기계화 명세를 통한 안전한 자바스크립트 언어 생태계 확보 기술

컴퓨터학과 박지혁 교수

2025.04.18 @ SWCS2025

프로그래밍 언어 연구실 PLRG

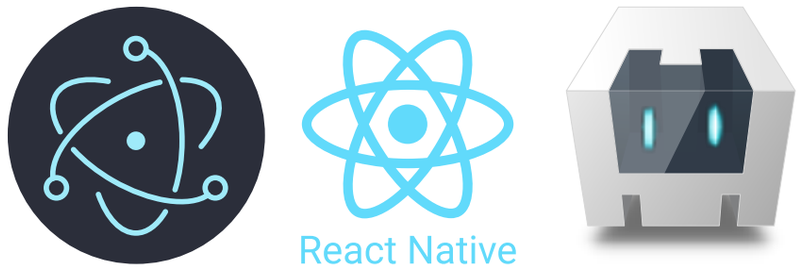
- **Members:** 3 Master Students / 6 Undergraduate Students

- **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)

어디에나 있는 자바스크립트

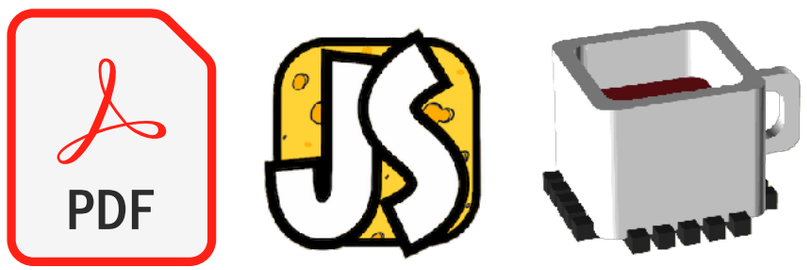
Client-Side Programming



Mobile/Desktop Applications

JS

Sever-Side Programming

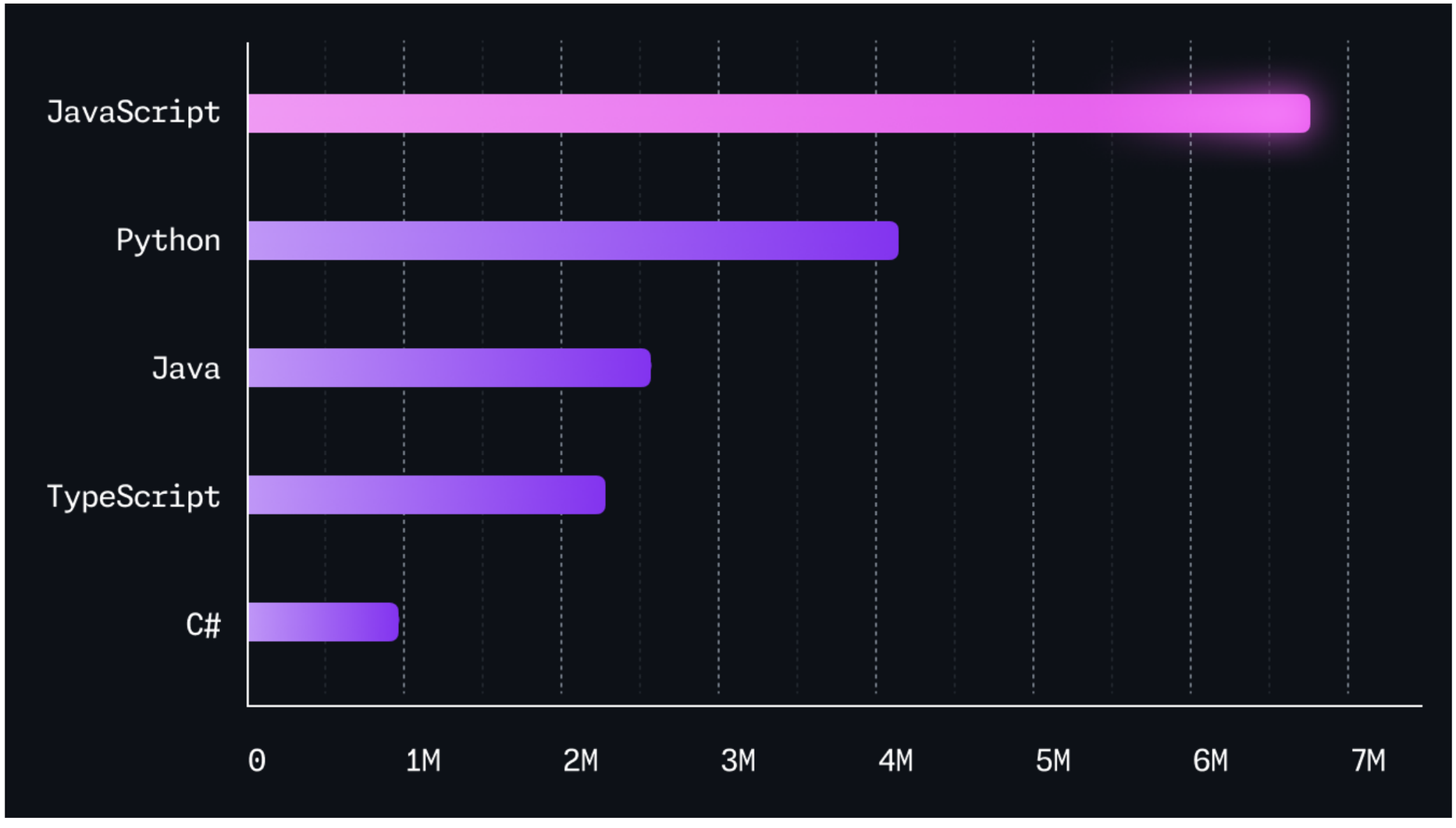


Others (PDF, IoT, Microcontrollers, etc.)

어디에나 있는 자바스크립트

GitHub

- **Top 5 languages** most commonly used in repositories used in repositories created within the last 12 months on GitHub

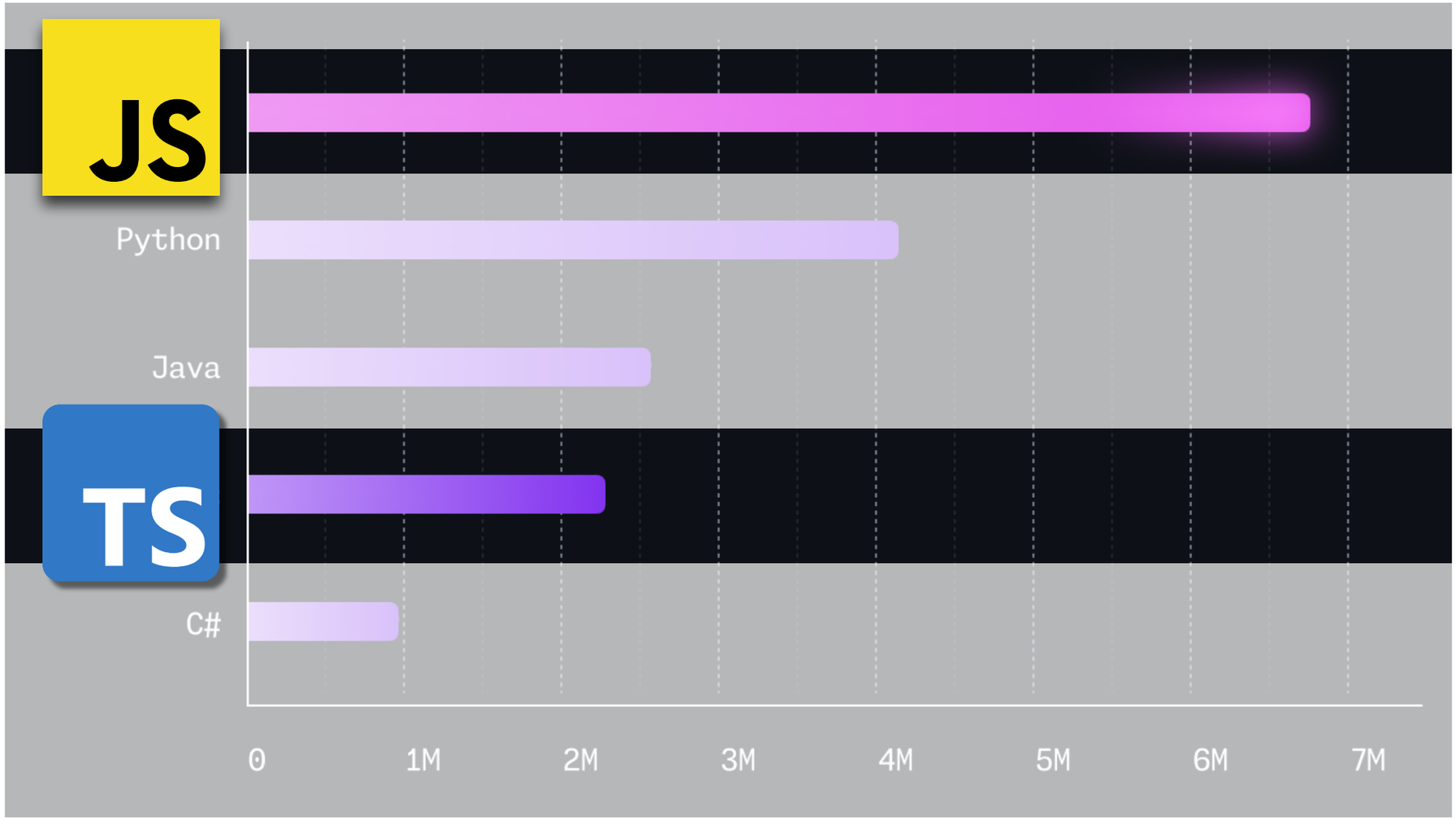


https://github.blog/news-insights/octoverse/octoverse-2024/

어디에나 있는 자바스크립트

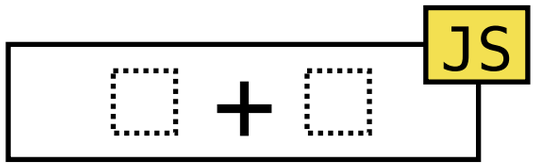
GitHub

- **Top 5 languages** most commonly used in repositories used in repositories created within the last 12 months on GitHub

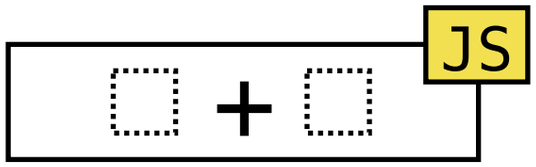


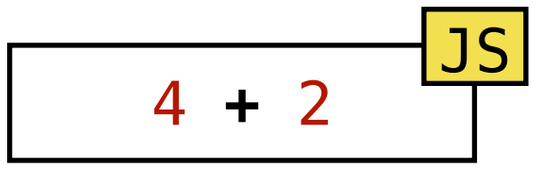
https://github.blog/news-insights/octoverse/octoverse-2024/

하지만, 복잡한 자바스크립트

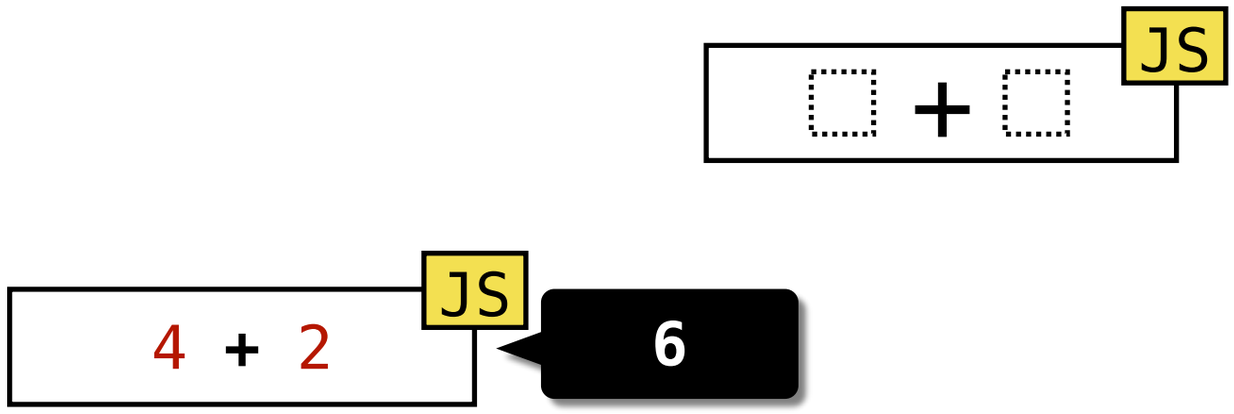


하지만, 복잡한 자바스크립트

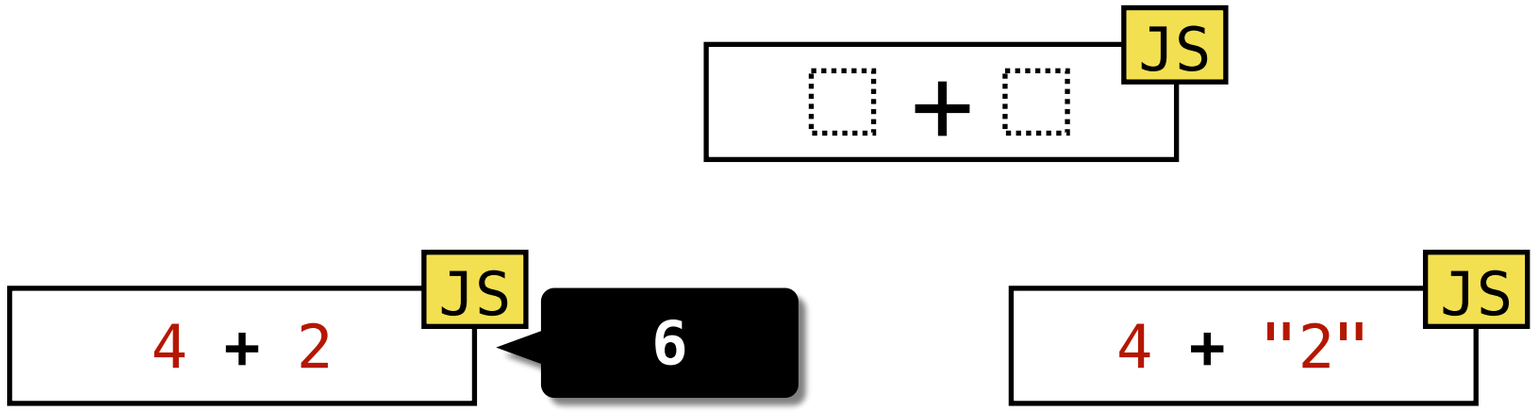




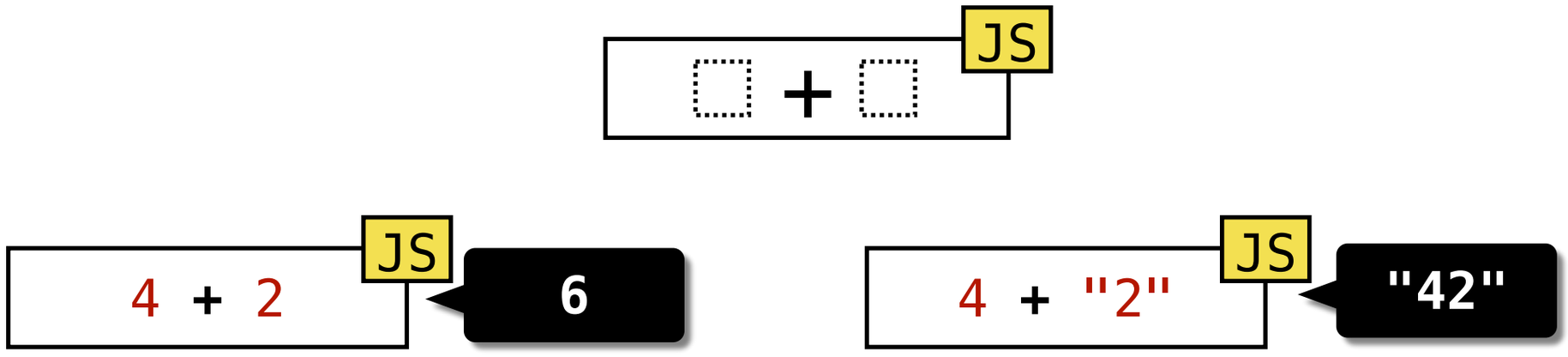
하지만, 복잡한 자바스크립트



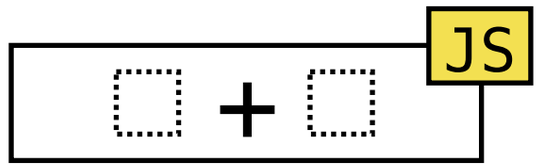
하지만, 복잡한 자바스크립트

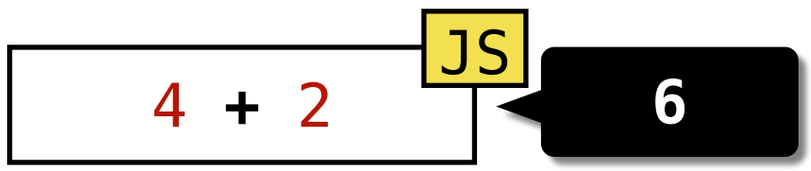


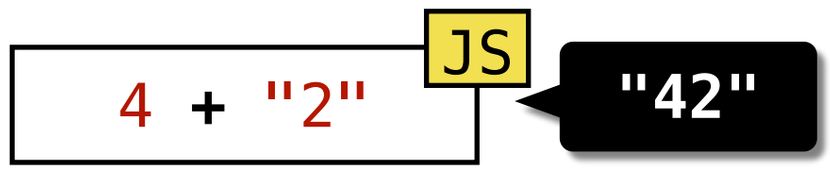
하지만, 복잡한 자바스크립트

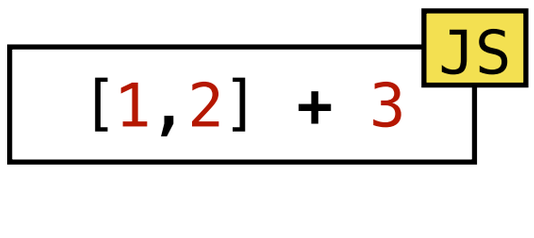


하지만, 복잡한 자바스크립트

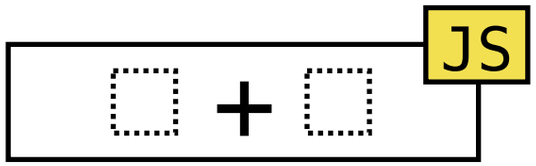


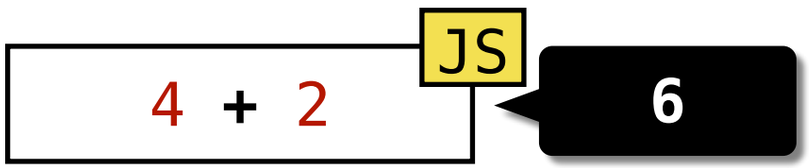


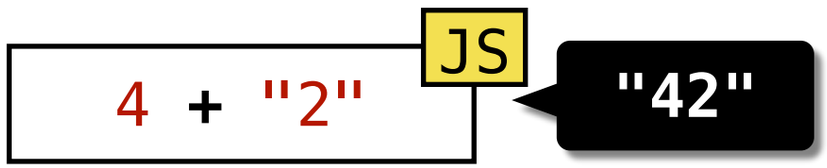


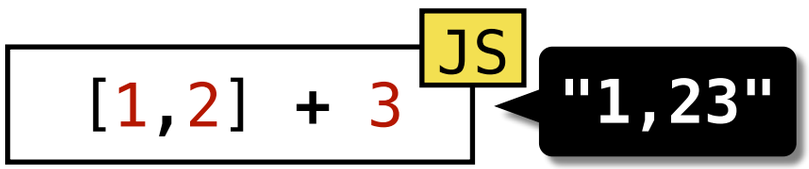


하지만, 복잡한 자바스크립트

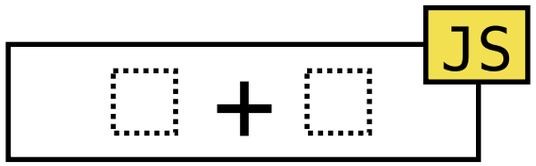


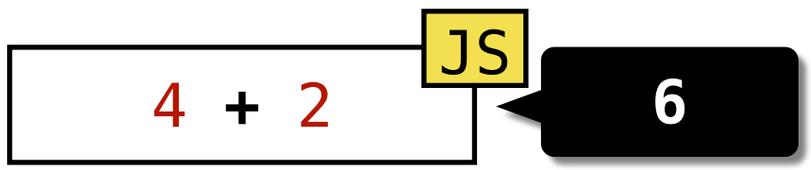


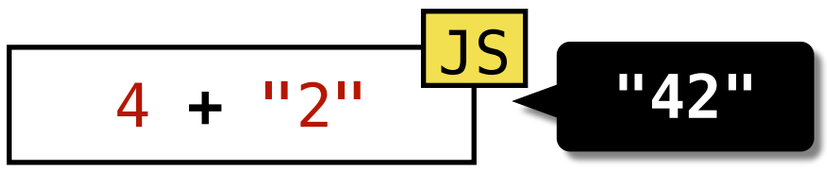


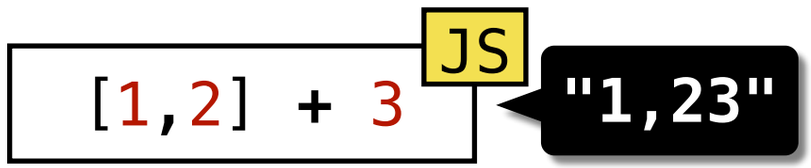


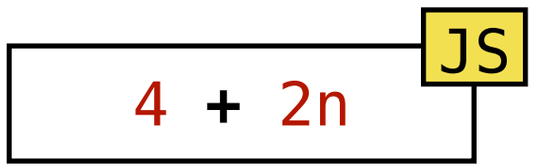
하지만, 복잡한 자바스크립트











하지만, 복잡한 자바스크립트

하지만, 복잡한 자바스크립트

JS

□ + □

JS

```
4 + 2
```

6

JS

```
4 + "2"
```

"42"

JS

```
[1,2] + 3
```

"1,23"

JS

```
4 + 2n
```

TypeError

JS

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

하지만, 복잡한 자바스크립트

JS

□ + □

JS

```
4 + 2
```

6

JS

```
4 + "2"
```

"42"

JS

```
[1,2] + 3
```

"1,23"

JS

```
4 + 2n
```

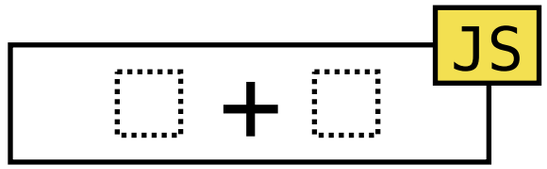
TypeError

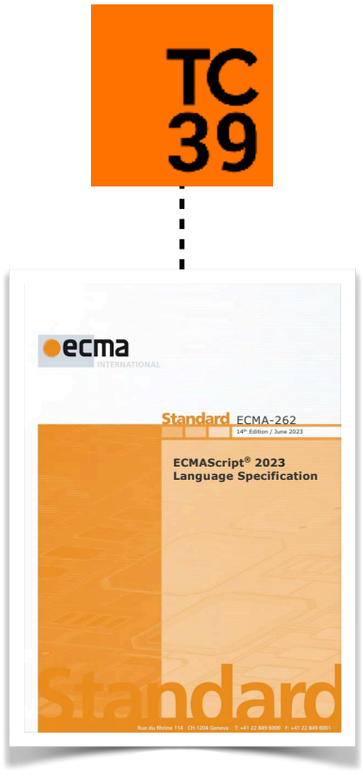
JS

```
(![]+[])[+[]]                    +      // "f"
(![]+[])[+!+[]]                  +      // "a"
([![]]+[][[]])[+!+[]+[+[]]] +           // "i"
(![]+[])[!+[]+!+[]]                     // "l"
```

"fail"

ECMA-262 - 자바스크립트 공식 언어 명세





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

TypeError

JS

```
!0 + 2n
```

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

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

TypeError

JS

```
!0 + 2n
```

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

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

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

TypeError

JS

```
!0 + 2n
```

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

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

Expr `!0` + Expr `2n`

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

TypeError

JS

```
!0 + 2n
```

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

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

Expr `!0` | `+` | Expr `2n`

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

`Evaluate Left  (lval = true)`

TypeError

JS

```
!0 + 2n
```

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

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

Expr `!0` | `+` | Expr `2n`

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** | **+** | Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

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

1. If *opText* is **+**, then
    a. Let *lprim* be ? ToPrimitive(*lval*).
    b. Let *rprim* be ? ToPrimitive(*rval*).
    c. If *lprim* is a String or *rprim* is a String, then …
    d. Set *lval* to *lprim*.
    e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.

…

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** | **+** | Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** | **+**

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

BigInt **2n**

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).
   c. If *lprim* is a String or *rprim* is a String, then …
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.

…

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** | **+** | Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** | **+**

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

BigInt **2n**

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*)

Conversion to Primitive
(lprim = **true** and rprim = **2n**)

   c. If *lprim* is a String or *rprim* is a String, then **...**
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.

...

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** | **+** | Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** | **+**

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

BigInt **2n**

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).

Conversion to Primitive
(lprim = **true** and rprim = **2n**)

   c. If *lprim* is a String or *rprim* is a String, then **...**
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).

Conversion to Numeric
(lnum = **1** and rnum = **2n**)

5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.

...

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** + Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** + BigInt **2n**

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

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).

Conversion to Primitive (lprim = **true** and rprim = **2n**)

   c. If *lprim* is a String or *rprim* is a String, then ...
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.

Conversion to Numeric (lnum = **1** and rnum = **2n**)

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

...

Number

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** | **+** | Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** | **+**

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

BigInt **2n**

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).

Conversion to Primitive
(lprim = **true** and rprim = **2n**)

   c. If *lprim* is a String or *rprim* is a String, then **...**
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.

Conversion to Numeric
(lnum = **1** and rnum = **2n**)

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

...

Number | BigInt

TypeError

JS

```
!0 + 2n
```

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

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

Expr **!0** + Expr **2n**

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

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).

Evaluate Left (lval = **true**)

3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).

Evaluate Right (rval = **2n**)

5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Boolean **true** + BigInt **2n**

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

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).
   c. If *lprim* is a String or *rprim* is a String, then ...
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeri
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.

...

Conversion to Primitive (lprim = **true** and rprim = **2n**)

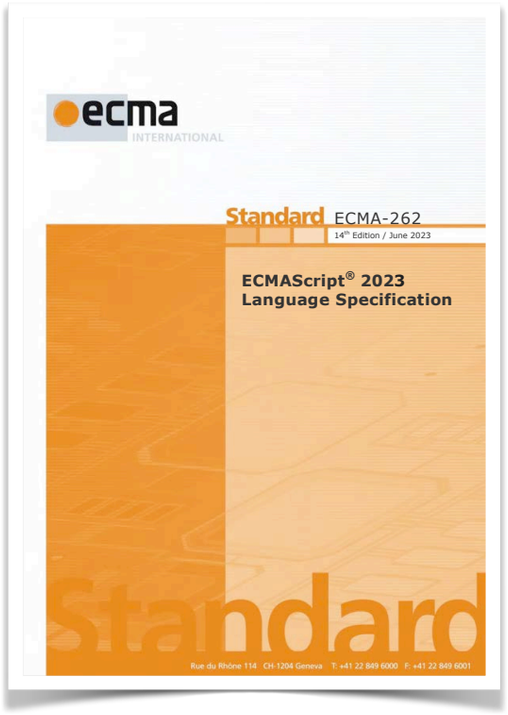
Conversion to Numeric (lnum = **1** and rnum = **2n**)

TypeError

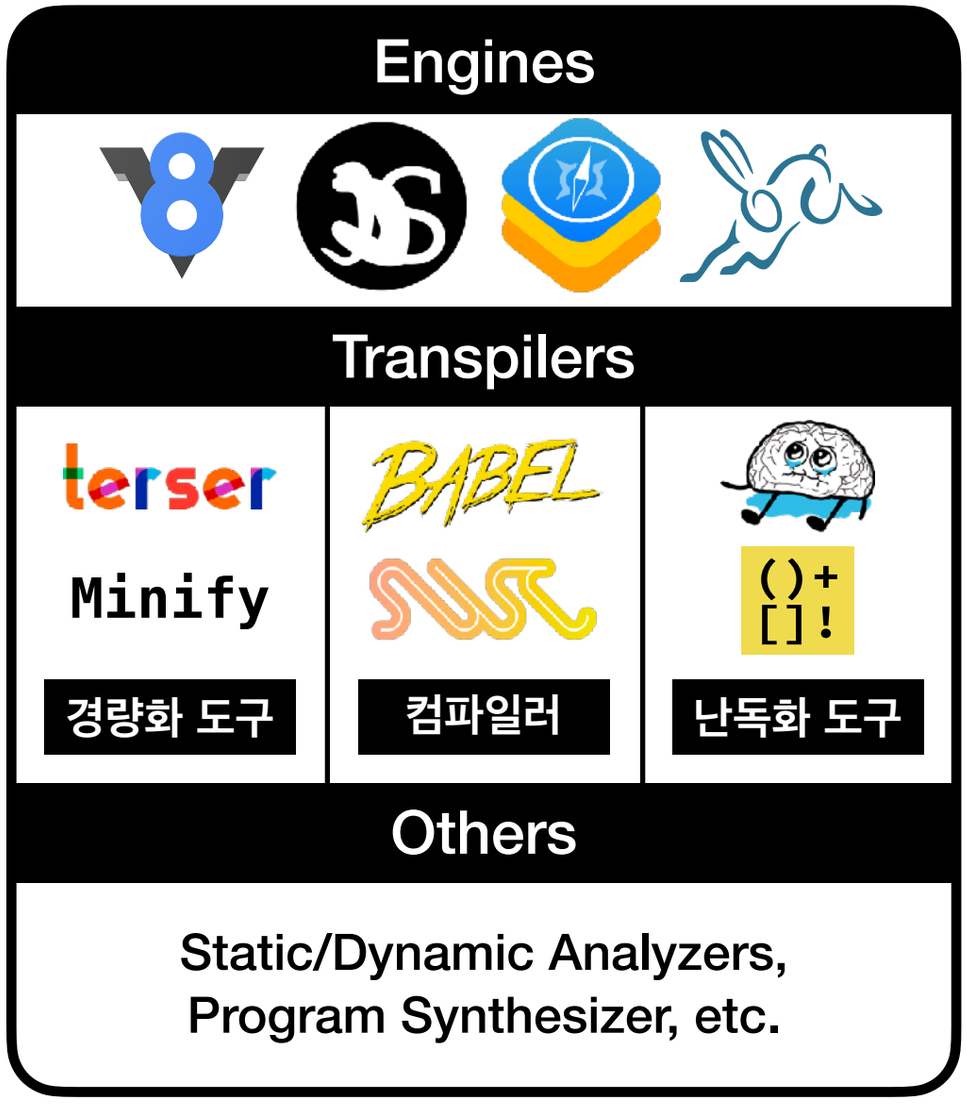
Number

BigInt

자바스크립트 언어의 설계 및 구현



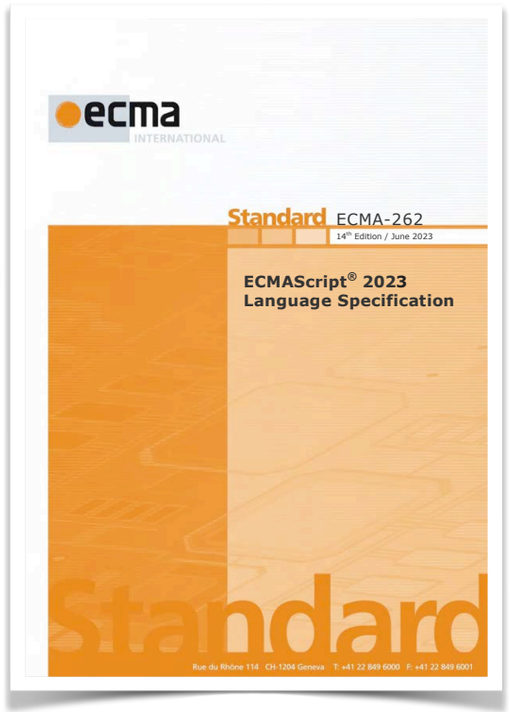
ECMA-262
(JavaScript Spec.)



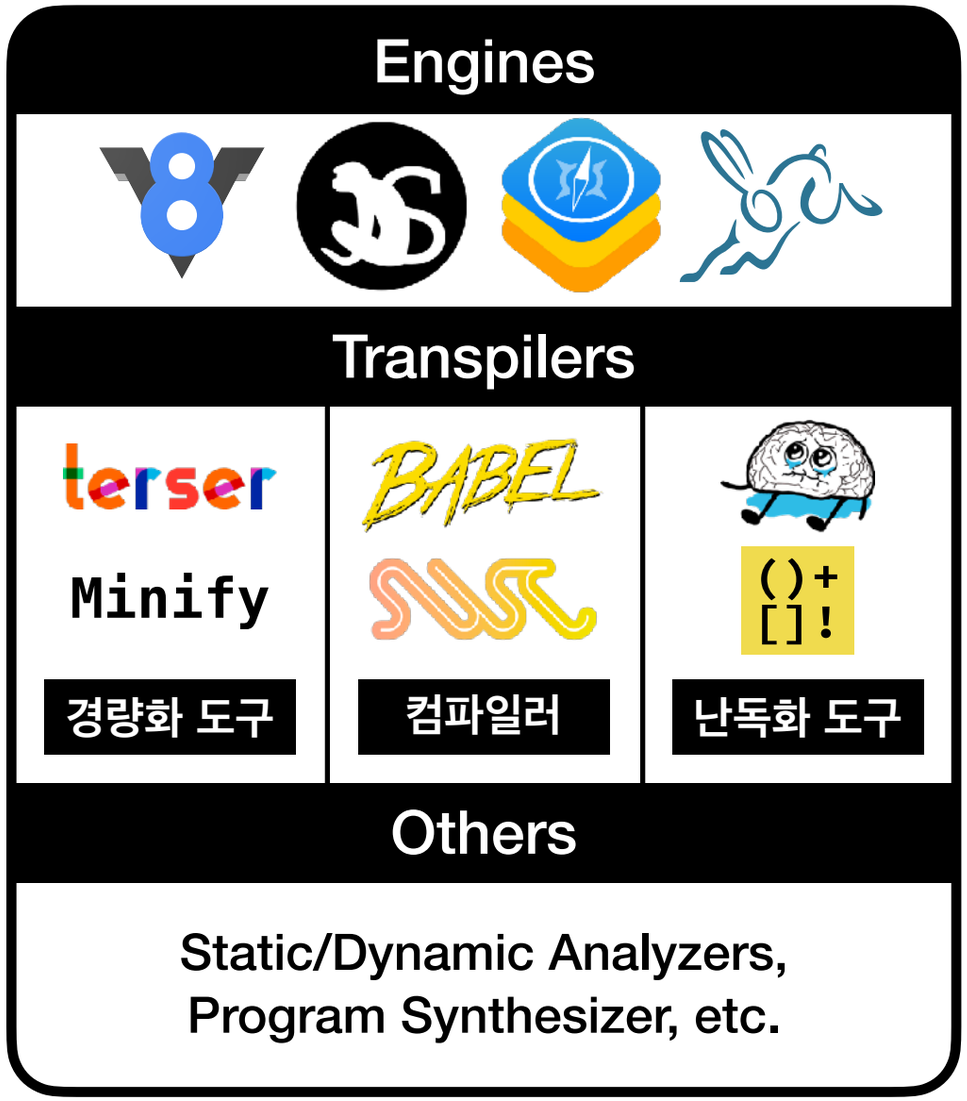
JavaScript Implementations

자바스크립트 언어의 설계 및 구현

Update



ECMA-262
(JavaScript Spec.)

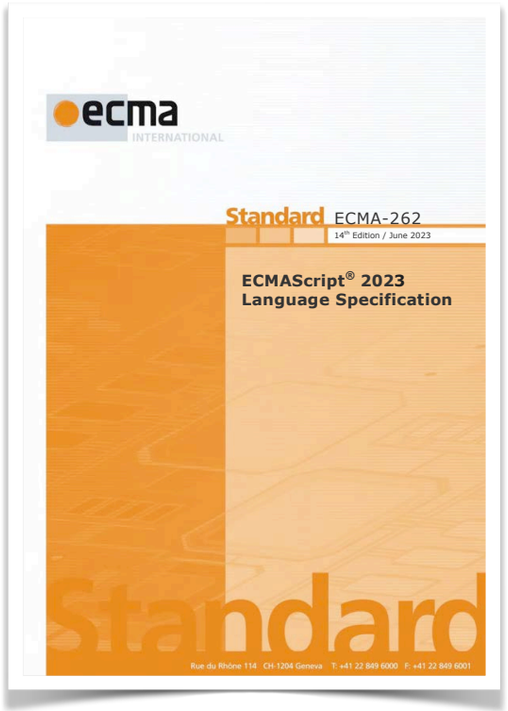


JavaScript Implementations

Update

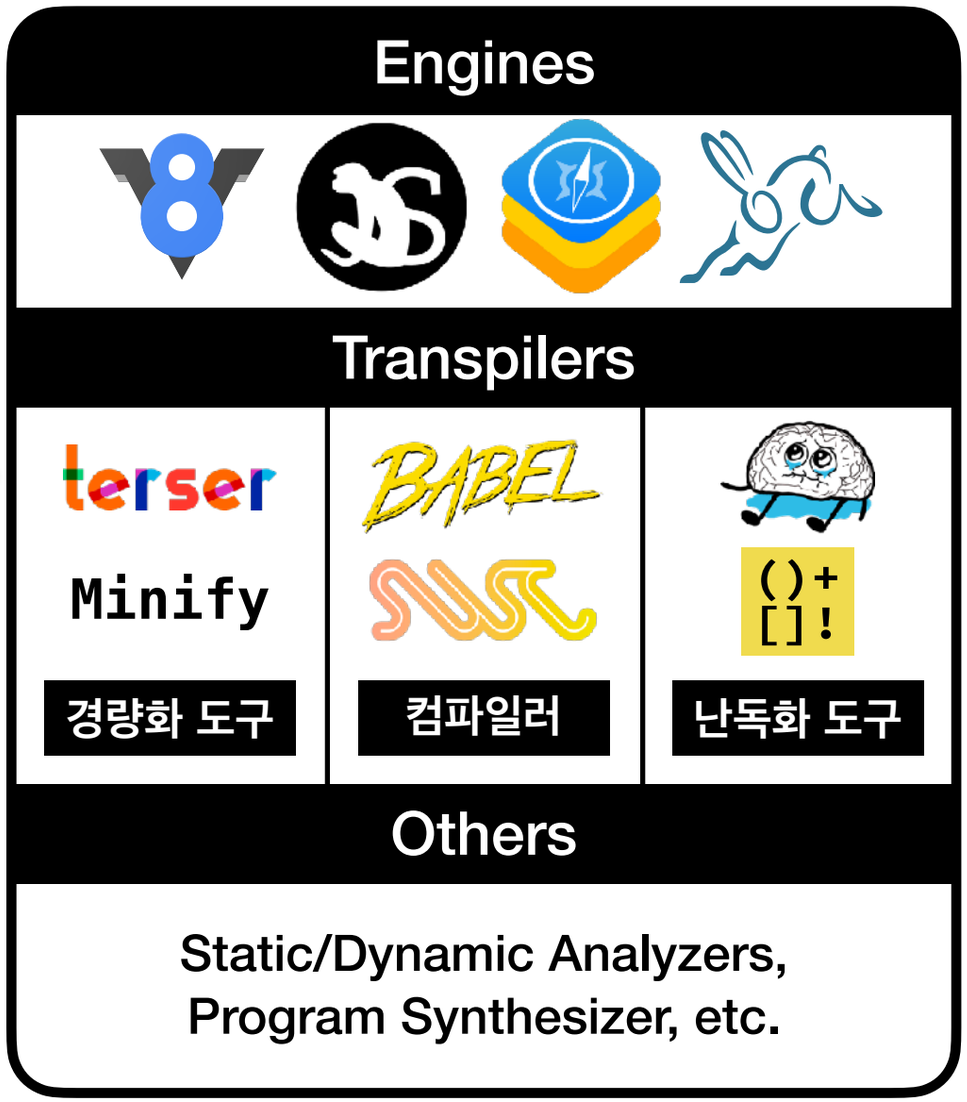
자바스크립트 언어의 설계 및 구현

Update



ECMA-262
(JavaScript Spec.)

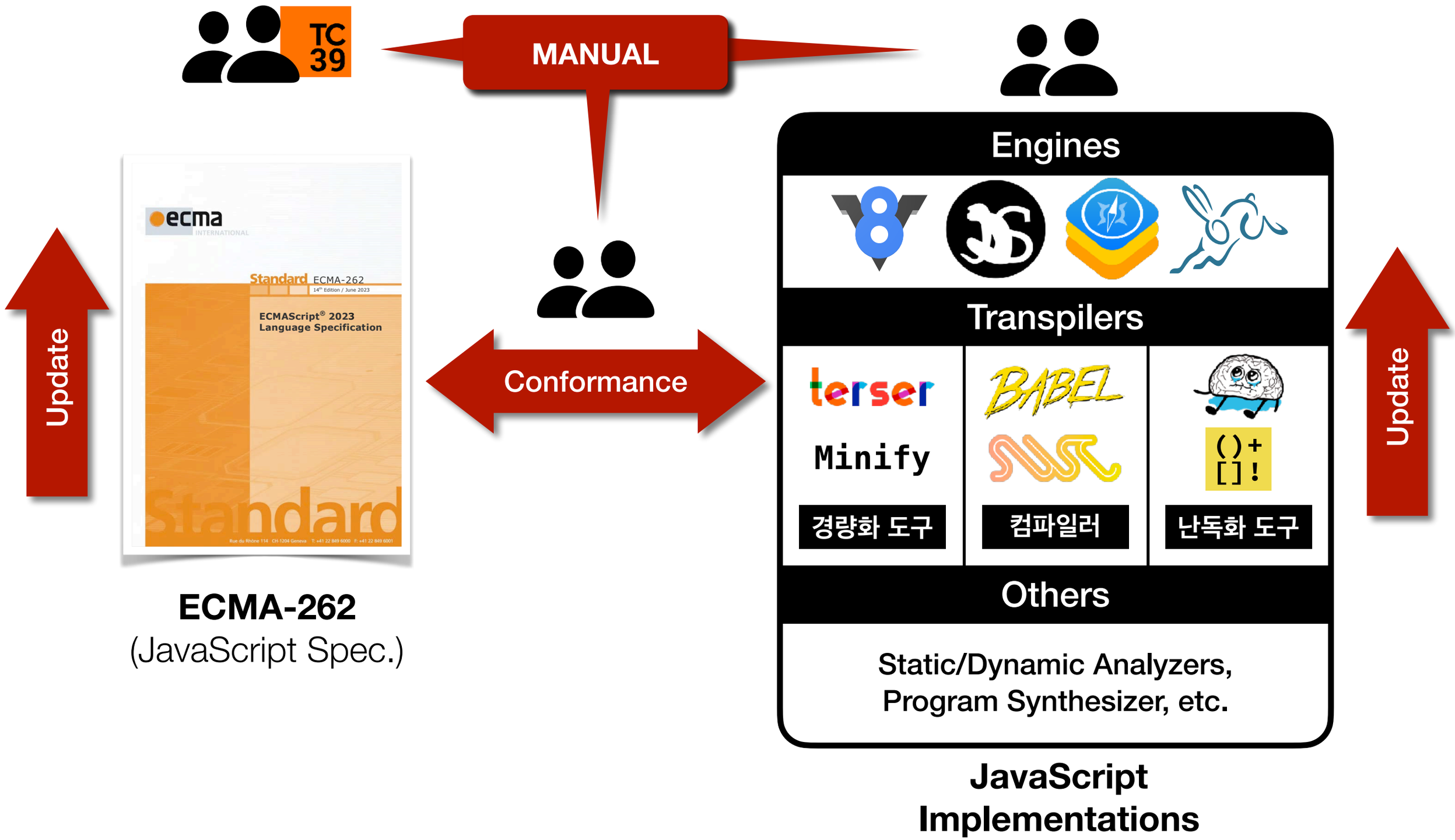
Conformance



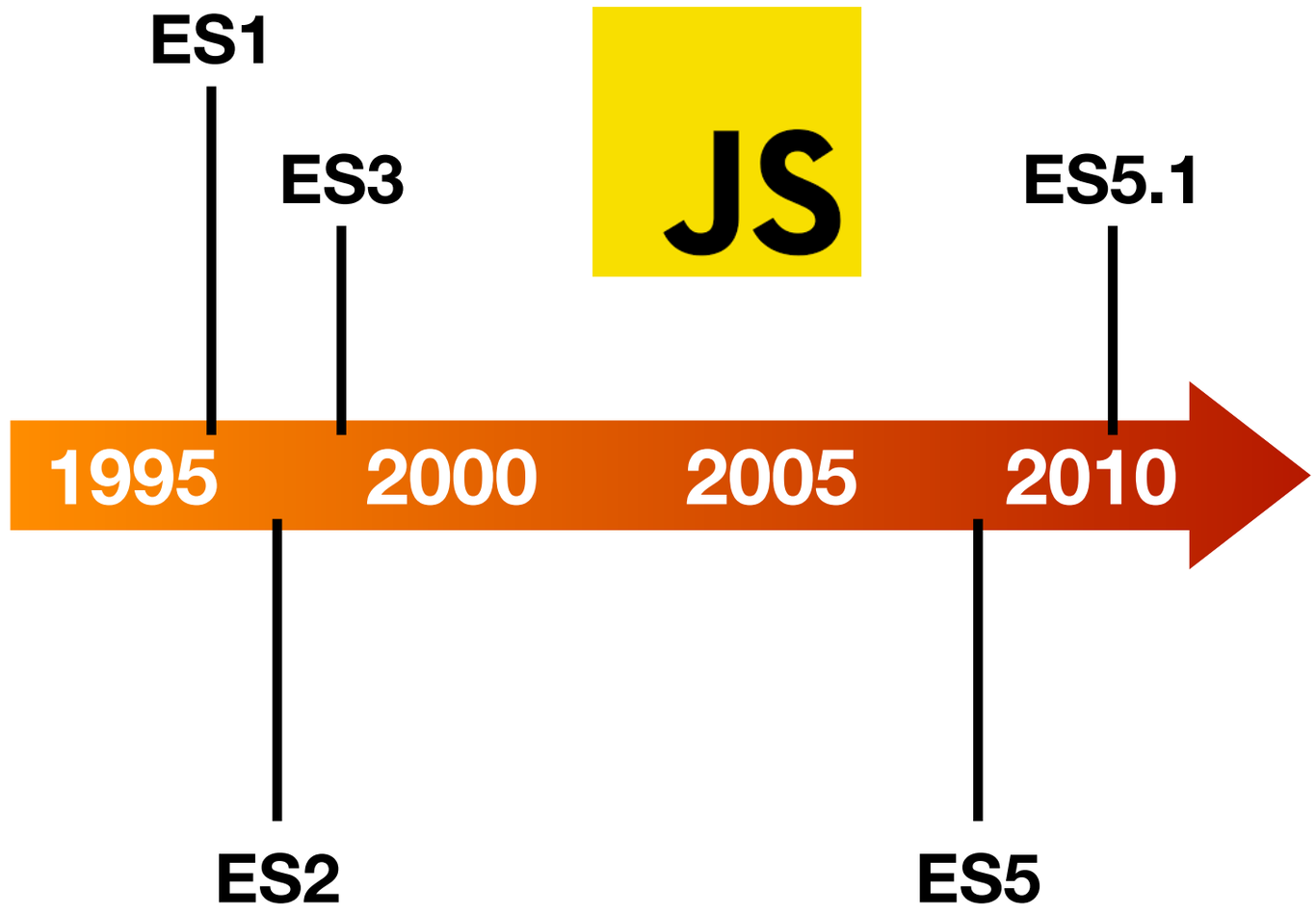
JavaScript Implementations

Update

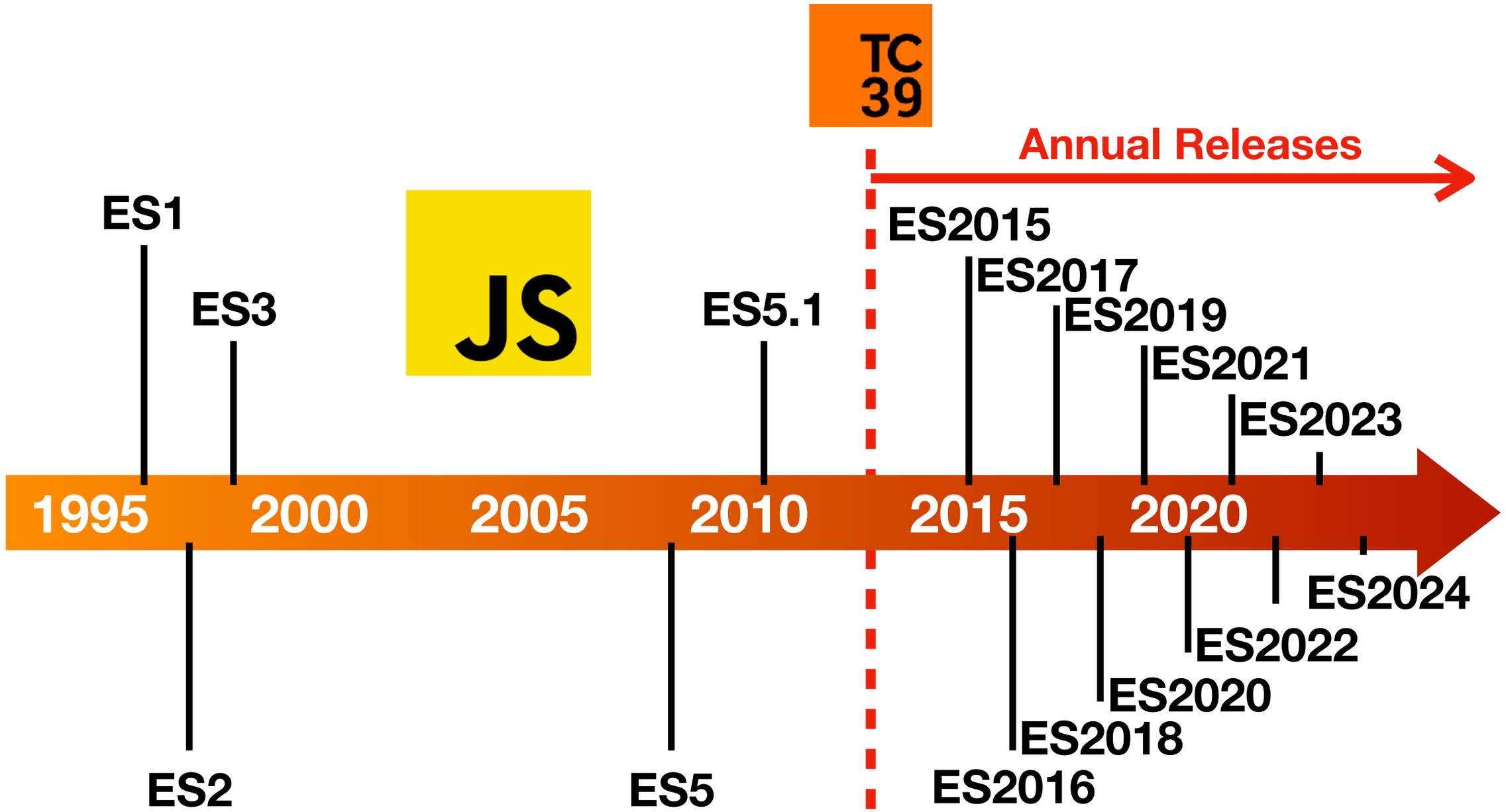
자바스크립트 언어의 설계 및 구현



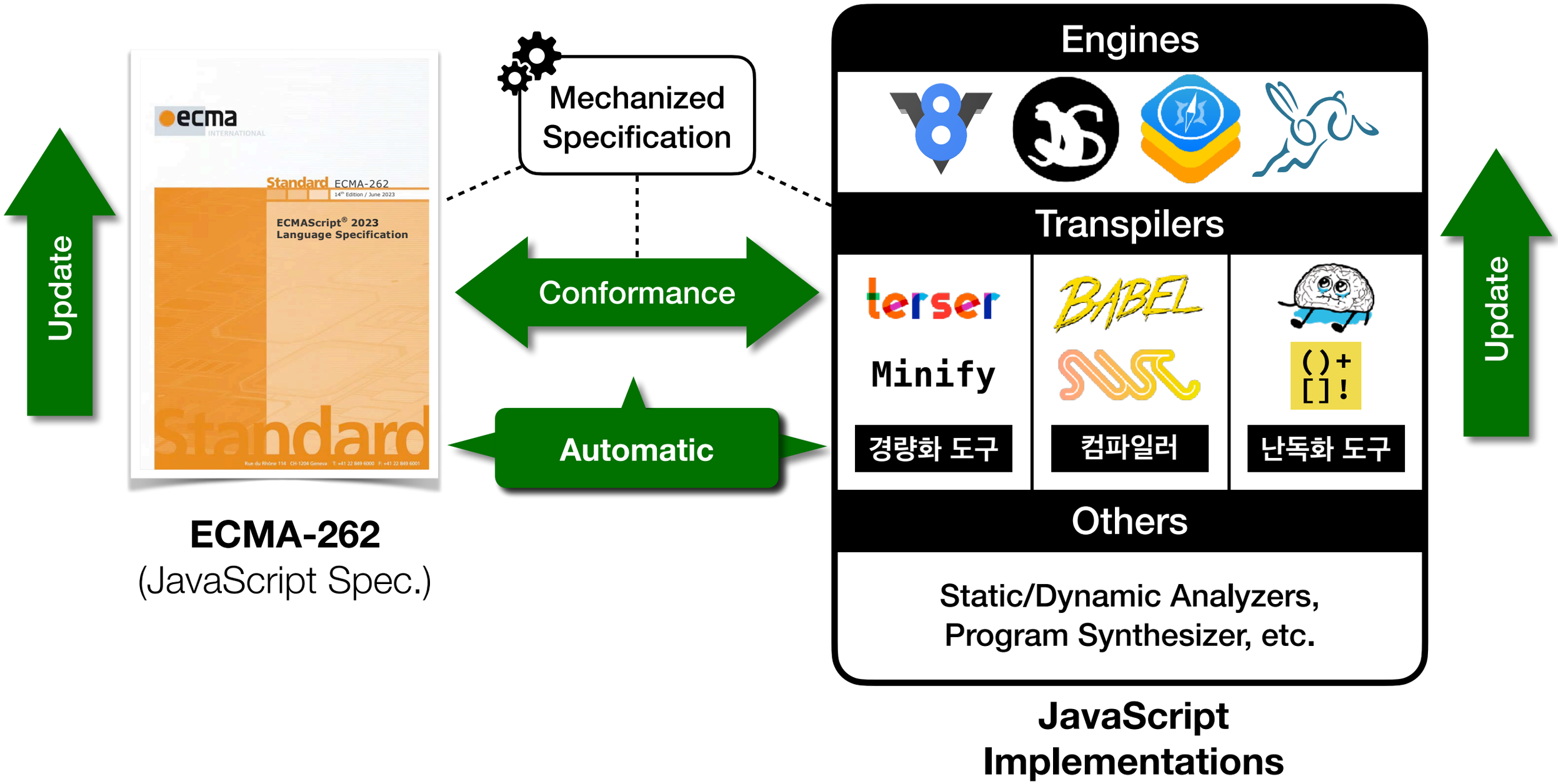
문제 - 빠르게 성장하는 자바스크립트

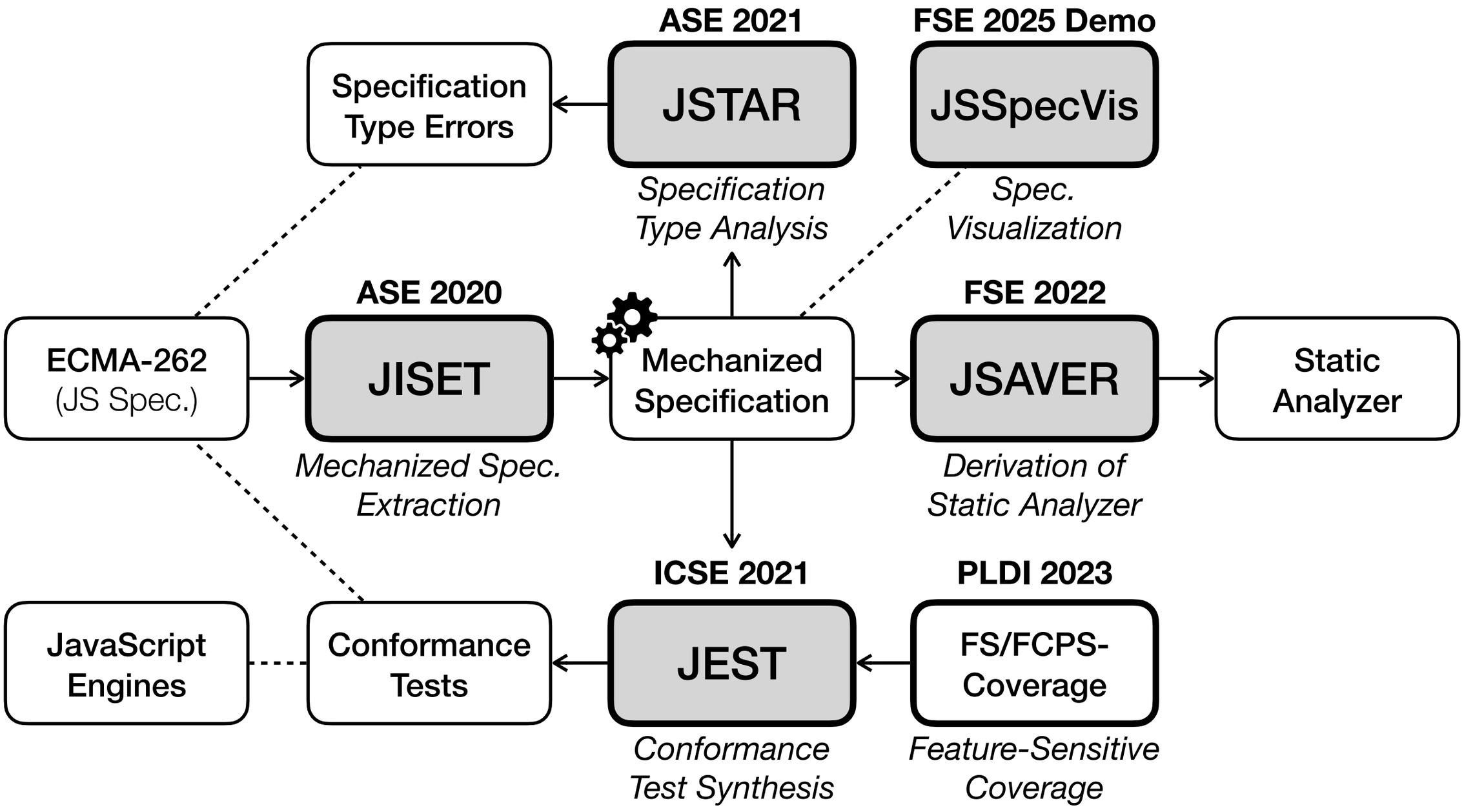


문제 - 빠르게 성장하는 자바스크립트

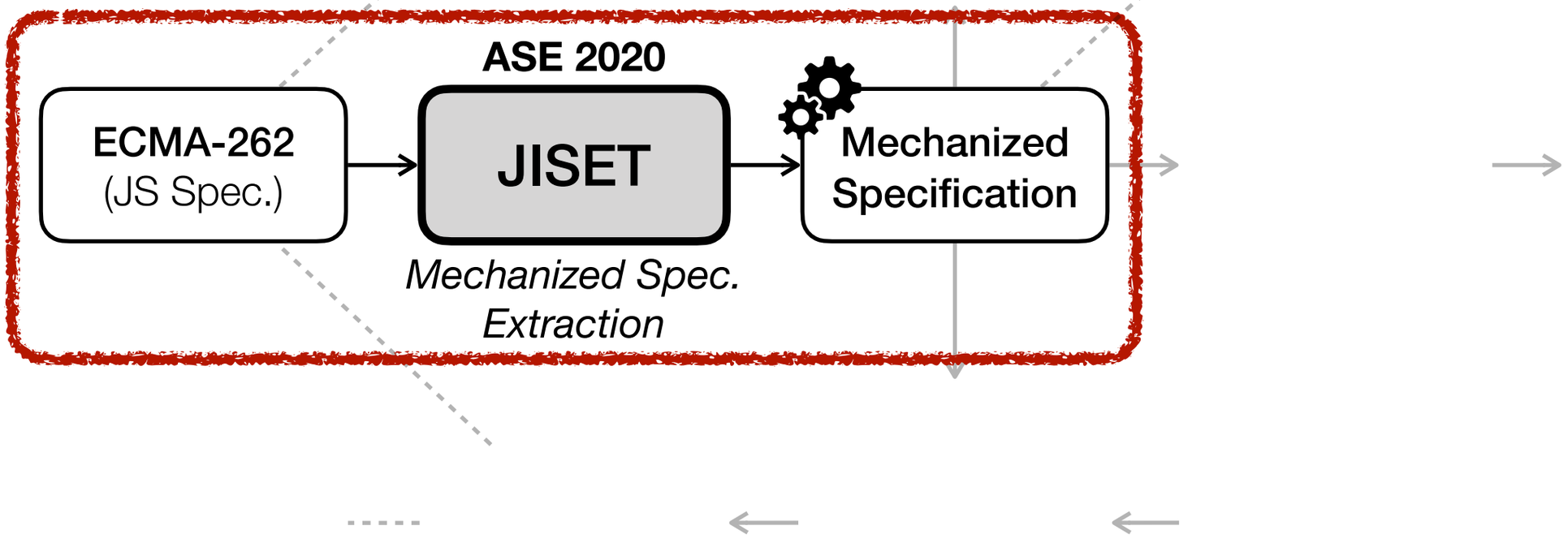


자바스크립트 기계화 명세로 자동화

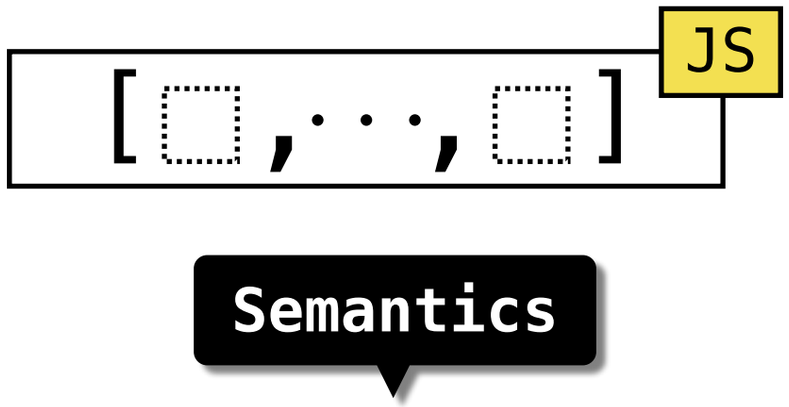



ASE 2021
FSE 2025 Demo
Specification Type Errors
JSTAR
JSSpecVis
Specification Type Analysis
Spec. Visualization
ASE 2020
FSE 2022
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
ICSE 2021
PLDI 2023
JavaScript Engines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

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



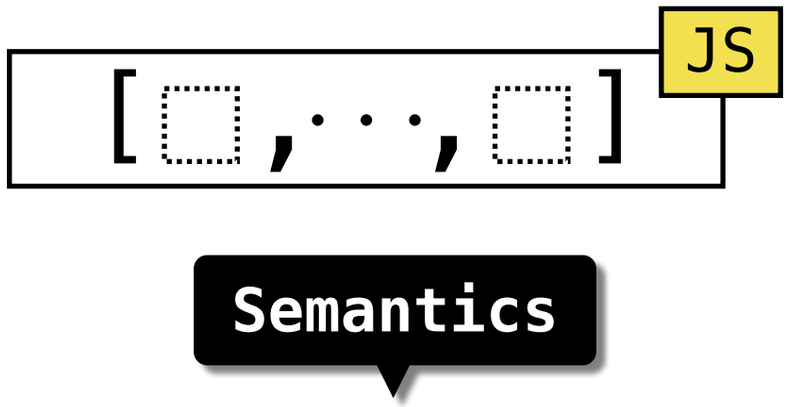
JISET - 명세 알고리즘 속 패턴



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*.

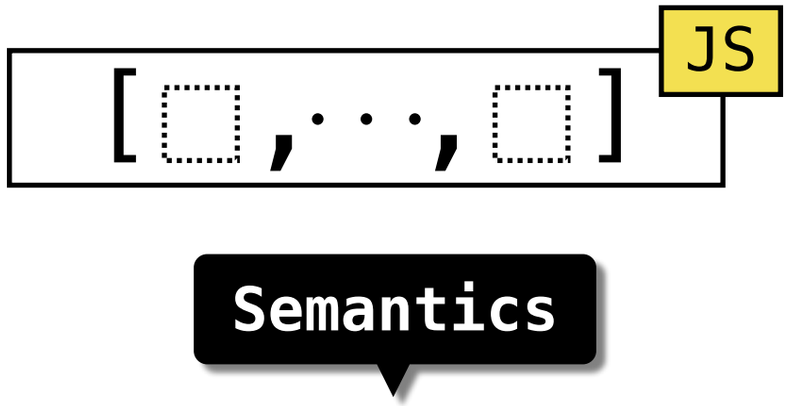
JISET - 명세 알고리즘 속 패턴



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*.

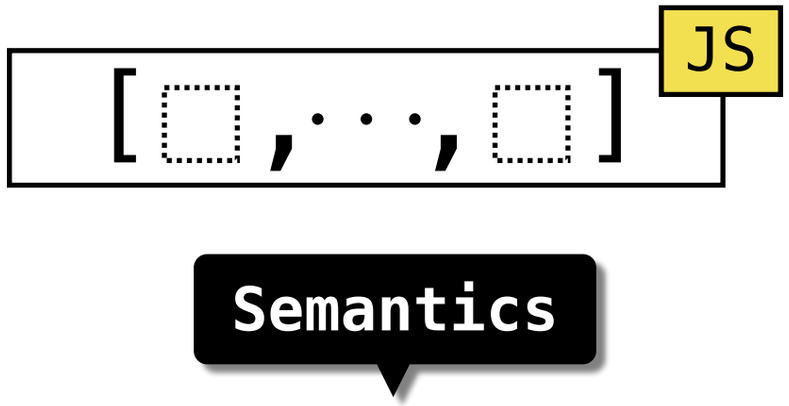
JISET - 명세 알고리즘 속 패턴



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 - 명세 알고리즘 속 패턴



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 - ECMA-262를 위한 **Metalanguage**

IR_{ES} - Intermediate Representation for ECMA-262

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

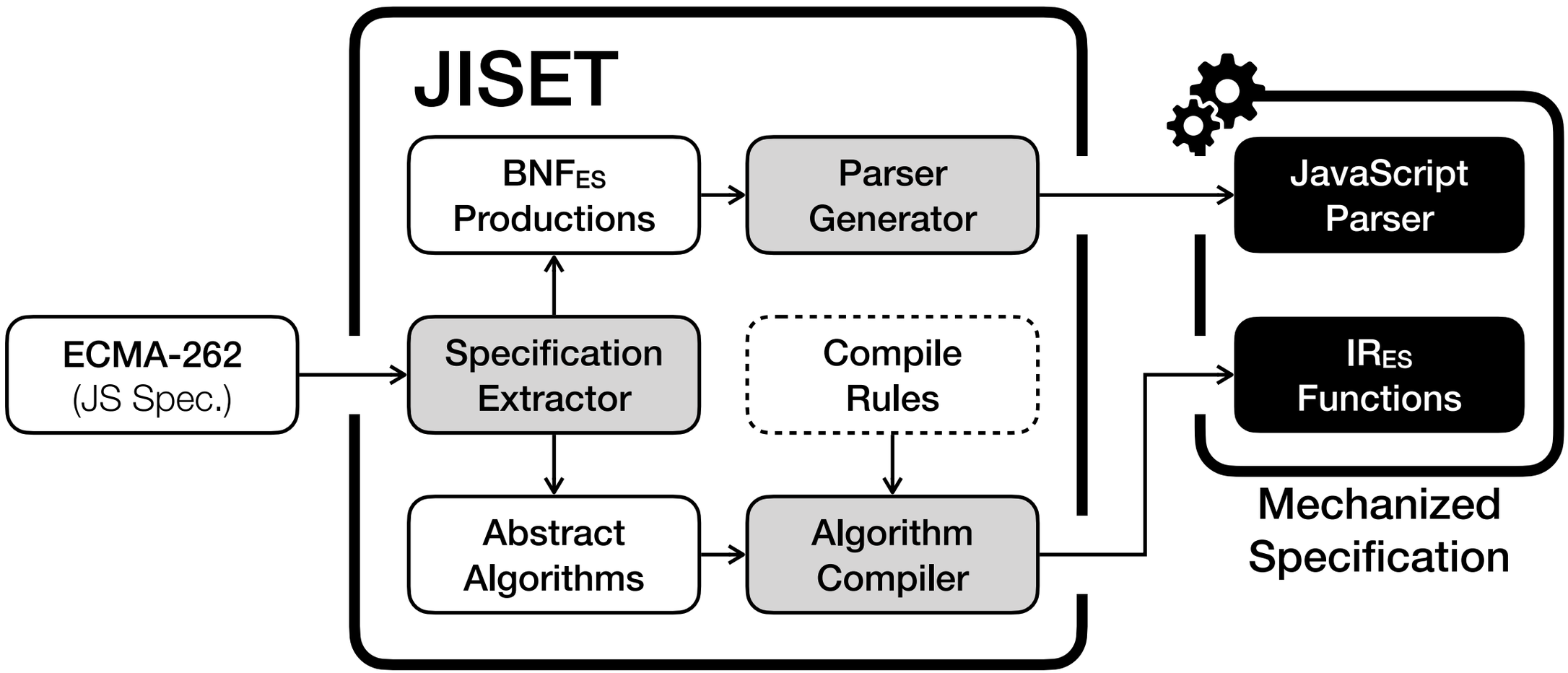
JISET - ECMA-262를 위한 **Metalanguage**

IR_{ES} - Intermediate Representation for ECMA-262

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

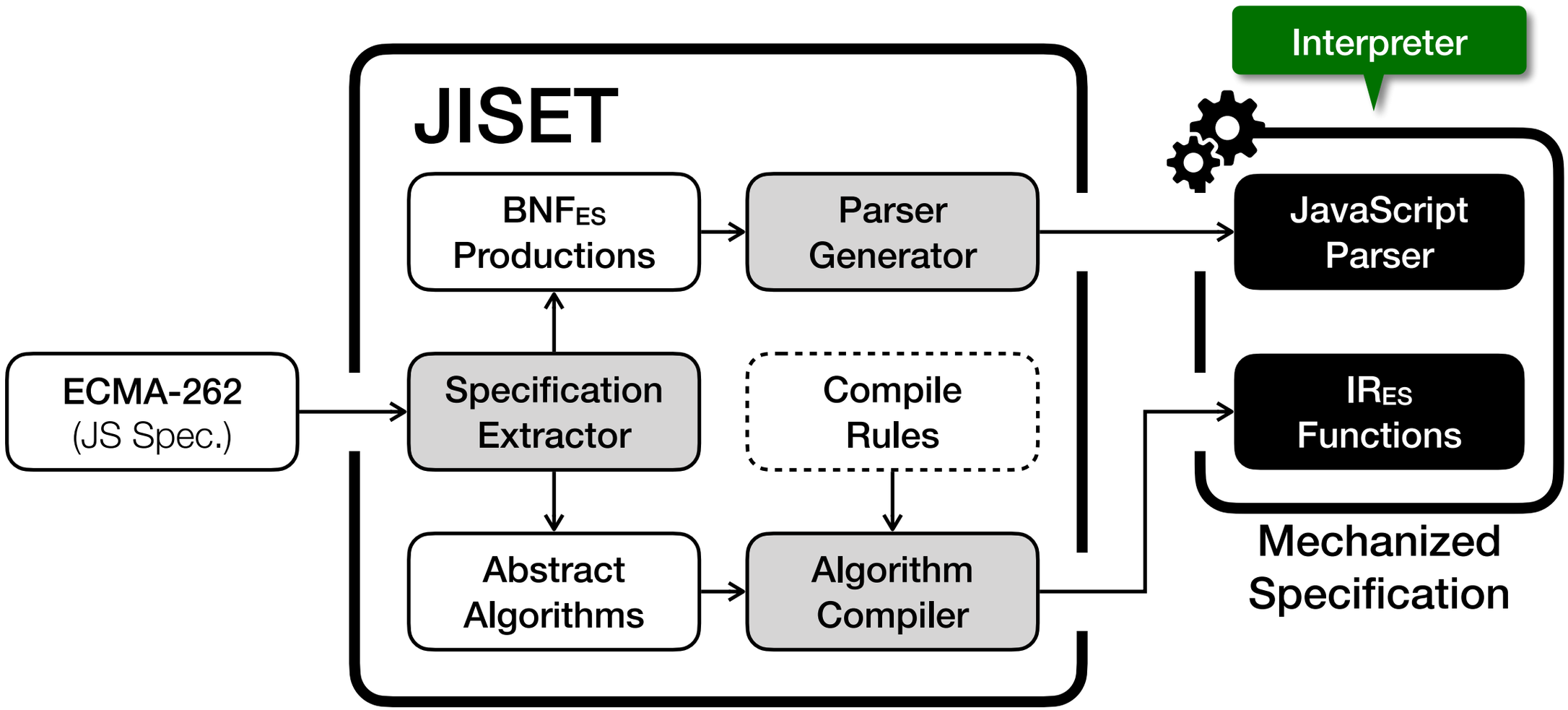
JISET

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



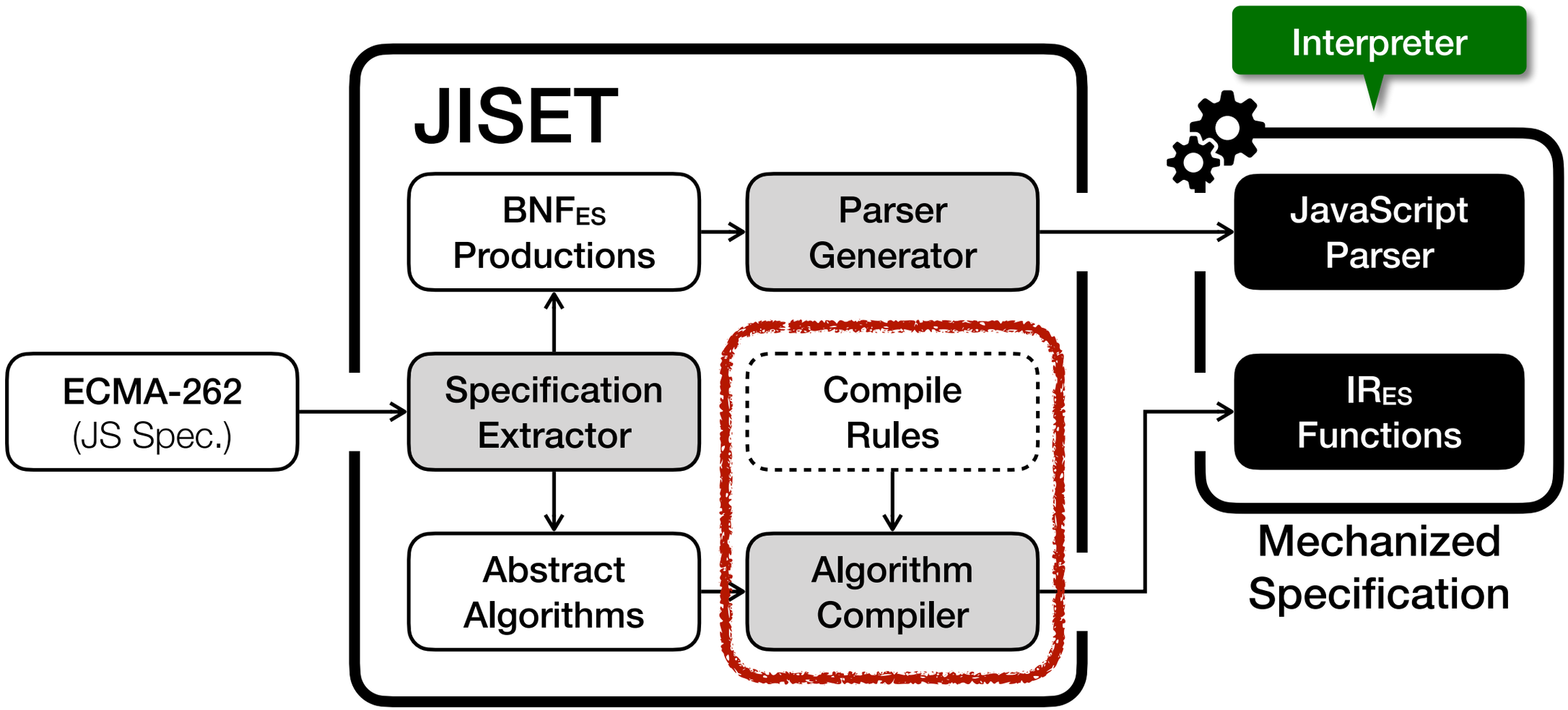
JISET

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



JISET

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

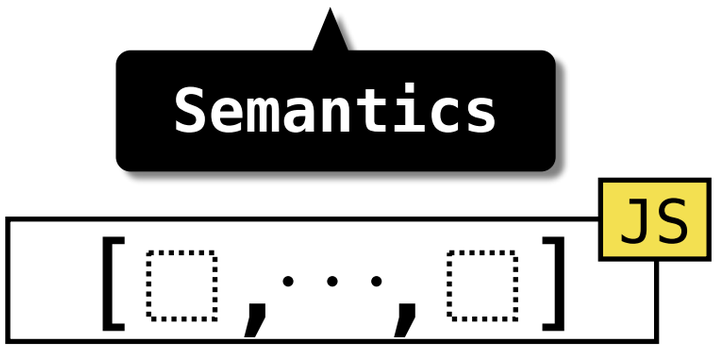


JISET - 알고리즘 컴파일러

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 - 알고리즘 컴파일러

Abstract algorithm for *ArrayLiteral* in ES13

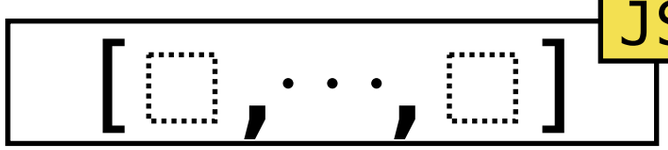
ArrayLiteral : `[` *ElementList* `,` *Elision*$_{opt}$ `]`

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

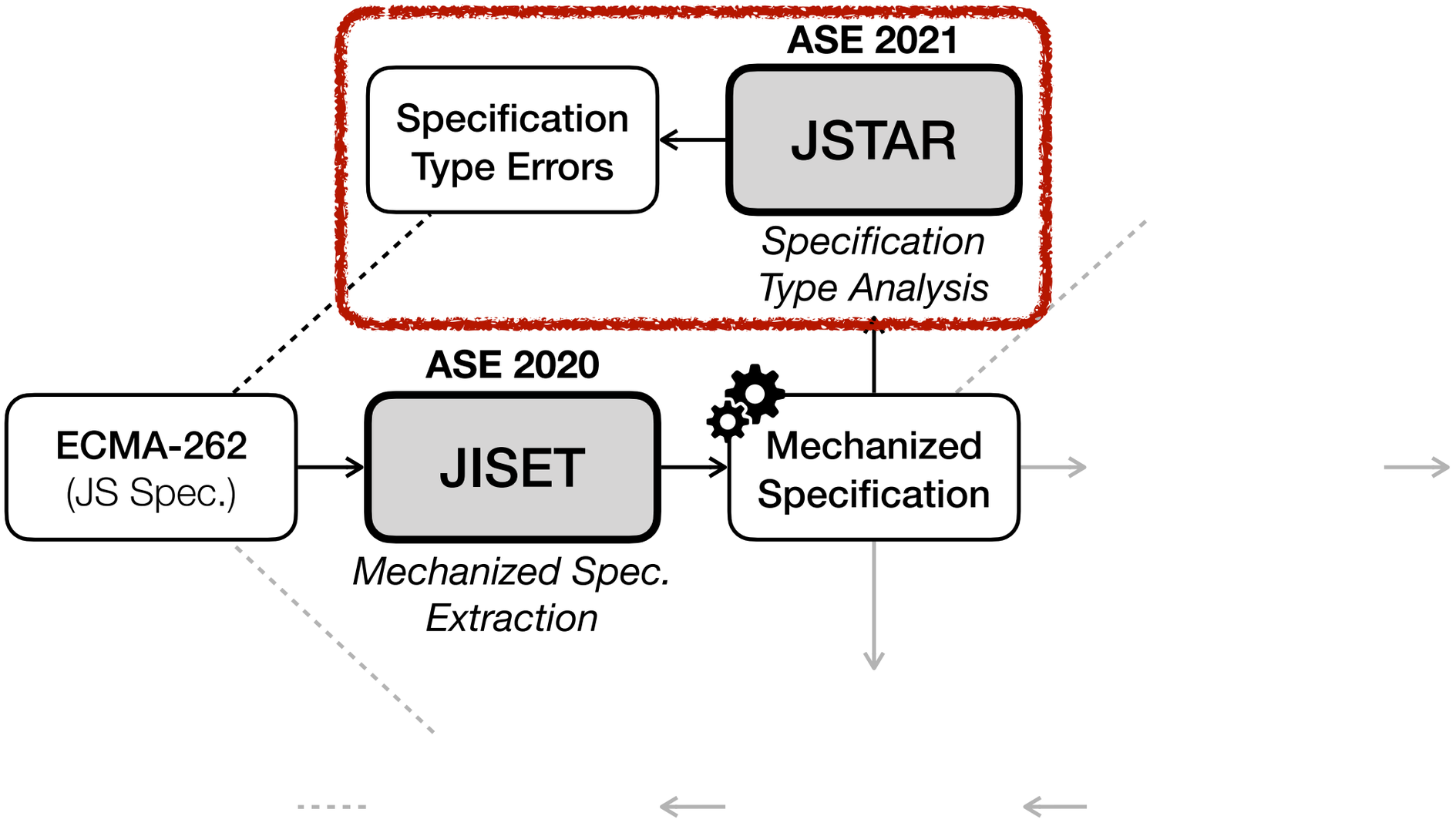
JS



```
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 - 명세 타입 분석

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* ≥ -0.5, return **-0**.

• • •

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

JSTAR - 명세 타입 분석

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 - 명세 타입 분석

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 - 명세 타입 분석

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 - 명세 타입 분석

20.3.2.28 Math.round (*x*)

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

n : Number

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 - 명세 타입 분석

20.3.2.28 Math.round (x)

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

n : Number

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 ≥ -0.5, return -0.

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

• • •

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

JSTAR - 명세 타입 분석

20.3.2.28 Math.round (x)

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

n : Number

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 - 명세 타입 분석

20.3.2.28 Math.round (x)

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

n : Number

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 - 실험 결과

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

59.2% Precision

93 Errors Detected

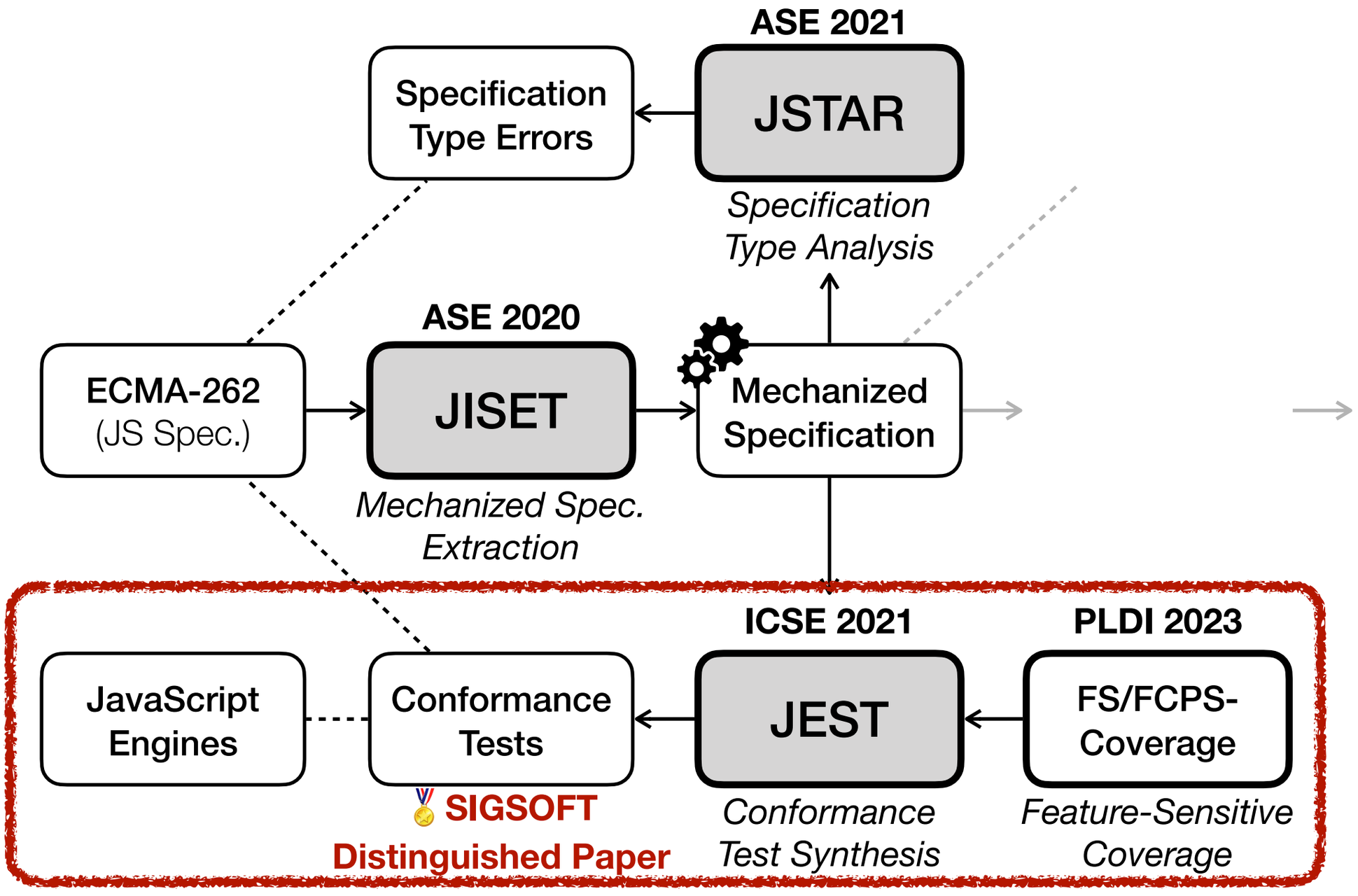
Checker	Bug Kind	Precision = (# True Bugs) / (# Detected Bugs)					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25	/ -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

[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”



자바스크립트 일치성 테스트

ECMA-262
(JavaScript Spec.)

Conformance

GraalVM

QuickJS

moddable

JavaScript Engines

자바스크립트 일치성 테스트

ECMA-262
(JavaScript Spec.)

How?

Conformance

GraalVM

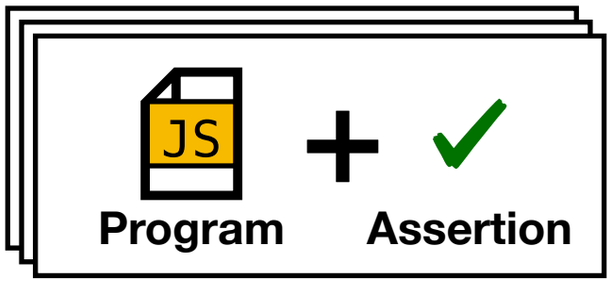
QuickJS

moddable

JavaScript Engines

자바스크립트 일치성 테스트

ECMA-262
(JavaScript Spec.)



Conformance Tests

Conformance

Test262
(Official Test Suite)

GraalVM

QuickJS

moddable

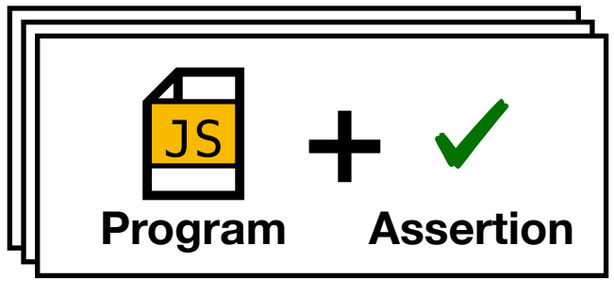
JavaScript Engines

자바스크립트 일치성 테스트

ECMA-262
(JavaScript Spec.)

Example: 4 + 2n JS

Assertion
TypeError



Conformance Tests

Conformance

Test262
(Official Test Suite)

GraalVM

QuickJS

moddable

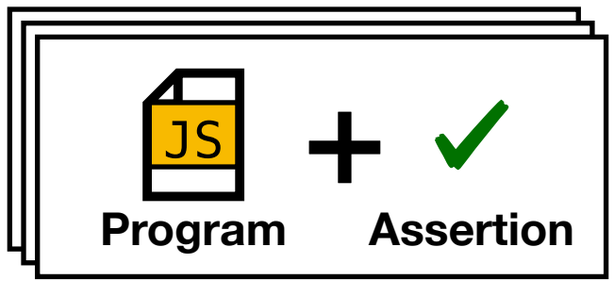
JavaScript Engines

문제 - 일치성 테스트 수동 작성

ECMA-262
(JavaScript Spec.)

Example: `4 + 2n` JS

Assertion: `TypeError`



Conformance Tests

Conformance

Test262
(Official Test Suite)

`Manual`

GraalVM

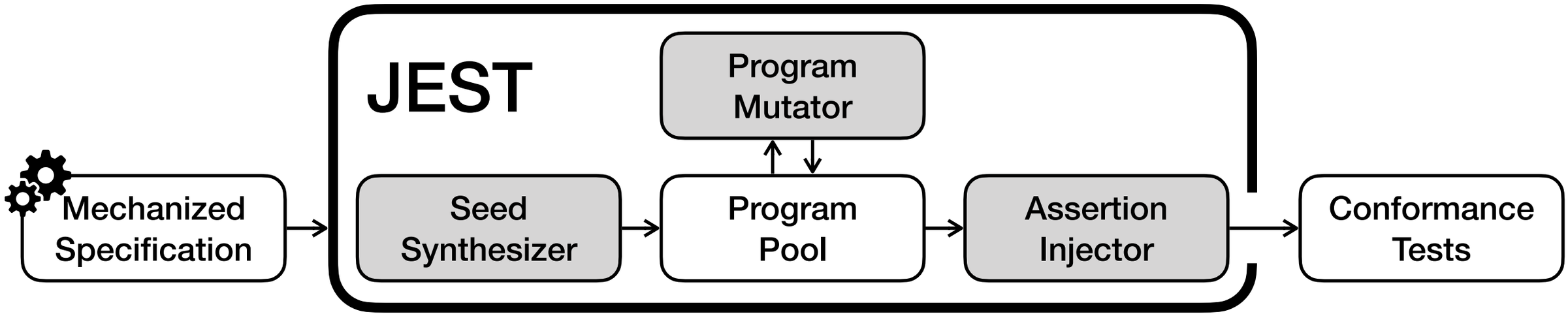
`QuickJS`

moddable

JavaScript Engines

JEST

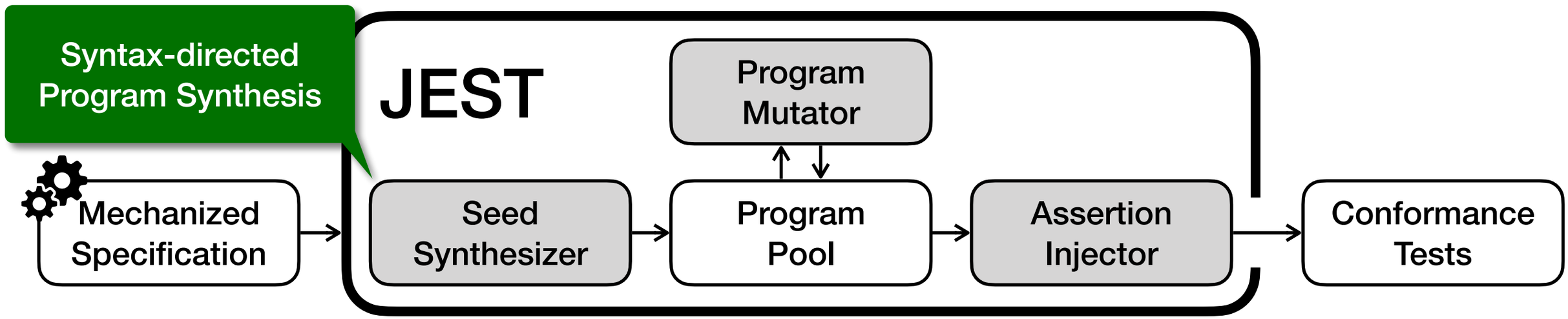
(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

JEST

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

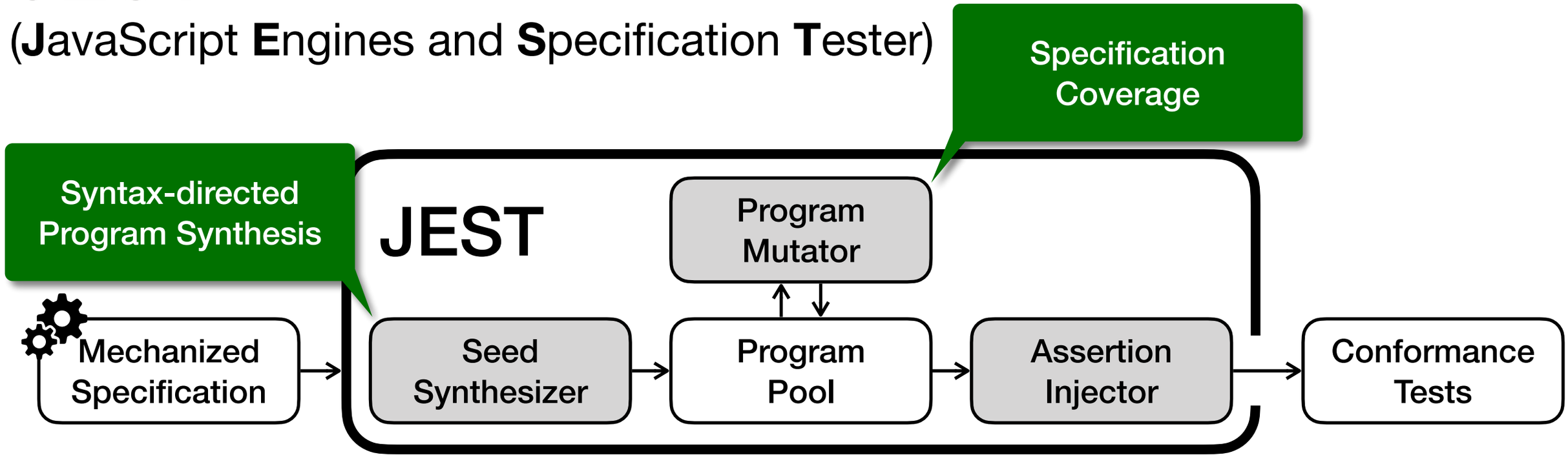


Program Pool

```
let x = 42;
```

JEST

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



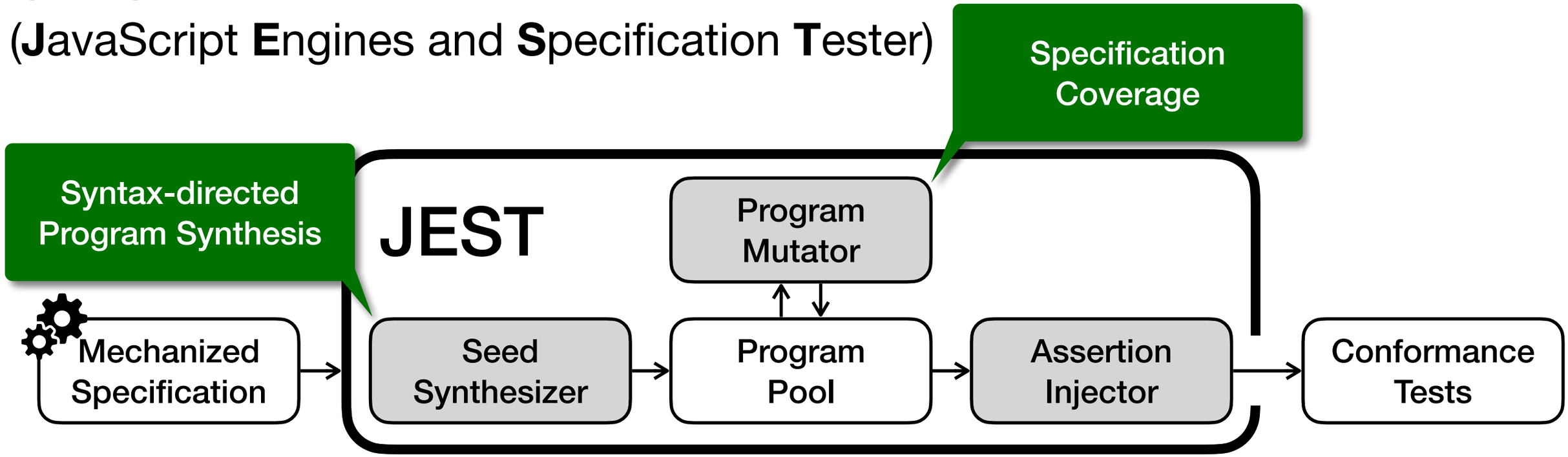
Program Pool

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

```
let x = 42;
```

JEST

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



Program Pool

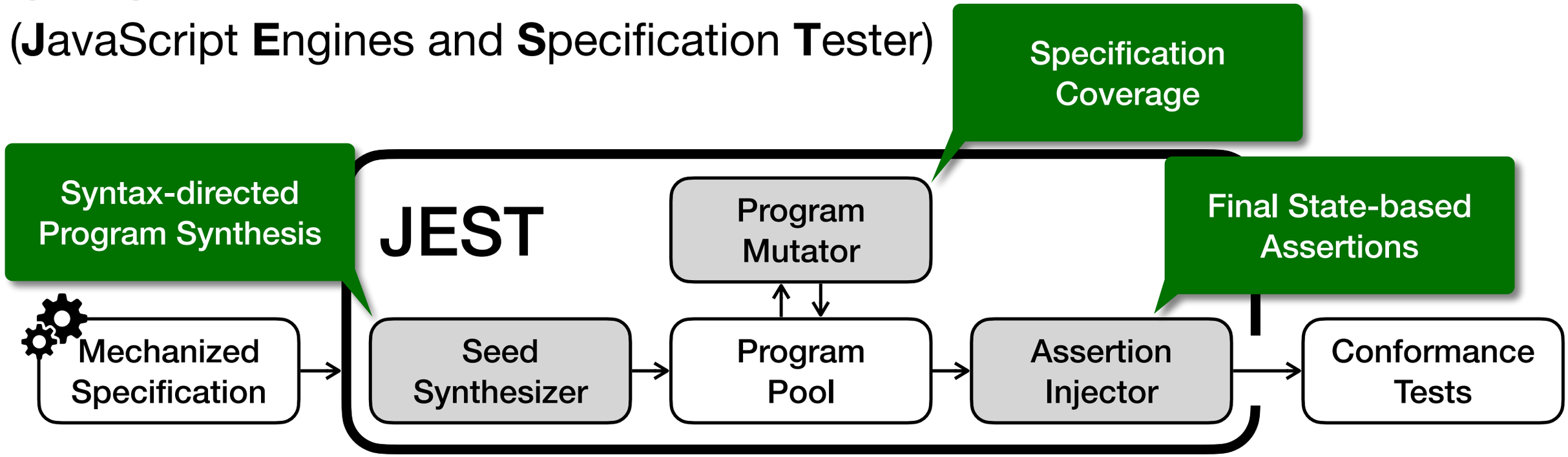
```
let x = 1 + 2;
```

```
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*)

...

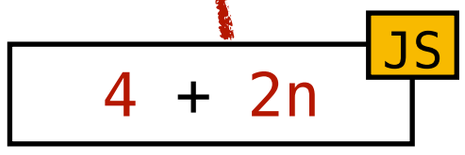
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then
   ...
7. Else,
   ...

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,
   …



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`

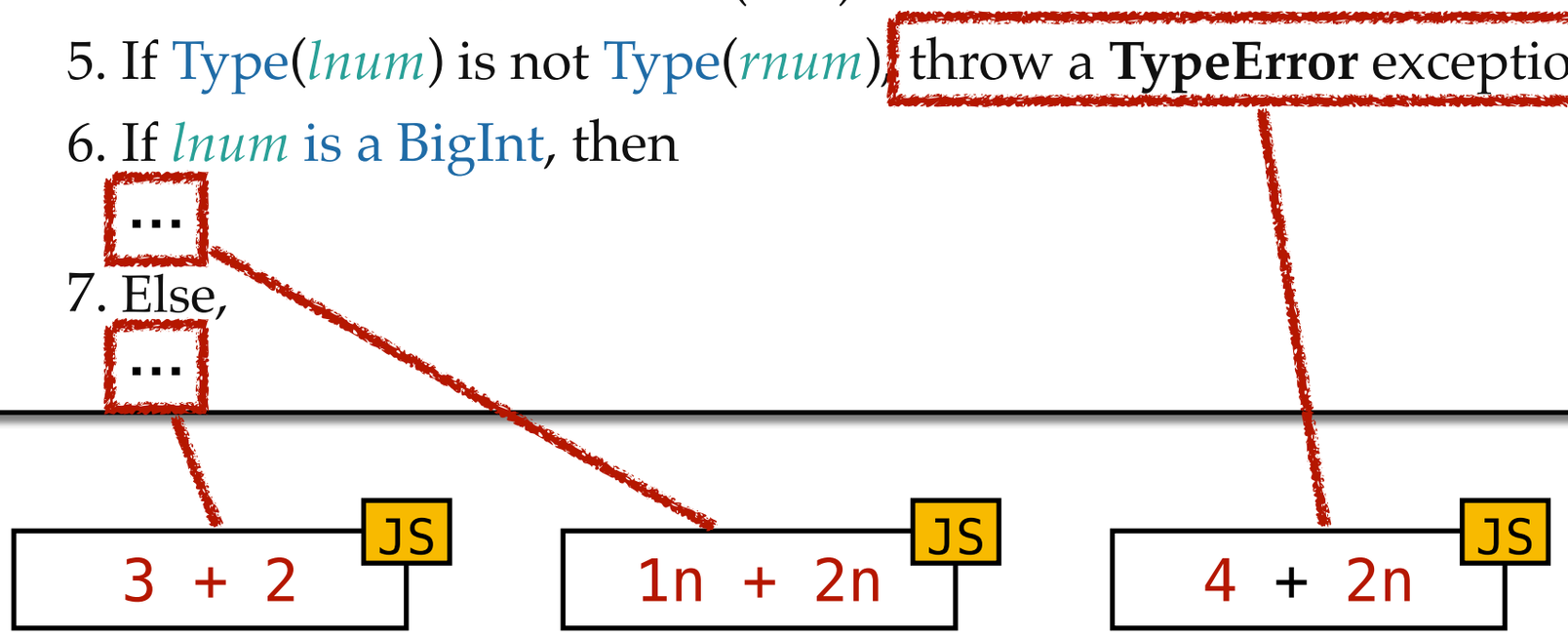
JS `4 + 2n`

JEST - Specification Coverage

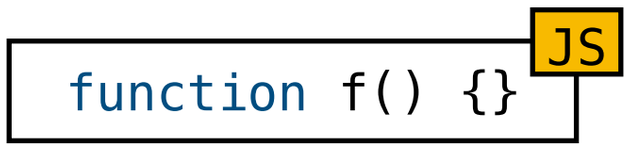
ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

…

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,
   …

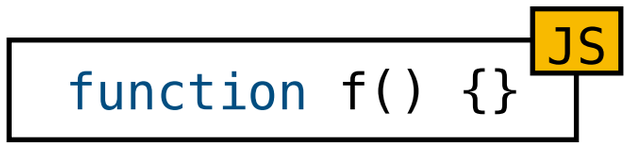


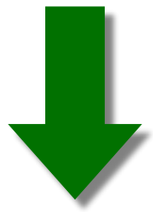
JEST - Final State-based Assertion Injection




```
  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

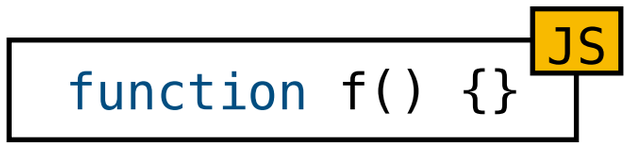


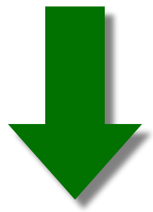


Prototype Chain

```
  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



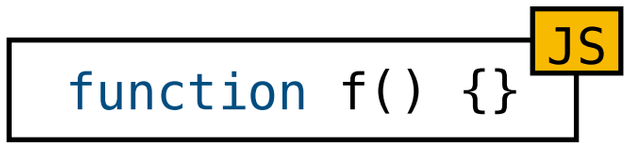


```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

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

+ ...
```

Prototype Chain

Property Descriptor

JEST - Final State-based Assertion Injection



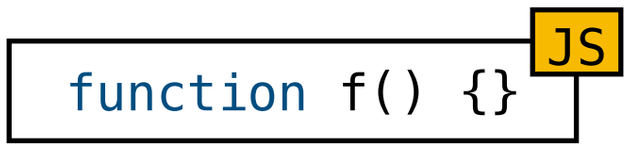

```
  function f() {}
+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);
+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $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.

JEST - 실험 결과

27 Bugs
In Spec.

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

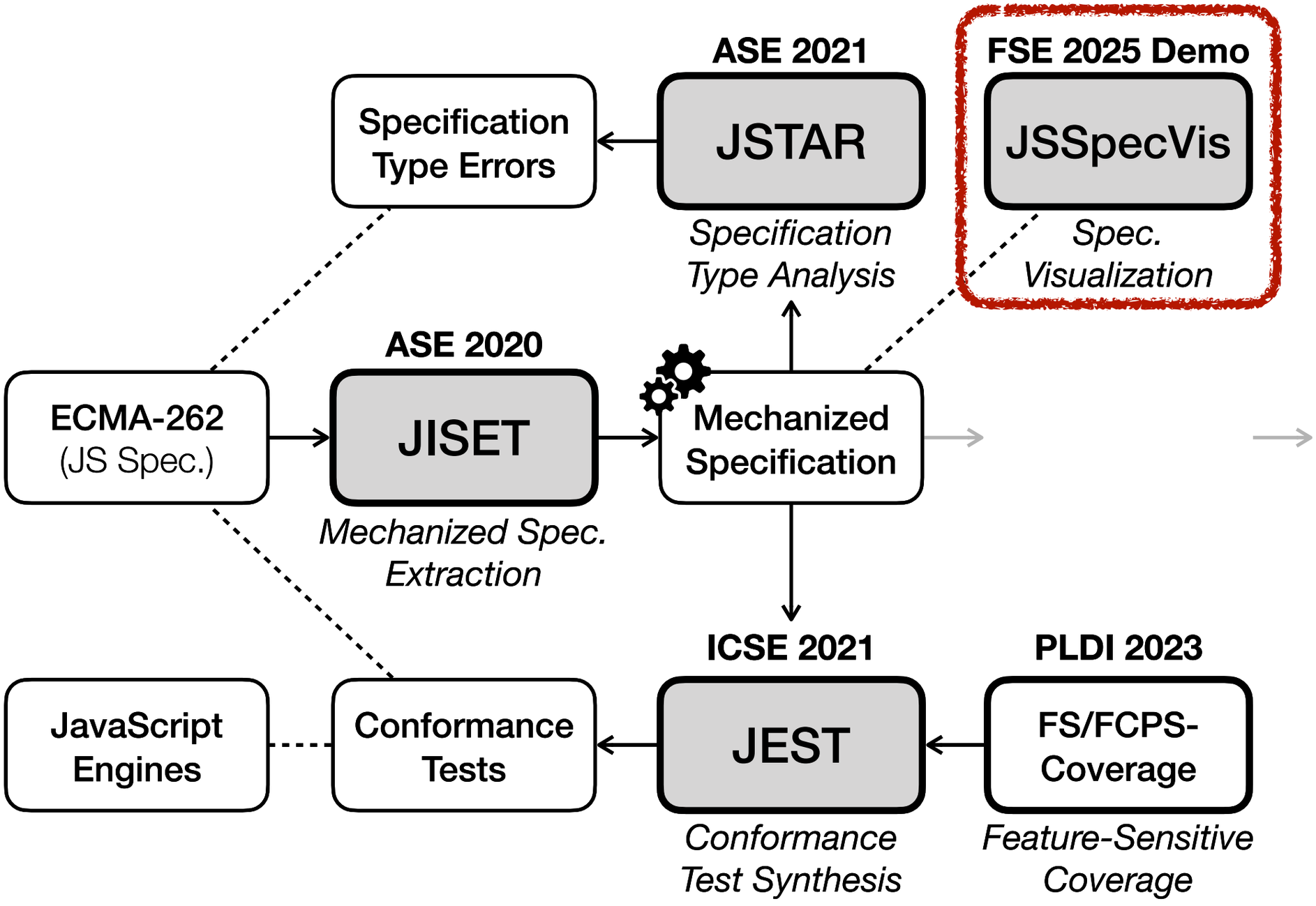
Name	Feature	#	Assertion	Known	Created	Resolved	Existed
ES11-1	Function	12	Key	O	2019-02-07	2020-04-11	429 days
ES11-2	Function	8	Key	O	2015-06-01	2020-04-11	1,776 days
ES11-3	Loop	1	Exc	O	2017-10-17	2020-04-30	926 days
ES11-4	Expression	4	Abort	O	2019-09-27	2020-04-23	209 days
ES11-5	Expression	1	Exc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

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 - 명세 부분별 일치성 테스트

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

1. If *opText* is **+**, then
    a. Let *lprim* be ? ToPrimitive(*lval*).
    b. Let *rprim* be ? ToPrimitive(*rval*).
    c. If *lprim* is a String or *rprim* is a String, then
        i. Let *lstr* be ? ToString(*lprim*).

Program Run on Double Debugger ▷

```
1 0 + { [ Symbol . toPrimitive ] : 0 } ;
```

Test262 7 found Download All

built-ins/String/prototype/valueOf/non-generic.js

language/expressions/addition/S11.6.1_A2.2_T1.js

language/expressions/addition/bigint-toprimitive.js

CallPath Clear

#	name	step
0	**13.15.4 EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)**	5
1	*AdditiveExpression* : *AdditiveExpression* + *MultiplicativeExpression*	1

JSSpecVis - 명세 기반 자바스크립트 실행

ESMeta Double Debugger Playground ✓ READY ES2024 (0B24A049)

RUN | QUIT | SPEC STEP | OVER | OUT | CONTINUE | SPEC BACK | BACK OVER | BACK OUT | REWIND | JS STEP | OVER | OUT

JavaScript Editor

```
1 0 + { [Symbol.toPrimitive]: 0 };
```

ECMAScript Specification

ApplyStringOrNumericBinaryOperator(lval, opText, rval)

1. If *opText* is +, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).
   c. If *lprim* is a String or *rprim* is a String, then
      i. Let *lstr* be ? ToString(*lprim*).
      ii. Let *rstr* be ? ToString(*rprim*).
      iii. Return the string-concatenation of *lstr* and *rstr*.
   d. Set *lval* to *lprim*.
   e. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
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
   a. If *opText* is **, return ? BigInt::exponentiate(*lnum*,

State Viewer — Env | Heap | Breaks | Callstack

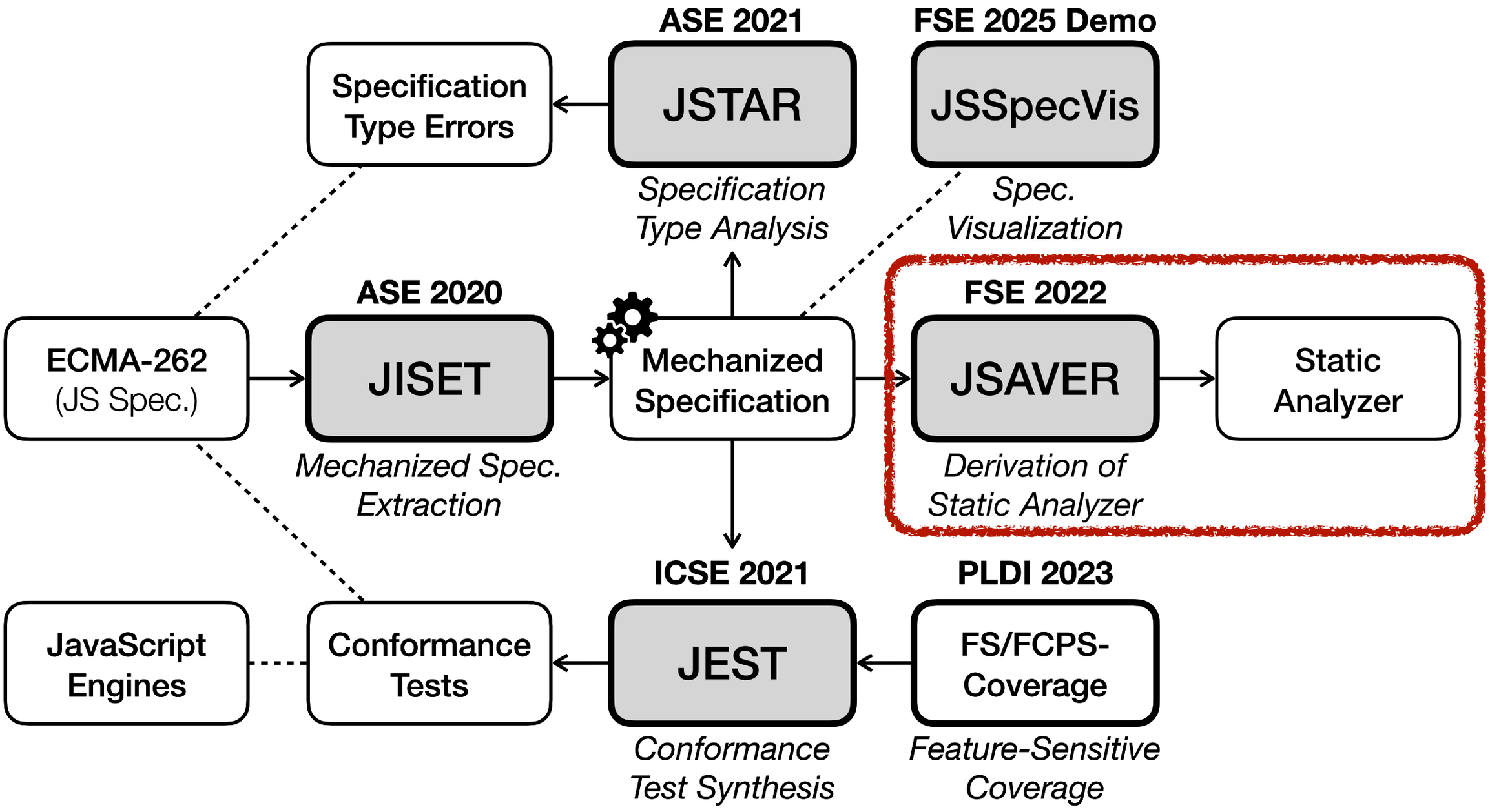
Specification Environment

- **lprim** : 0
- **lval** : 0
- **opText** : "+"
- **rval** : Record[Object]

JavaScript Environment

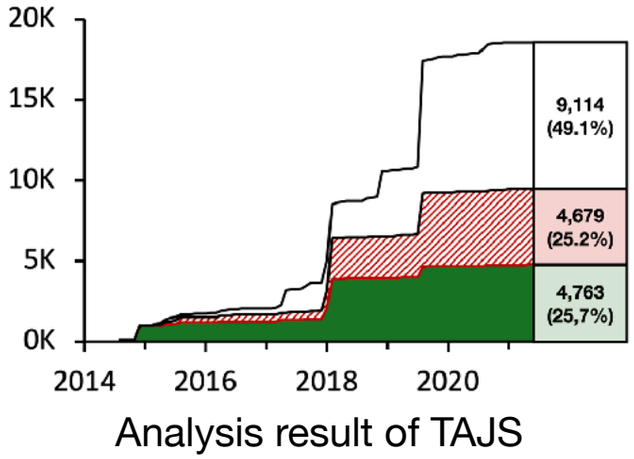
- **AggregateError** : Record[PropertyDescriptor]
  - **Value** : Record[BuiltinFunctionObject]
  - **Writable** : true
  - **Enumerable** : false
  - **Configurable** : true
- **Array** : Record[PropertyDescriptor]
  - **Value** : Record[BuiltinFunctionObject]
  - **Writable** : true
  - **Enumerable** : false
  - **Configurable** : true
- **ArrayBuffer** : Record[PropertyDescriptor]
  - **Value** : Yet[ArrayBuffer]
  - **Writable** : true
  - **Enumerable** : false

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

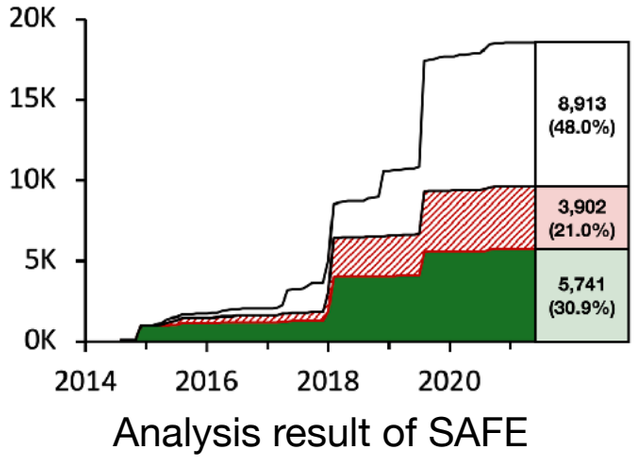


JSAVER - 정적 분석기 자동 생성

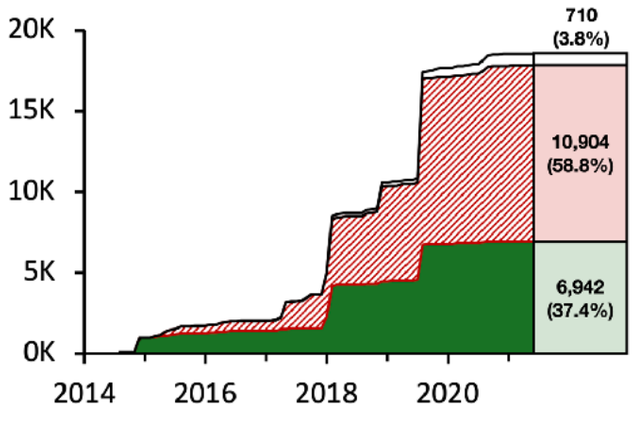
Manual



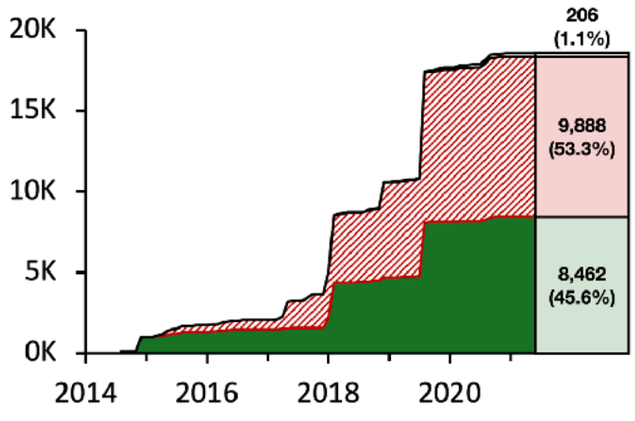
Analysis result of TAJS



Analysis result of SAFE

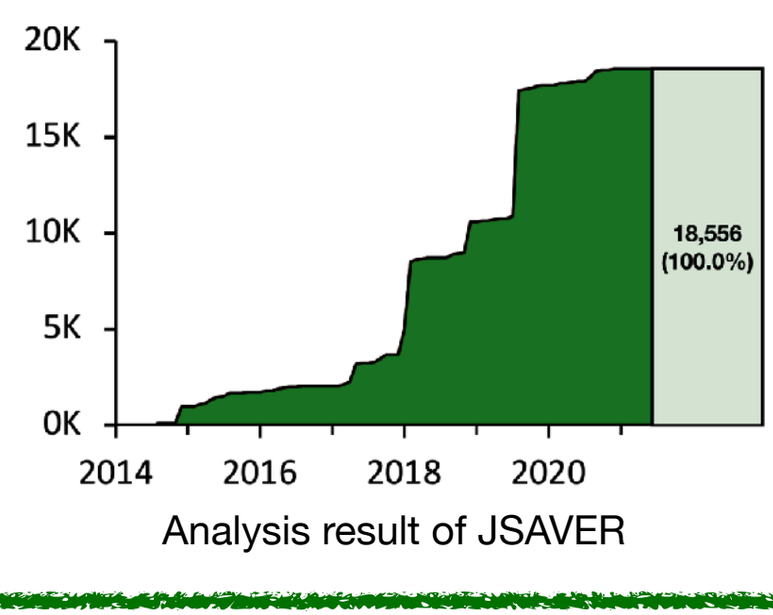


Analysis result of TAJS with Babel

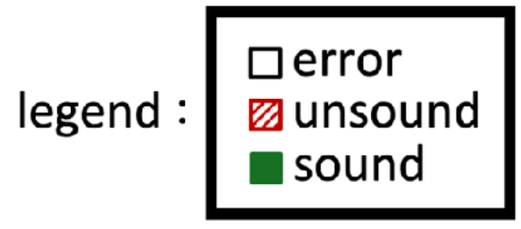


Analysis result of SAFE with Babel

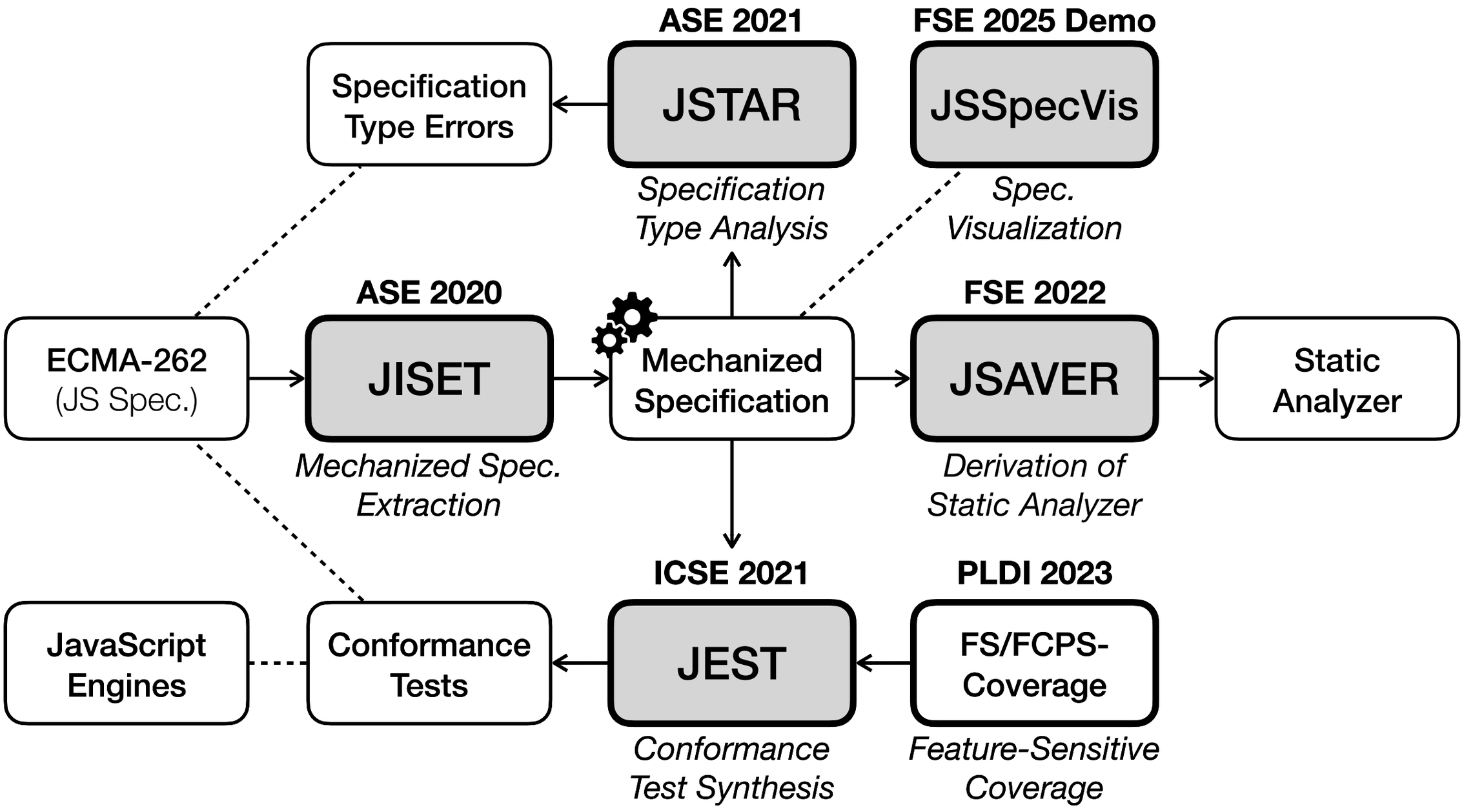
Generated

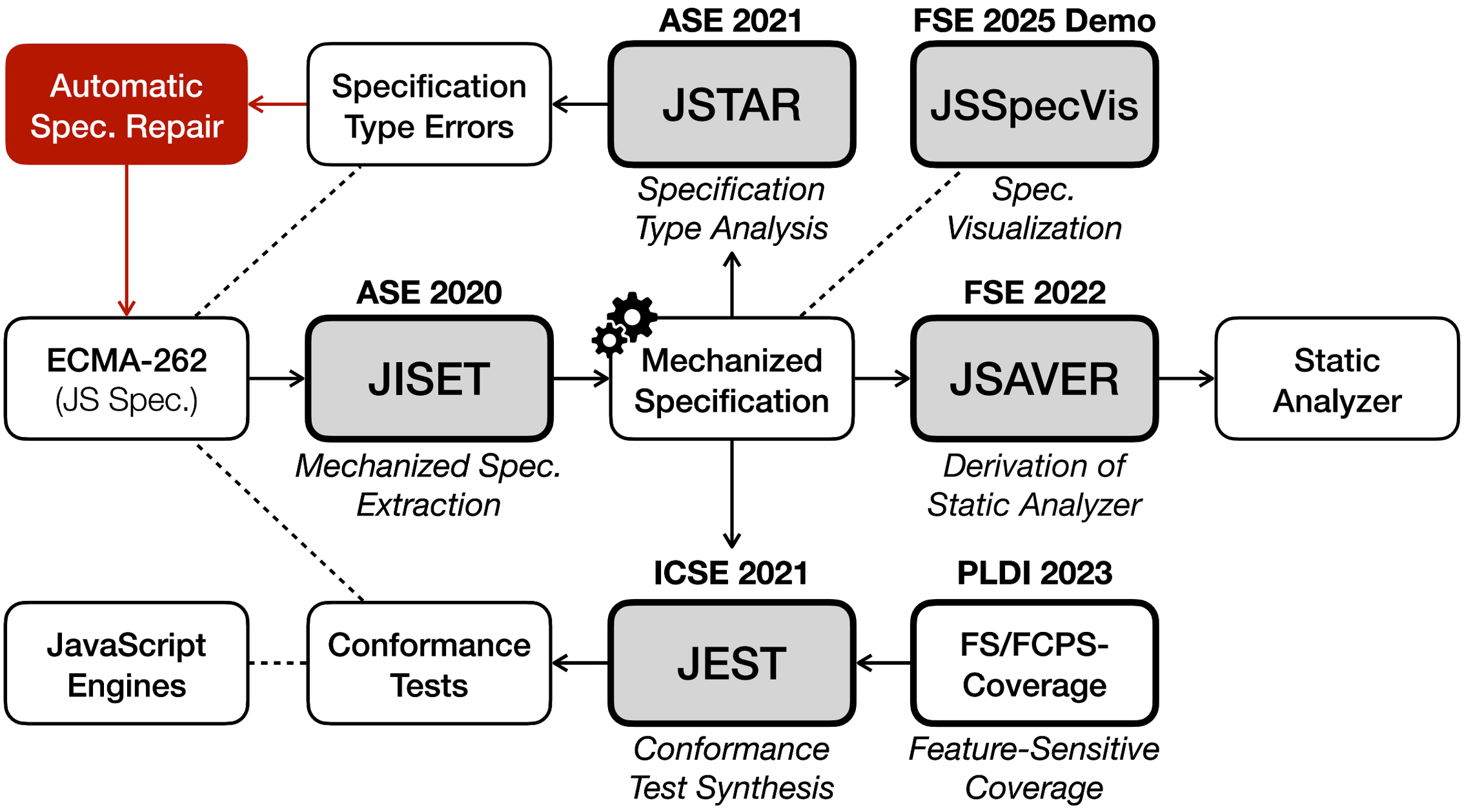


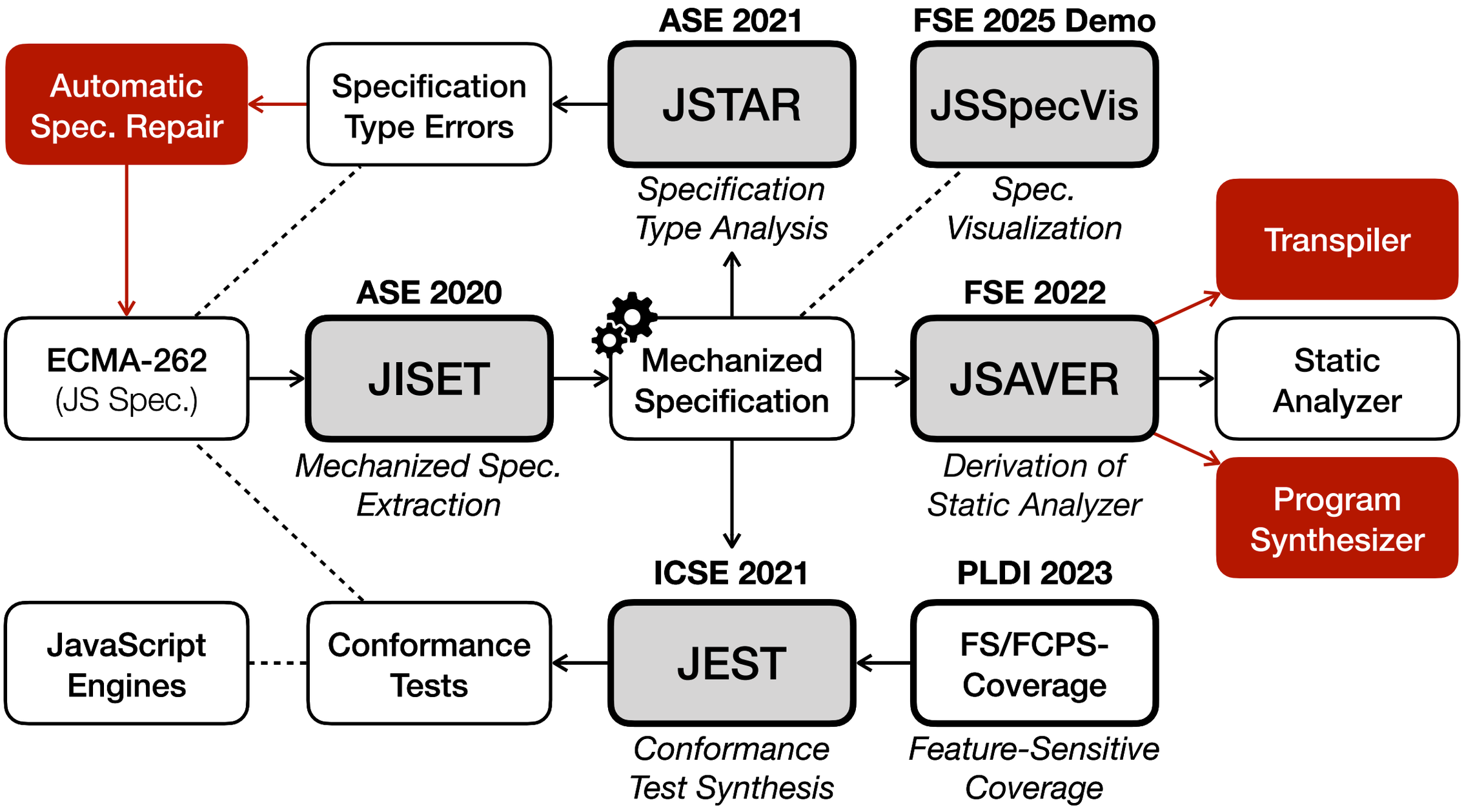
Analysis result of JSAVER

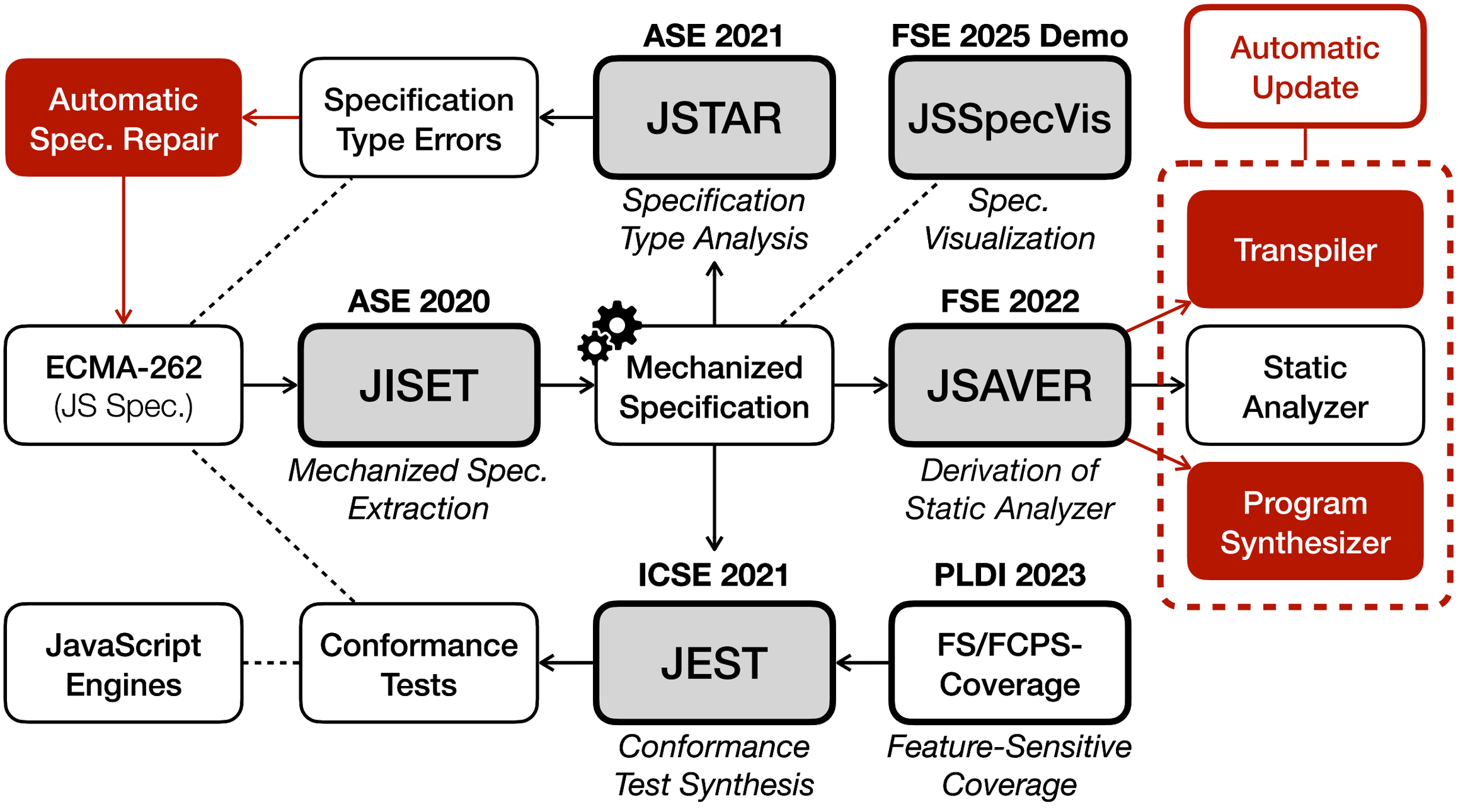


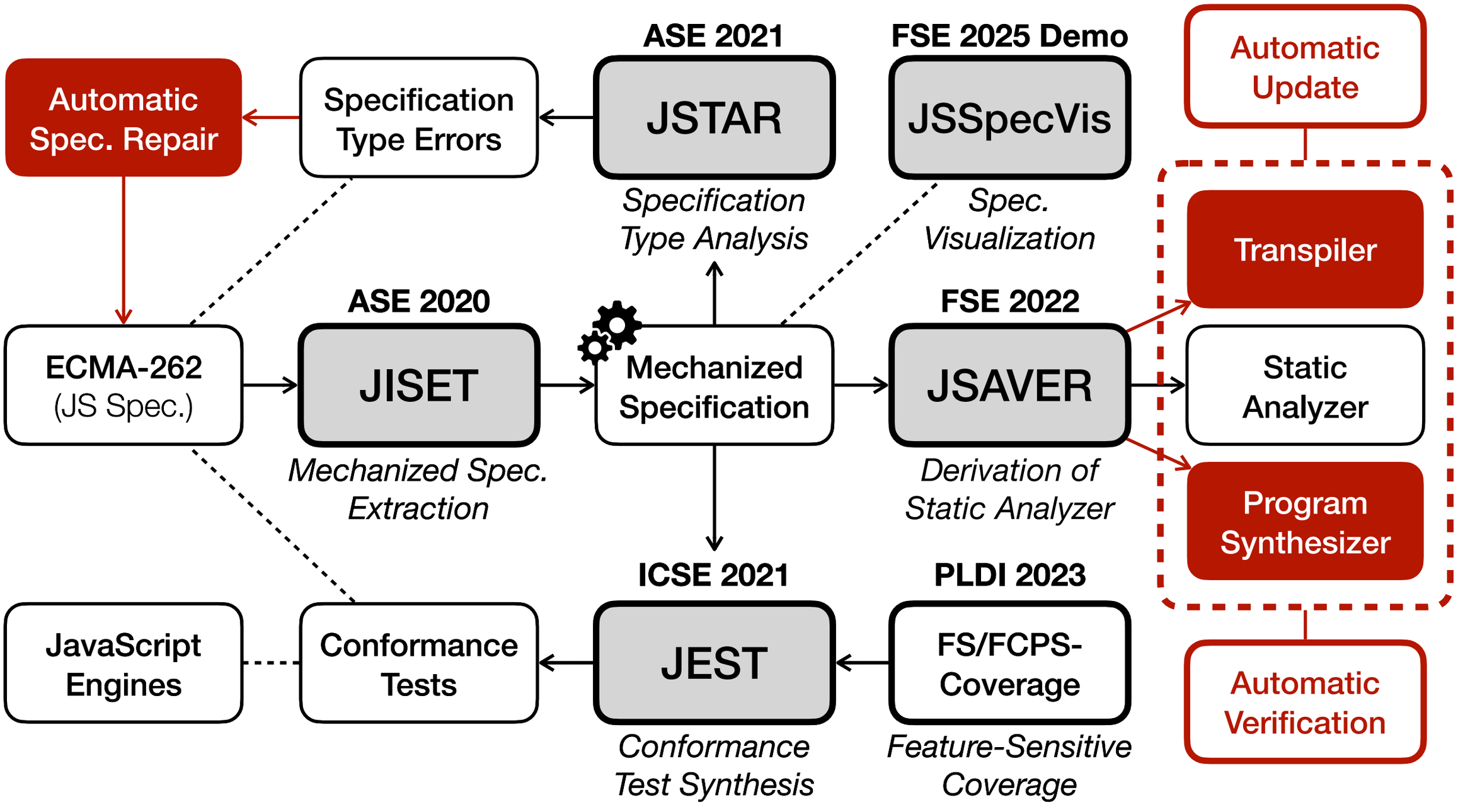
x-axis : creation time (year)
y-axis : # tests


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

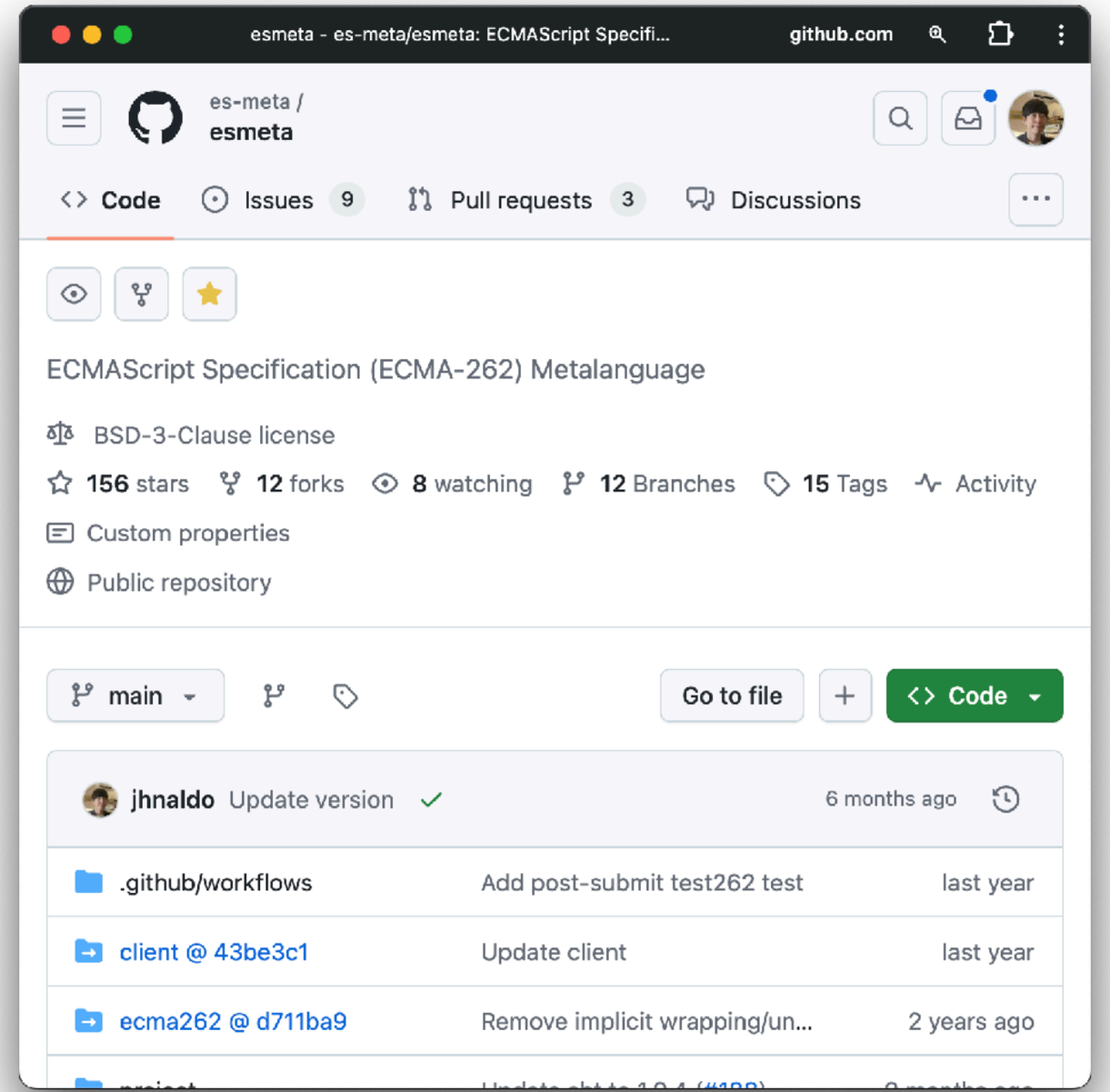

ASE 2021
FSE 2025 Demo
Automatic Spec. Repair
Specification Type Errors
JSTAR
JSSpecVis
Specification Type Analysis
Spec. Visualization
ASE 2020
FSE 2022
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
ICSE 2021
PLDI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


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


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


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

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



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

ESMETA

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