KOREA UNIVERSITY

기계화 명세를 이용한 자바스크립트 언어의 설계와 구현

컴퓨터학과 2023년 가을학기 콜로퀴움

박지혁

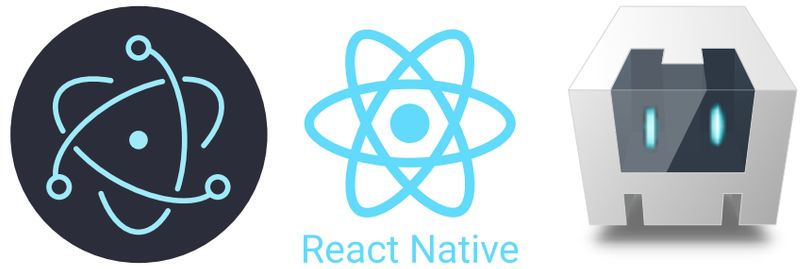
고려대학교 컴퓨터학과
프로그래밍 언어 연구실

2023. 10. 11

자바스크립트는 어디에나 있다

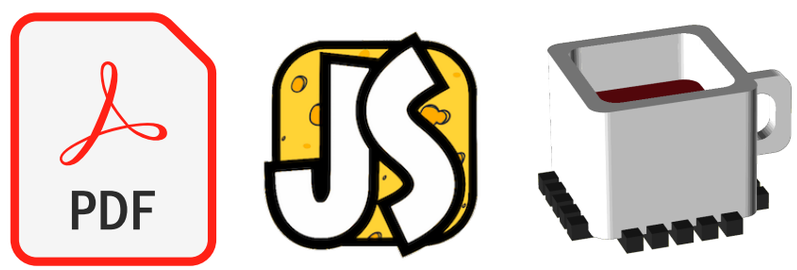
클라이언트-사이드 프로그래밍

JS



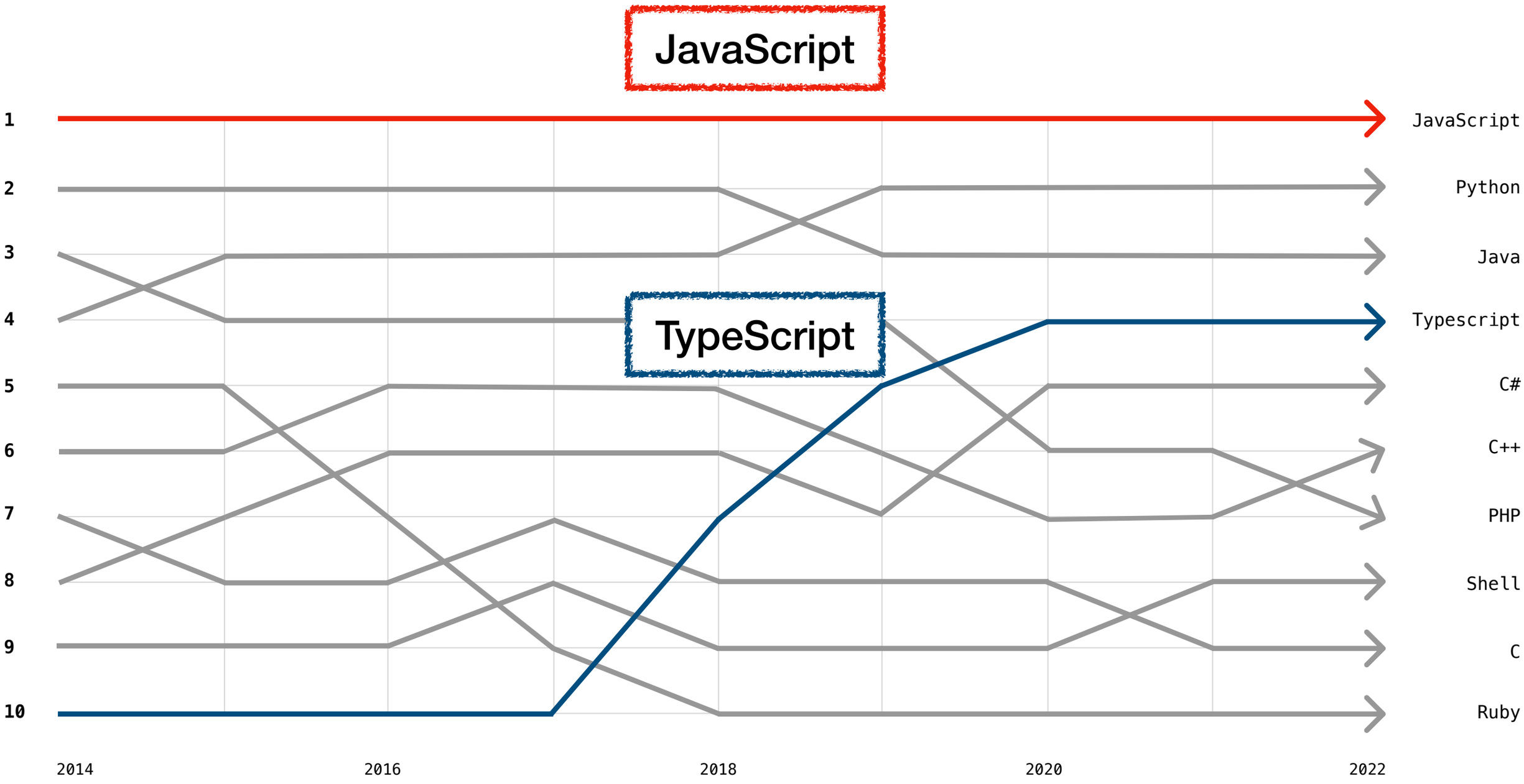
모바일/데스크톱 어플리케이션

서버-사이드 프로그래밍



그 외 (PDF, 사물 인터넷, 마이크로컨트롤러, etc.)

자바스크립트는 어디에나 있다



https://octoverse.github.com/

하지만, 자바스크립트는 복잡하다..

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

언제나 `false`를 반환할까?

NO!!

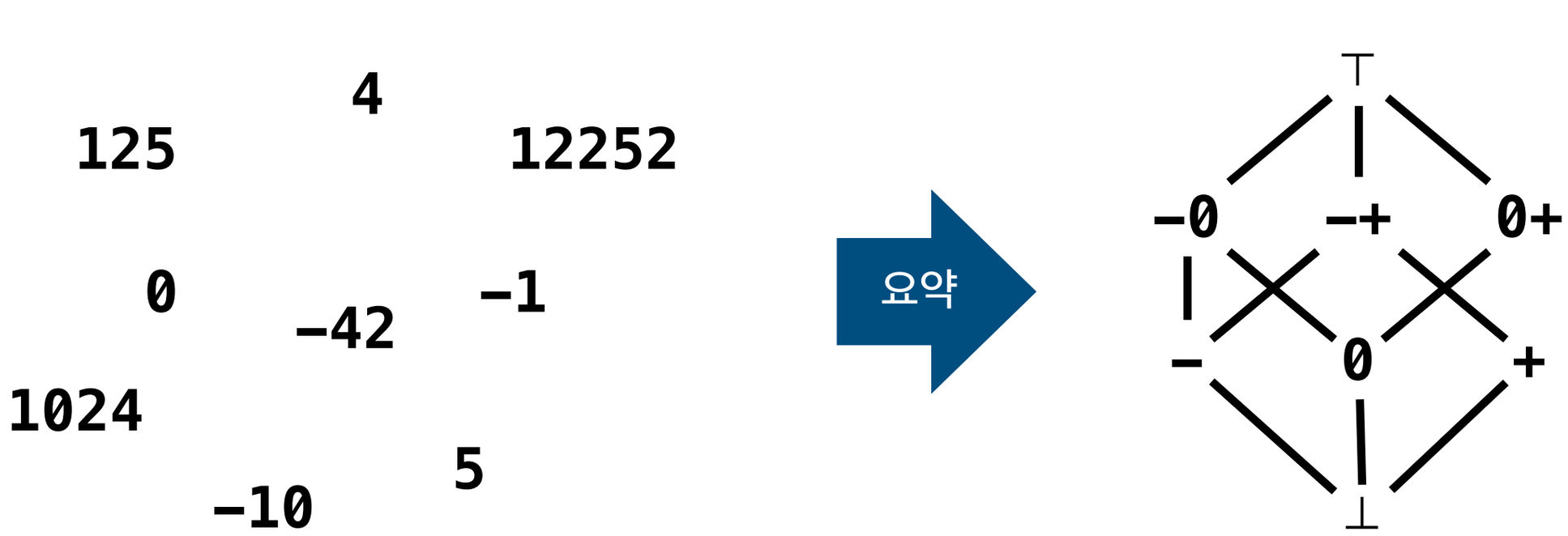
```
f([]) -> [] == ![]
      -> [] == false
      -> +[] == +false
      -> 0 == 0
      -> true
```

정적 분석 (Static Analysis)

- 주어진 프로그램을 **실행하지 않고 (정적으로)** 분석하는 기법

- 프로그램의 행동을 **요약해서 해석**

  예를 들어, 정수를 요약해보자



정적 분석 (Static Analysis)

```
function f(x) {
  // x == ⊤
  if (x == 0) {
    // x == 0
    return 0;
    // [RETURN] 0
  } else if (x < 0) {
    // x == -
    return -x;
    // [RETURN] +
  } else {
    // x == +
    return x;
    // [RETURN] +
  }
}// [RETURN] 0+
```

실제 실행

f(-4) = 4 **f(0) = 0**

. . . **f(-42) = 42**

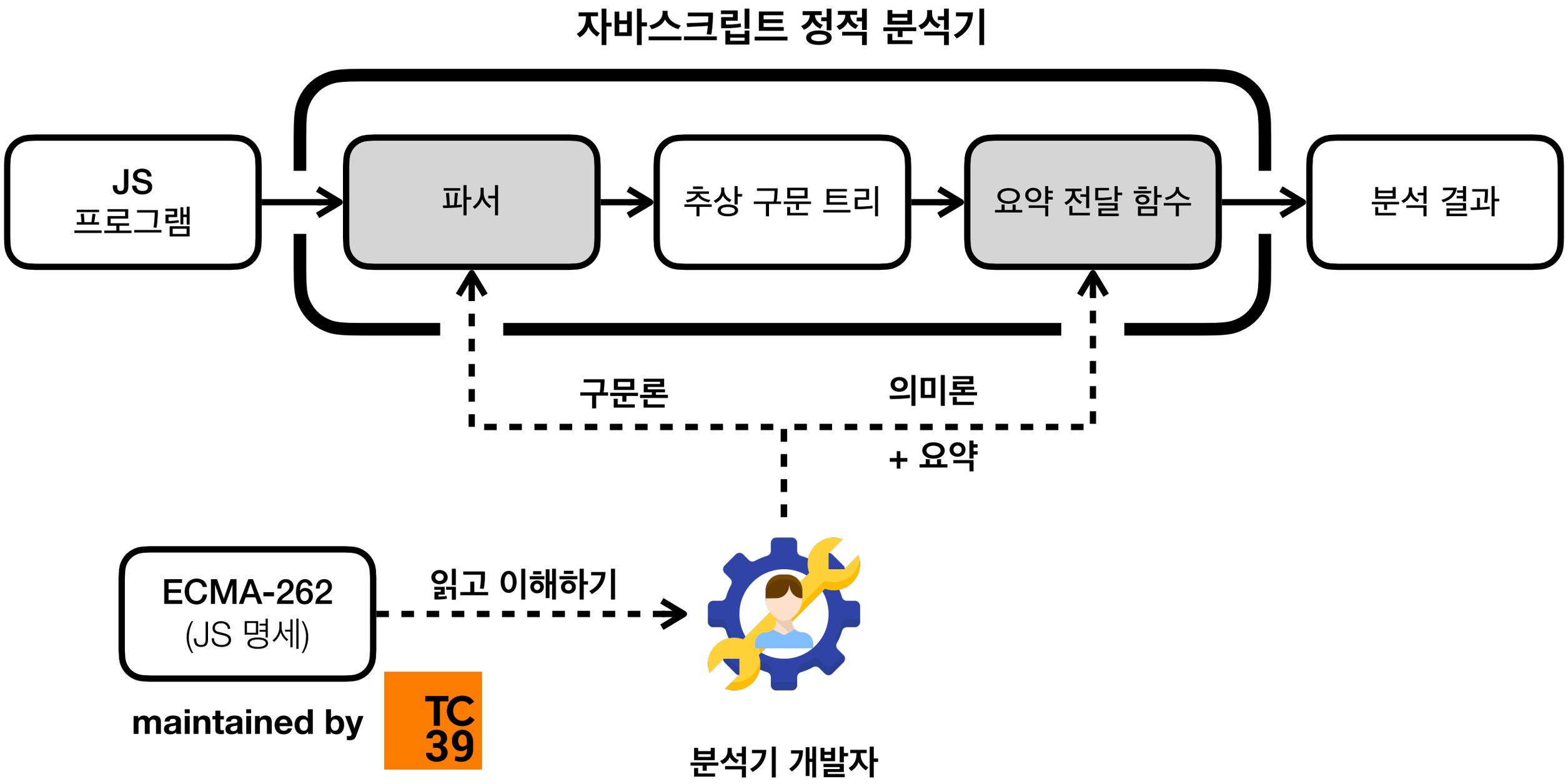
f(5) = 5 **. . .**

요약 해석

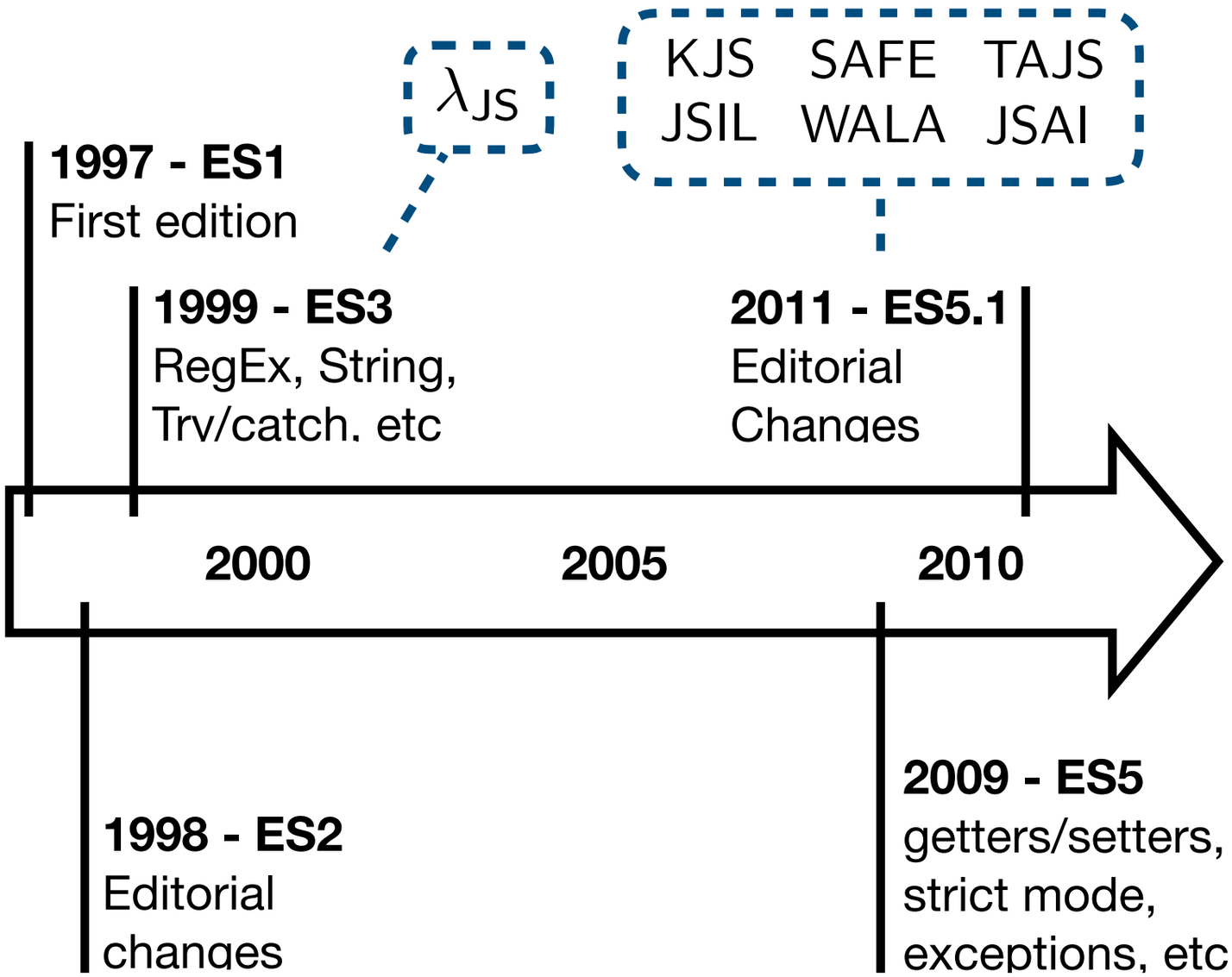
f(⊤) = 0+

타입 추론 (분석) ⊆

자바스크립트를 위한 정적 분석



문제: 빠르게 성장하는 자바스크립트



문제: 빠르게 성장하는 자바스크립트

TC 39

λ_{JS}

KJS SAFE TAJS
JSIL WALA JSAI

1997 - ES1
First edition

1998 - ES2
Editorial changes

1999 - ES3
RegEx, String, Try/catch, etc

2009 - ES5
getters/setters, strict mode, exceptions, etc

2011 - ES5.1
Editorial Changes

2015 - ES6
classes, modules, etc.

2016 - ES7
destructuring patterns, etc.

2017 - ES8
object manipulation, etc.

2018 - ES9

2019 - ES10

2020 - ES11

2021 - ES12

2022 - ES13

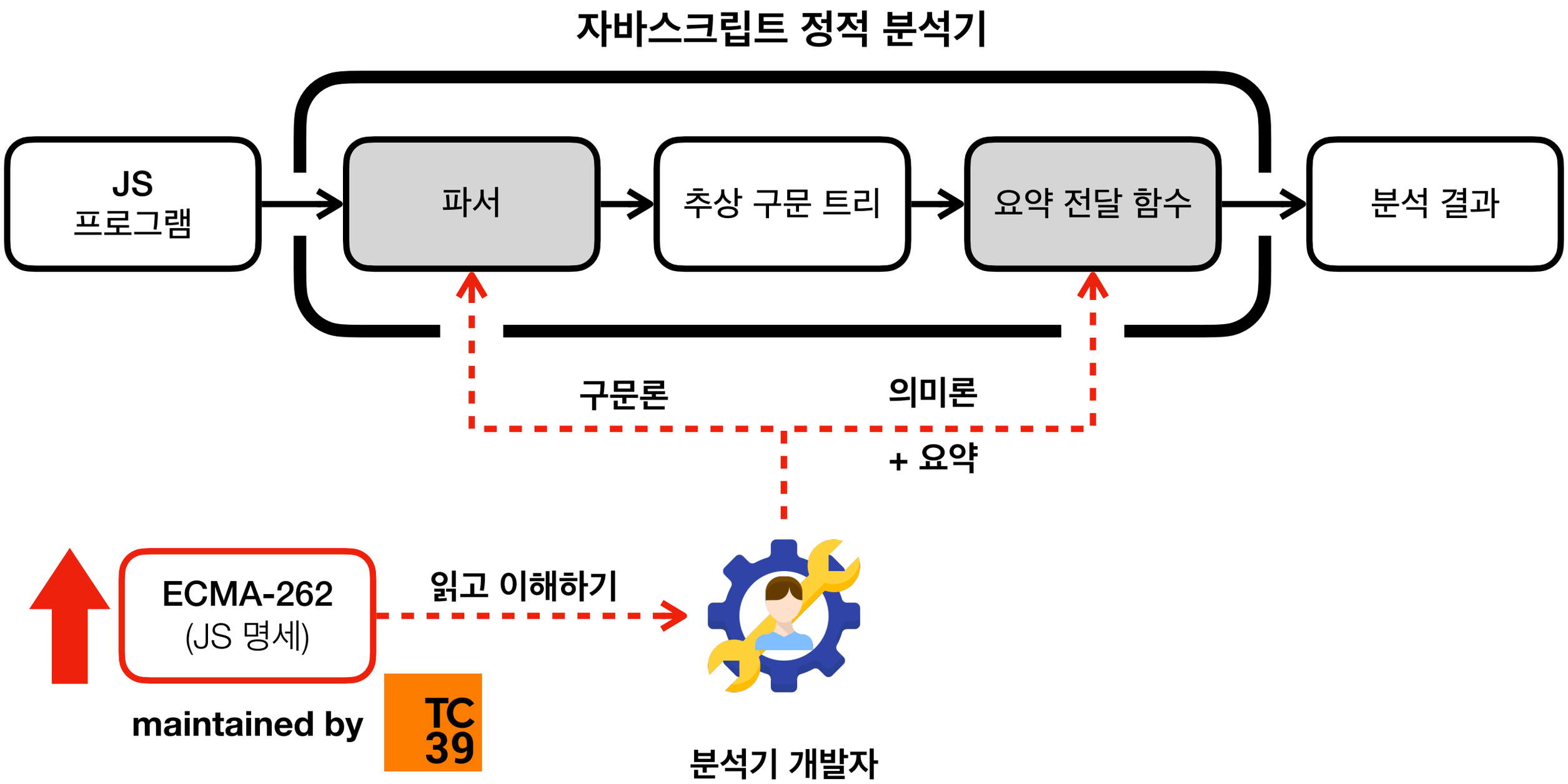
2023 - ES14

2000 2005 2010 2015 2020

ES2022 (ES13) - 840 pages

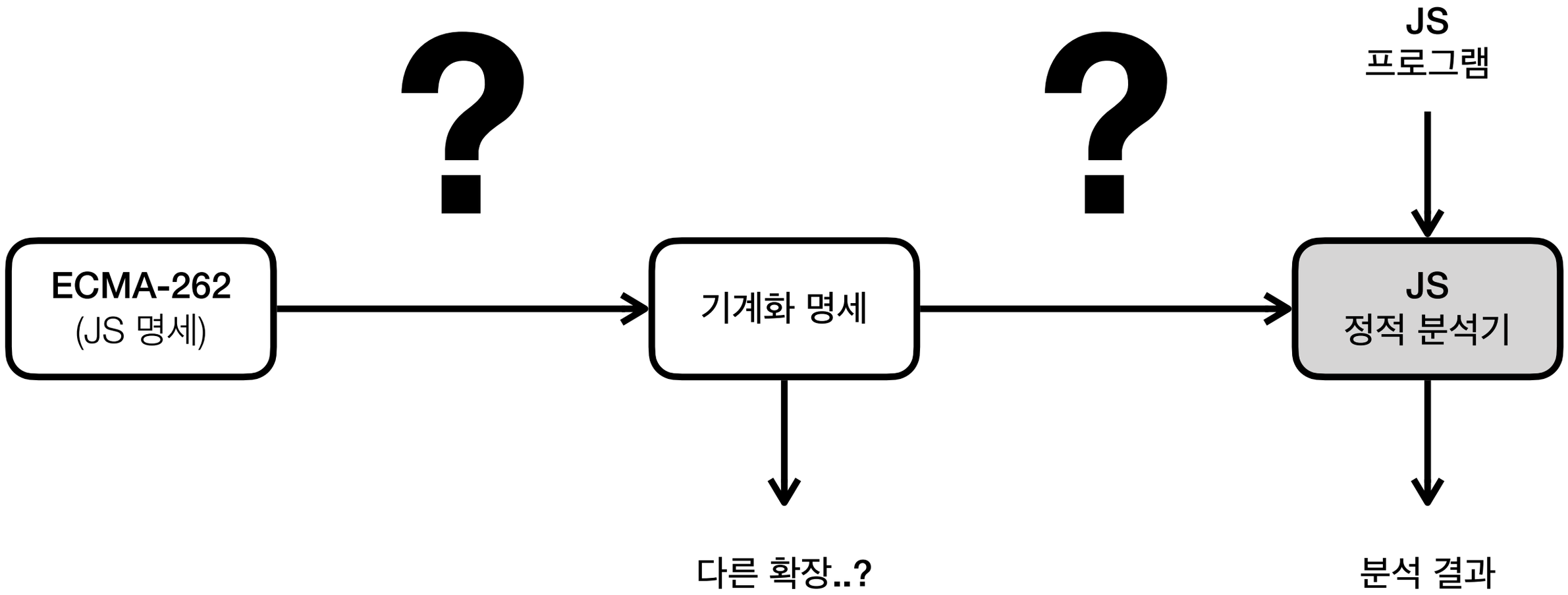
Annual Update

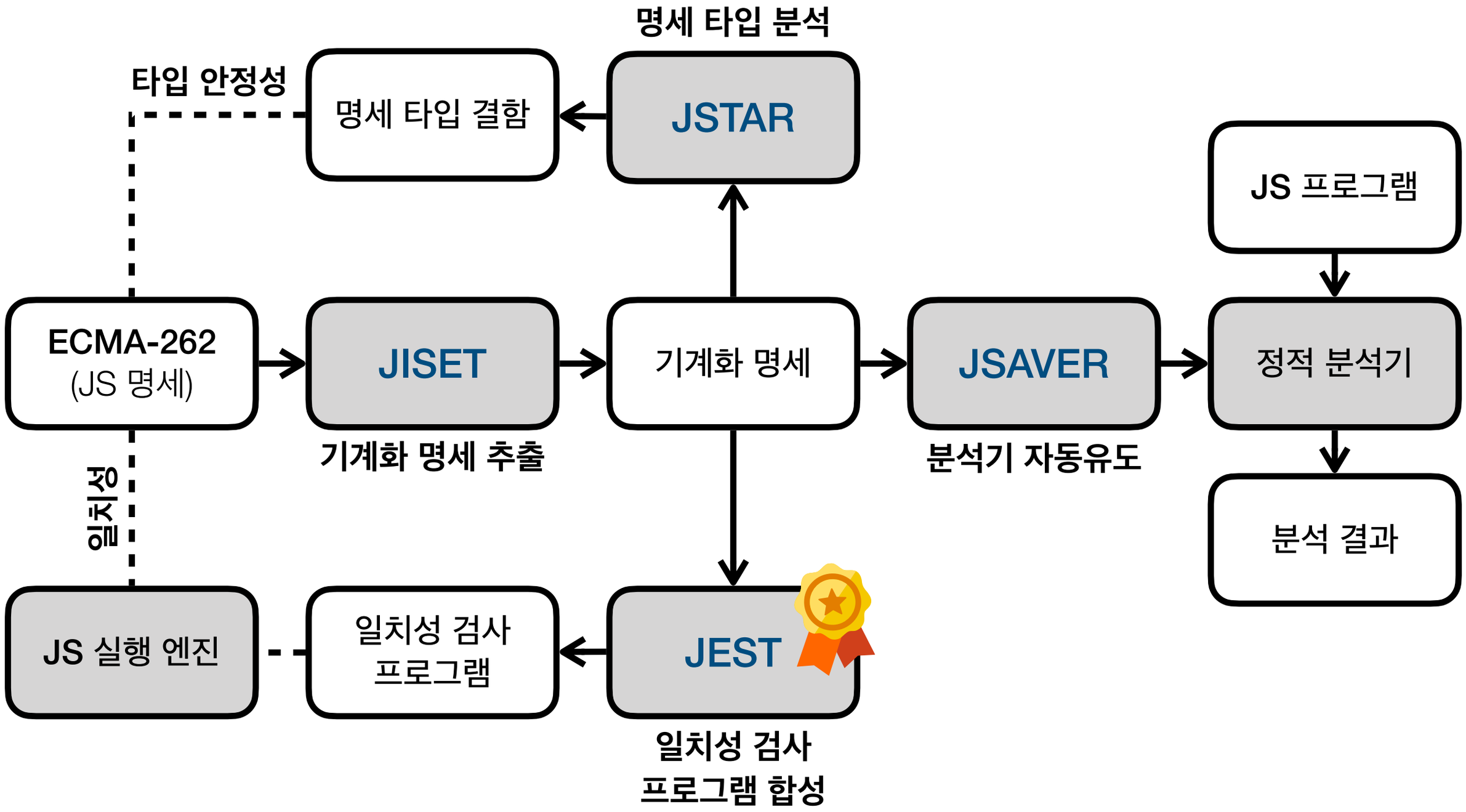
문제: 수동으로 분석기 갱신

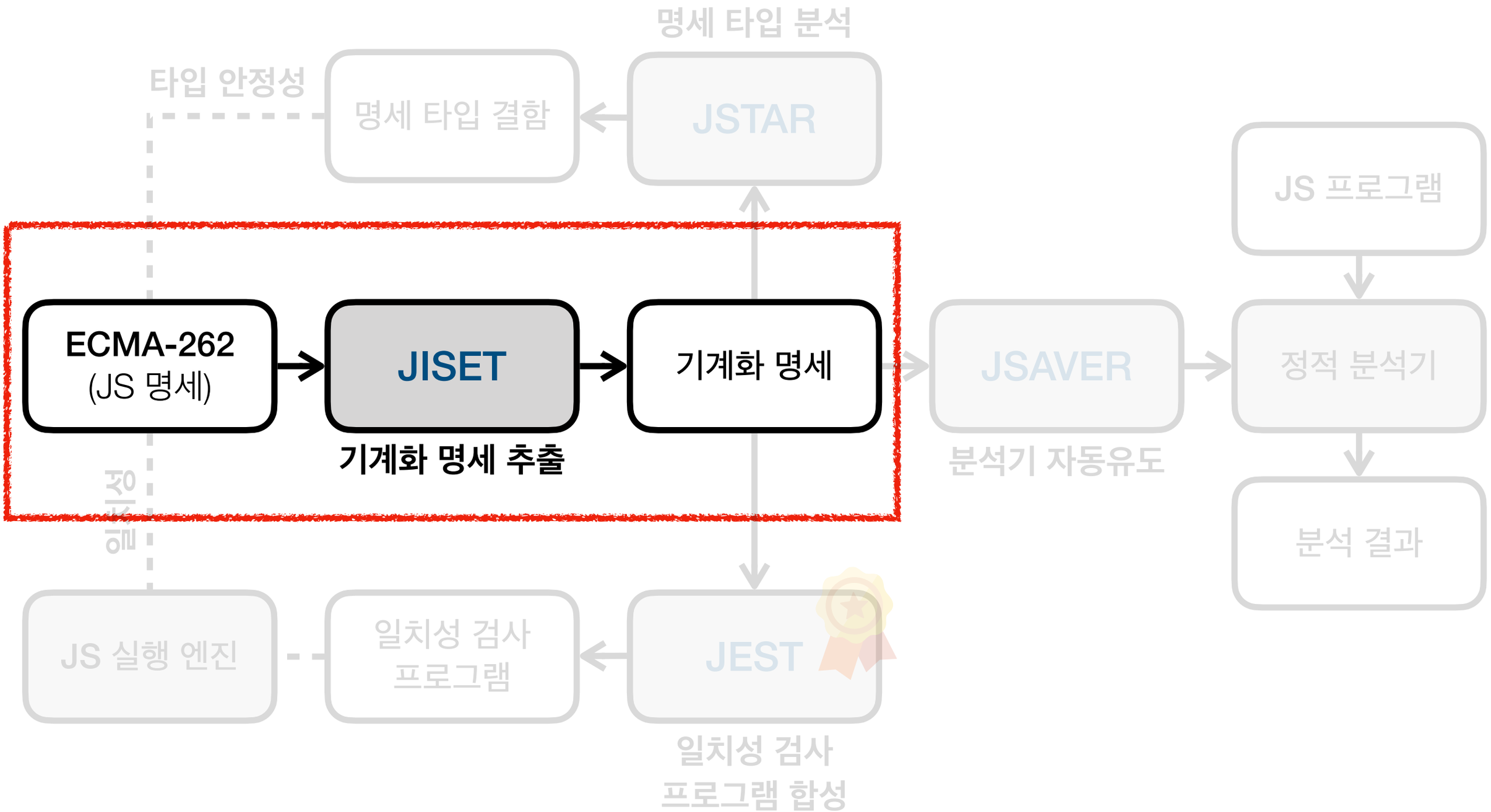


분석기를 자동으로 유도하기?

아이디어: 기계화 명세




명세 타입 분석
타입 안정성
명세 타입 결함
JSTAR
JS 프로그램
ECMA-262
(JS 명세)
JISET
기계화 명세
JSAVER
정적 분석기
기계화 명세 추출
분석기 자동유도
분석 결과
일치성
JS 실행 엔진
일치성 검사
프로그램
JEST
일치성 검사
프로그램 합성



[ASE'20] J. Park, et al. "JISET: JavaScript IR-based Semantics Extraction Toolchain"

ECMA-262 (자바스크립트 명세)

ArrayLiteral[Yield, Await] :

[*Elision*opt]

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

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

구문론

TC 39

The production of *ArrayLiteral* in ES13

13.2.5.2 Runtime Semantics: Evaluation

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

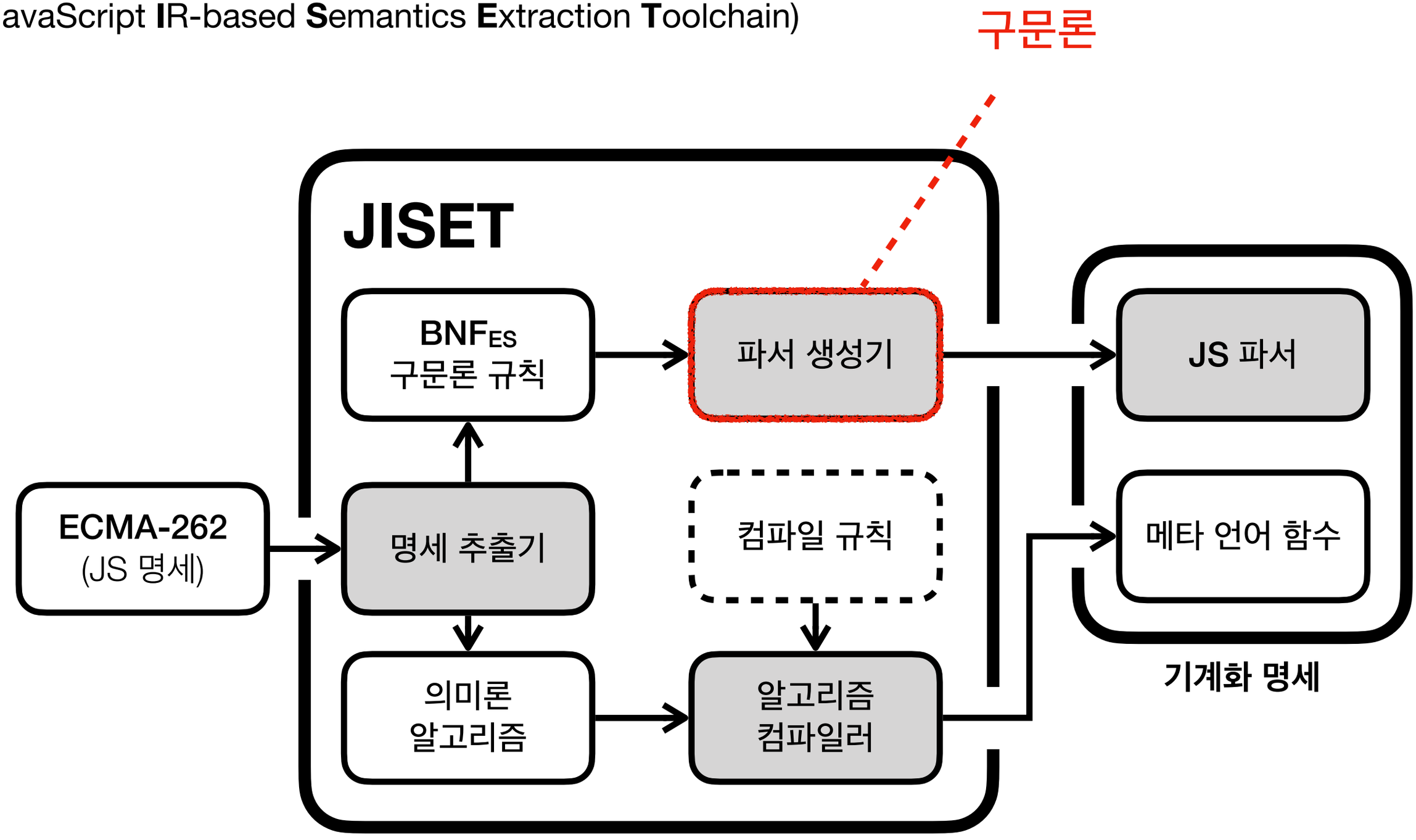
1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
   a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments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

의미론

The Evaluation algorithm for the third alternative of *ArrayLiteral* in ES13

JISET - ASE'20

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



JISET - 자동 파서 생성 (구문론)

ArrayLiteral[Yield, Await] :

[*Elision*opt]

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

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

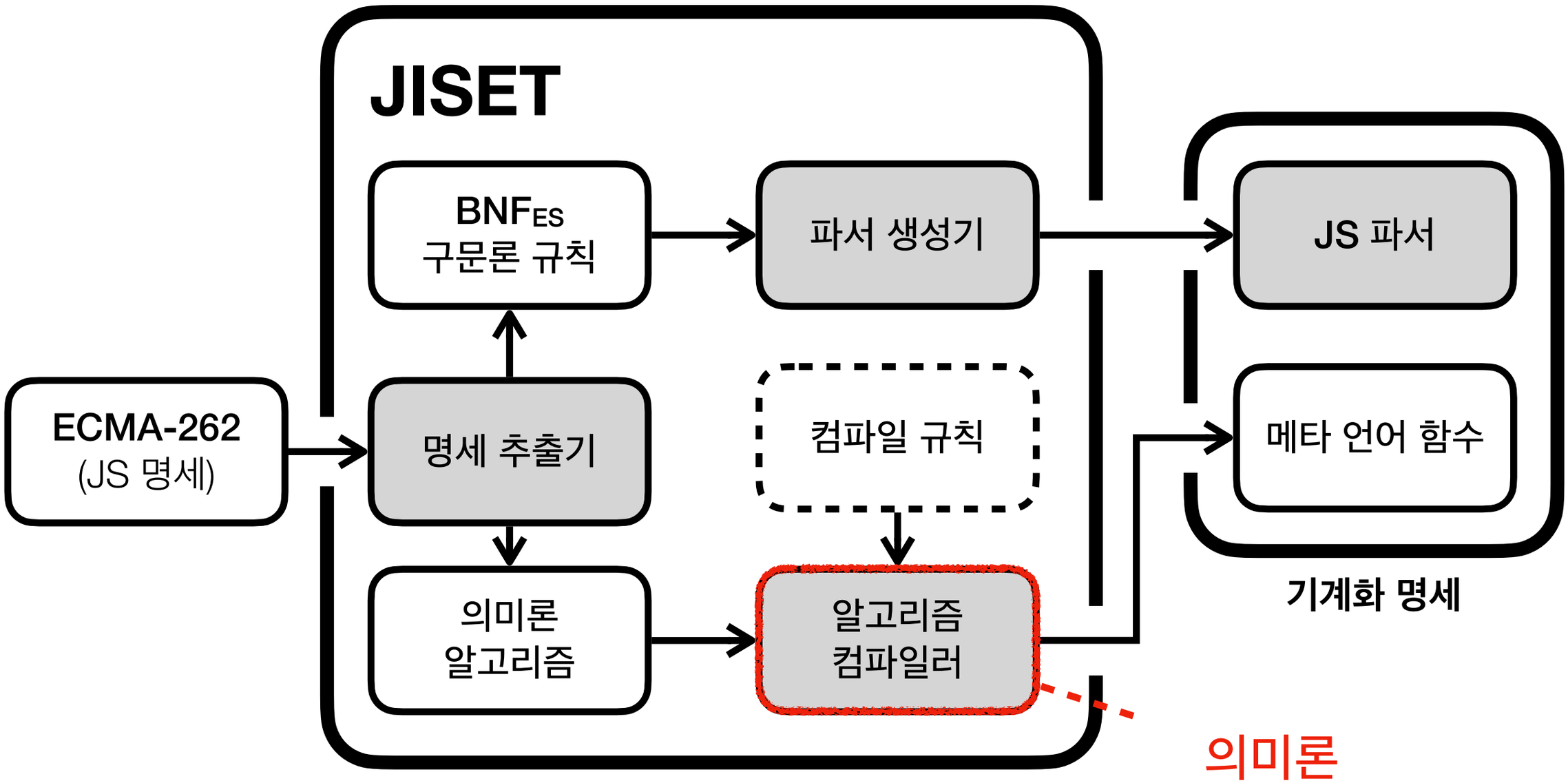
Parsing Expression Grammar (PEG)

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

[POPL'04] B. Ford, "Parsing Expression Grammars: A Recognition-based Syntactic Foundation"

JISET - ASE'20

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



JISET - ECMA-262를 위한 메타 언어

- ECMA-262의 추상 알고리즘을 표현하기 위한 **메타 언어**를 정의

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

• • •

$$
\begin{array}{lll}
\text{Values} & v \in \mathbb{V} & = \mathbb{A} \uplus \mathbb{V}^{\text{p}} \uplus \mathbb{T} \uplus \mathcal{F} \\
\text{Primitive Values} & v^{\text{p}} \in \mathbb{V}^{\text{p}} & = \mathbb{V}_{\text{bool}} \uplus \mathbb{V}_{\text{int}} \uplus \mathbb{V}_{\text{str}} \uplus \cdots \\
\text{JS ASTs} & t \in \mathbb{T}
\end{array}
$$

• • •

JISET - 알고리즘 컴파일러 (의미론)

13.2.5.2 Runtime Semantics: Evaluation

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

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

추상 알고리즘을 메타 언어로
컴파일하기 위한
118 가지 컴파일 규칙

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

JISET - Evaluation

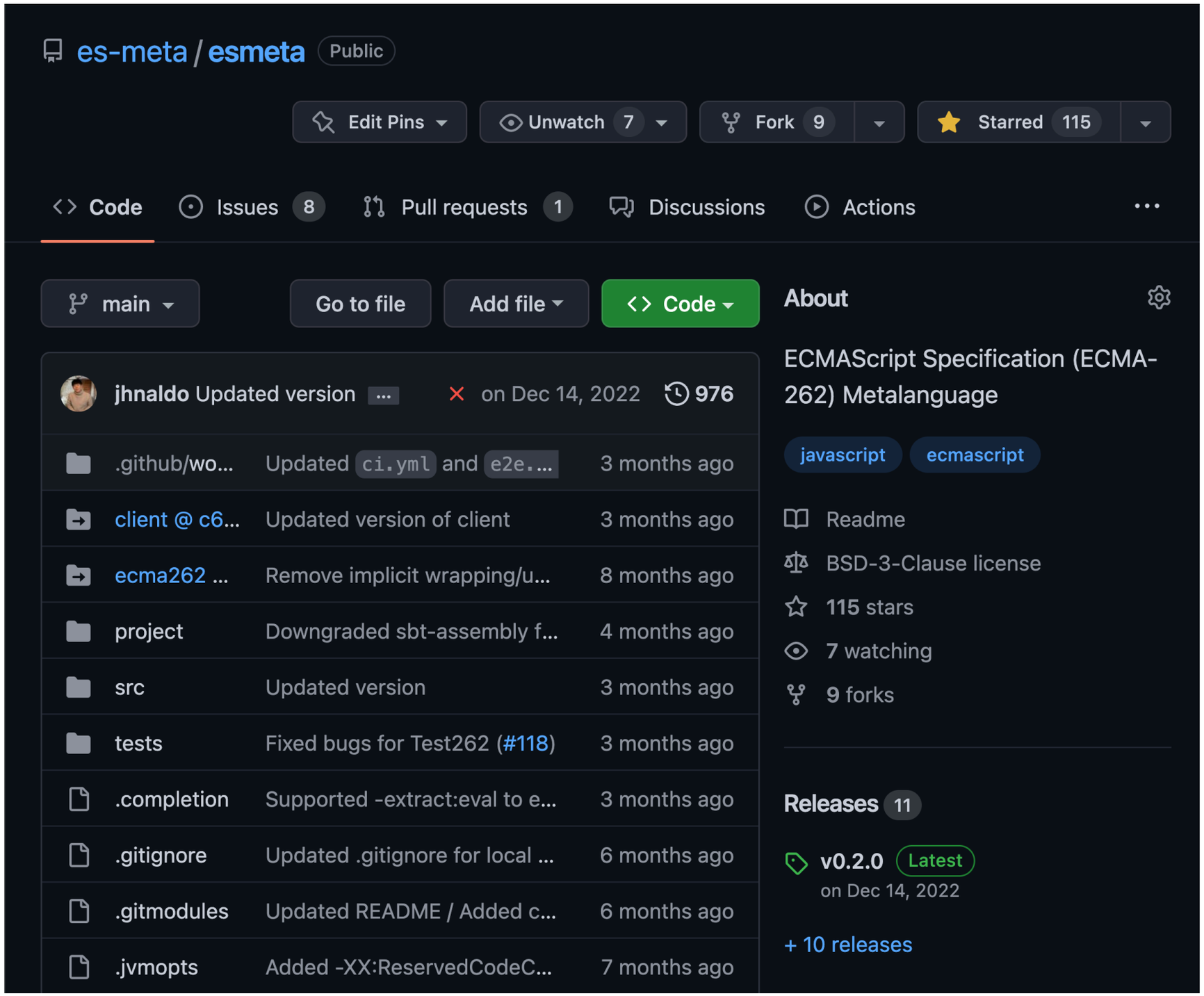
≈ 96% 자동 컴파일

auto manual

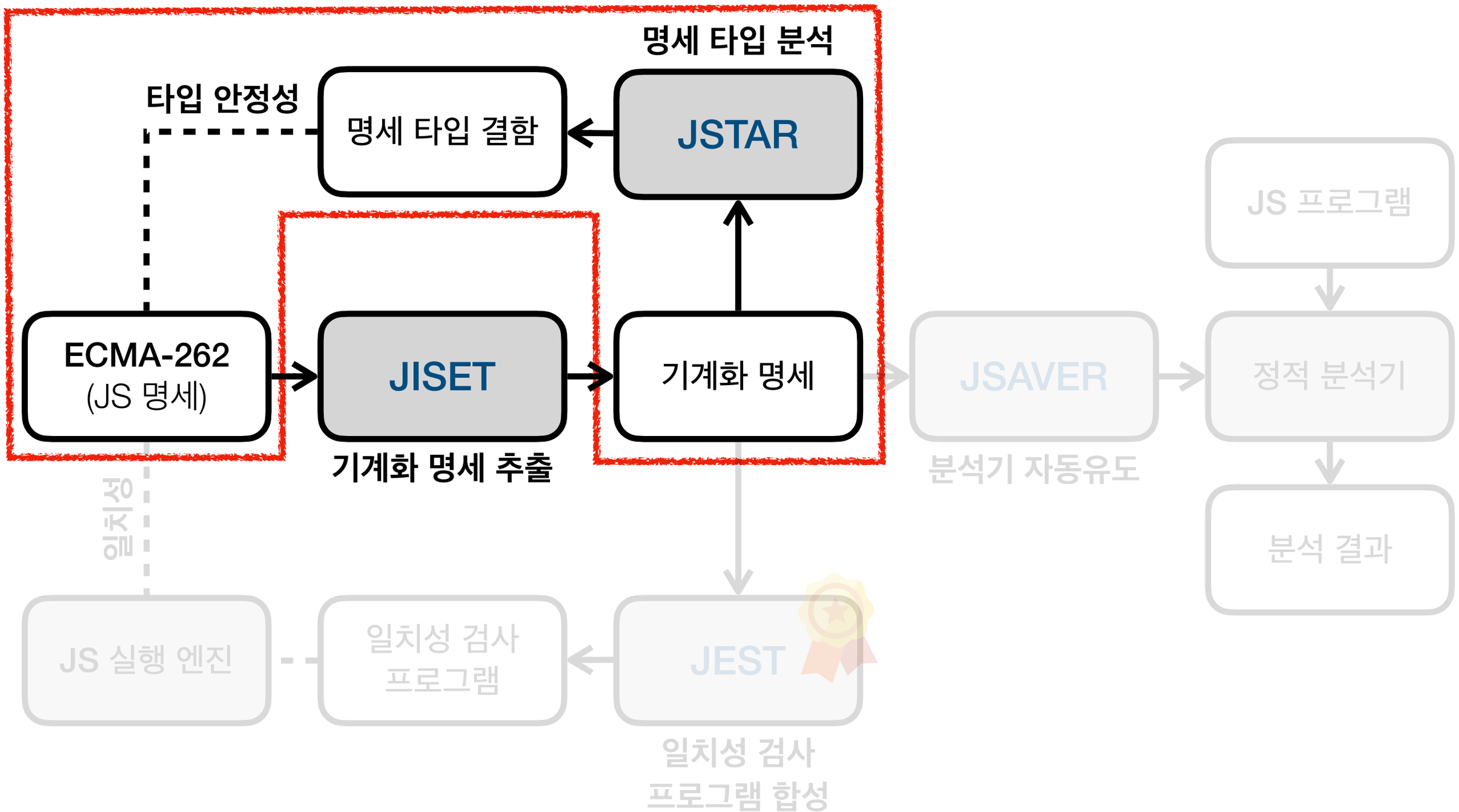
T: Total **L**: Core Language Semantics **B**: Built-in Libraries

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

ESMeta



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



[ASE'21] J. Park, et al. "JSTAR: JavaScript Specification Type Analyzer using Refinement"

Types in ECMA-262

20.3.2.28 Math.round (*x*) `x: (String ∨ Boolean ∨ Number ∨ Object ∨ ...)`

1. Let *n* be ? ToNumber(*x*). `ToNumber(x): (Number ∨ Exception) ∧ n: (Number)`
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**.

`>` 연산 타입 불일치
(숫자만 가능)

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

• • •

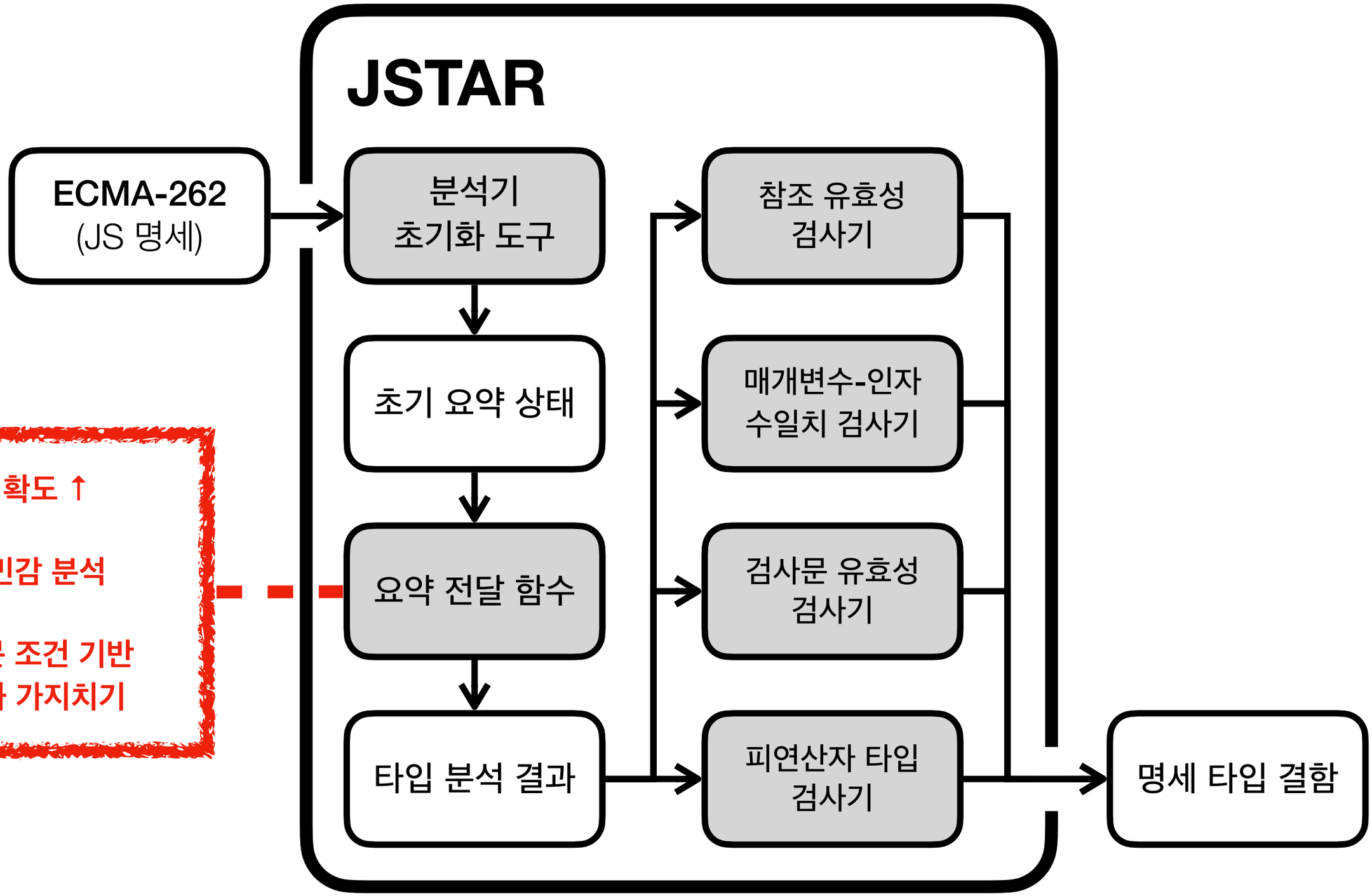
3. If $n < 0.5$ and $n > 0$, return **+0**.
4. If $n < 0$ and $n \geq -0.5$, return **-0**.

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

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

JSTAR - ASE'21

(**J**avaScript **S**pecification **T**ype **A**nalyzer using **R**efinement)



정확도 ↑

1. 타입 민감 분석

2. 분기문 조건 기반
분석 결과 가지치기

JSTAR - 실험 결과

59.2% 정확도

타입 결함 93개 검출

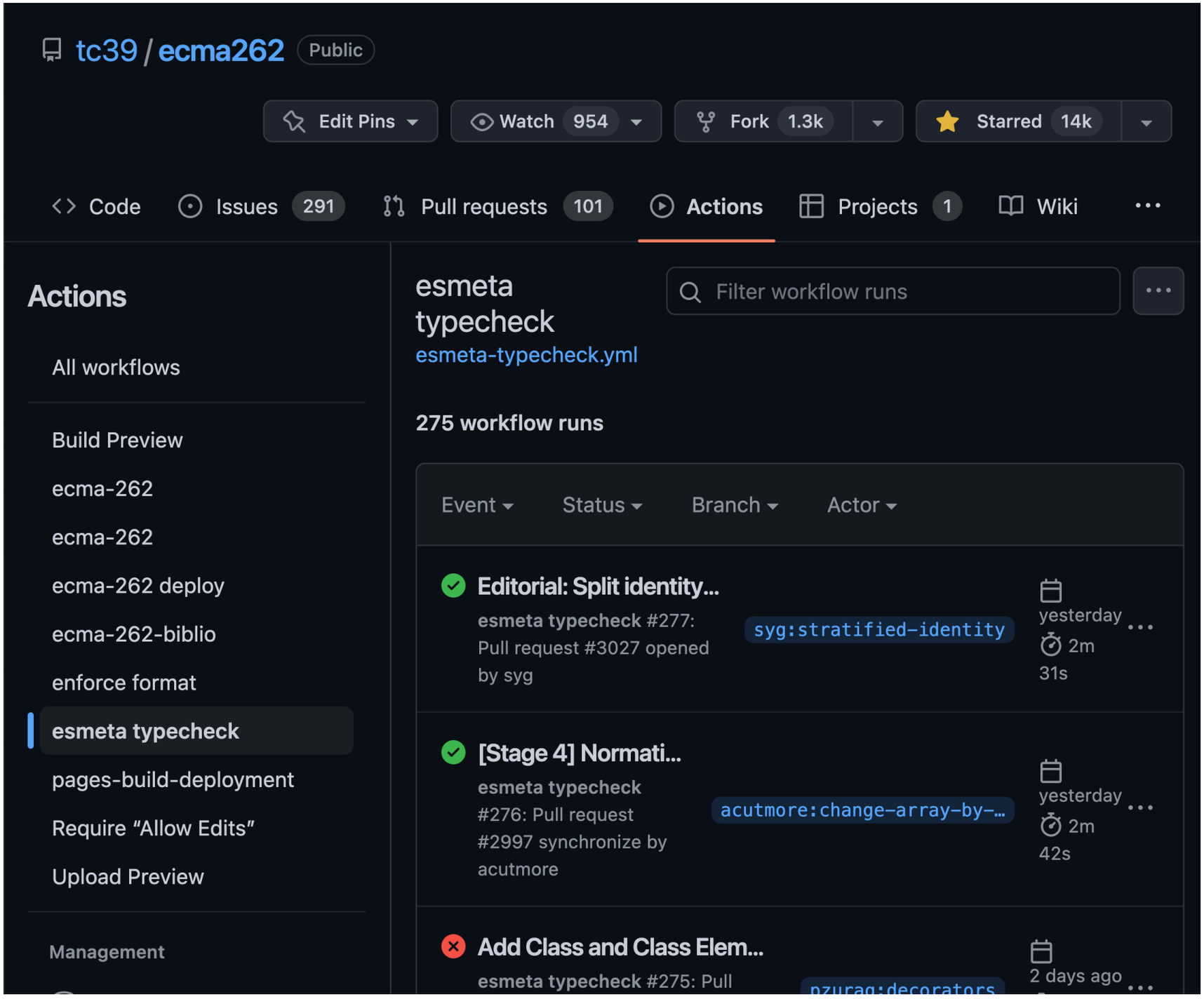
- **검사 대상**: 3년 동안 864개의 다른 ECMA-262 버전

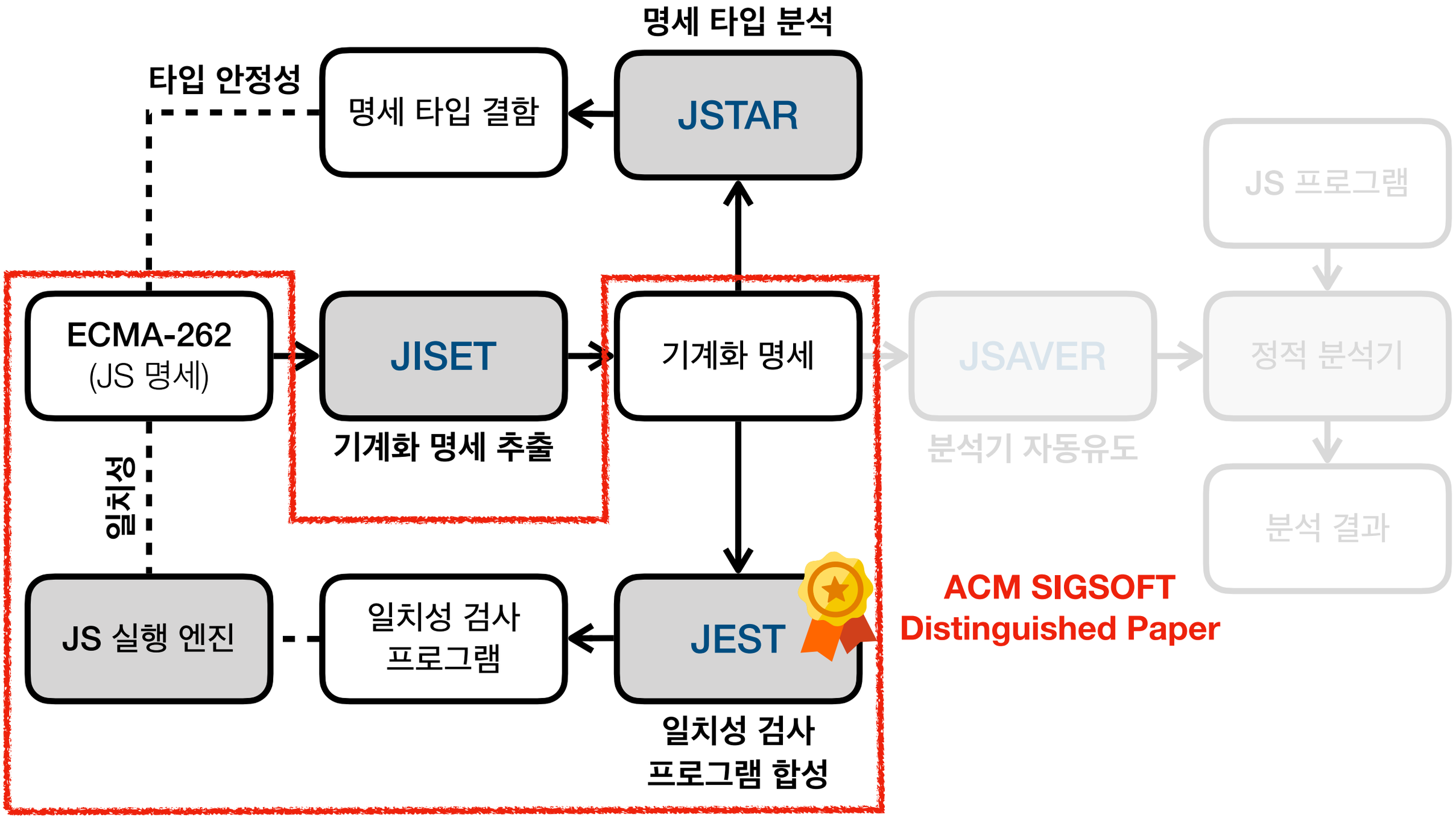
Checker	Bug Kind	Precision = (# True Bugs) / (# Detected Bugs)					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25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

ES12에서 결함 14개

ECMA-262 공식 CI 시스템으로 선정





[ICSE'21] J. Park, et al. "JEST: N+1-version Differential Testing of Both JavaScript Engines"
[PLDI'23] J. Park, et al. "Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations"

JS 명세와 실행 엔진 간의 일치성

ECMA-262
(JS 명세)

?

일치

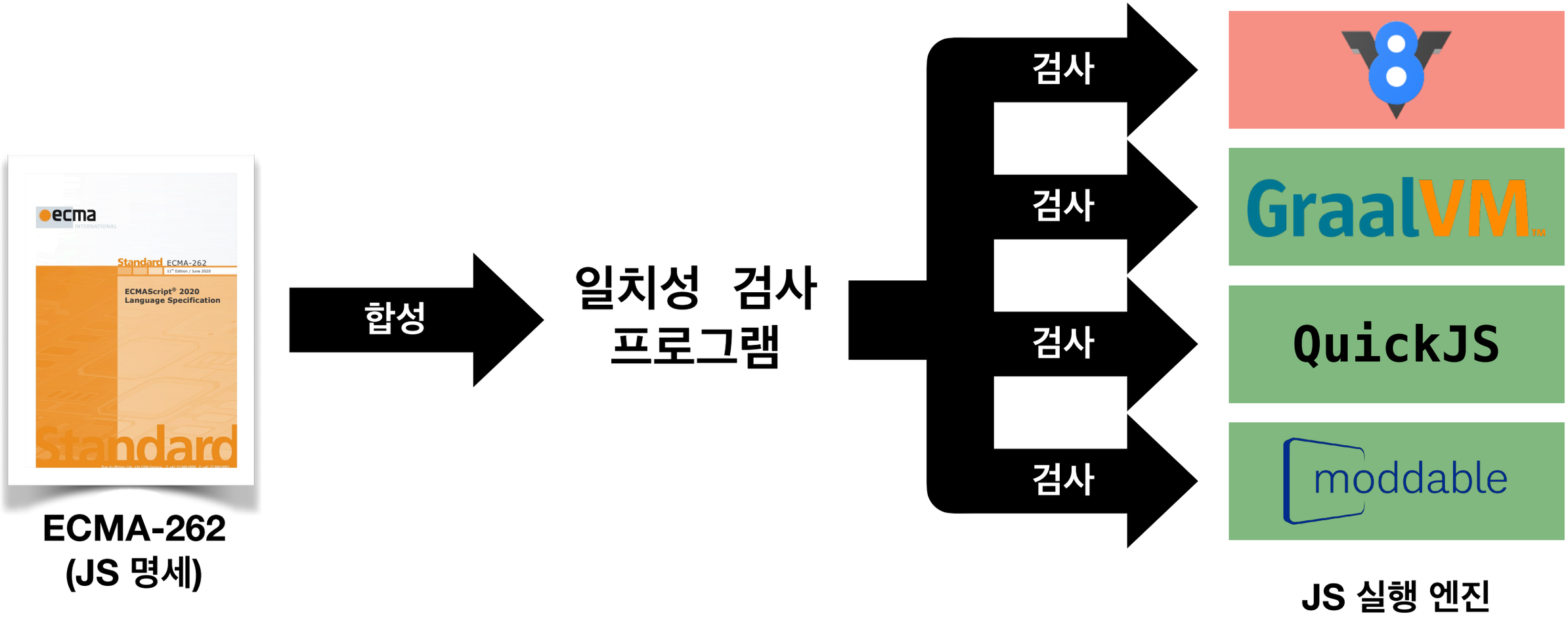
GraalVM

QuickJS

moddable

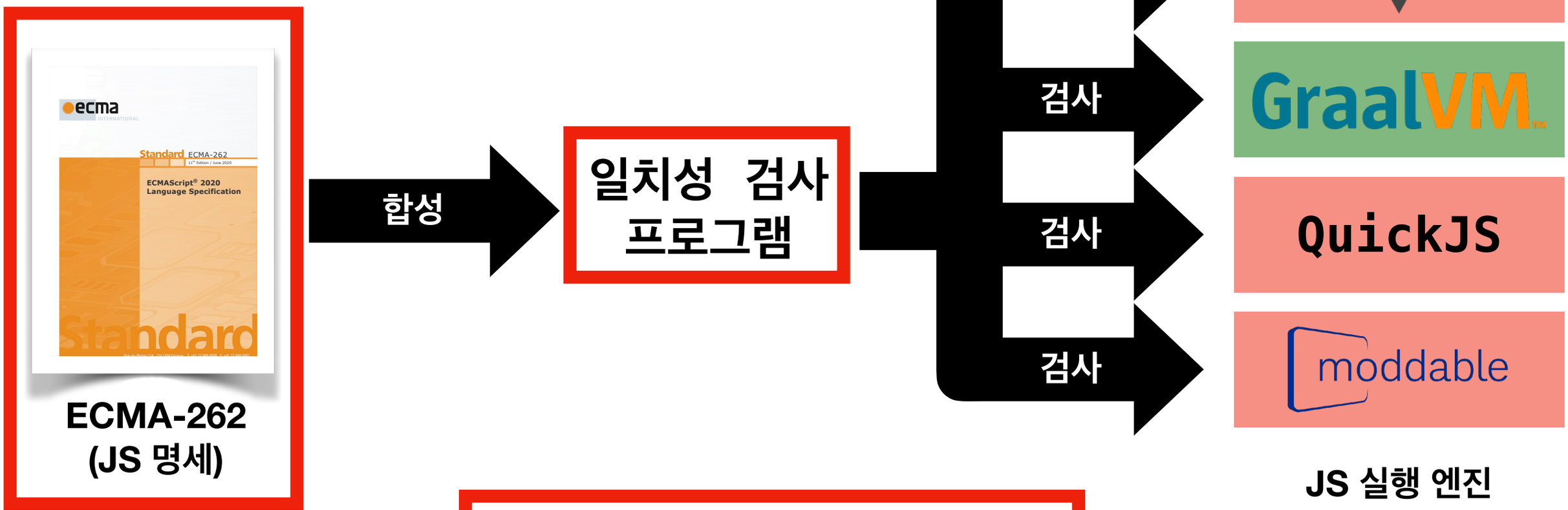
JS 실행 엔진

아이디어: N+1-버전 차분 테스팅



JS 실행 엔진의 결함

아이디어: N+1-버전 차분 테스팅

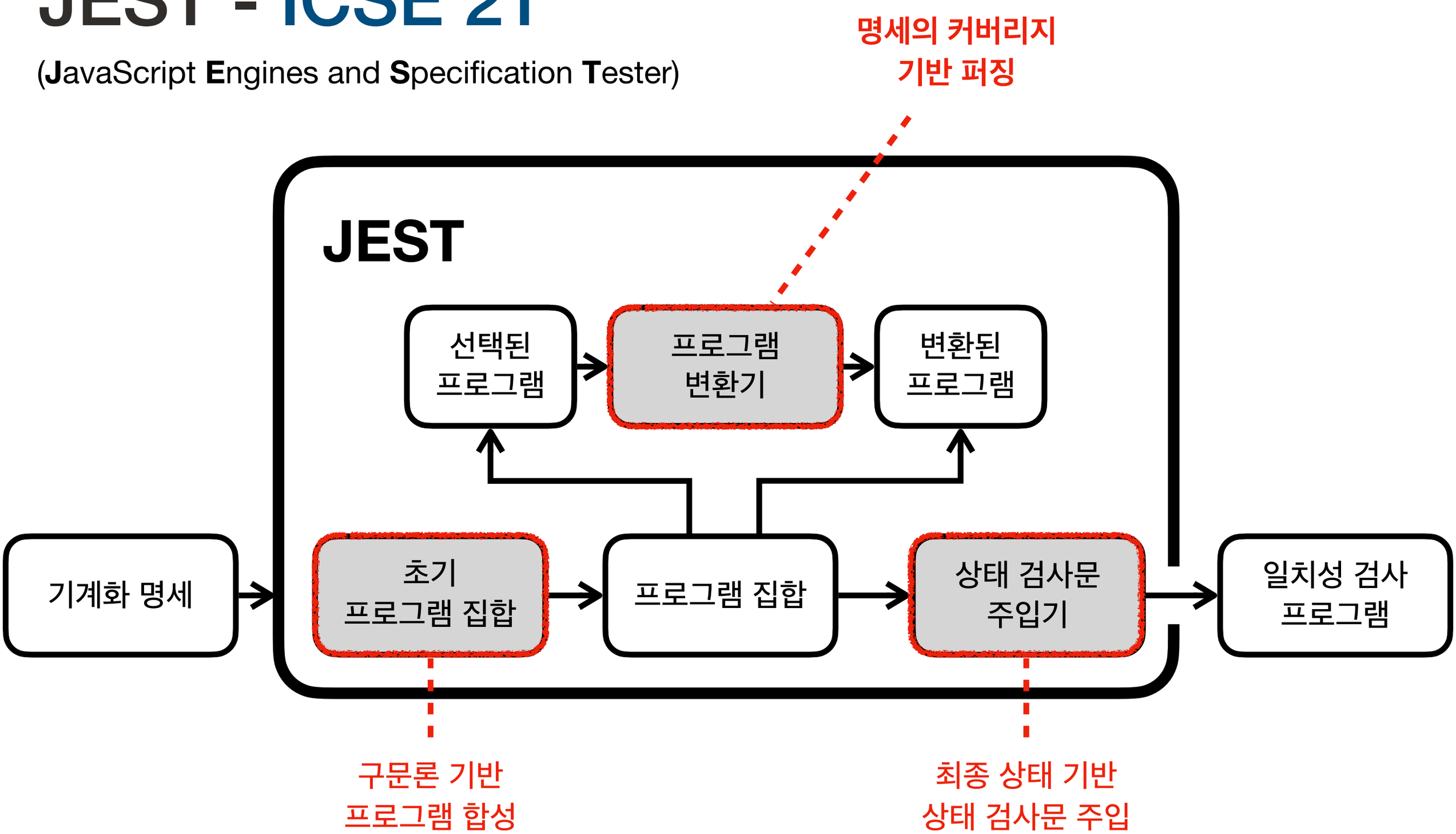


ECMA-262 내의 명세 결함 존재

JS 실행 엔진의 결함 GraalVM

JEST - ICSE'21

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



JEST - 명세의 커버리지 기반 퍼징

```
0 + { valueOf() { return 1; }
```

7.1.3 ToNumeric (*value*)

1. Let *primValue* be ? ToPrimitive(*value*, number)
2. If Type(*primValue*) is BigInt, return *primValue*.
3. Return ? ToNumber(*primValue*).

```
0 + { valueOf() { throw 42; }
```

JEST - 최종 상태 기반 상태 검사문 주입

```
  function f() {}

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

JEST - 실험 결과

- JEST가 성공적으로 ES11에서 1,700 일치성 테스트를 합성

TABLE II: The number of engine bugs detected by JEST

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

실행 엔진 결함 44개

명세 결함 27개

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

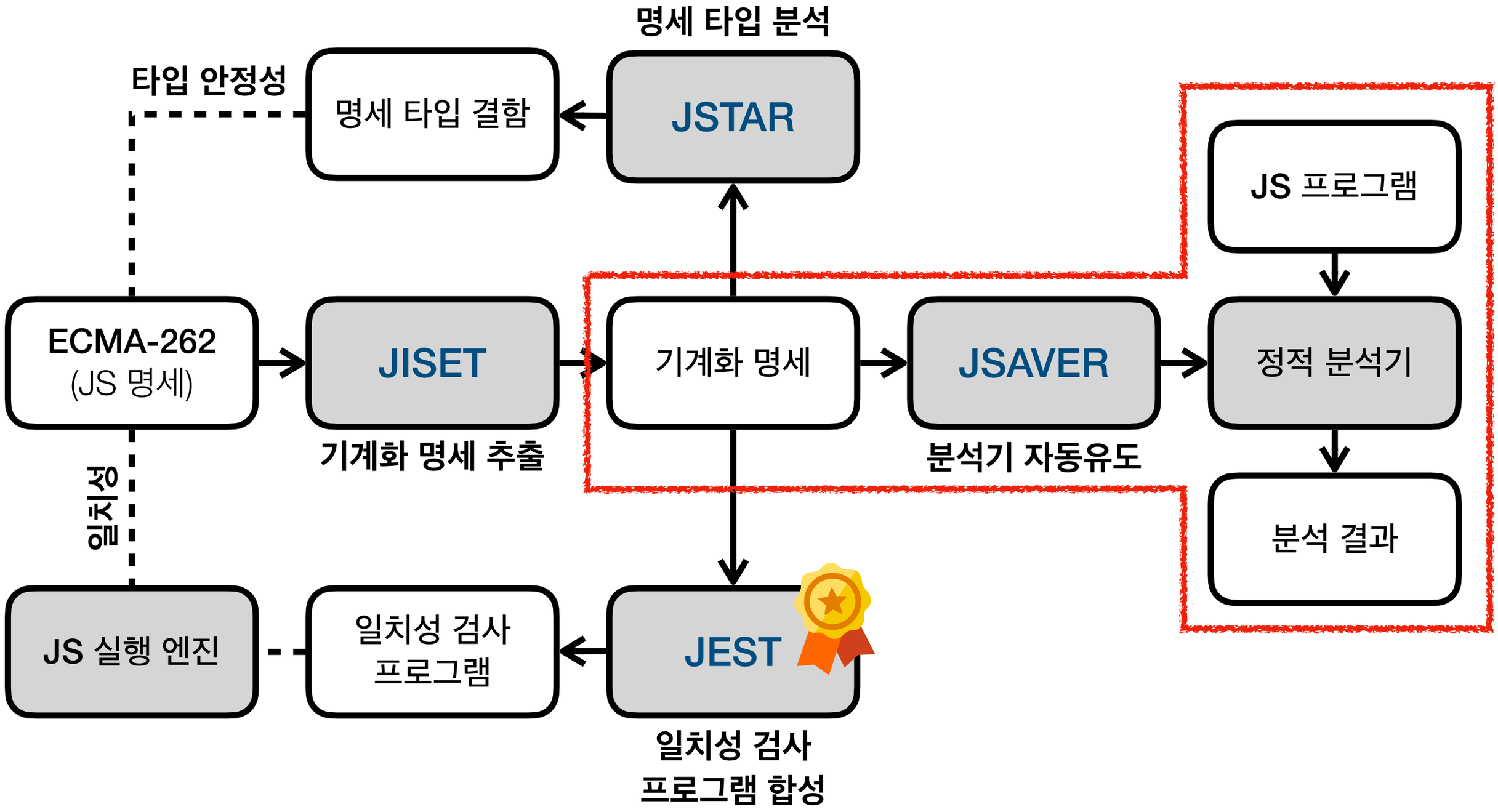
Name	Feature	#	Assertion	Known	Created	Resolved	Existed
ES11-1	Function	12	Key	O	2019-02-07	2020-04-11	429 days
ES11-2	Function	8	Key	O	2015-06-01	2020-04-11	1,776 days
ES11-3	Loop	1	Exc	O	2017-10-17	2020-04-30	926 days
ES11-4	Expression	4	Abort	O	2019-09-27	2020-04-23	209 days
ES11-5	Expression	1	Exc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

JEST - Example in GraalVM


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

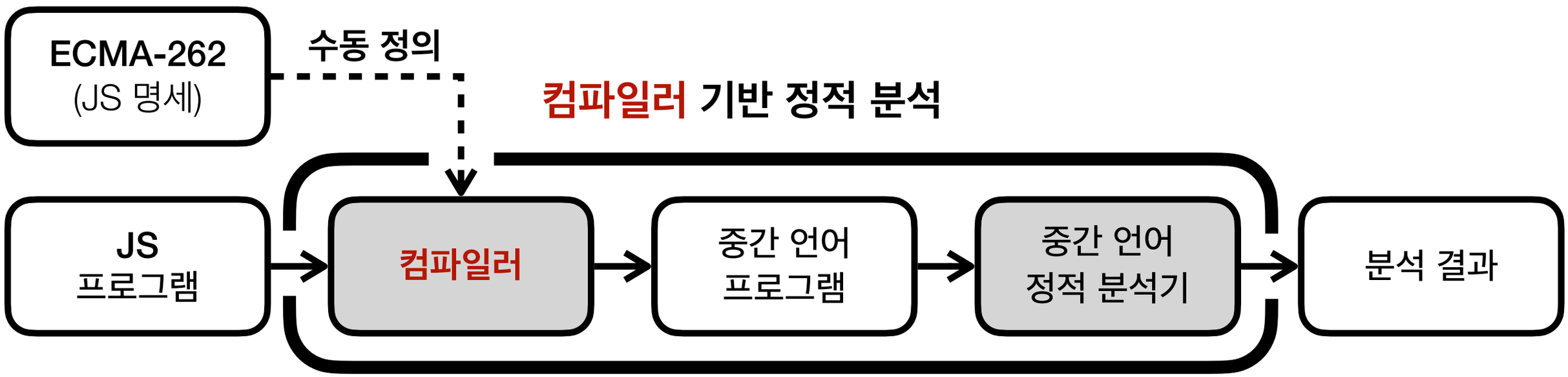
*"Right now, we are running Test262 and the V8 and Nashorn unit test suites in our CI for every change, it might make sense to **add your suite as well**."*

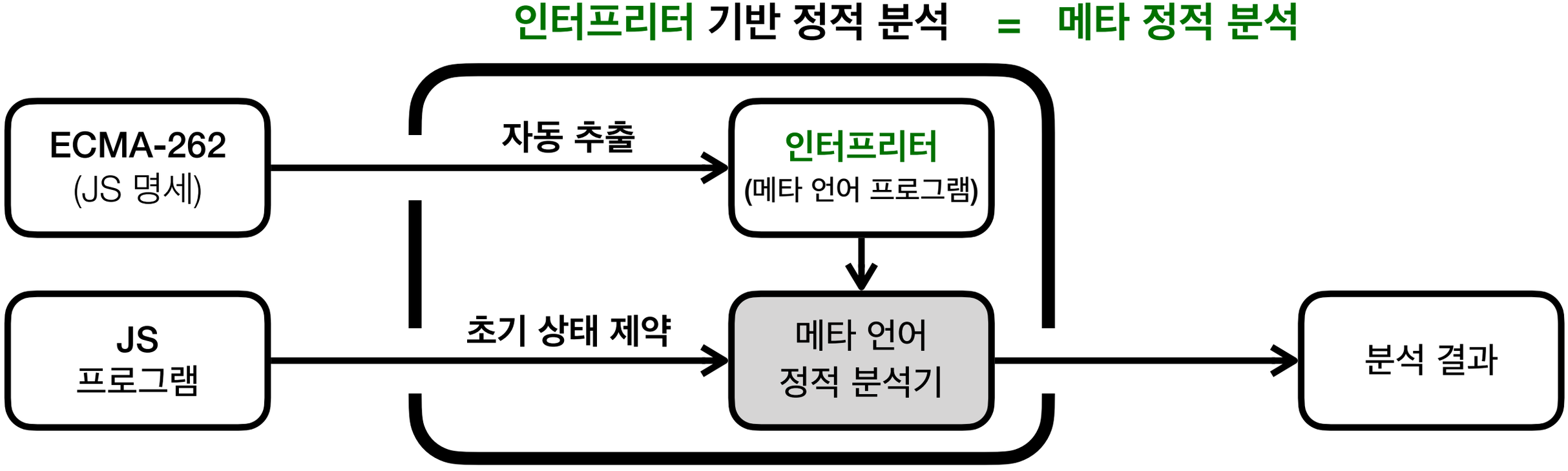
- A Developer of GraalVM



[FSE'22] J. Park, et al. "Automatically Deriving JavaScript Static Analyzers from Language Specifications"

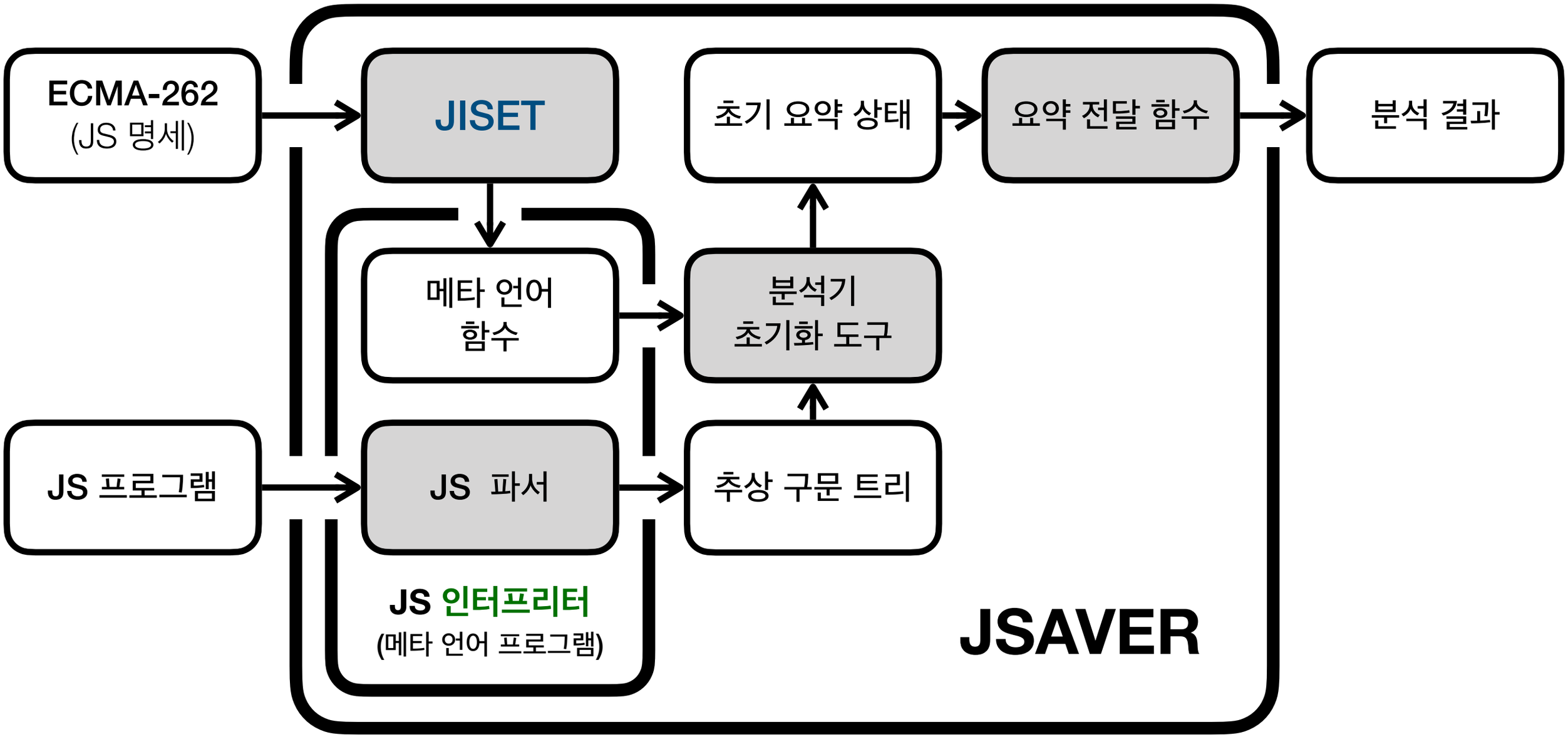
아이디어: 인터프리터 기반 정적 분석



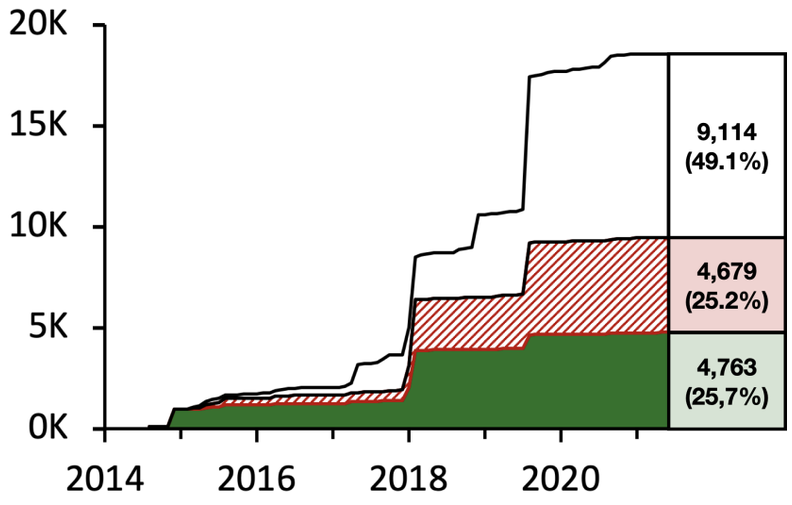


JSAVER - FSE'22

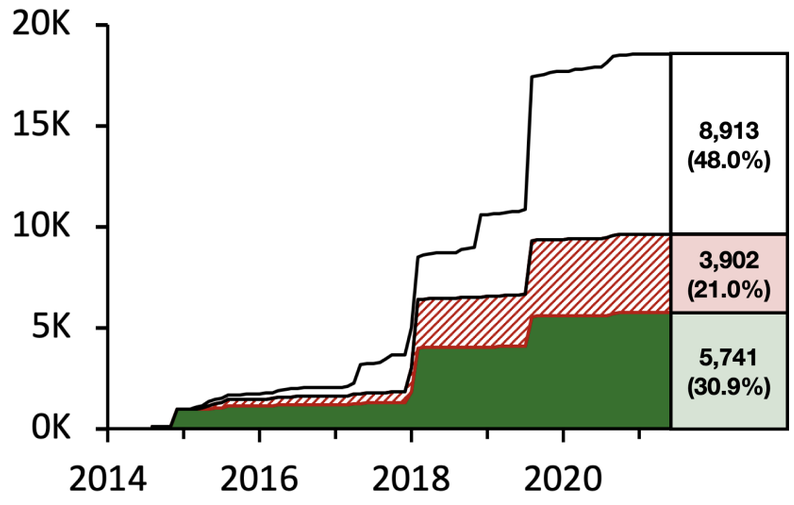
(**J**avaScript **S**tatic **A**nalyzer **v**ia **E**CMAScript **R**epresentation)



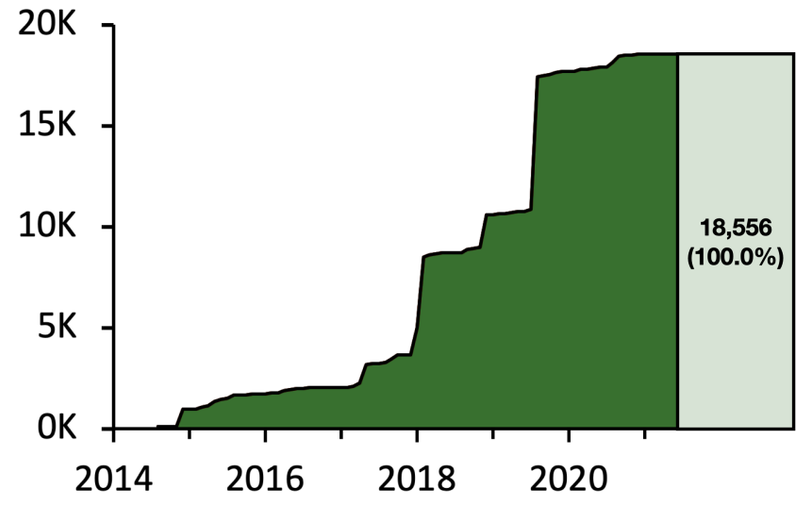
JSAVER - 평가 (RQ1: 올바른 분석)



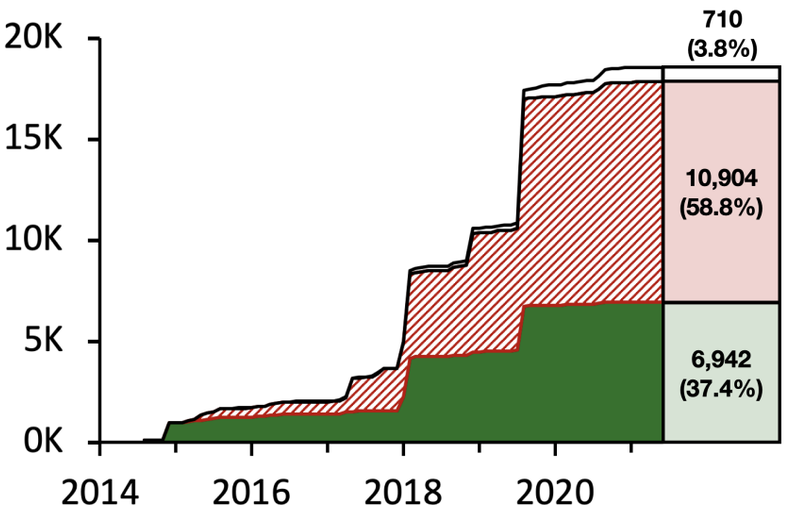
(a) Analysis results of TAJS



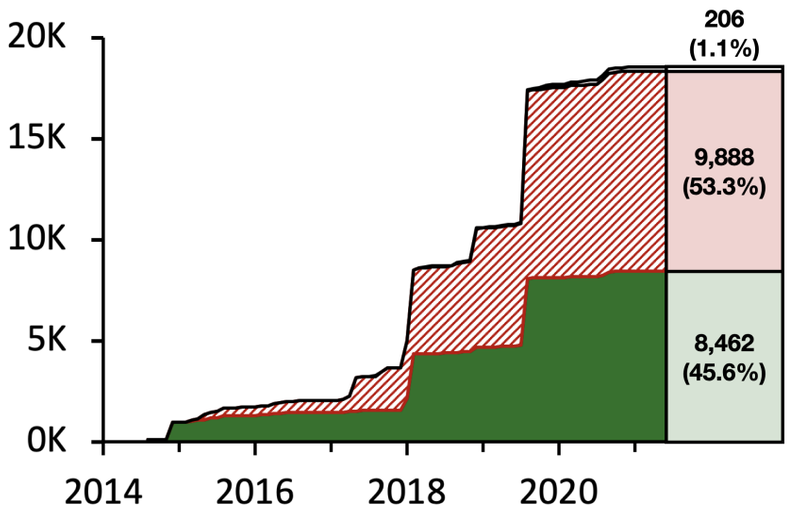
(b) Analysis results of SAFE



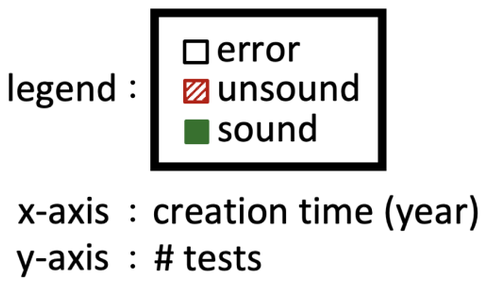
(c) Analysis results of JSA_{ES12}



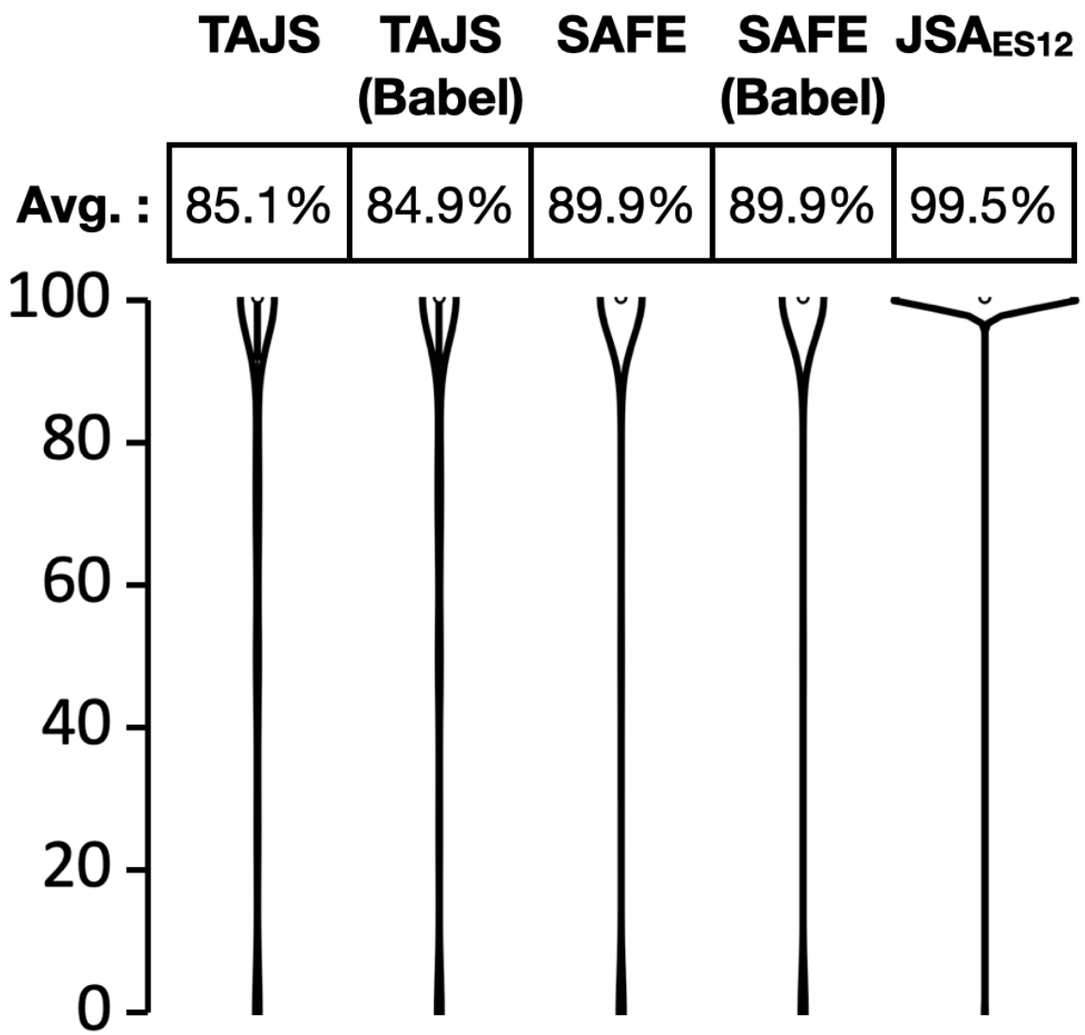
(d) Analysis results of TAJS with Babel



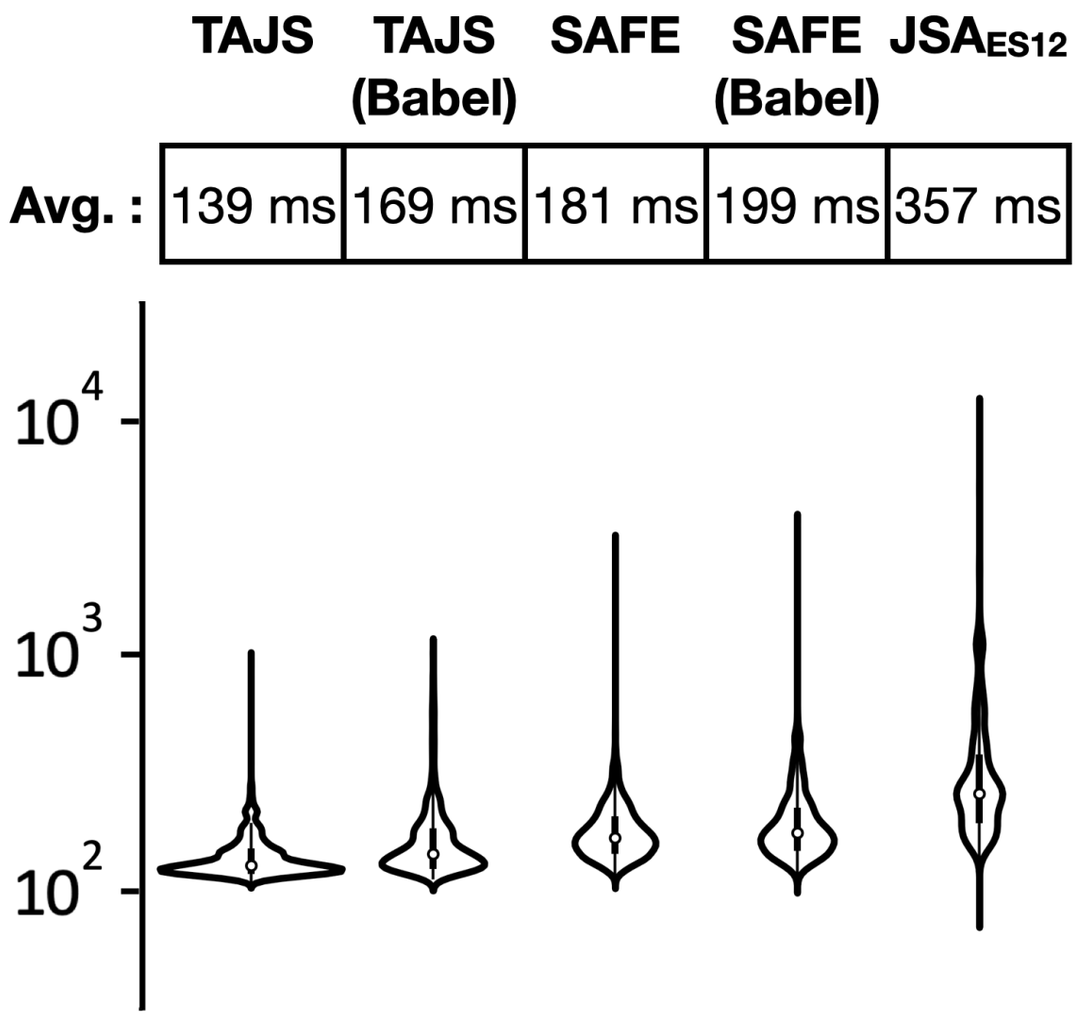
(e) Analysis results of SAFE with Babel



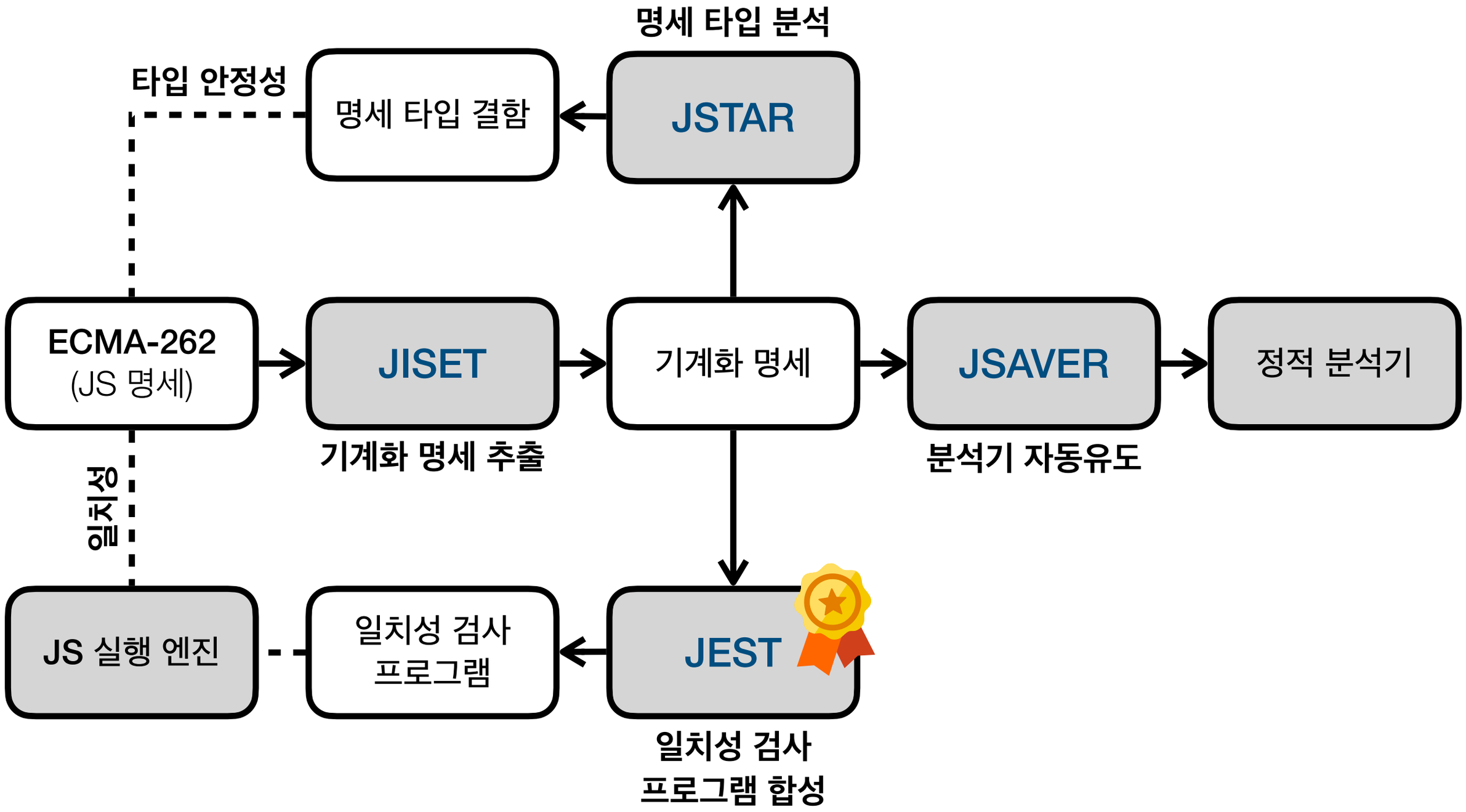
JSAVER - 평가 (RQ2: 정확도 / 속도)

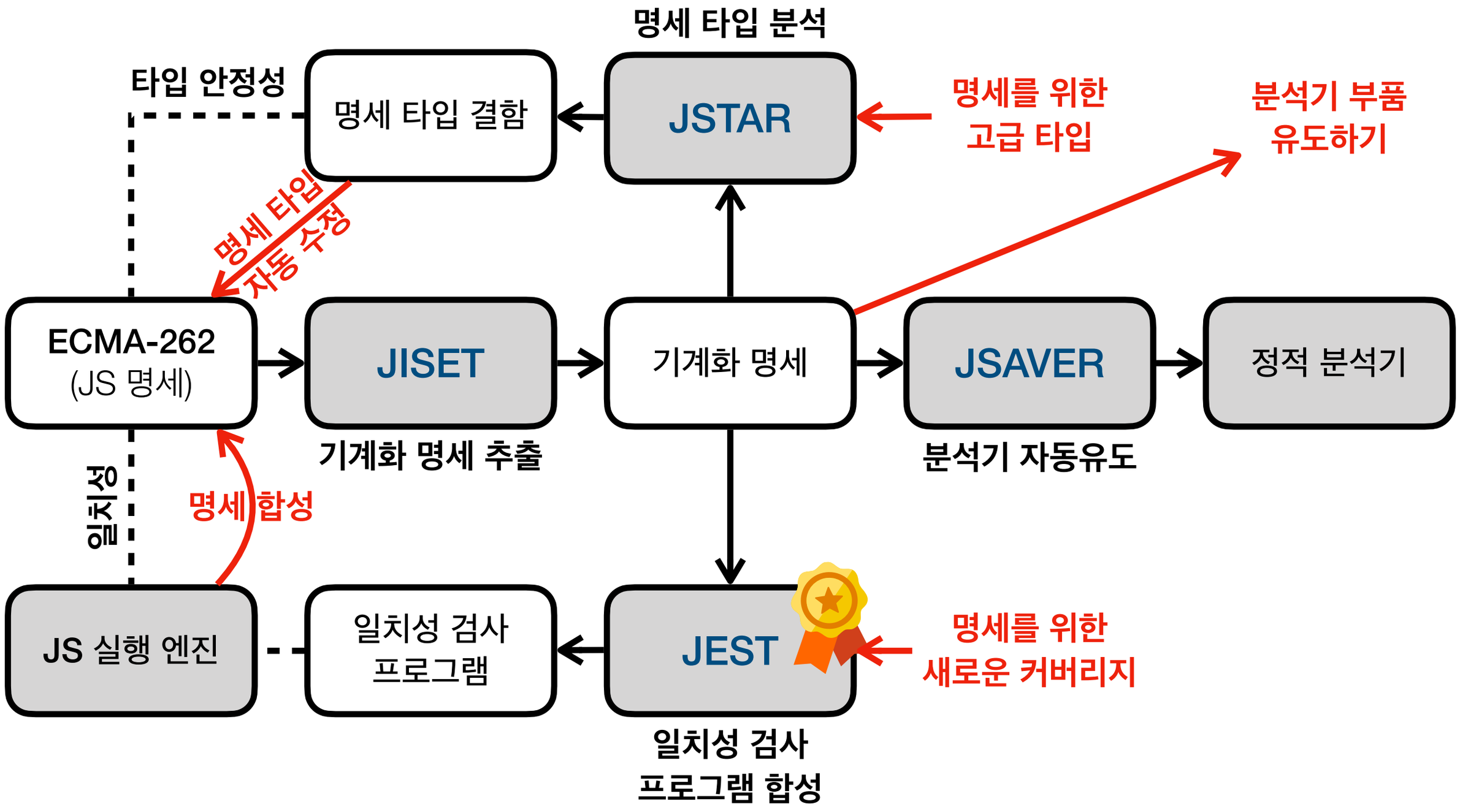


(a) The analysis precision

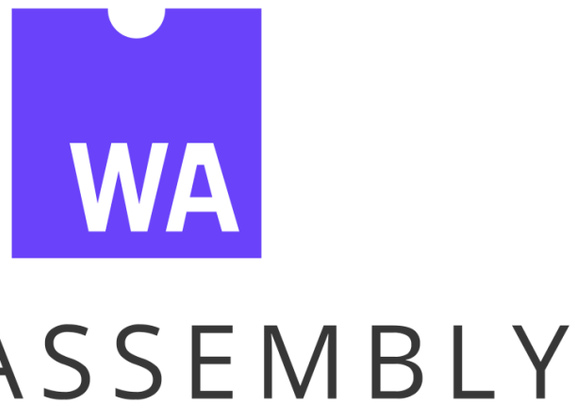


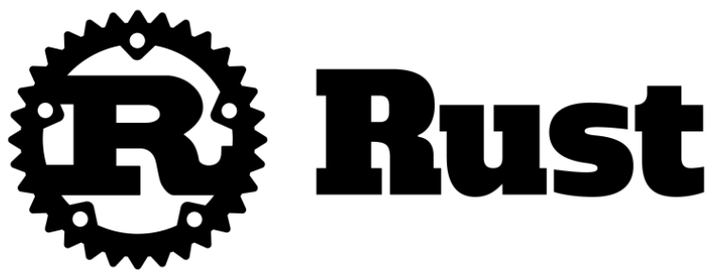
(b) The analysis performance


명세 타입 분석
타입 안정성
명세 타입 결함
JSTAR
ECMA-262
(JS 명세)
JISET
기계화 명세
JSAVER
정적 분석기
기계화 명세 추출
분석기 자동유도
일치성
JS 실행 엔진
일치성 검사
프로그램
JEST
일치성 검사
프로그램 합성


명세 타입 분석
타입 안정성
명세 타입 결함
JSTAR
명세를 위한
고급 타입
분석기 부품
유도하기
명세 타입
자동 수정
ECMA-262
(JS 명세)
JISET
기계화 명세
JSAVER
정적 분석기
기계화 명세 추출
분석기 자동유도
일치성
명세 합성
JS 실행 엔진
일치성 검사
프로그램
JEST
명세를 위한
새로운 커버리지
일치성 검사
프로그램 합성

추후 연구 - 다른 프로그래밍 언어로 확장





추후 연구 - 국내 및 해외 연구진들과 협업

Oracle Labs

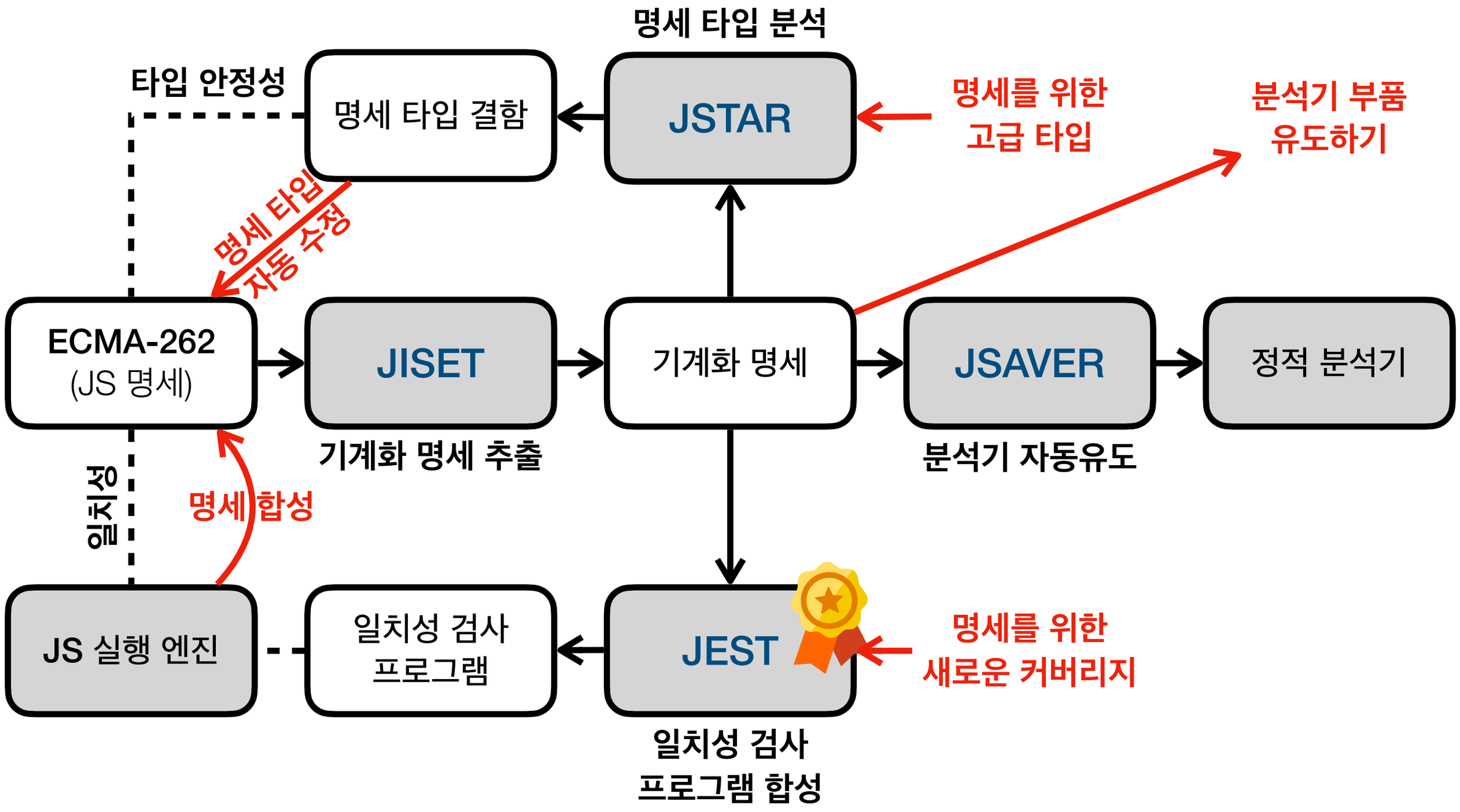
KAIST
KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
1971
한국과학기술원

PLRG

prime video

1794
ENS

PSL


명세 타입 분석
타입 안정성
명세 타입 결함
JSTAR
명세를 위한
고급 타입
분석기 부품
유도하기
명세 타입
자동 수정
ECMA-262
(JS 명세)
JISET
기계화 명세
JSAVER
정적 분석기
기계화 명세 추출
분석기 자동유도
일치성
명세 합성
JS 실행 엔진
일치성 검사
프로그램
JEST
명세를 위한
새로운 커버리지
일치성 검사
프로그램 합성