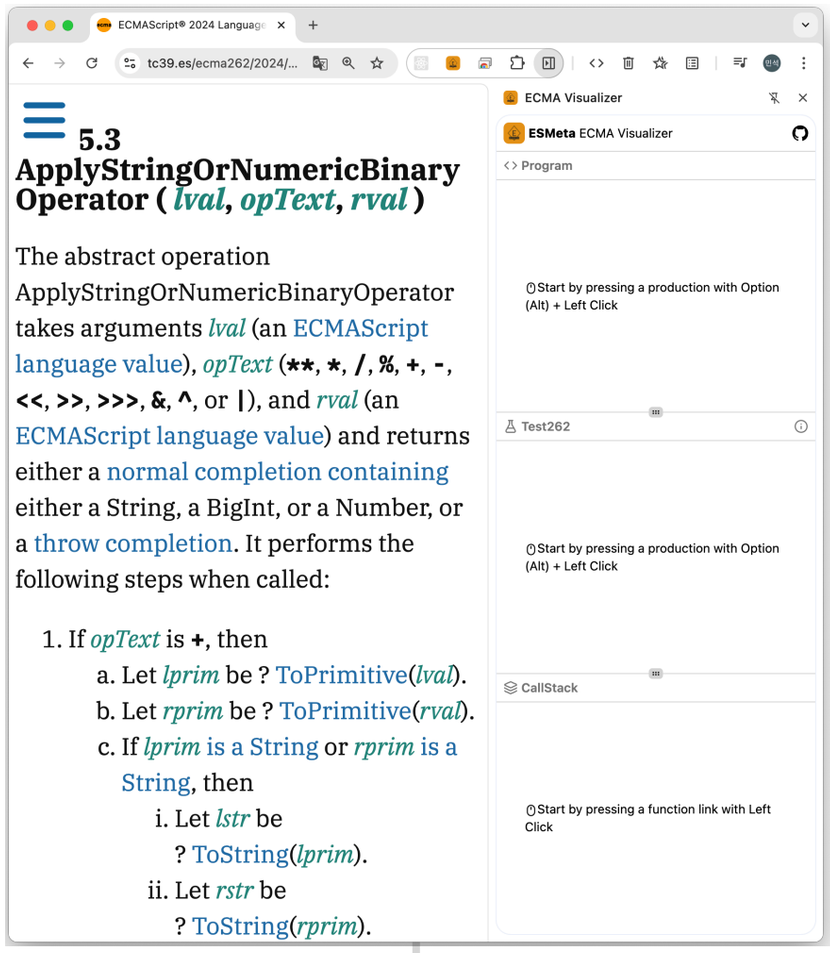
[FSE 2025 Demo] M. Choe et al., “JSSpecVis: A JavaScript Language Specification Visualization Tool”



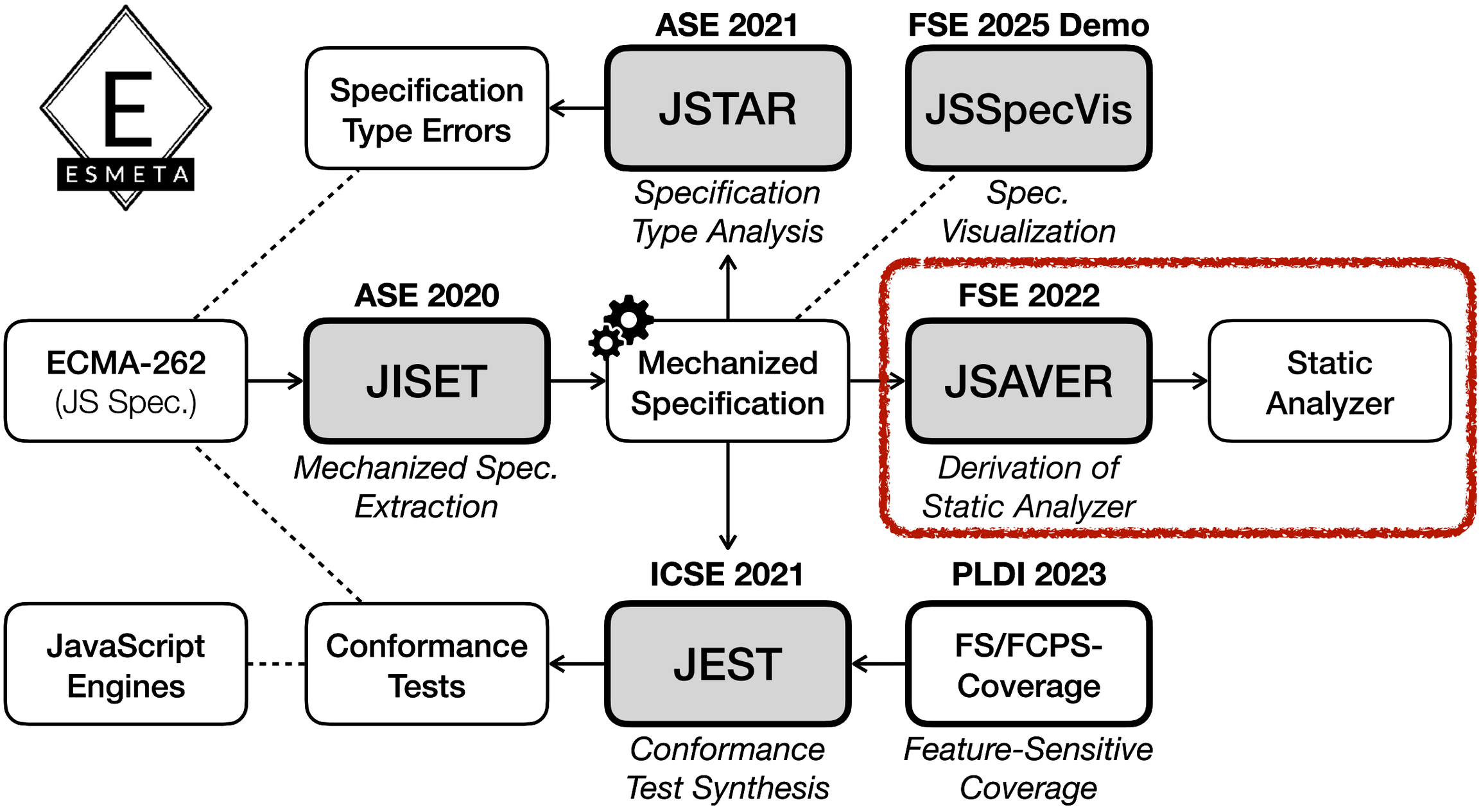
JSSpecVis - Specification Visualizer







[FSE 2022] J. Park et al., “Automatically Deriving JavaScript Static Analyzers from Specifications using Meta-level Static Analysis”



Meta-Level Static Analysis

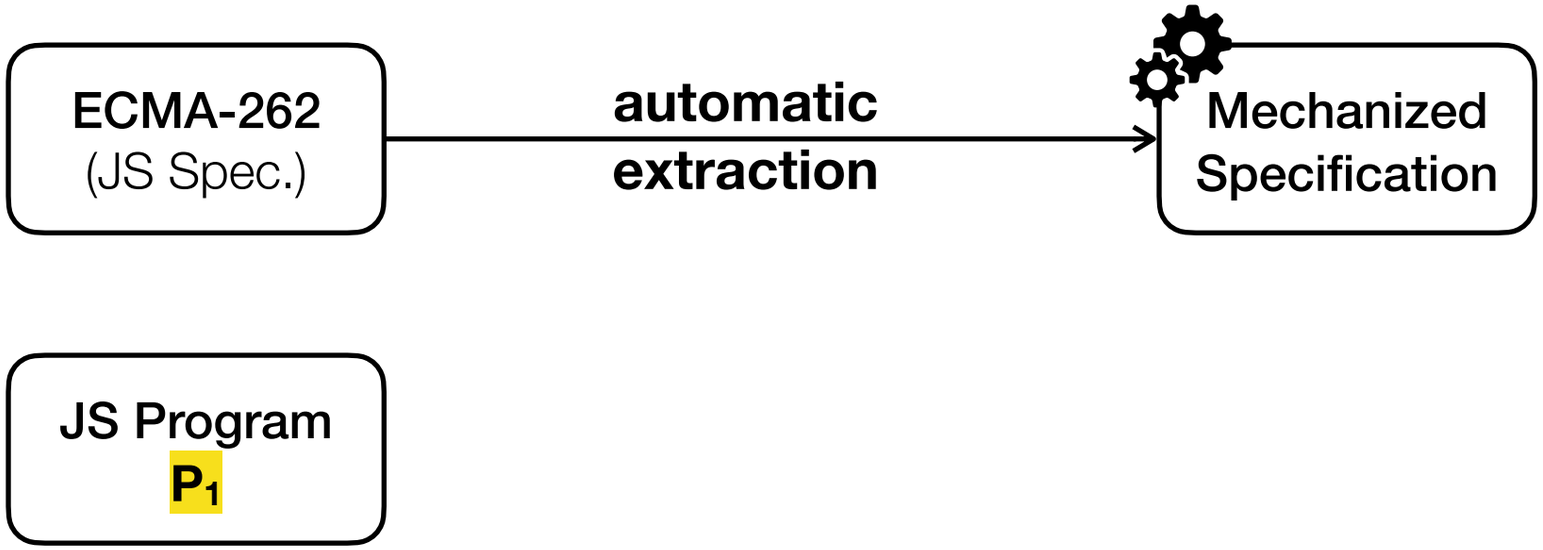
How to perform **static analysis** on **JavaScript** **programs** using **language specification**?

ECMA-262
(JS Spec.)

JS Program
P_1

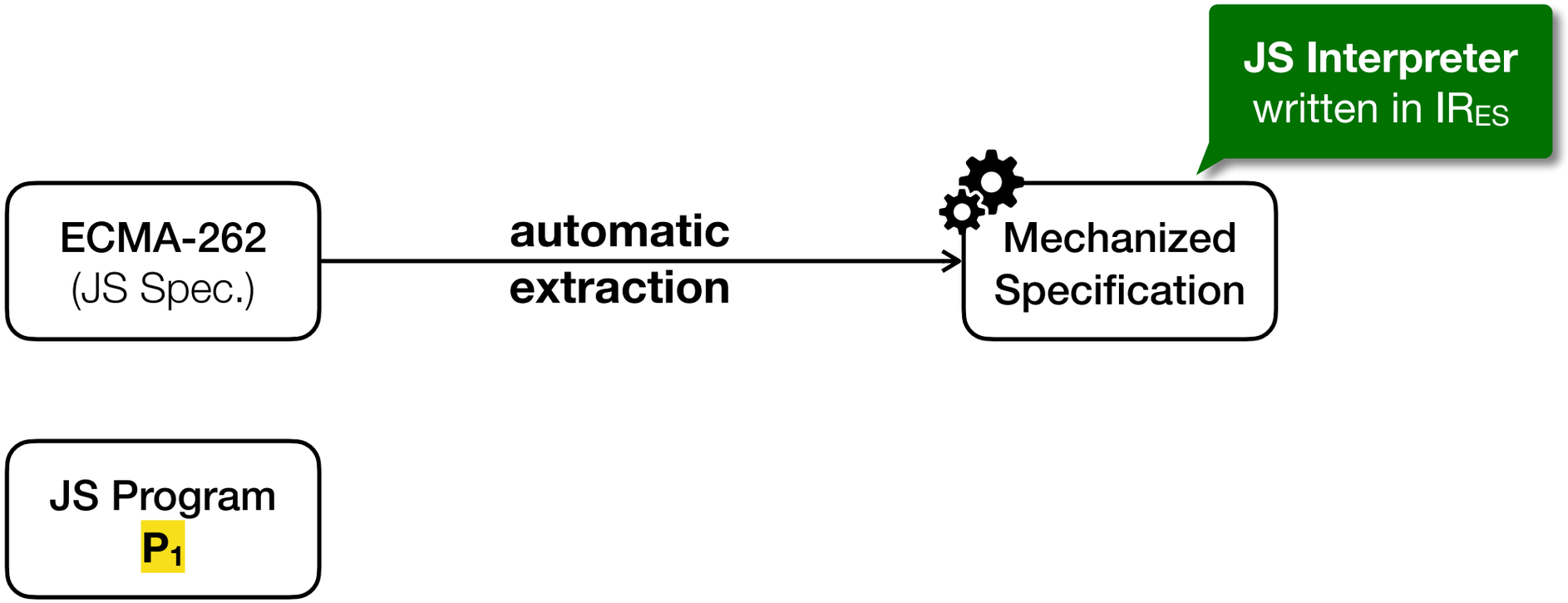
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



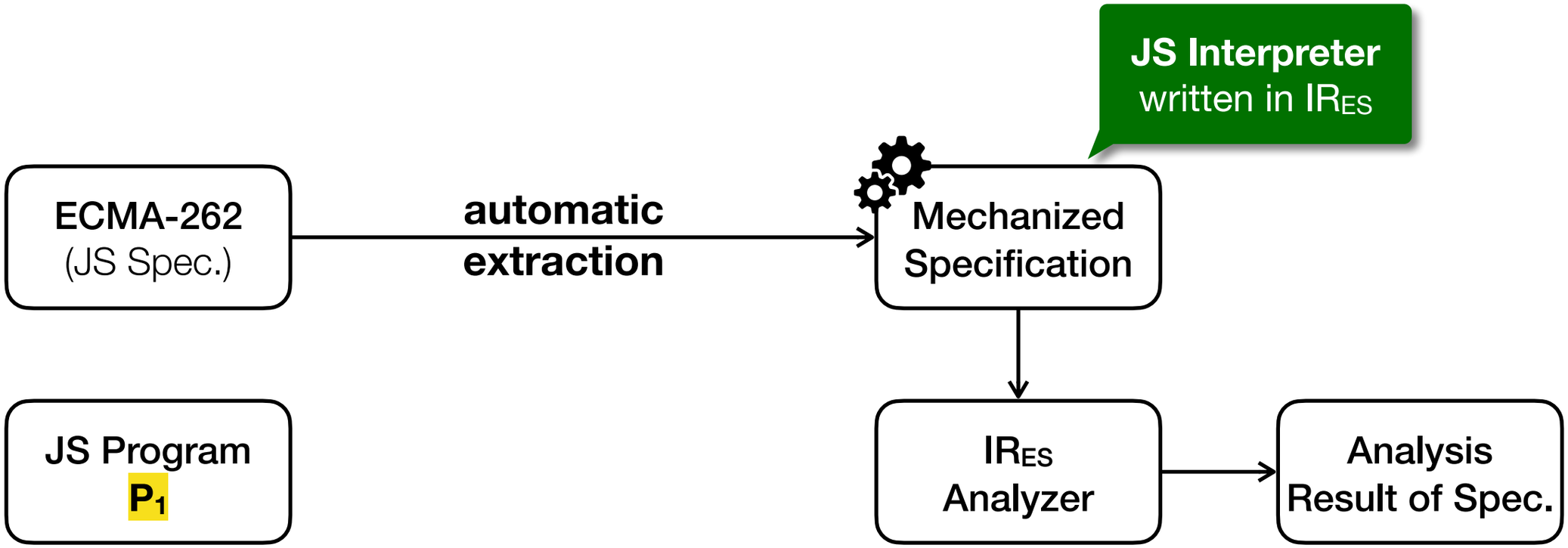
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



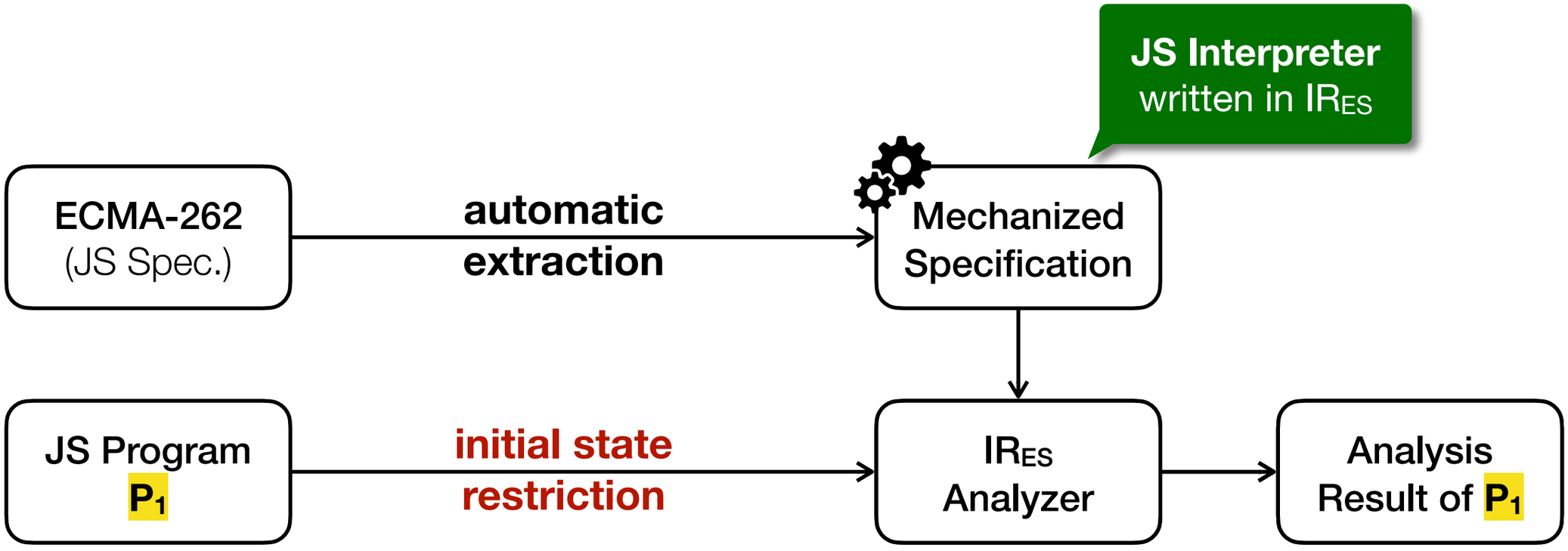
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



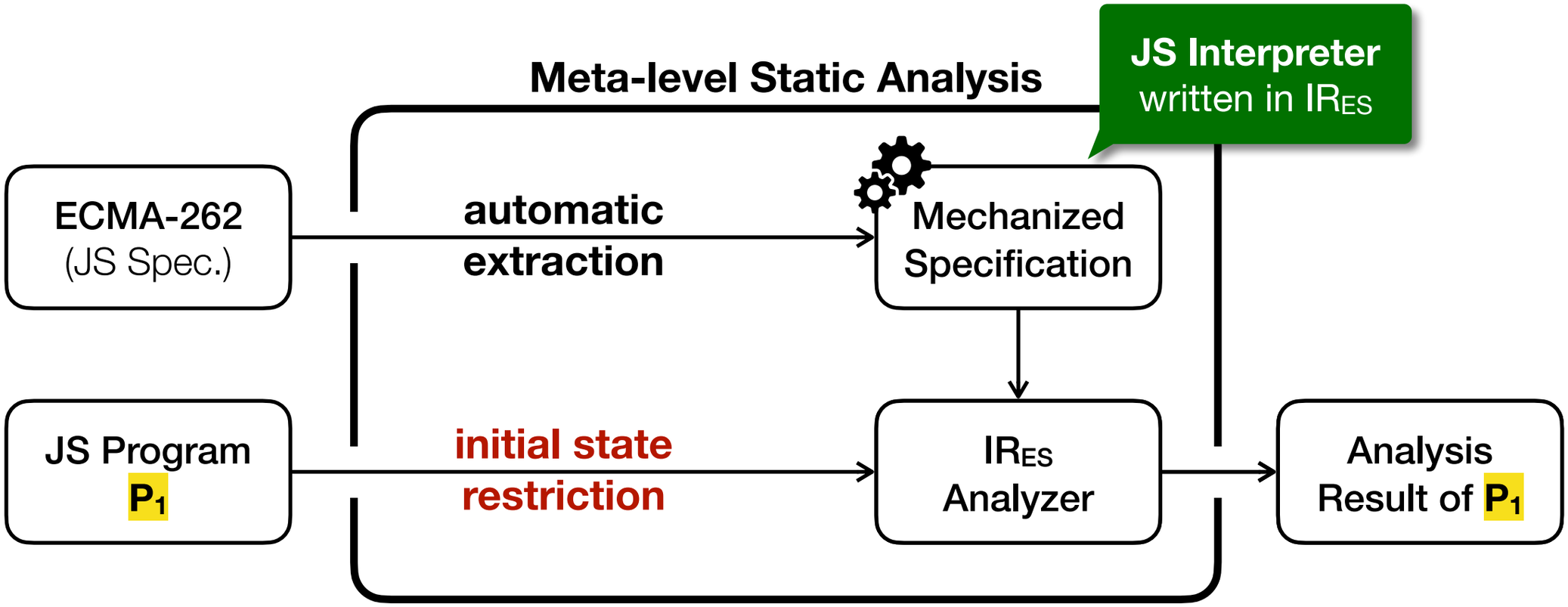
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



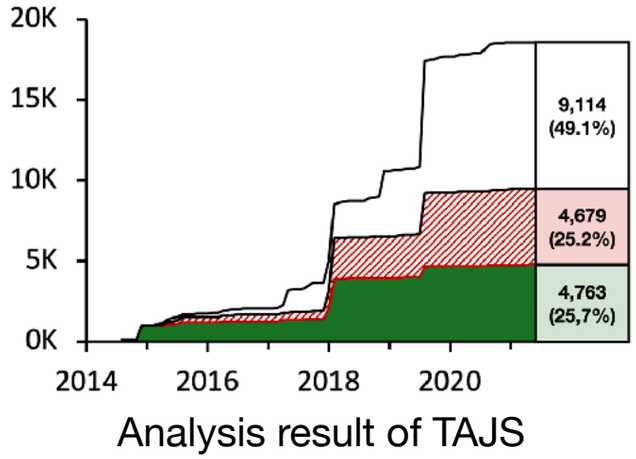
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?

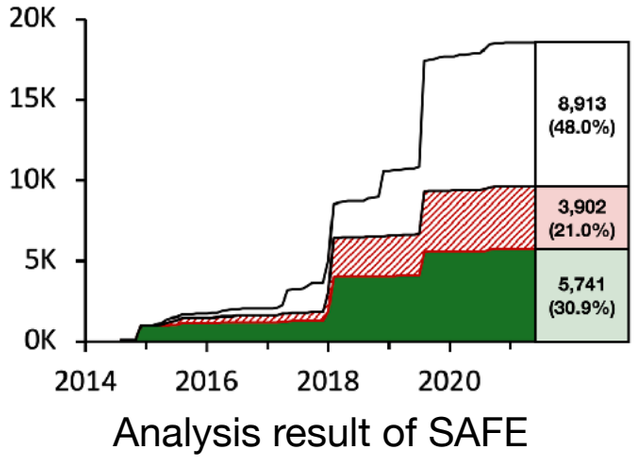


JSAVER - Evaluation

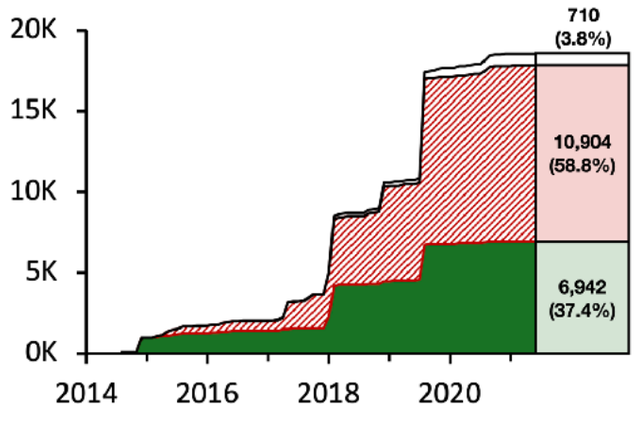
Manual



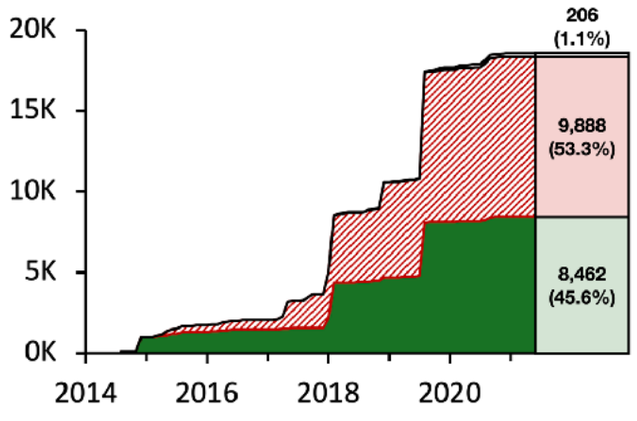
Analysis result of TAJS



Analysis result of SAFE

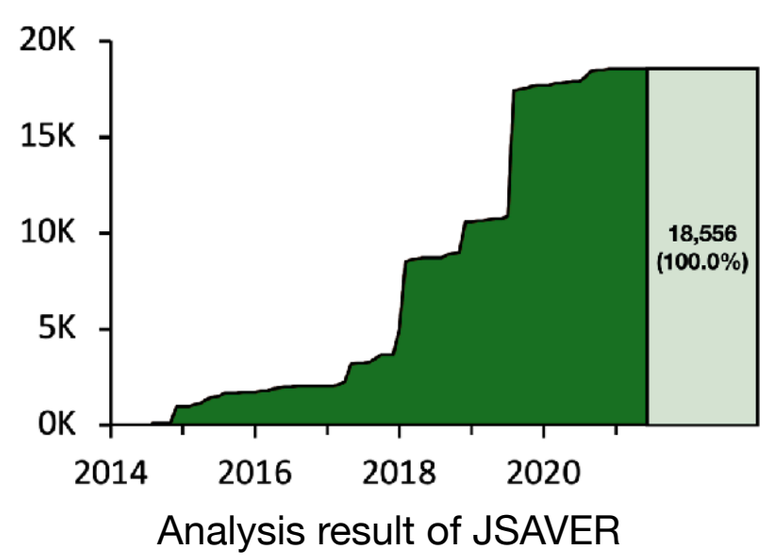


Analysis result of TAJS with Babel

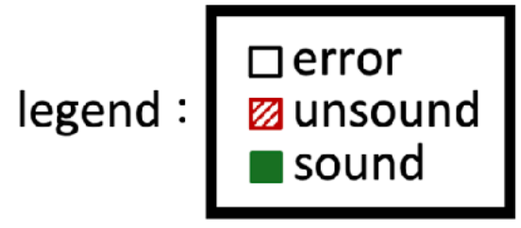


Analysis result of SAFE with Babel

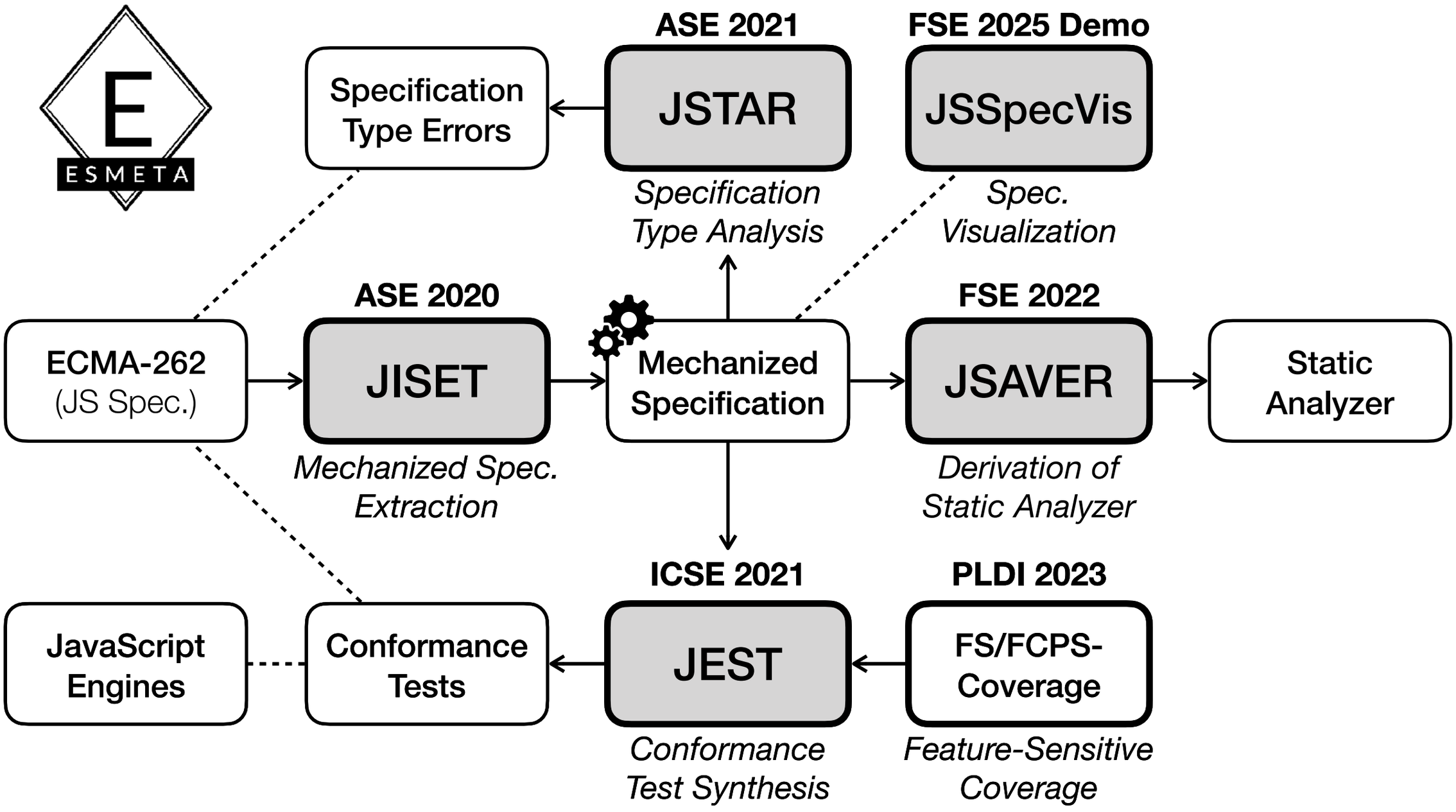
Derived

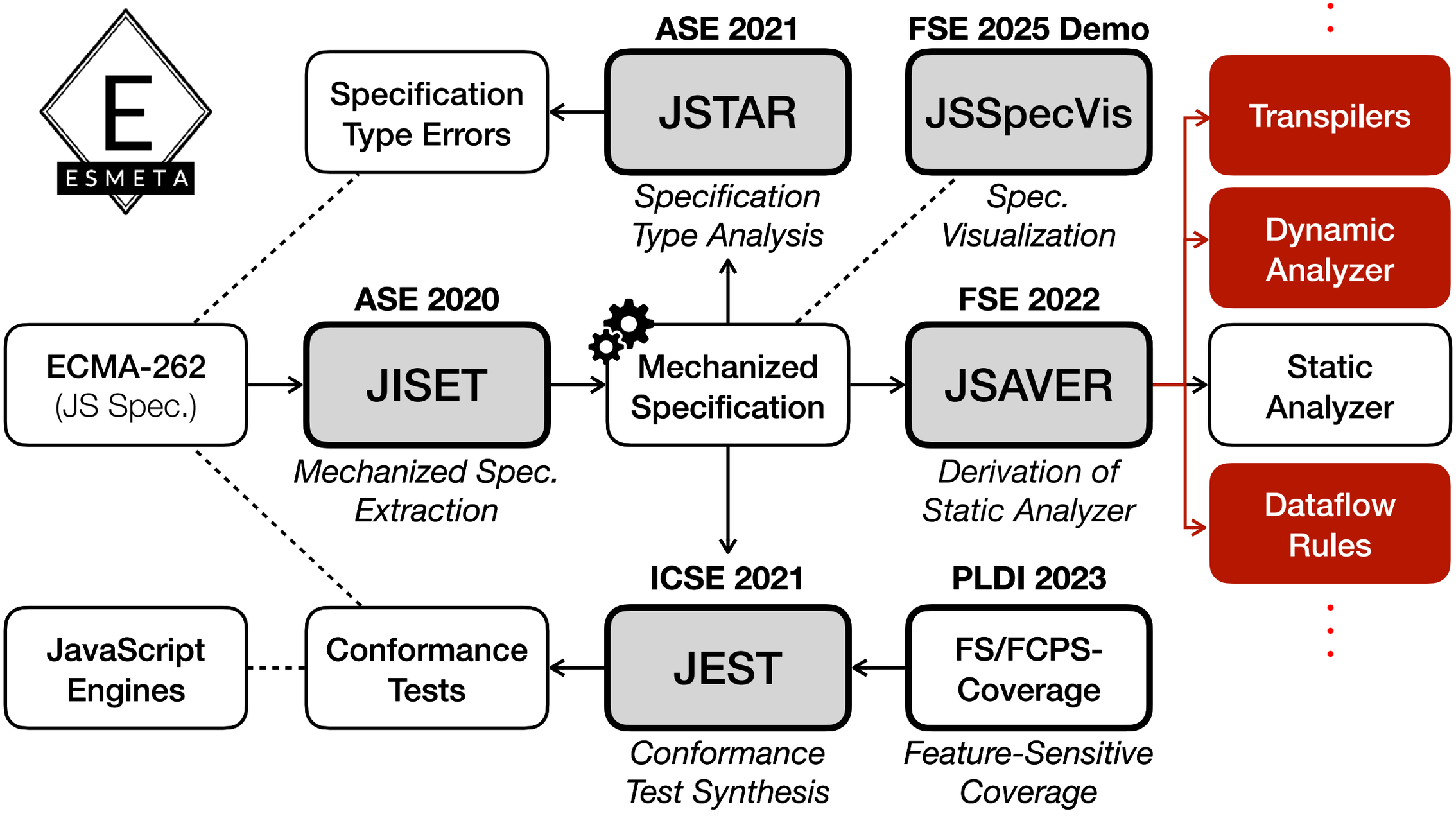


Analysis result of JSAVER



x-axis : creation time (year)
y-axis : # tests

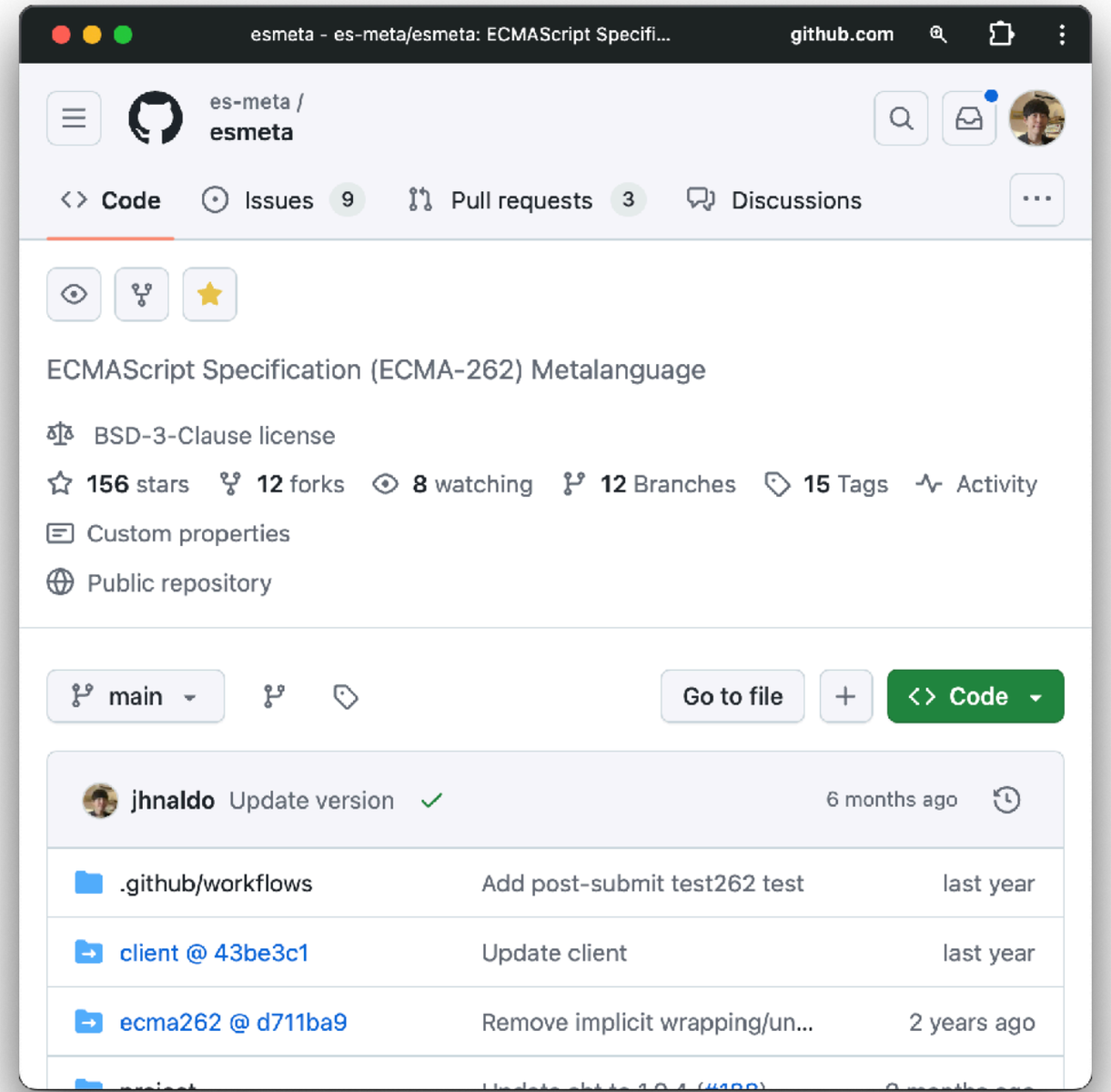

ESMETA
Specification Type Errors
ASE 2021
JSTAR
Specification Type Analysis
FSE 2025 Demo
JSSpecVis
Spec. Visualization
ECMA-262 (JS Spec.)
ASE 2020
JISET
Mechanized Spec. Extraction
Mechanized Specification
FSE 2022
JSAVER
Derivation of Static Analyzer
Static Analyzer
ICSE 2021
JEST
Conformance Test Synthesis
PLDI 2023
FS/FCPS-Coverage
Feature-Sensitive Coverage
JavaScript Engines
Conformance Tests


ESMETA
Specification Type Errors
ASE 2021
JSTAR
Specification Type Analysis
FSE 2025 Demo
JSSpecVis
Spec. Visualization
Transpilers
Dynamic Analyzer
Static Analyzer
Dataflow Rules
ECMA-262 (JS Spec.)
ASE 2020
JISET
Mechanized Spec. Extraction
Mechanized Specification
FSE 2022
JSAVER
Derivation of Static Analyzer
ICSE 2021
JEST
Conformance Test Synthesis
PLDI 2023
FS/FCPS-Coverage
Feature-Sensitive Coverage
JavaScript Engines
Conformance Tests

Official tool used in CI system of ECMA-262 and Test262

ESMETA

https://github.com/es-meta/esmeta

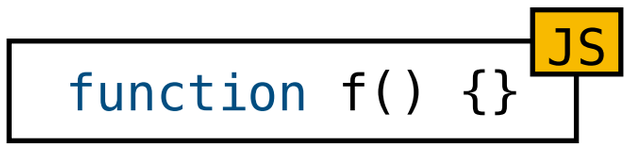


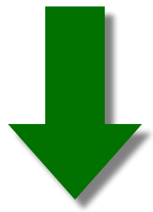
Backup Slides

PLRG @ Korea University

- **Assistant Professor @ CSE Dept. in Korea University**
- **Programming Language Research Group (PLRG)**
- **Members:** 3 Master Students / 2 Undergraduate Students
- **Research Areas:** Programming Languages (PL) and Software Engineering (SE)
  - Program Analysis
  - Mechanized Language Specification
  - Automated Testing
  - Program Synthesis & Repair
- **Publications:** PL and SE
  - **PL**: PLDI (2023 / 2024)
  - **SE**: ICSE (2021) / FSE (2021, 2022, 2025-Demo), ASE (2020, 2021)

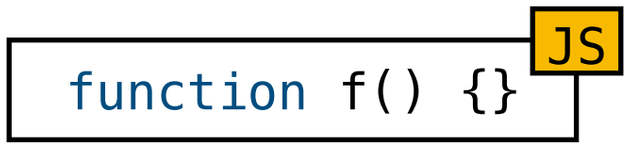
JEST - Final State-based Assertion Injection

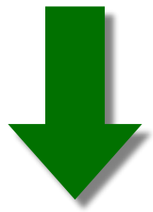




```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

JEST - Final State-based Assertion Injection

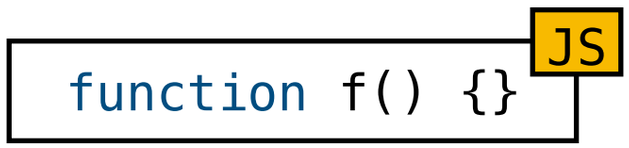


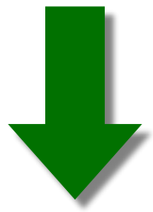


Prototype Chain

```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

JEST - Final State-based Assertion Injection



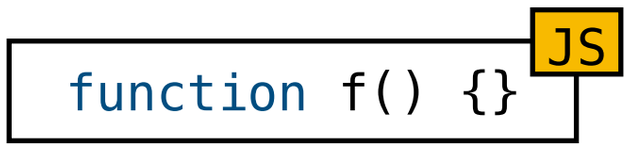


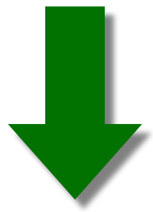
```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

Prototype Chain

Property Descriptor

JEST - Final State-based Assertion Injection





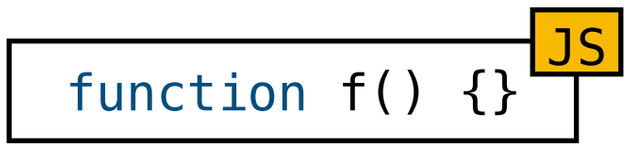
```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

Prototype Chain

Property Descriptor

Property Order

JEST - Final State-based Assertion Injection




```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

Prototype Chain

Property Descriptor

Property Order

Etc.