안전한 자바스크립트 언어 생태계 확보 기술

박지혁
고려대학교 컴퓨터학과

KOREA UNIVERSITY

2025.08.21 @ SIGPL 2025 여름학교

프로그래밍 언어 연구실 @ 고려대학교

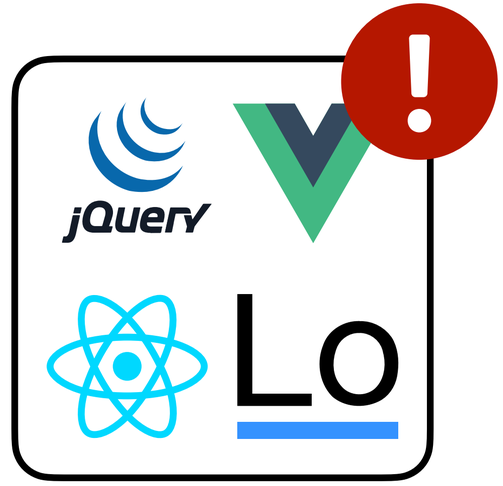
- **구성원** - 석사 3명 / 학부 4명
- **연구 분야 - 프로그래밍 언어 / 소프트웨어 공학**
  - 기계화 언어 명세
  - 프로그램 분석
  - 테스트 자동화
  - 프로그램 합성 및 패치
- **주요 연구 실적**
  - PL: **PLDI** (’23, ’24)
  - SE: **ICSE** (’17-Demo, ’21) / **FSE** (’21, ’22, ’25-Demo) / **ASE** (’20, ’21, ‘25)

자바스크립트 언어의 생태계

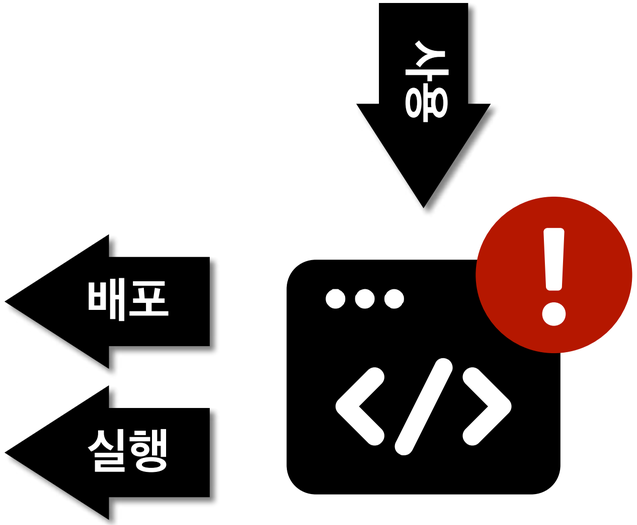
ECMA-262
(언어 명세)

구현

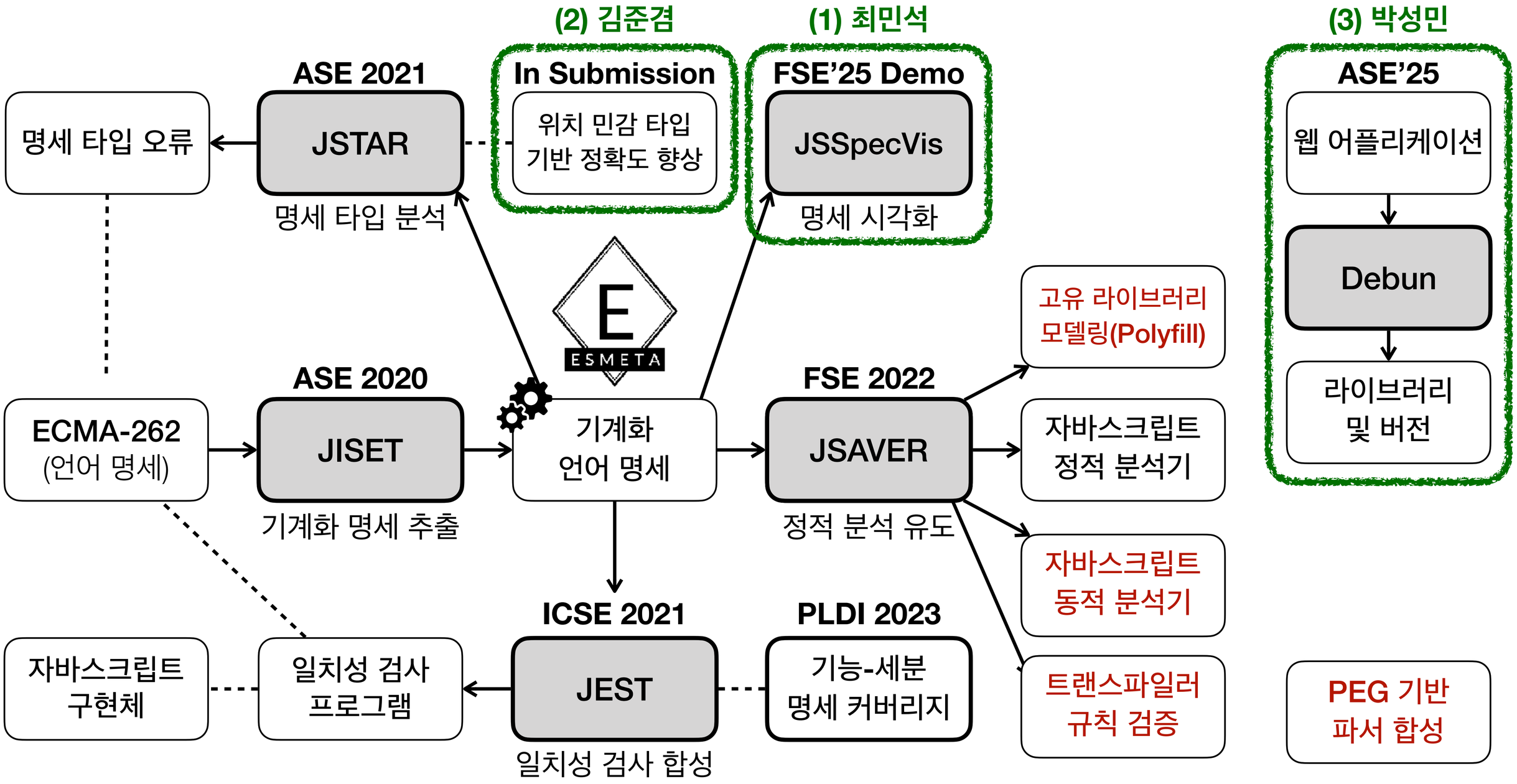
언어 구현체 및 개발 도구



라이브러리



어플리케이션


(2) 김준겸
(1) 최민석
(3) 박성민
ASE 2021
In Submission
FSE'25 Demo
ASE'25
명세 타입 오류
JSTAR
위치 민감 타입
기반 정확도 향상
JSSpecVis
웹 어플리케이션
Debun
라이브러리
및 버전
명세 타입 분석
명세 시각화
ESMETA
ASE 2020
FSE 2022
고유 라이브러리
모델링(Polyfill)
ECMA-262
(언어 명세)
JISET
기계화
언어 명세
JSAVER
자바스크립트
정적 분석기
기계화 명세 추출
정적 분석 유도
자바스크립트
동적 분석기
ICSE 2021
PLDI 2023
자바스크립트
구현체
일치성 검사
프로그램
JEST
기능-세분
명세 커버리지
트랜스파일러
규칙 검증
PEG 기반
파서 합성
일치성 검사 합성

JSSpecVis

자바스크립트 언어 명세 시각화 도구

최민석*, 송경호*, 김현준, 박지혁

프로그래밍언어연구실 @ 고려대학교

JavaScript 언어 명세 : ECMA-262

- 기본적으로 영어 산문체지만, 일부분은 **코드 스타일로 작성**
- 이는 명세를 대상으로 한 **자동화 된 작업(기계화 명세)**을 가능하게 함

ApplyStringOrNumericBinaryOperator (*lVal*, *opText*, *rVal*)
It performs the following steps when called:

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
      1. Let *lStr* be ? ToString(*lPrim*).
      2. Let *rStr* be ? ToString(*rPrim*).
      3. 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. …

JavaScript 언어 명세는 복잡하다

- ‘더하기’를 이해하는 것조차 쉽지 않다: ⬚ + ⬚
  - 최소 **120개 이상의 명세 단계를 거쳐야함**

예를 들어: 왜 `!0 + 1n` 를 계산하면 **TypeError**일까?

Syntax

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

Semantic

Runtime Semantics: Evaluation
AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*
1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

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

Expr !0

+

Expr 1n

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*).

왼쪽을 평가, (**lval** = Boolean true)

오른쪽을 평가, (**rval** = BigInt 1n)

Boolean true

+

BigInt 1n

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

A. Let *lprim* be ? ToPrimitive(*lval*).

B. Let *rprim* be ? ToPrimitive(*rval*).

C. If *lprim* is a String or *rprim* is a String, then

   1. Let *lstr* be ? ToString(*lprim*).
   2. Let *rstr* be ? ToString(*rprim*).
   3. 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

…

Primitive 값으로 변환

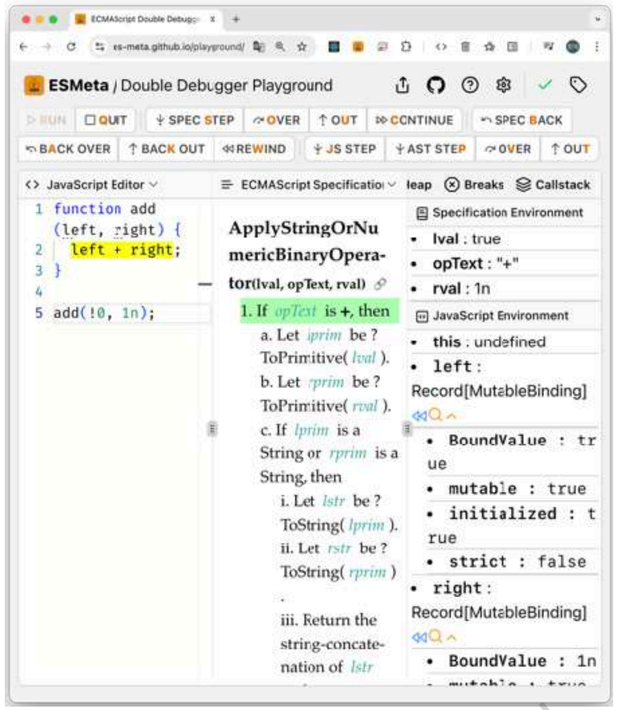
(**lprim** = Boolean true, **rprim** = BigInt

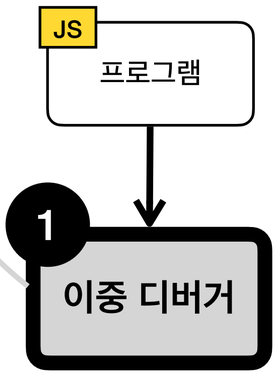
Numeric 값으로 변환

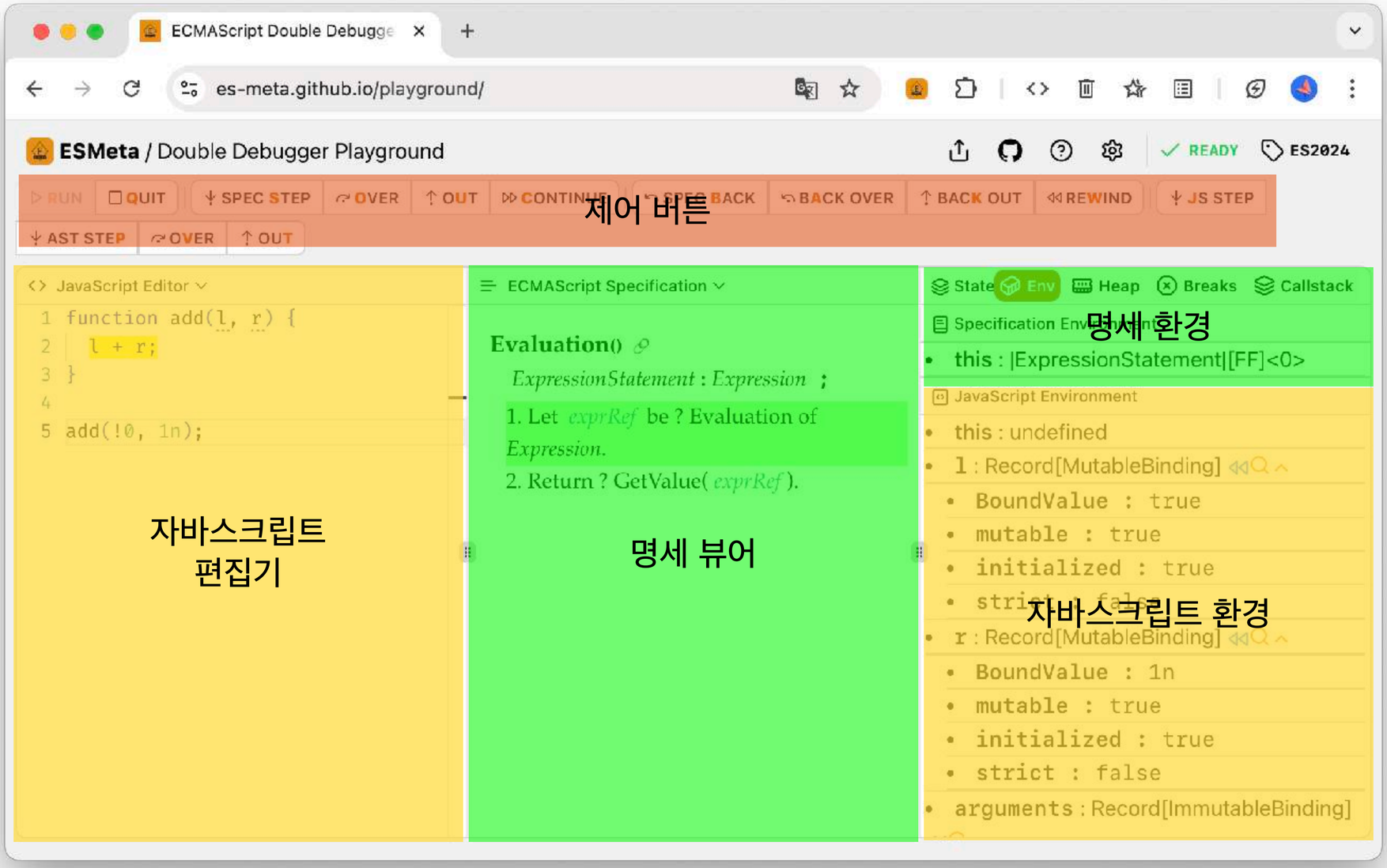
(**lnum** = Number 1, **rnum** = BigInt 1n)

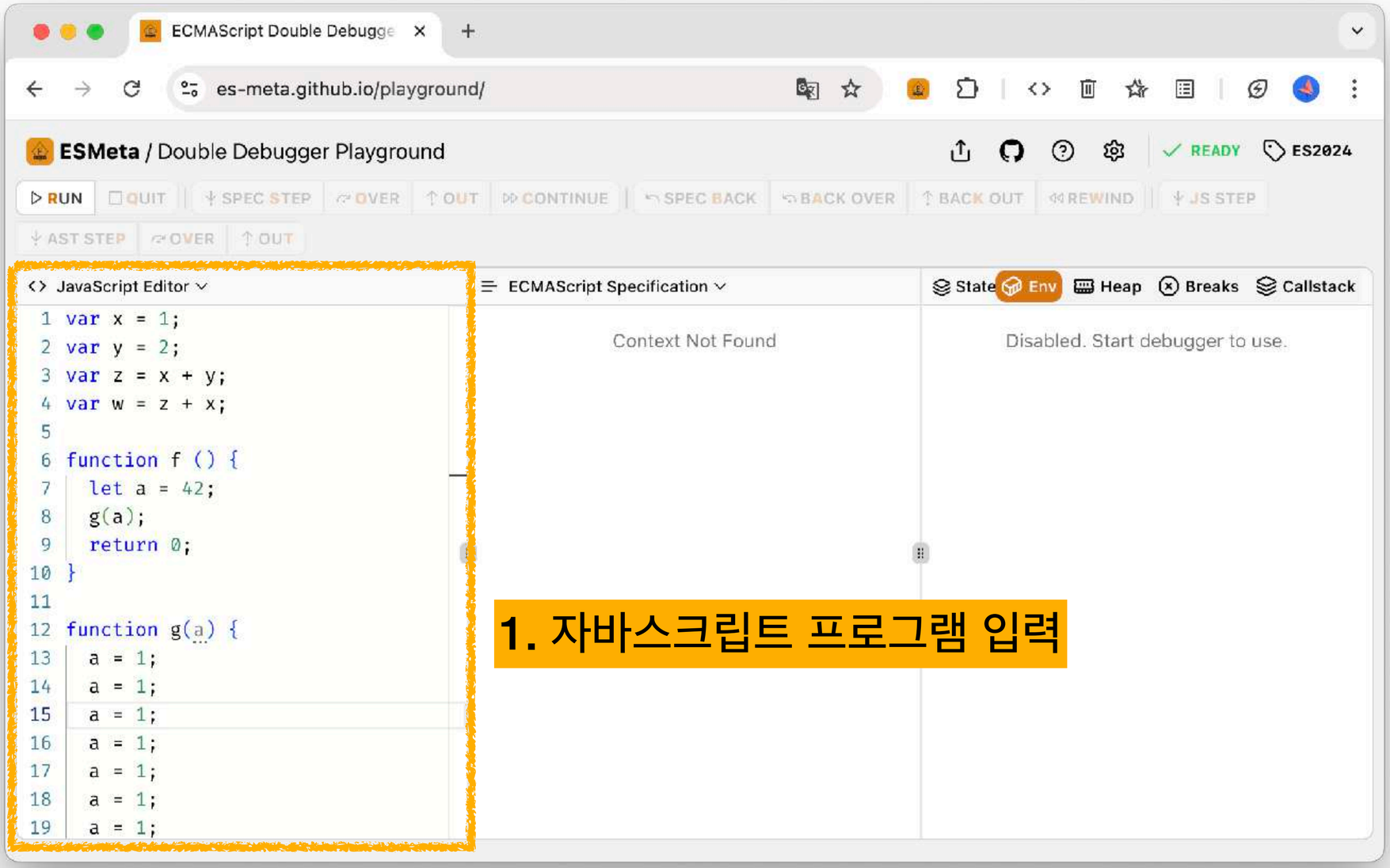
명세-레벨 디버깅을 더 자동적이고 상호작용적인 방식으로 할 수 있다면 어떨까?

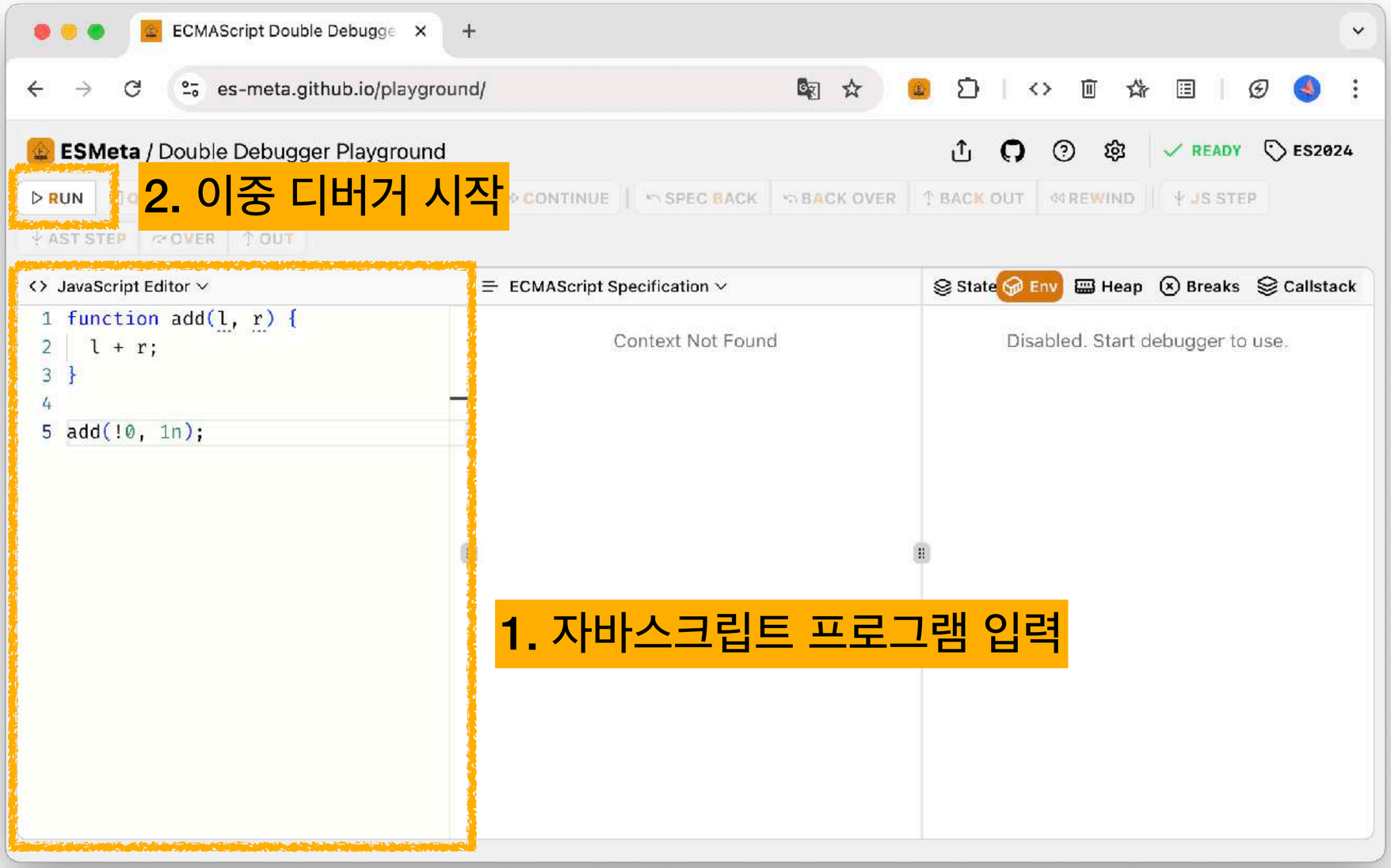
Solution 1. 이중 디버거

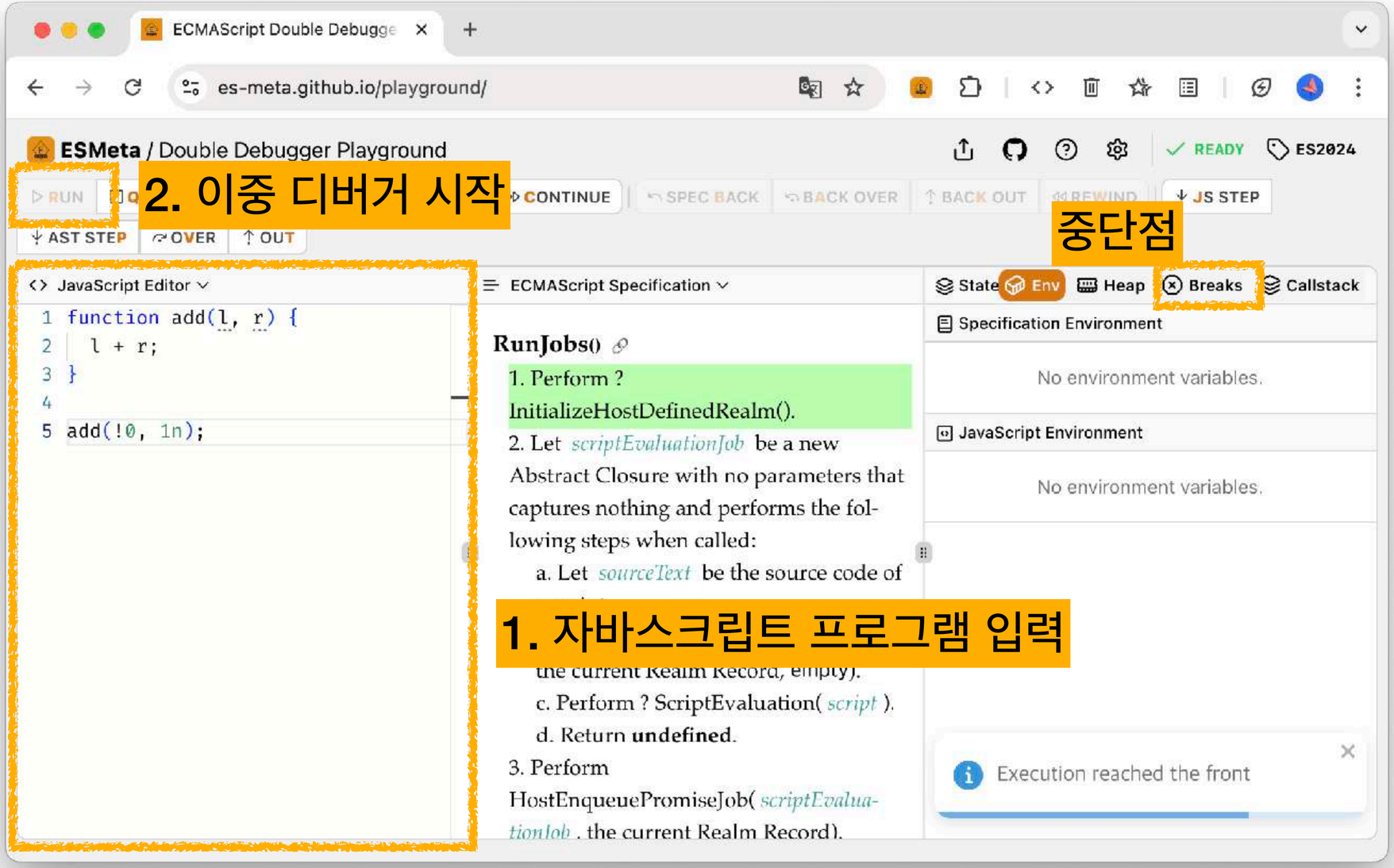


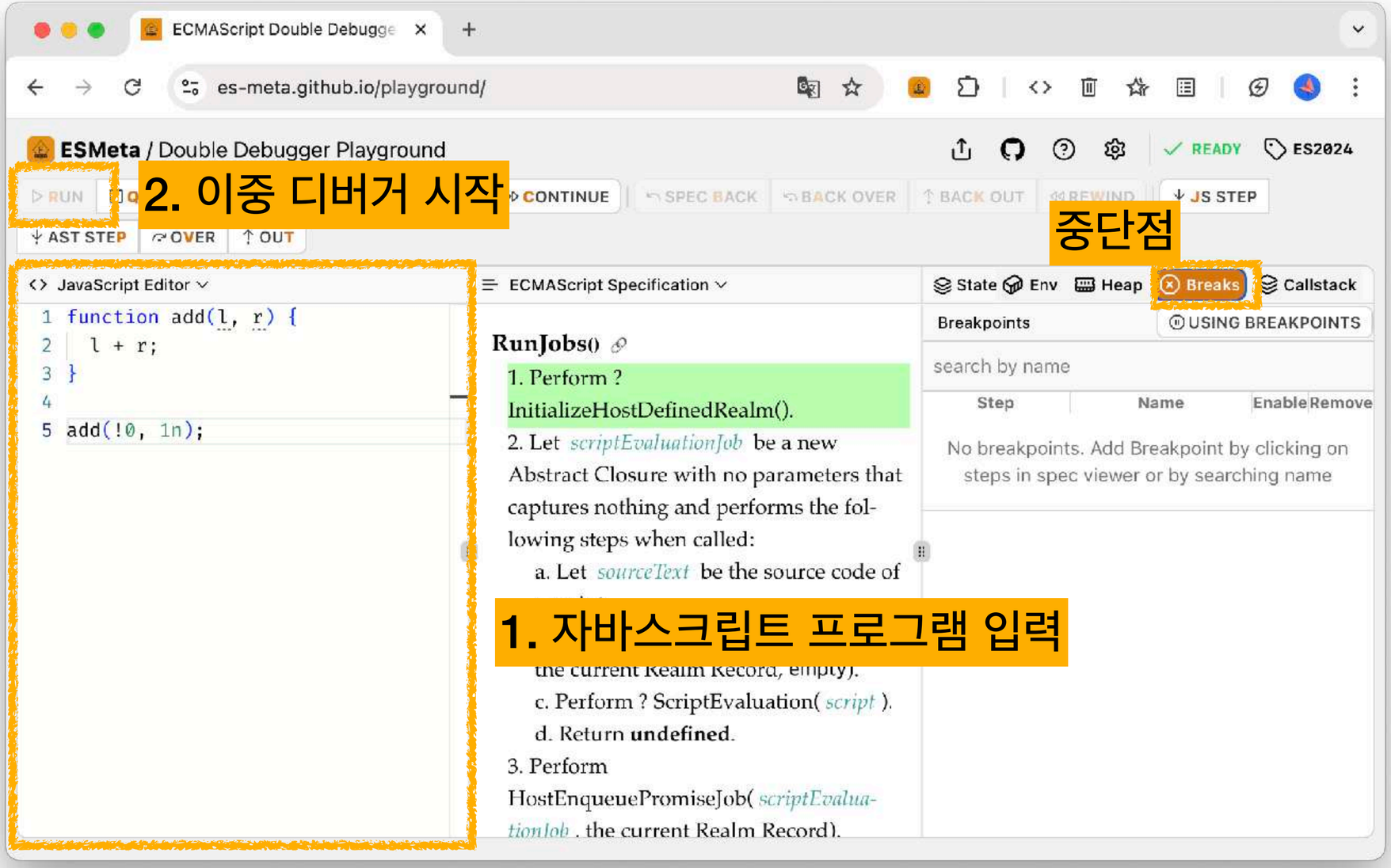


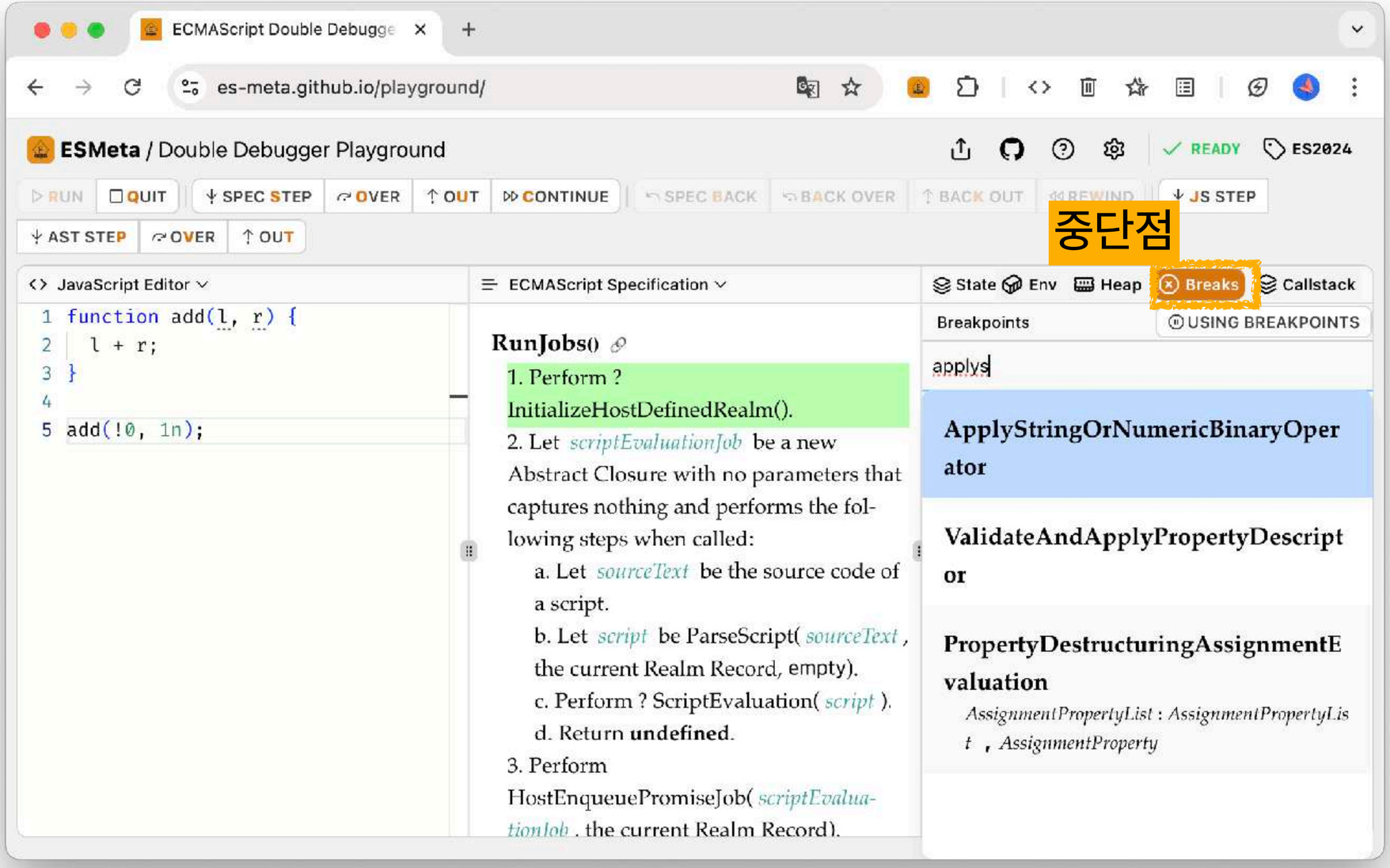

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
제어 버튼
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
자바스크립트
편집기
ECMAScript Specification
Evaluation()
ExpressionStatement : Expression ;
1. Let exprRef be ? Evaluation of Expression.
2. Return ? GetValue(exprRef).
명세 뷰어
State
Env
Heap
Breaks
Callstack
Specification Environment
명세 환경
this : |ExpressionStatement|[FF]<0>
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true
initialized : true
strict : false
arguments : Record[ImmutableBinding]
자바스크립트 환경

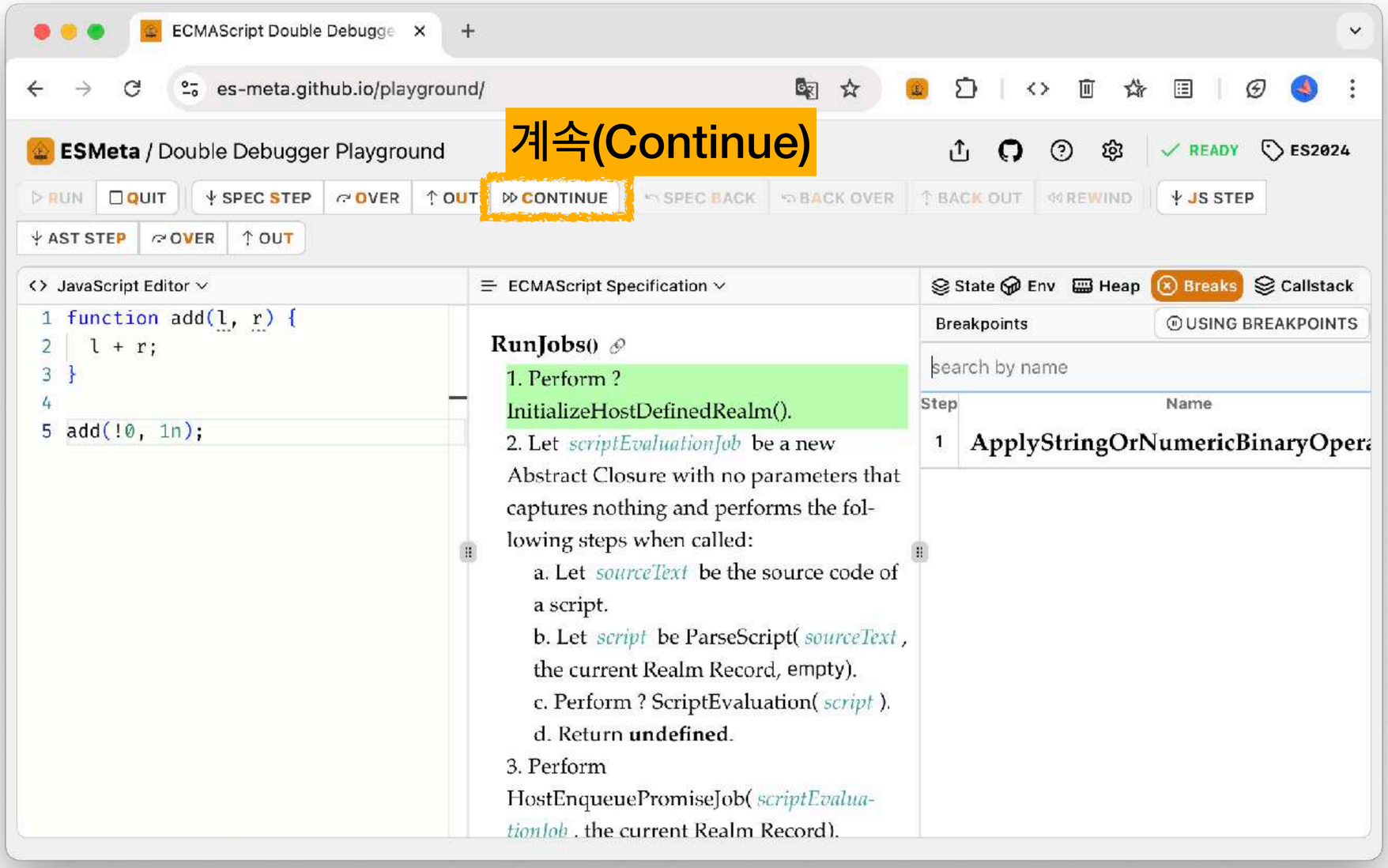

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
ECMAScript Specification
State
Env
Heap
Breaks
Callstack
Context Not Found
Disabled. Start debugger to use.
1 var x = 1;
2 var y = 2;
3 var z = x + y;
4 var w = z + x;
5
6 function f () {
7 let a = 42;
8 g(a);
9 return 0;
10 }
11
12 function g(a) {
13 a = 1;
14 a = 1;
15 a = 1;
16 a = 1;
17 a = 1;
18 a = 1;
19 a = 1;
1. 자바스크립트 프로그램 입력

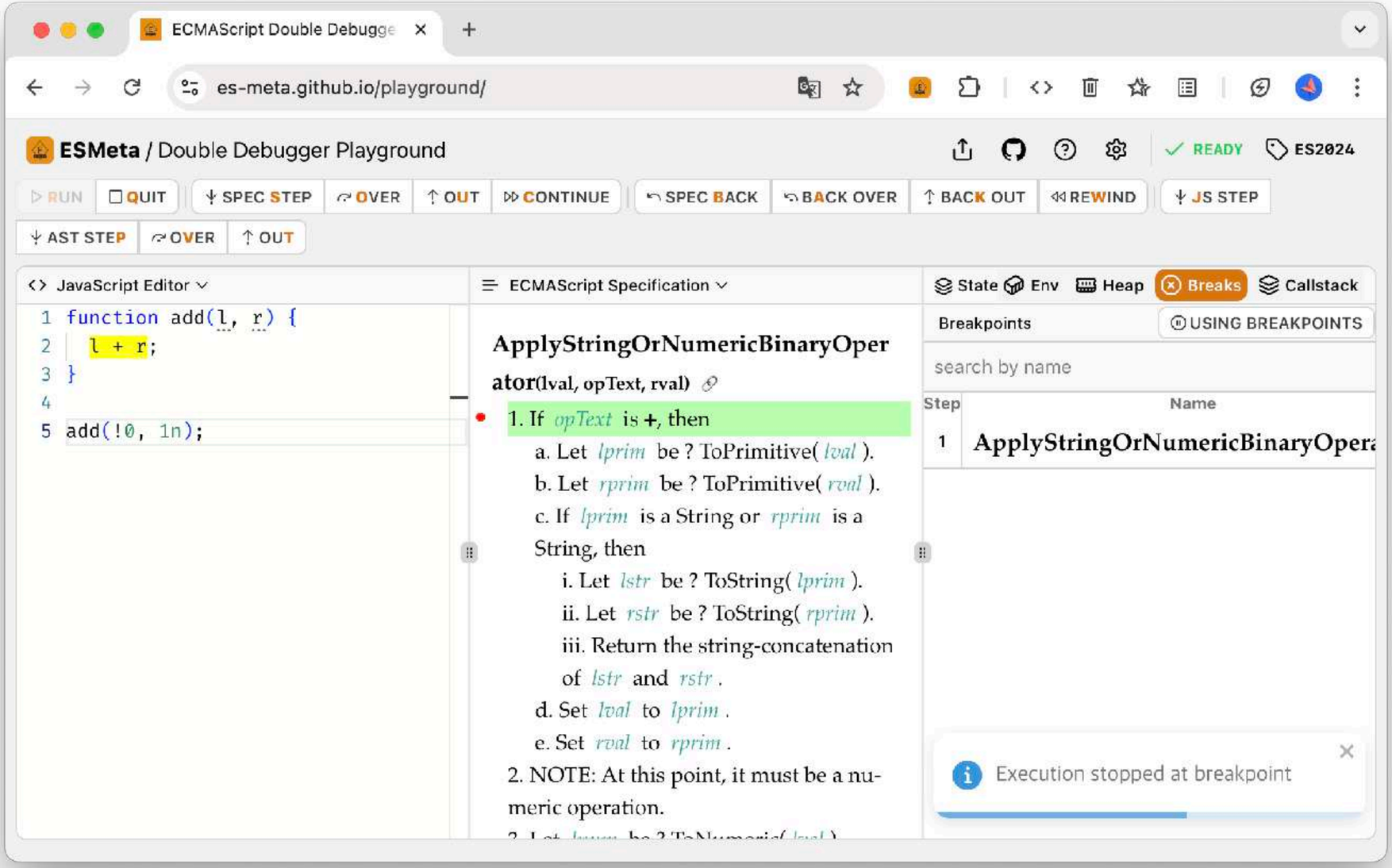

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
2. 이중 디버거 시작
JavaScript Editor
ECMAScript Specification
State
Env
Heap
Breaks
Callstack
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
Context Not Found
Disabled. Start debugger to use.
1. 자바스크립트 프로그램 입력

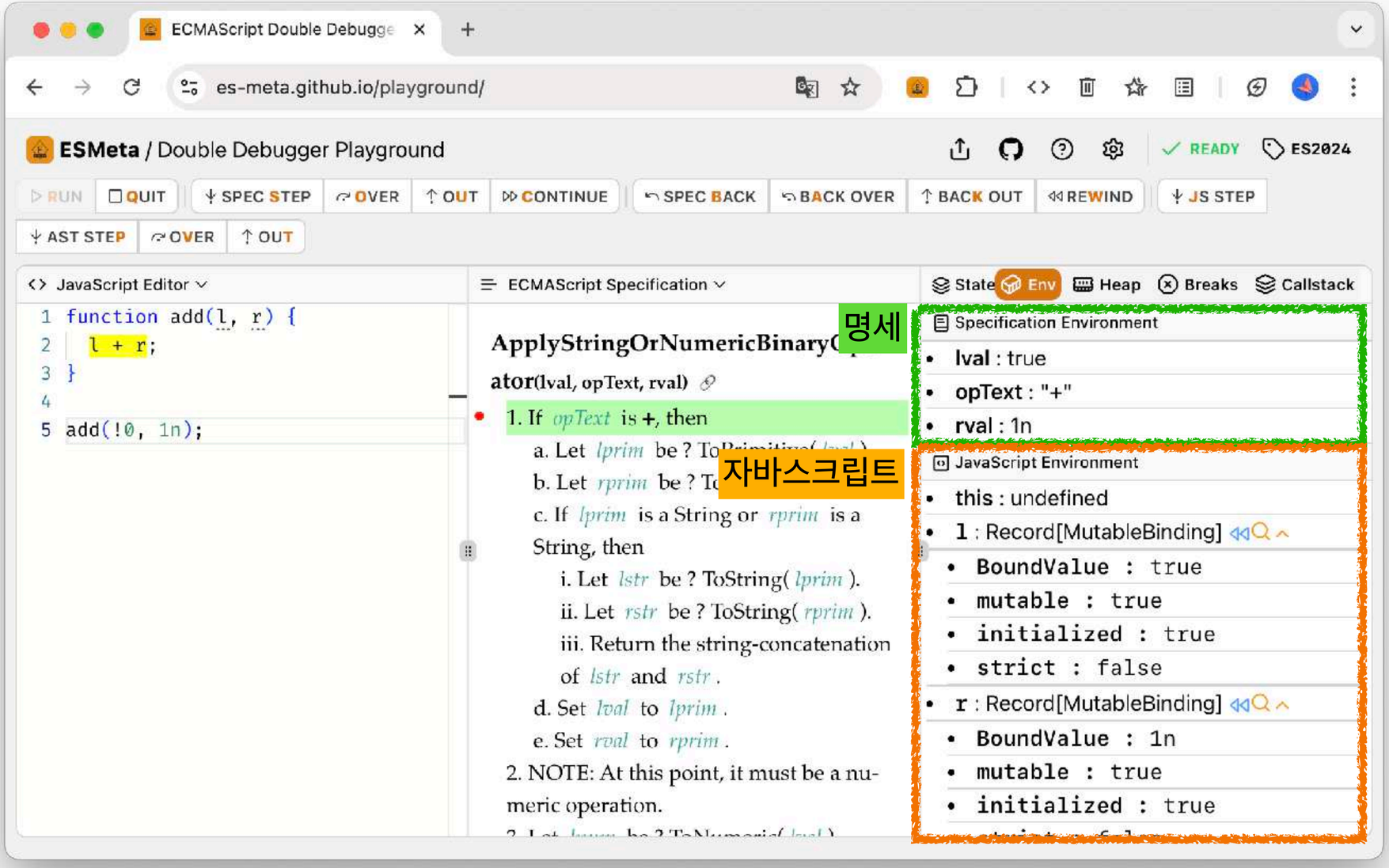

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
2. 이중 디버거 시작
중단점
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
RunJobs()
1. Perform ? InitializeHostDefinedRealm().
2. Let scriptEvaluationJob be a new Abstract Closure with no parameters that captures nothing and performs the following steps when called:
a. Let sourceText be the source code of
the current Realm Record, empty).
c. Perform ? ScriptEvaluation(script).
d. Return undefined.
3. Perform HostEnqueuePromiseJob(scriptEvaluationJob , the current Realm Record).
State
Env
Heap
Breaks
Callstack
Specification Environment
No environment variables.
JavaScript Environment
No environment variables.
1. 자바스크립트 프로그램 입력
Execution reached the front

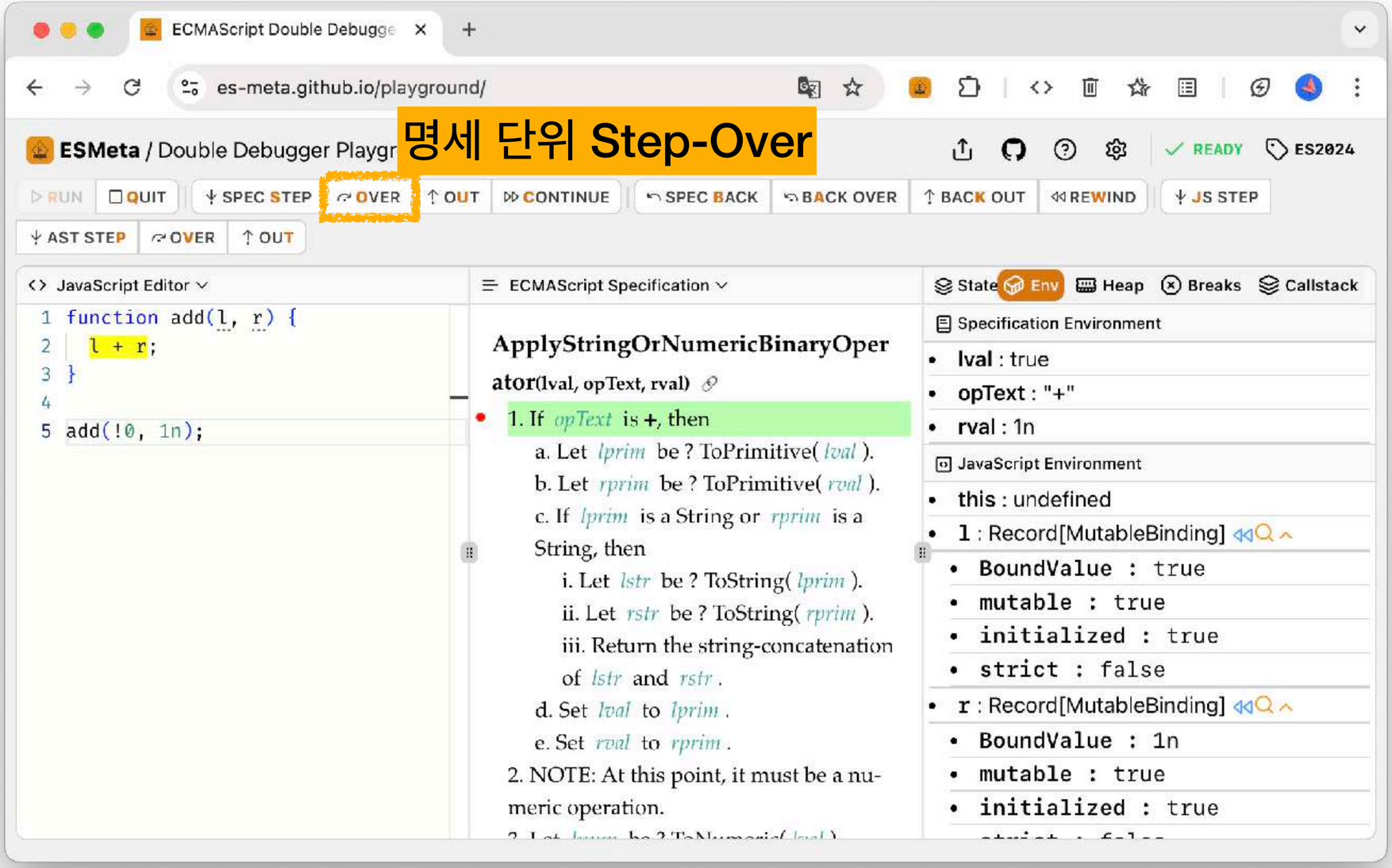

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
2. 이중 디버거 시작
중단점
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
RunJobs()
1. Perform ? InitializeHostDefinedRealm().
2. Let scriptEvaluationJob be a new Abstract Closure with no parameters that captures nothing and performs the following steps when called:
a. Let sourceText be the source code of
the current Realm Record, empty).
c. Perform ? ScriptEvaluation(script).
d. Return undefined.
3. Perform HostEnqueuePromiseJob(scriptEvaluationJob , the current Realm Record).
State
Env
Heap
Breaks
Callstack
Breakpoints
USING BREAKPOINTS
search by name
Step
Name
Enable
Remove
No breakpoints. Add Breakpoint by clicking on steps in spec viewer or by searching name
1. 자바스크립트 프로그램 입력

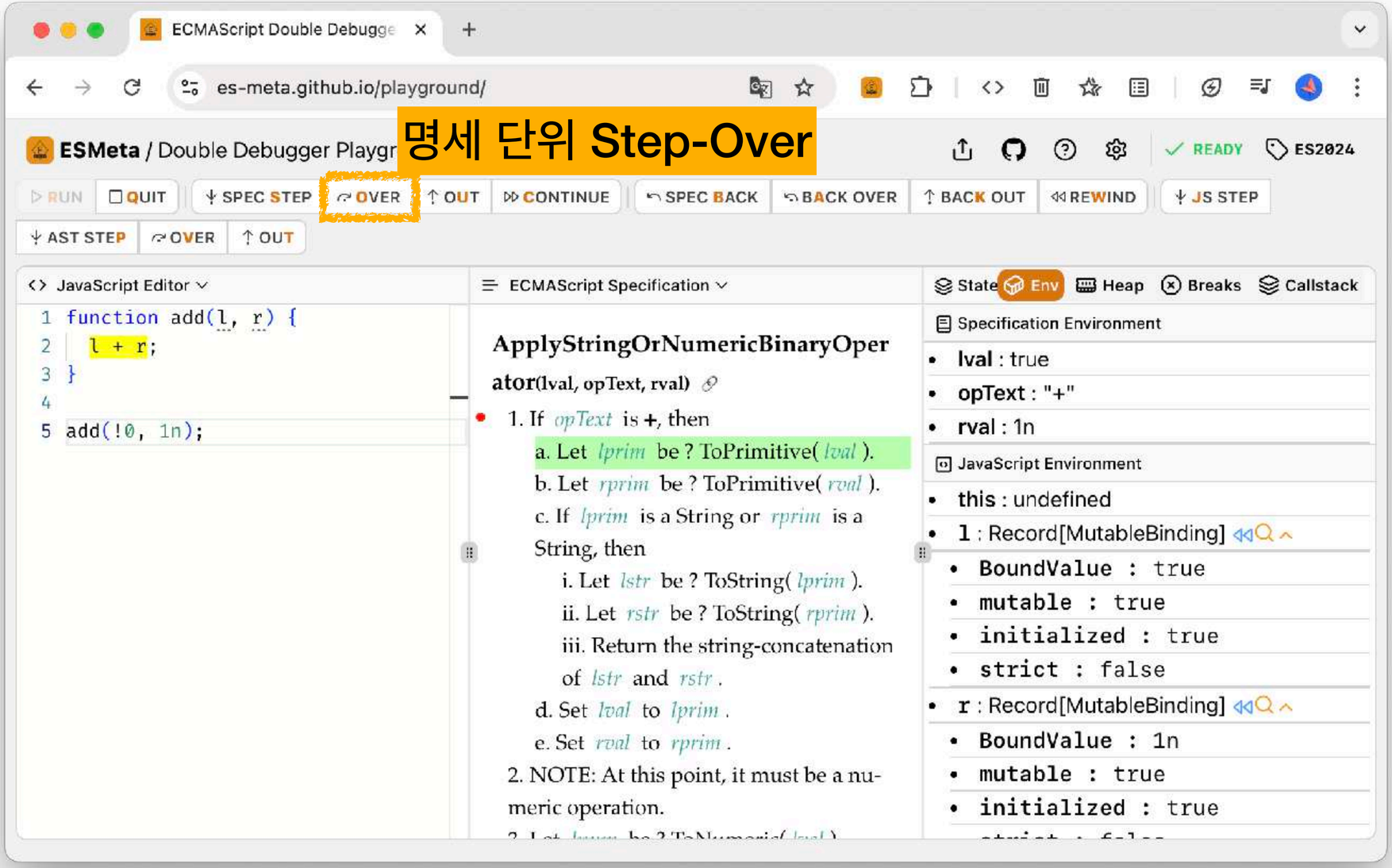

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
중단점
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
RunJobs()
1. Perform ? InitializeHostDefinedRealm().
2. Let scriptEvaluationJob be a new Abstract Closure with no parameters that captures nothing and performs the following steps when called:
a. Let sourceText be the source code of a script.
b. Let script be ParseScript(sourceText , the current Realm Record, empty).
c. Perform ? ScriptEvaluation(script).
d. Return undefined.
3. Perform HostEnqueuePromiseJob(scriptEvaluationJob , the current Realm Record).
State
Env
Heap
Breaks
Callstack
Breakpoints
USING BREAKPOINTS
applys
ApplyStringOrNumericBinaryOperator
ValidateAndApplyPropertyDescriptor
PropertyDestructuringAssignmentEvaluation
AssignmentPropertyList : AssignmentPropertyList , AssignmentProperty

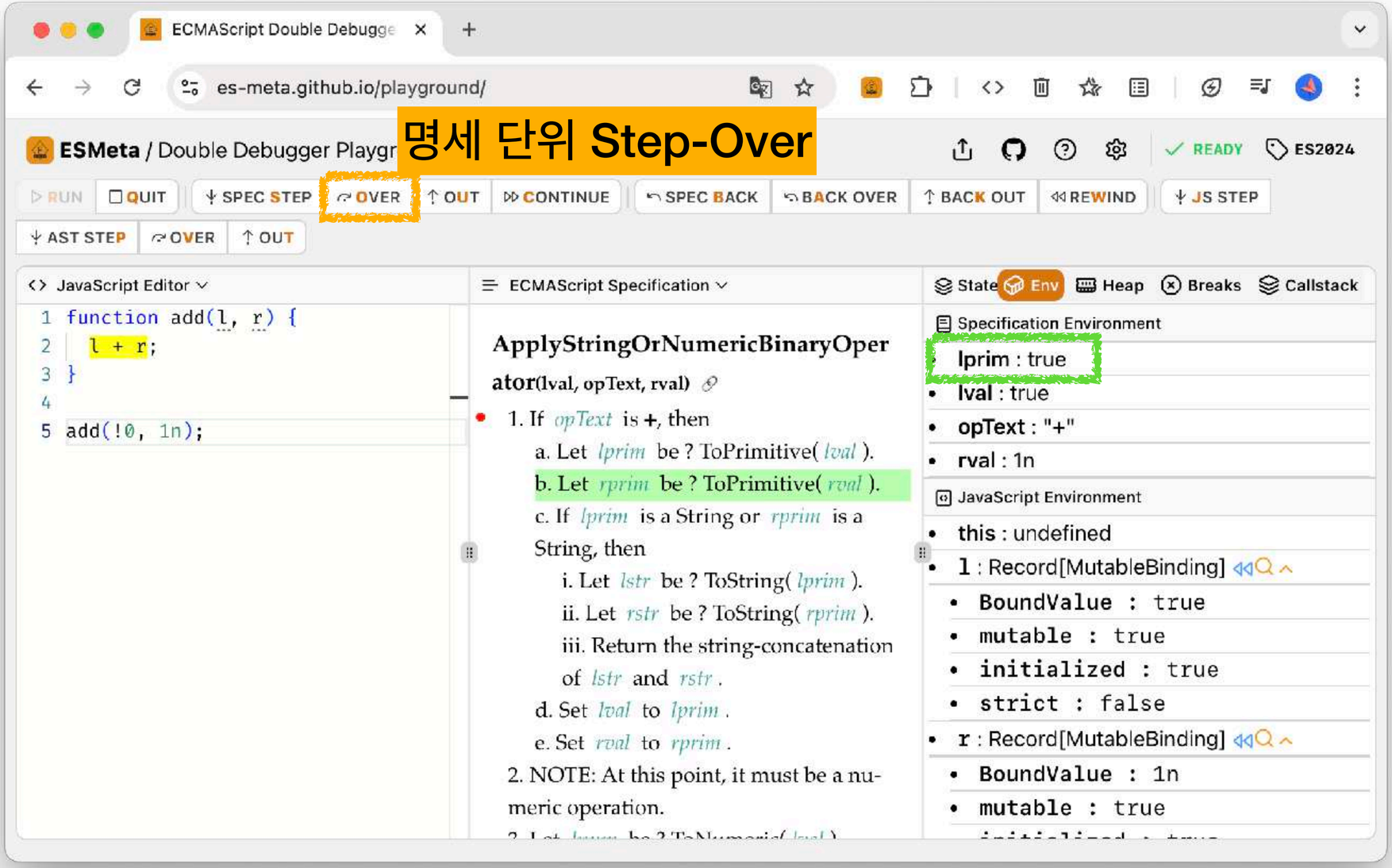

ECMAScript Double Debugge
es-meta.github.io/playground/
계속(Continue)
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
RunJobs()
1. Perform ? InitializeHostDefinedRealm().
2. Let scriptEvaluationJob be a new Abstract Closure with no parameters that captures nothing and performs the following steps when called:
a. Let sourceText be the source code of a script.
b. Let script be ParseScript(sourceText , the current Realm Record, empty).
c. Perform ? ScriptEvaluation(script).
d. Return undefined.
3. Perform HostEnqueuePromiseJob(scriptEvaluationJob , the current Realm Record).
State
Env
Heap
Breaks
Callstack
Breakpoints
USING BREAKPOINTS
search by name
Step
Name
1
ApplyStringOrNumericBinaryOpera

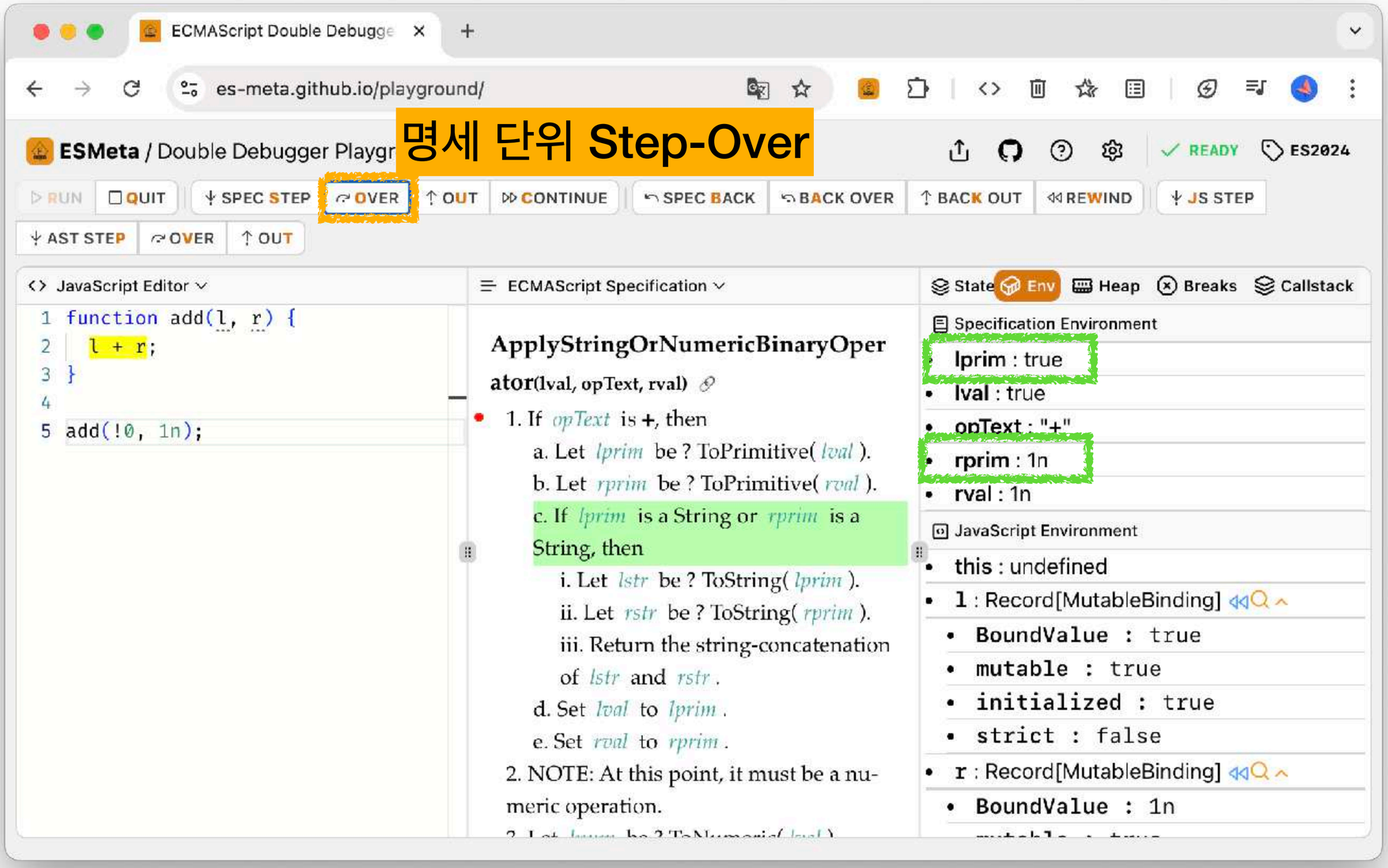

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
ApplyStringOrNumericBinaryOperator(lval, 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.
State
Env
Heap
Breaks
Callstack
Breakpoints
USING BREAKPOINTS
search by name
Step
Name
1
ApplyStringOrNumericBinaryOpera
Execution stopped at breakpoint

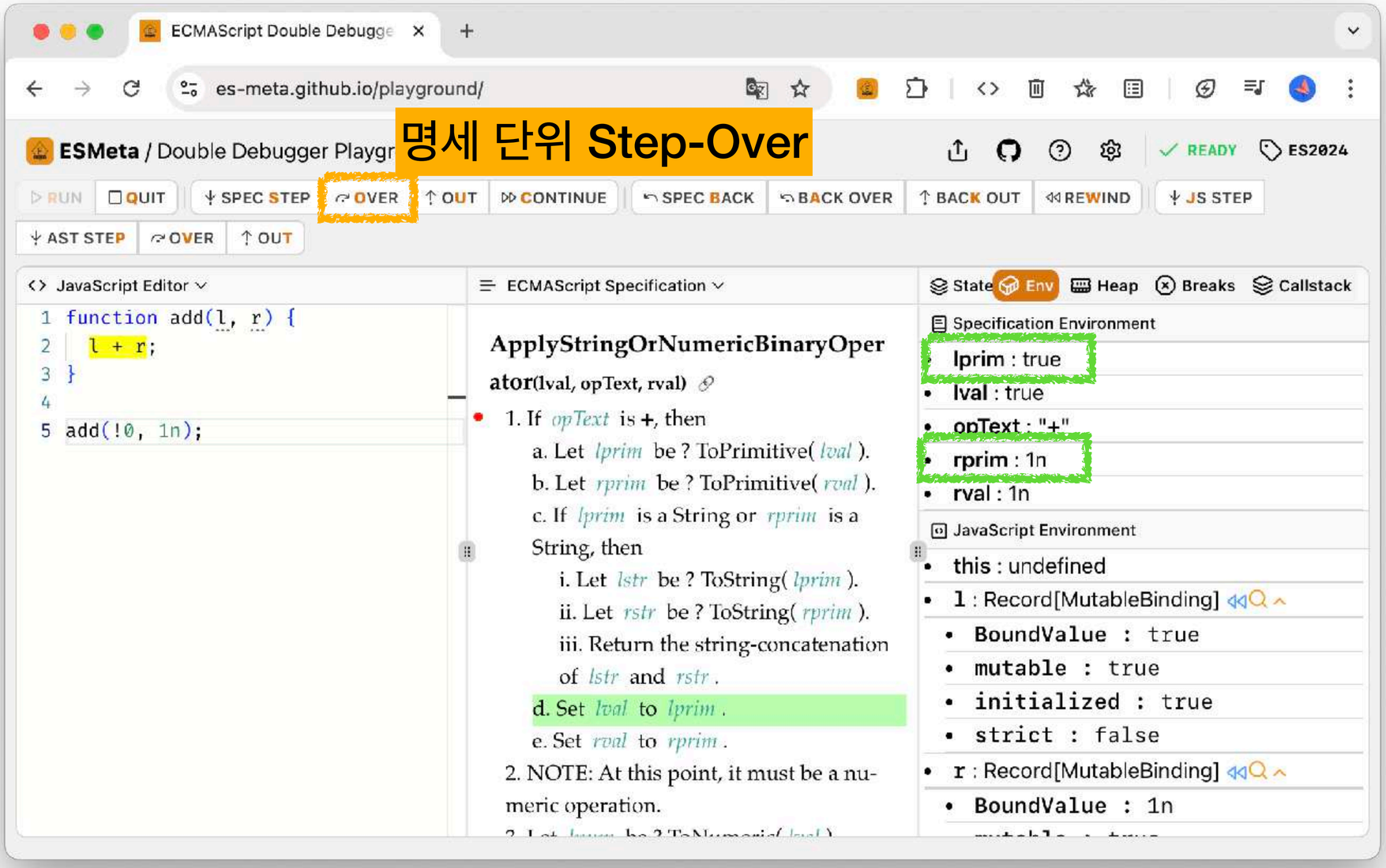

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
명세
ApplyStringOrNumericBinaryOperator(lval, opText, rval)
1. If opText is +, then
a. Let lprim be ? ToPrimitive(lval).
b. Let rprim be ? To
자바스크립트
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true
initialized : true

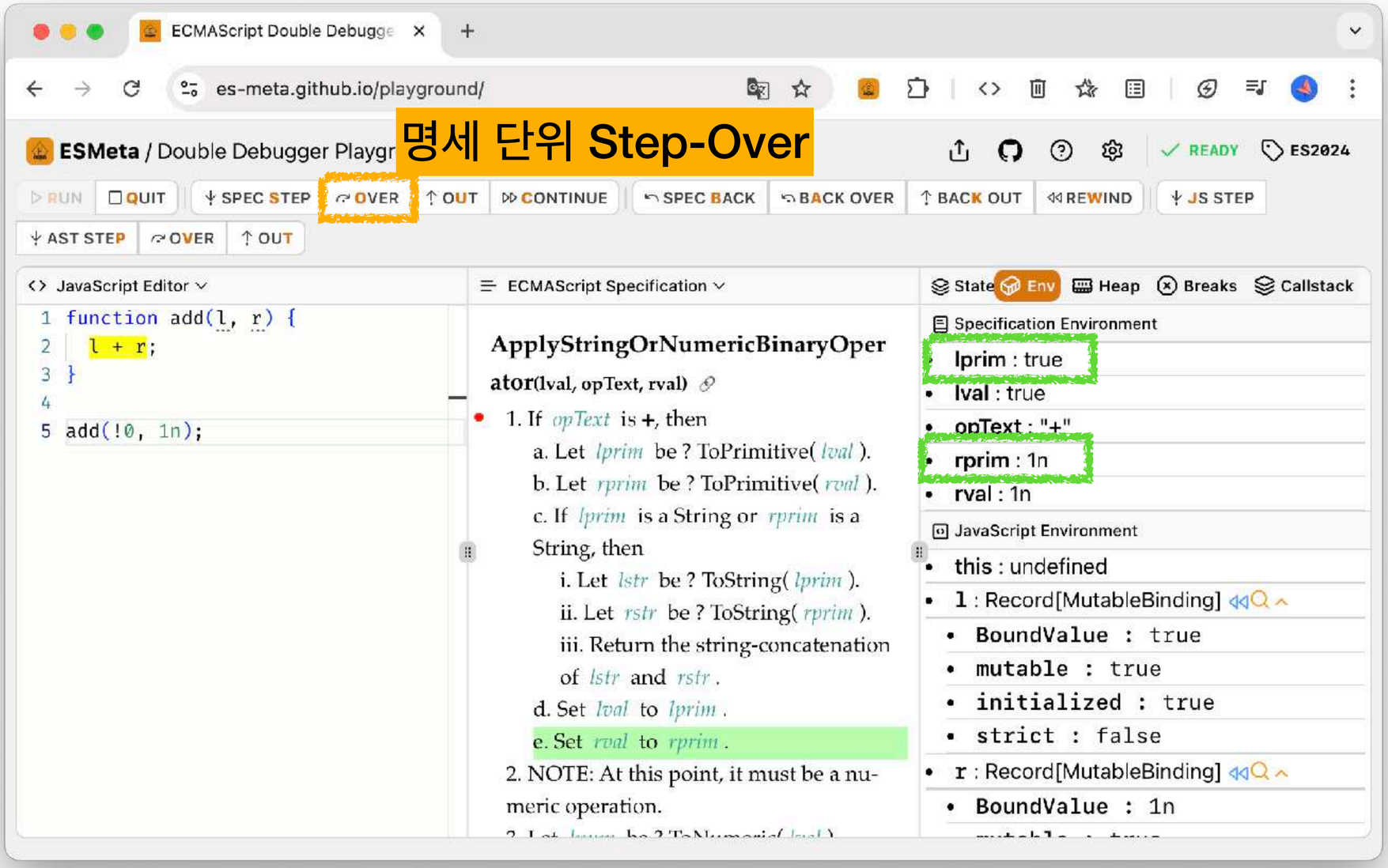

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true
initialized : true

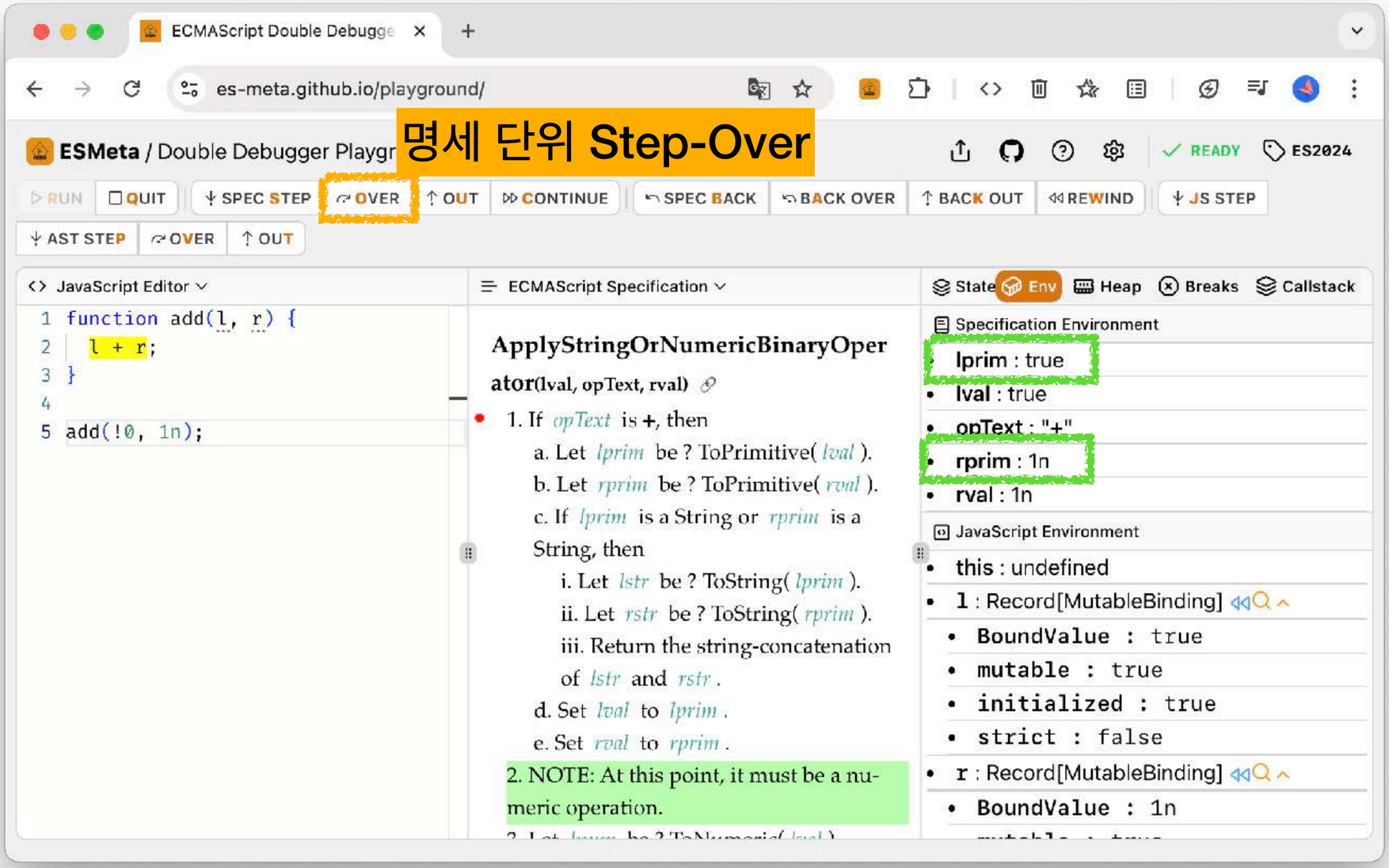

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true
initialized : true

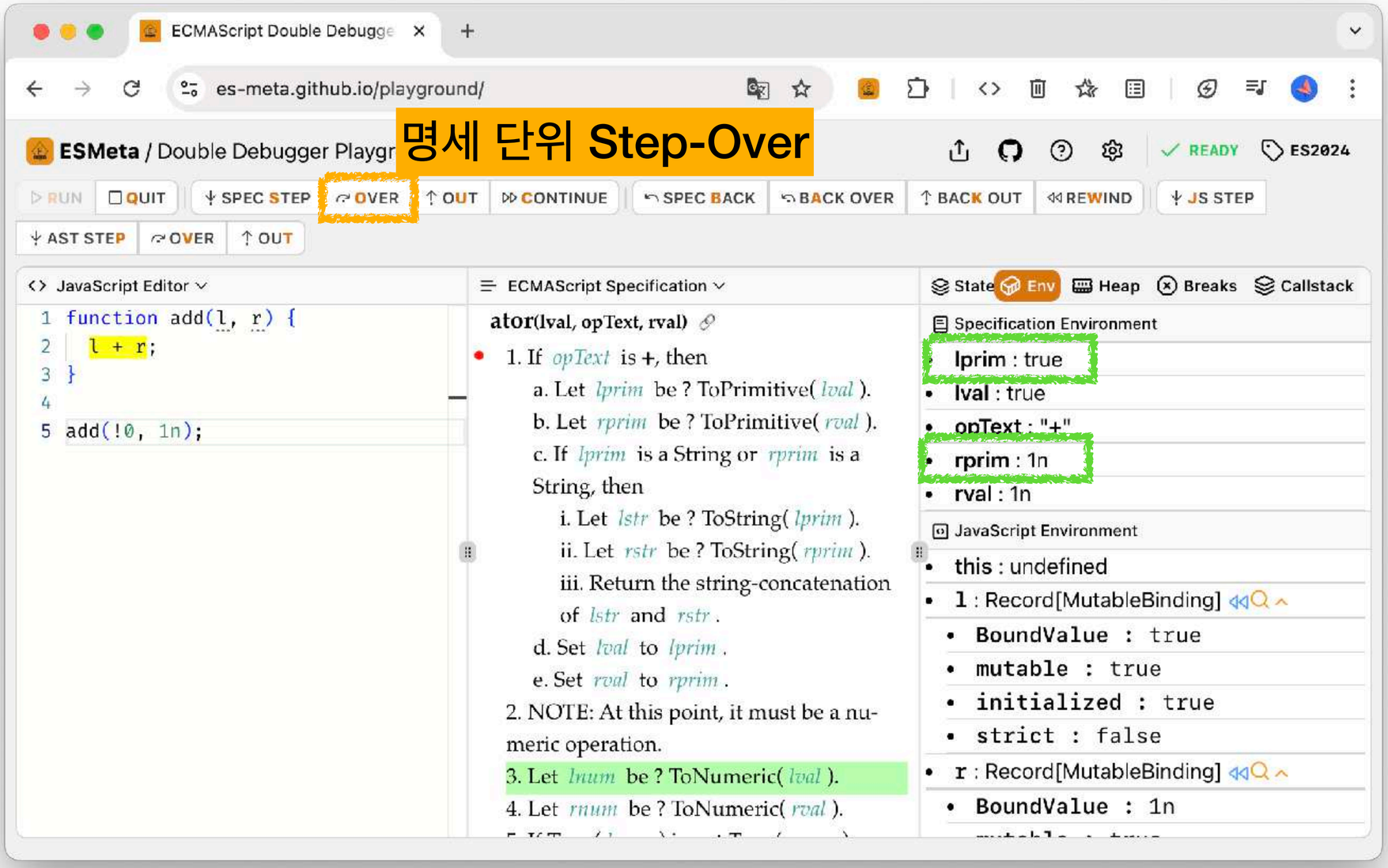

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true

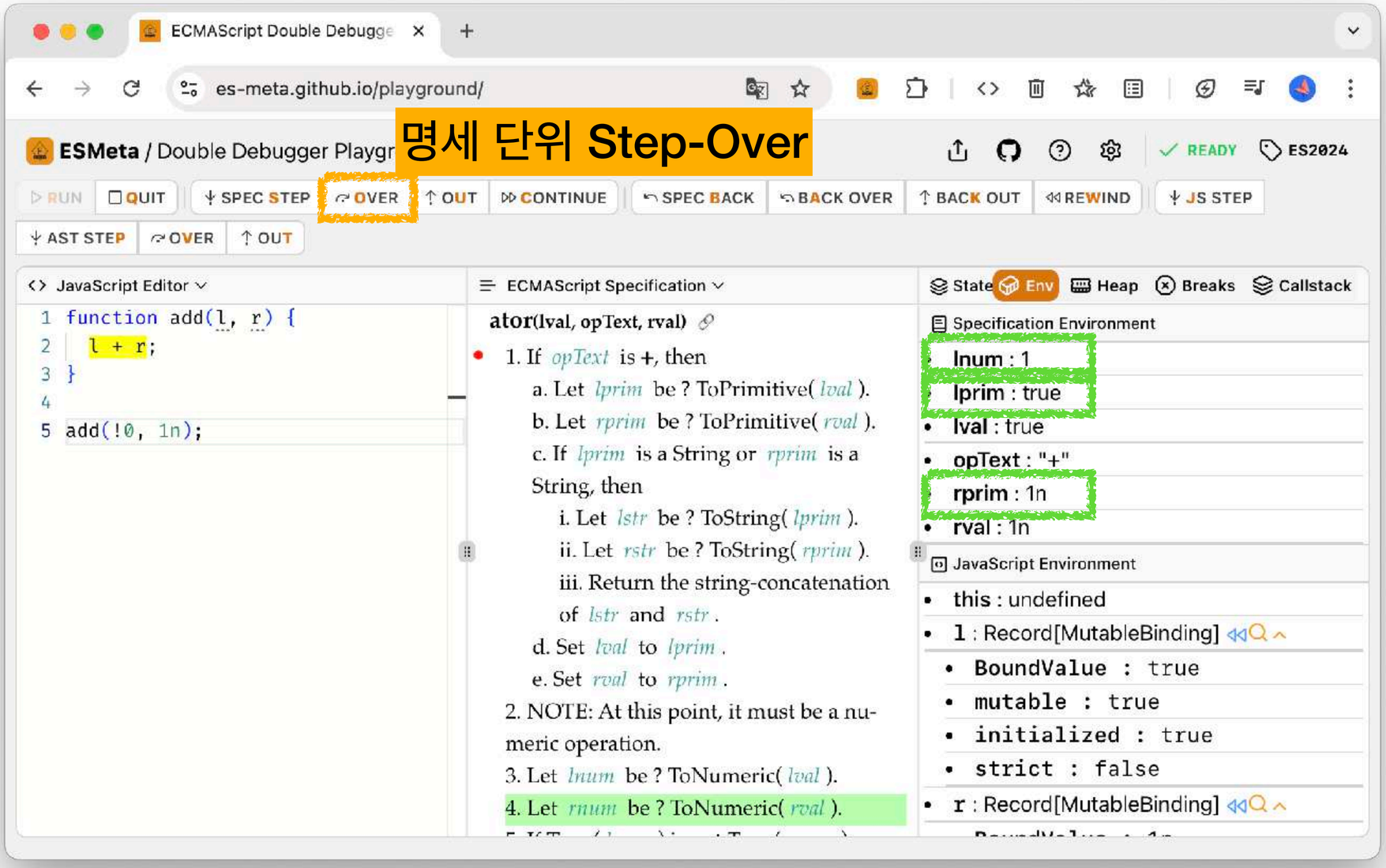

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n

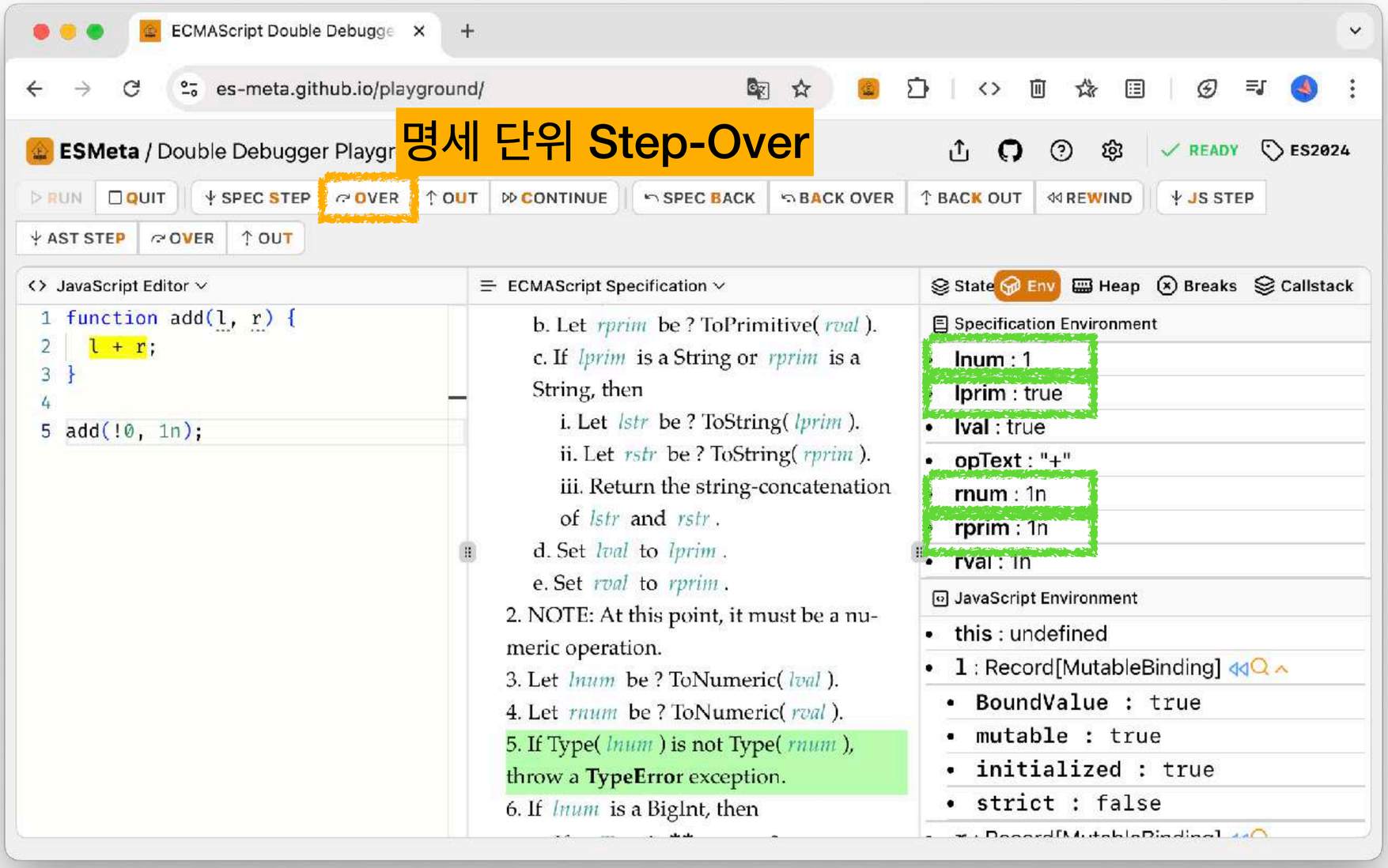

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n

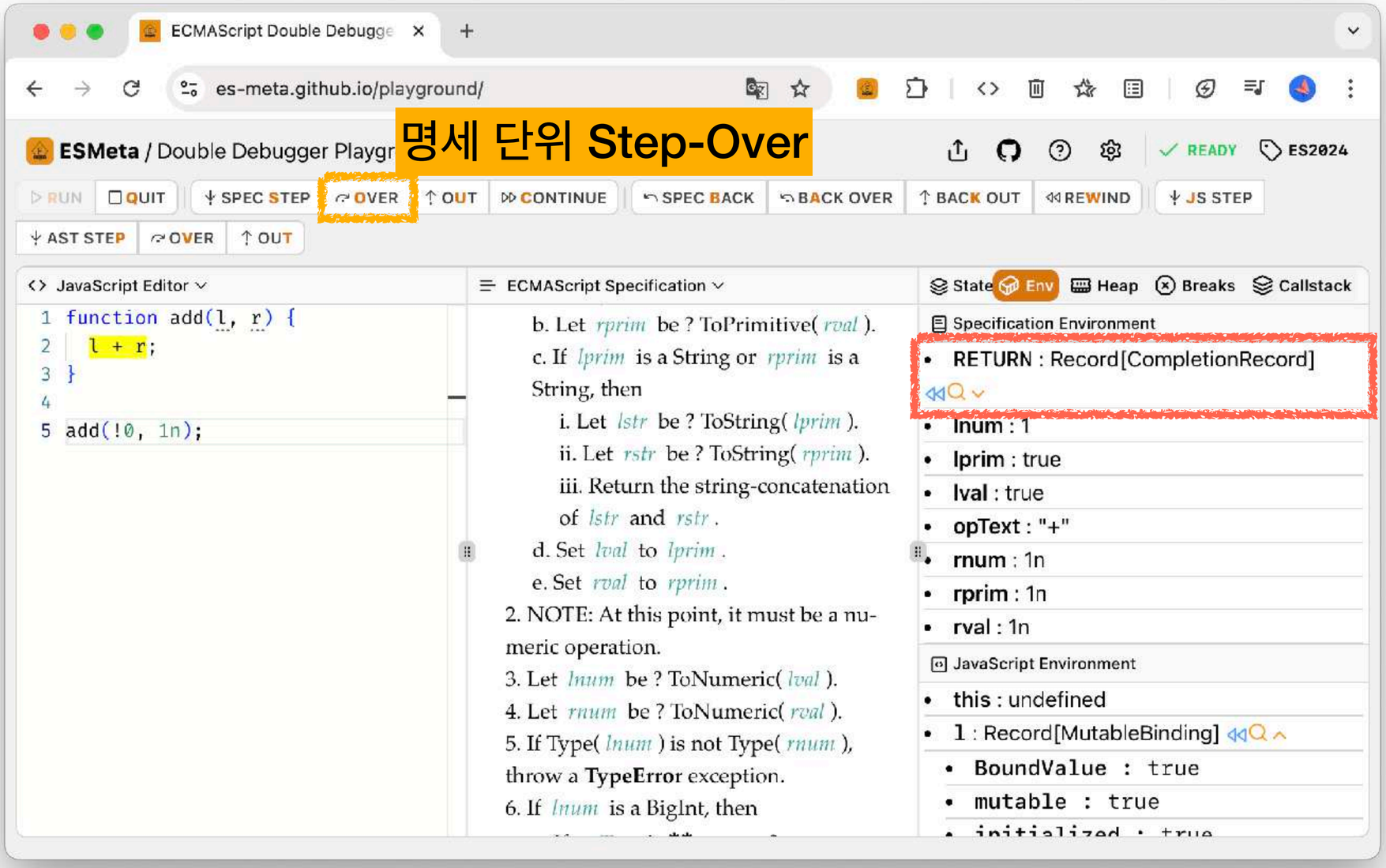

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n

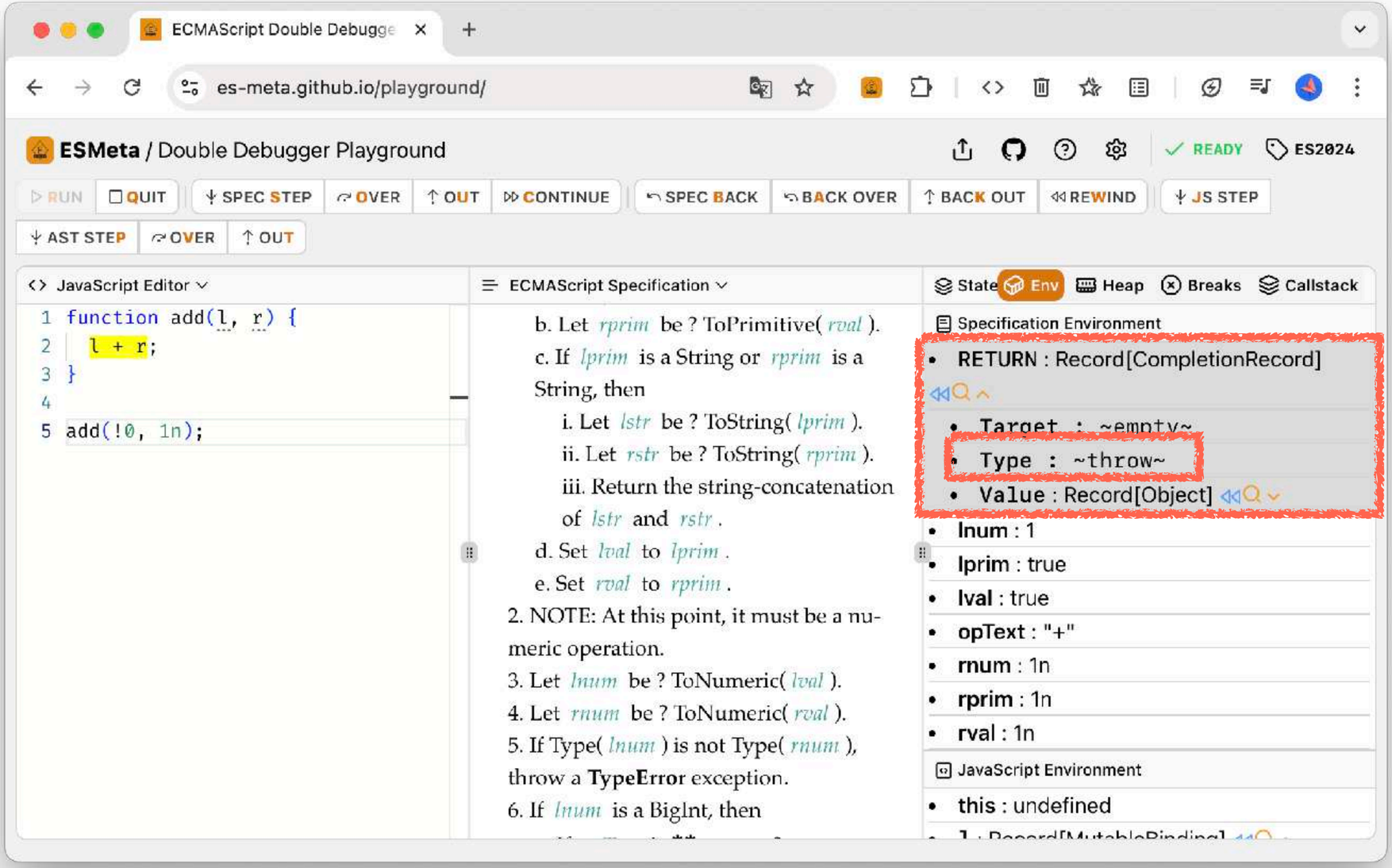

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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.
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n

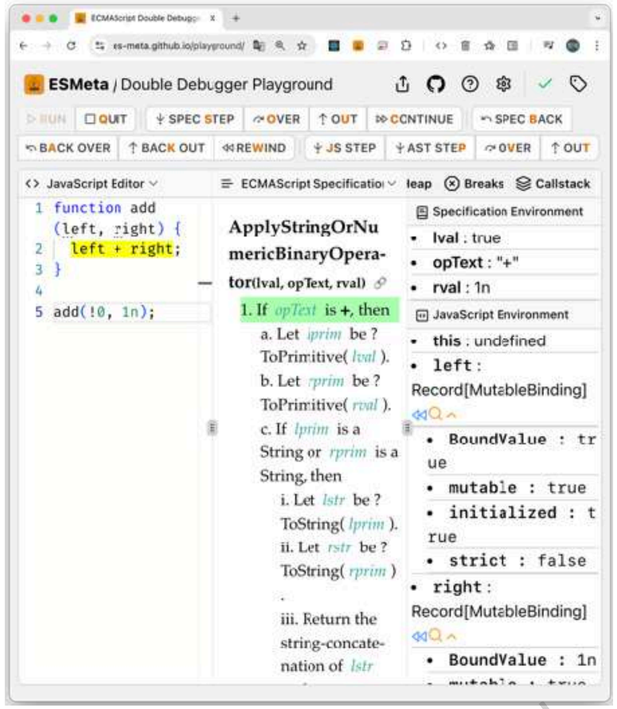

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
ator(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).
State
Env
Heap
Breaks
Callstack
Specification Environment
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n

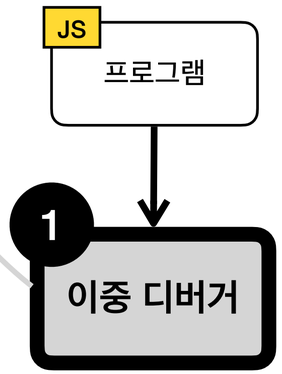

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
ator(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).
State
Env
Heap
Breaks
Callstack
Specification Environment
lnum : 1
lprim : true
lval : true
opText : "+"
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]


ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playgr
명세 단위 Step-Over
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
ECMAScript Specification
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
State
Env
Heap
Breaks
Callstack
Specification Environment
lnum : 1
lprim : true
lval : true
opText : "+"
rnum : 1n
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false


ECMAScript Double Debugge
es-meta.github.io/playground/
명세 단위 Step-Over
ESMeta / Double Debugger Playgr
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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
State
Env
Heap
Breaks
Callstack
Specification Environment
RETURN : Record[CompletionRecord]
lnum : 1
lprim : true
lval : true
opText : "+"
rnum : 1n
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined
1 : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true


ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
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
State
Env
Heap
Breaks
Callstack
Specification Environment
RETURN : Record[CompletionRecord]
Target : ~empty~
Type : ~throw~
Value : Record[Object]
lnum : 1
lprim : true
lval : true
opText : "+"
rnum : 1n
rprim : 1n
rval : 1n
JavaScript Environment
this : undefined


ESMeta / Double Debugger Playground
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
ECMAScript Specificatio
leap
Breaks
Callstack
1 function add
(left, right) {
2 left + right;
3 }
4
5 add(!0, 1n);
ApplyStringOrNu
mericBinaryOpera-
tor(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-concate-
nation of lstr
Specification Environment
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
left :
Record[MutableBinding]
BoundValue : tr
ue
mutable : true
initialized : t
rue
strict : false
right :
Record[MutableBinding]
BoundValue : 1n


JS
프로그램
1
이중 디버거

Runtime Semantics: Evaluation

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*).

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
      1. Let *lstr* be ? ToString(*lprim*).
      2. Let *rstr* be ? ToString(*rprim*).
      3. 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
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

…

[[Type]]이 **normal이 아니면** return, normal이면 [[Value]] 값 사용

syntactic sugar of:

return { [[Type]] : throw , [[Value]] : … , … }

Runtime Semantics: Evaluation

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*).

ApplyStringOrNumericBinaryOperator (*lval*, *opText*, ...

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
      1. Let *lstr* be ? ToString(*lprim*).
      2. Let *rstr* be ? ToString(*rprim*).
      3. Return the string-concatenation of *lstr* and *r*
   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

...

명세에 예제 자바스크립트 프로그램이 함께 제공된다면 어떨까?

```
1 + 1
```

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

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

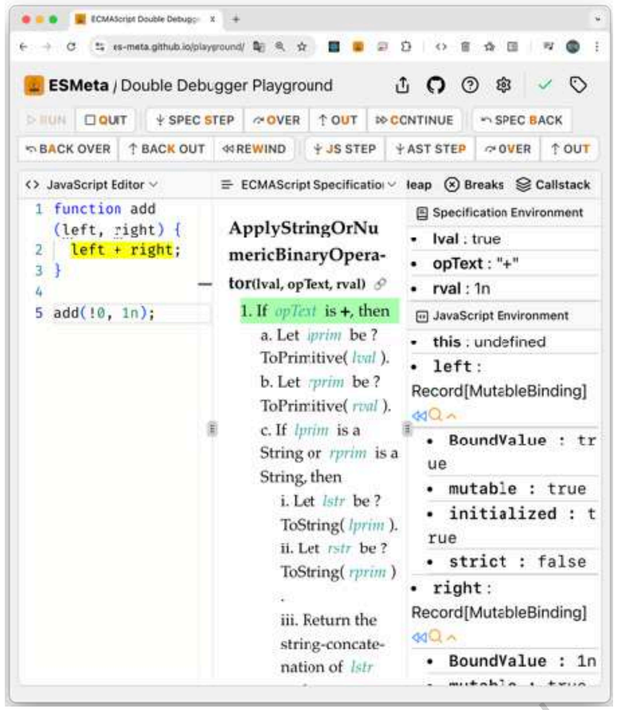
```
[] + 0
```

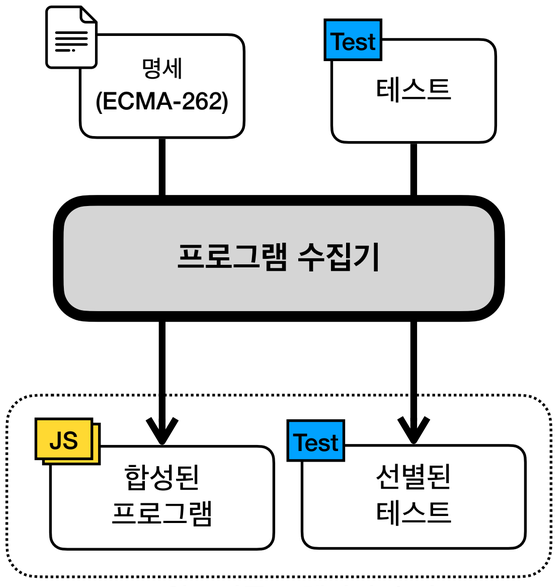
```
{ [Symbol.toPrimitive] : 0 } * 0
```

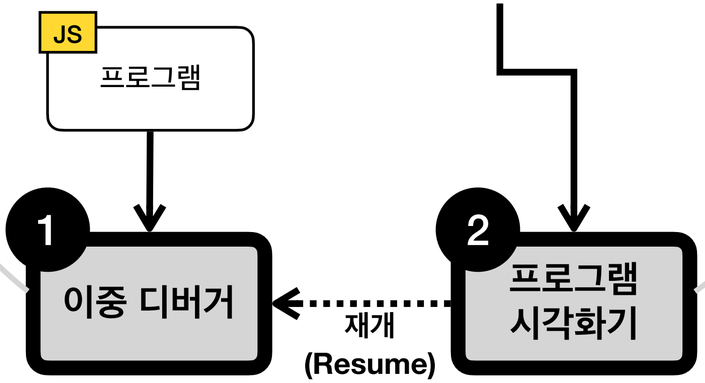
```
0 * { [Symbol.toPrimitive] : 0 }
```

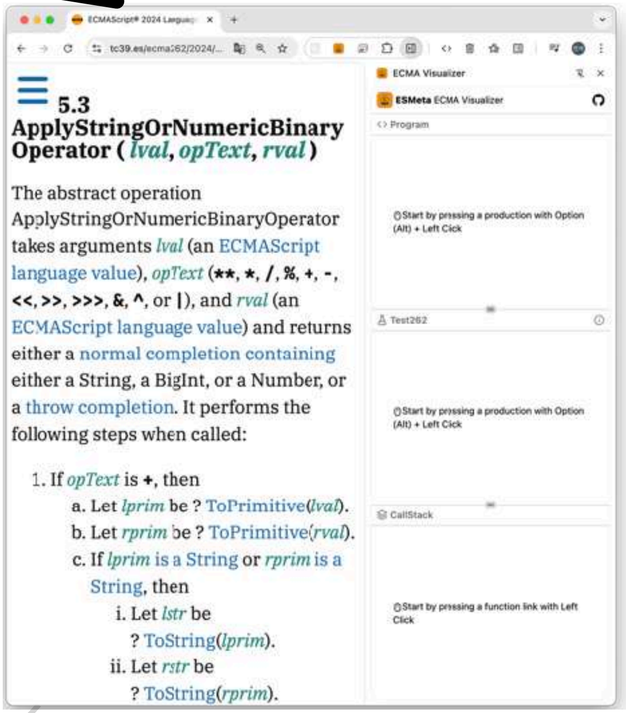
```
0 + 1n
```

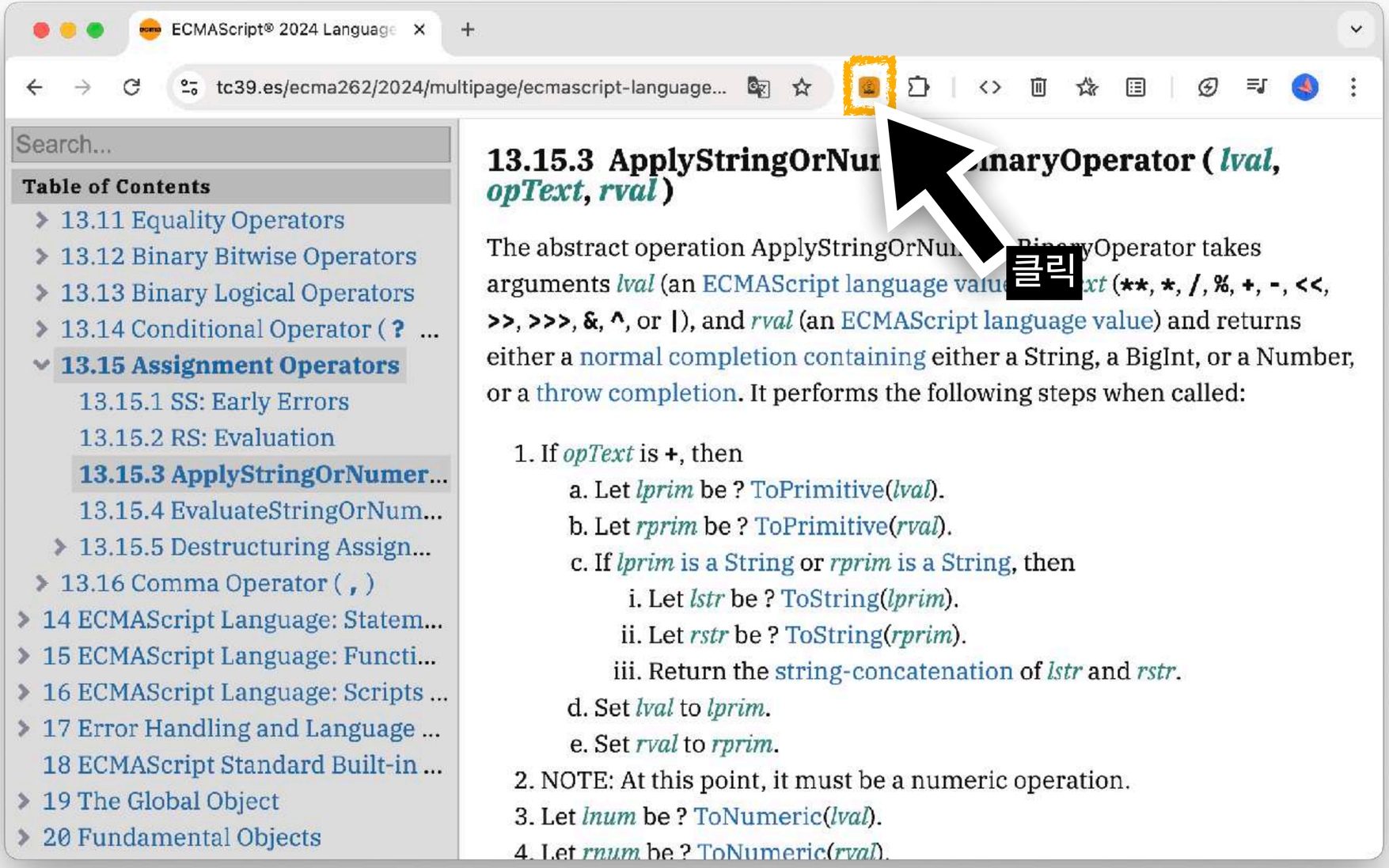
Solution 2. 프로그램 시각화기

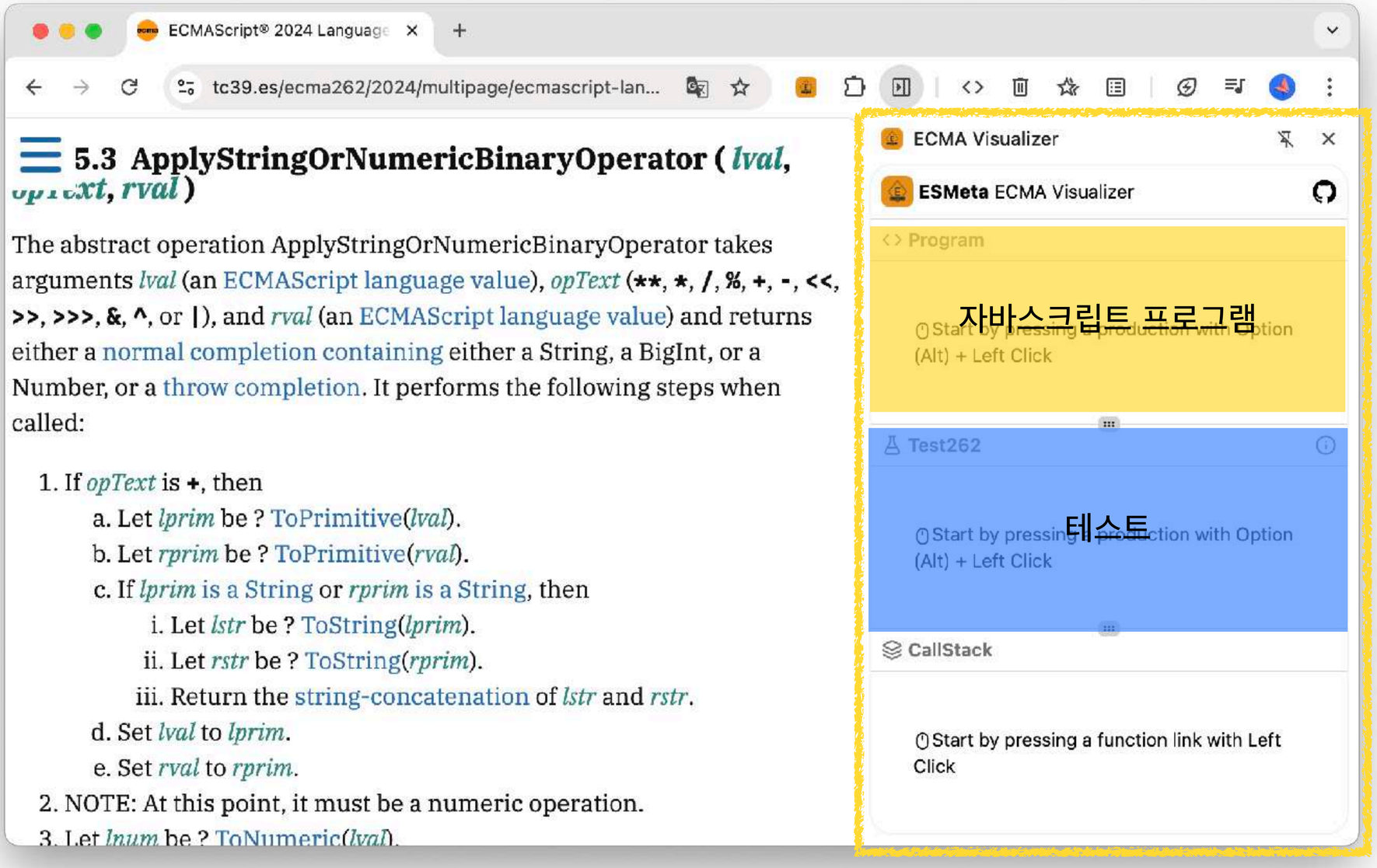


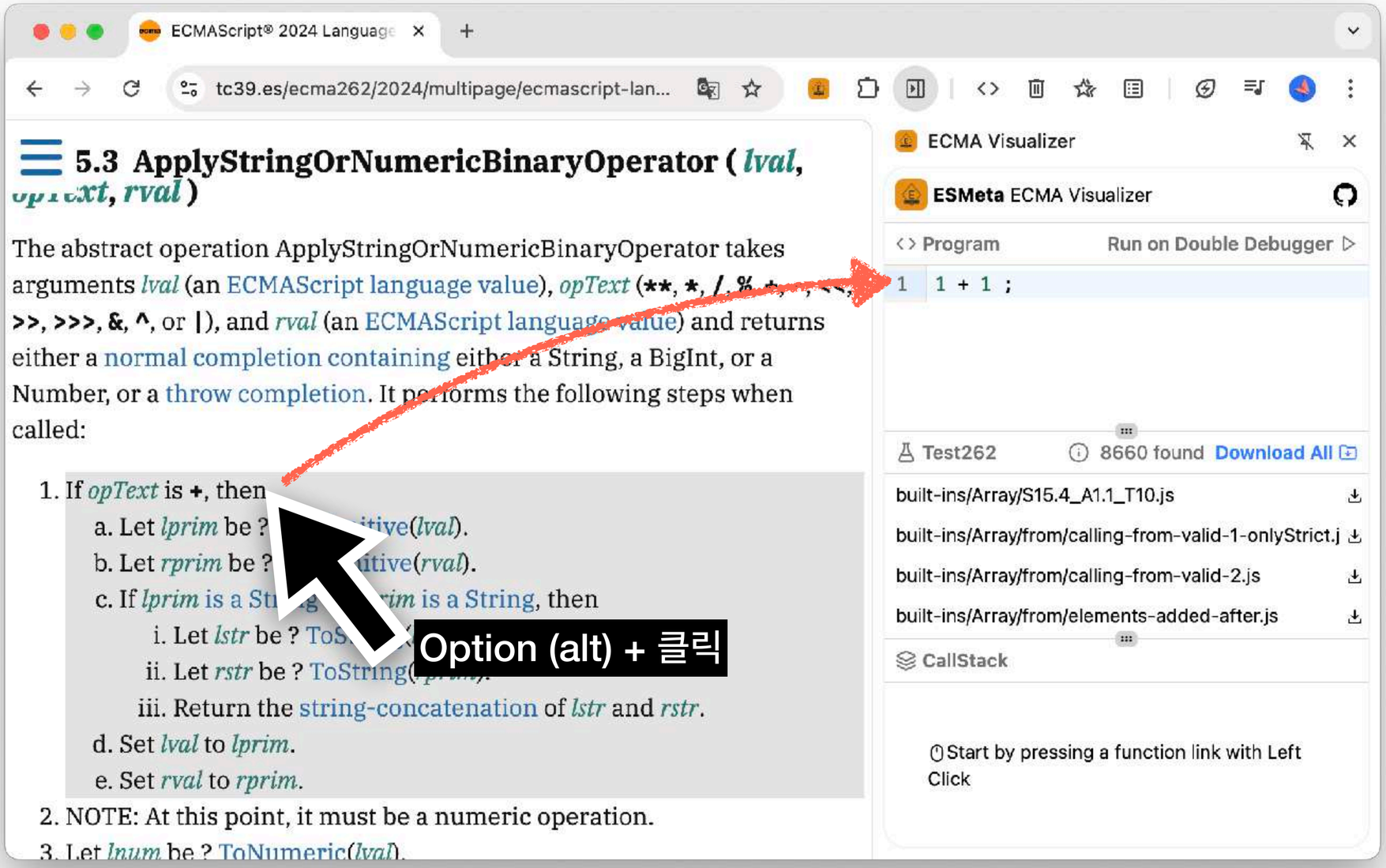


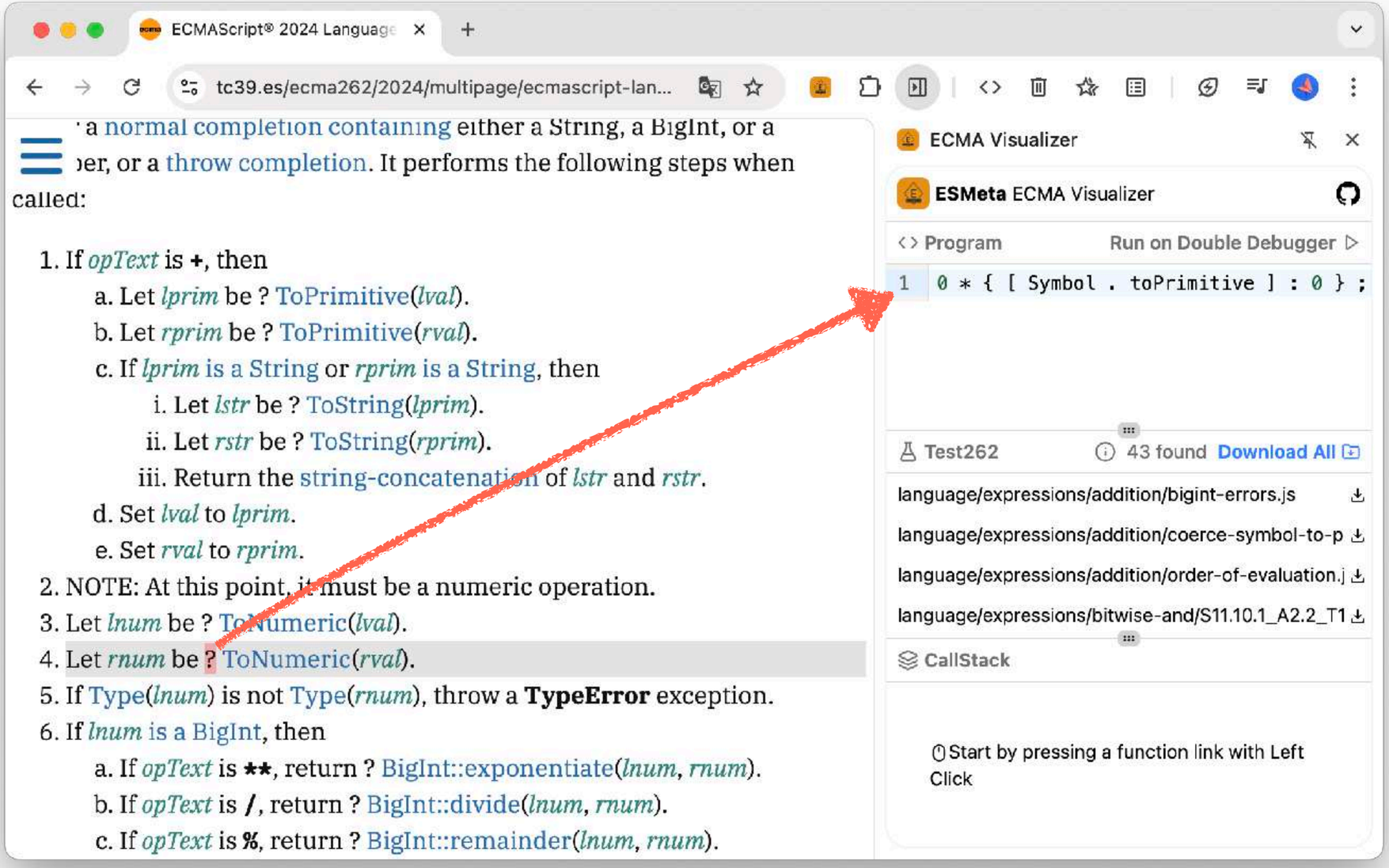


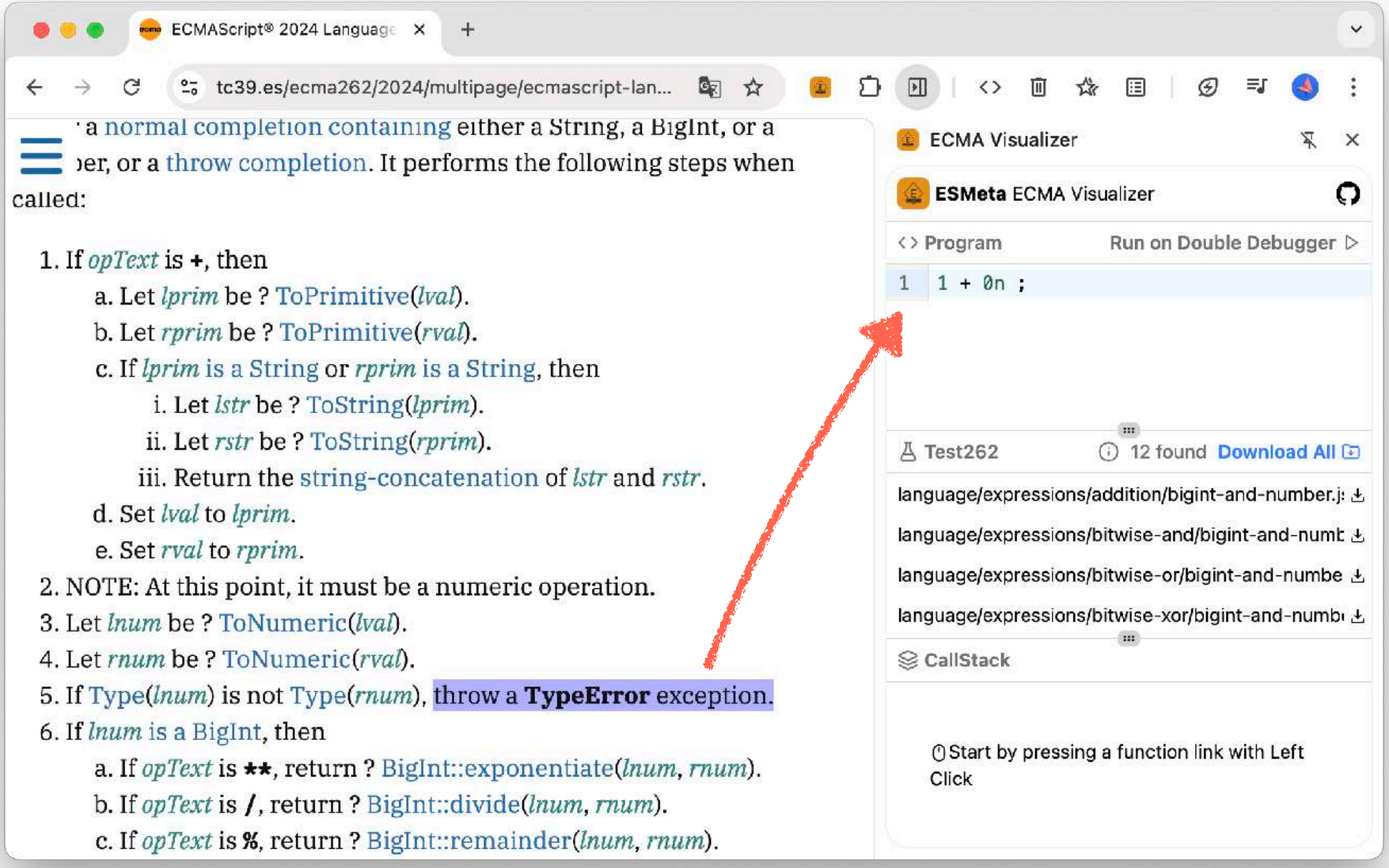


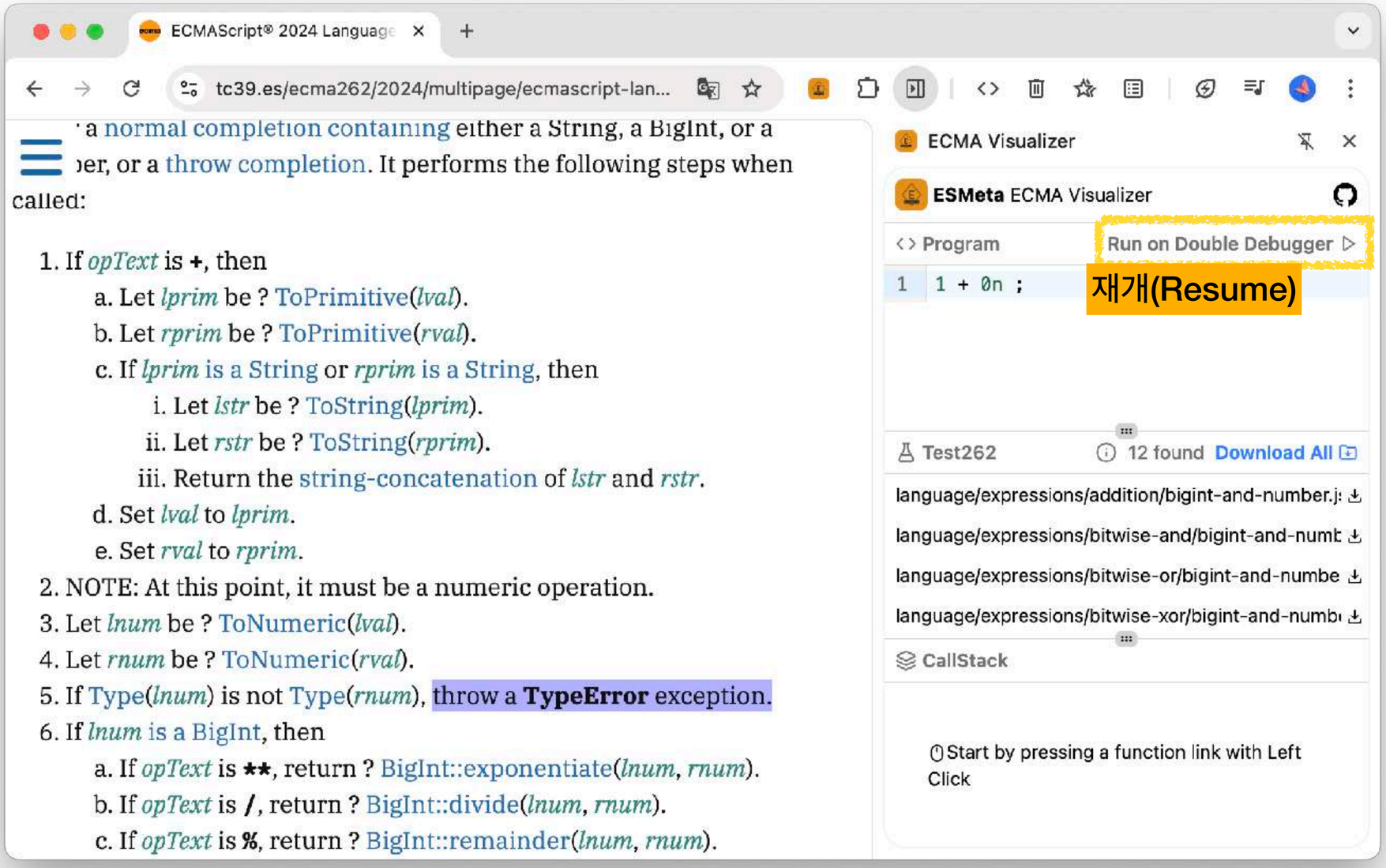

ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-language...
Search...
Table of Contents
13.11 Equality Operators
13.12 Binary Bitwise Operators
13.13 Binary Logical Operators
13.14 Conditional Operator (? ...
13.15 Assignment Operators
13.15.1 SS: Early Errors
13.15.2 RS: Evaluation
13.15.3 ApplyStringOrNumer...
13.15.4 EvaluateStringOrNum...
13.15.5 Destructuring Assign...
13.16 Comma Operator (,)
14 ECMAScript Language: Statem...
15 ECMAScript Language: Functi...
16 ECMAScript Language: Scripts ...
17 Error Handling and Language ...
18 ECMAScript Standard Built-in ...
19 The Global Object
20 Fundamental Objects
13.15.3 ApplyStringOrNu inaryOperator (lval, opText, rval)
The abstract operation ApplyStringOrNu BinaryOperator takes arguments lval (an ECMAScript language valu xt (**, *, /, %, +, -, <<, >>, >>>, &, ^, or |), and rval (an ECMAScript language value) and returns either a normal completion containing either a String, a BigInt, or a Number, or a throw completion. It performs the following steps when called:
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).
클릭

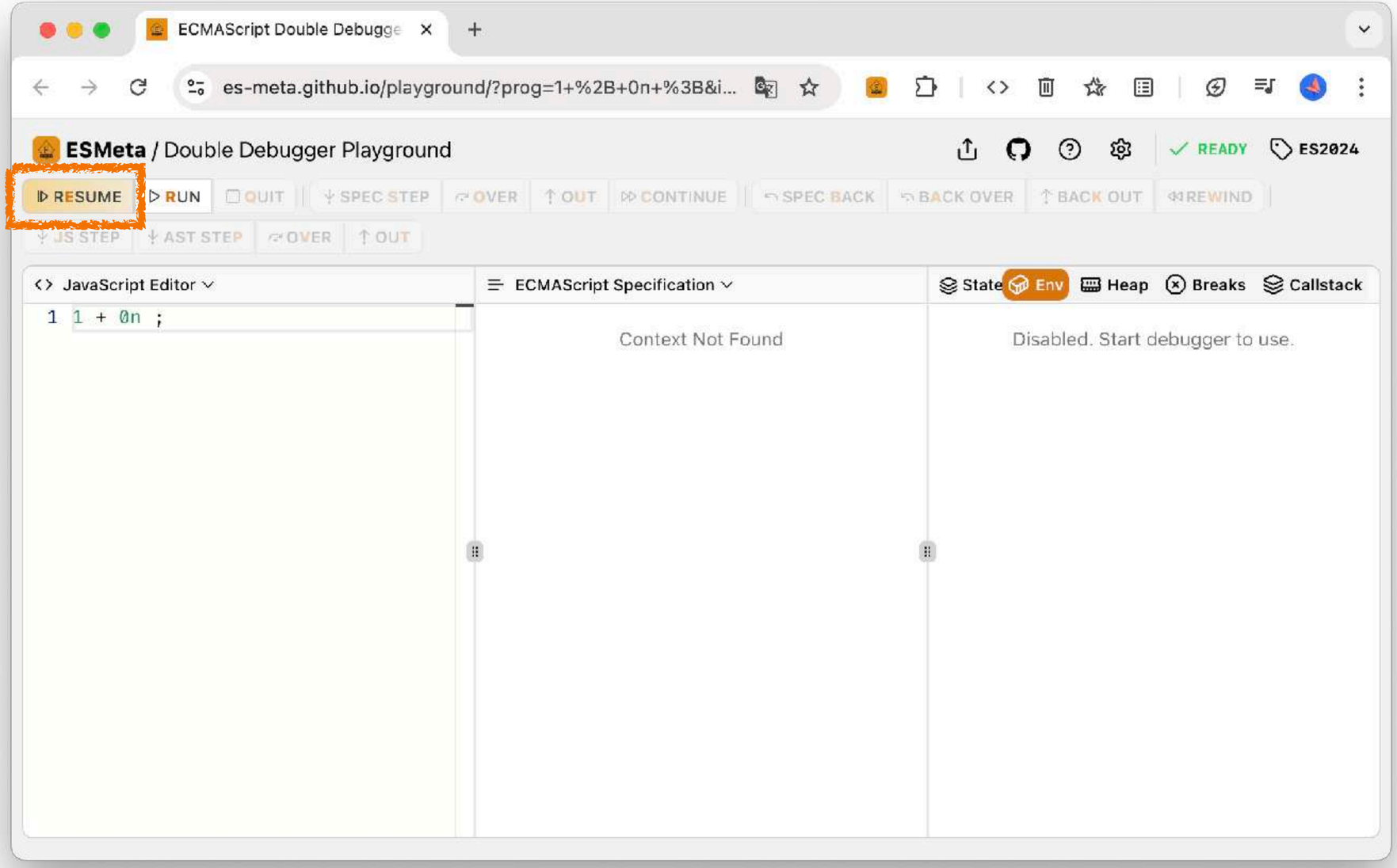

ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-lan...
5.3 ApplyStringOrNumericBinaryOperator (lval, opText, rval)
The abstract operation ApplyStringOrNumericBinaryOperator takes arguments lval (an ECMAScript language value), opText (**, *, /, %, +, -, <<, >>, >>>, &, ^, or |), and rval (an ECMAScript language value) and returns either a normal completion containing either a String, a BigInt, or a Number, or a throw completion. It performs the following steps when called:
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).
ECMA Visualizer
ESMeta ECMA Visualizer
Program
자바스크립트 프로그램
(Alt) + Left Click
Test262
테스트
(Alt) + Left Click
CallStack
Start by pressing a function link with Left Click

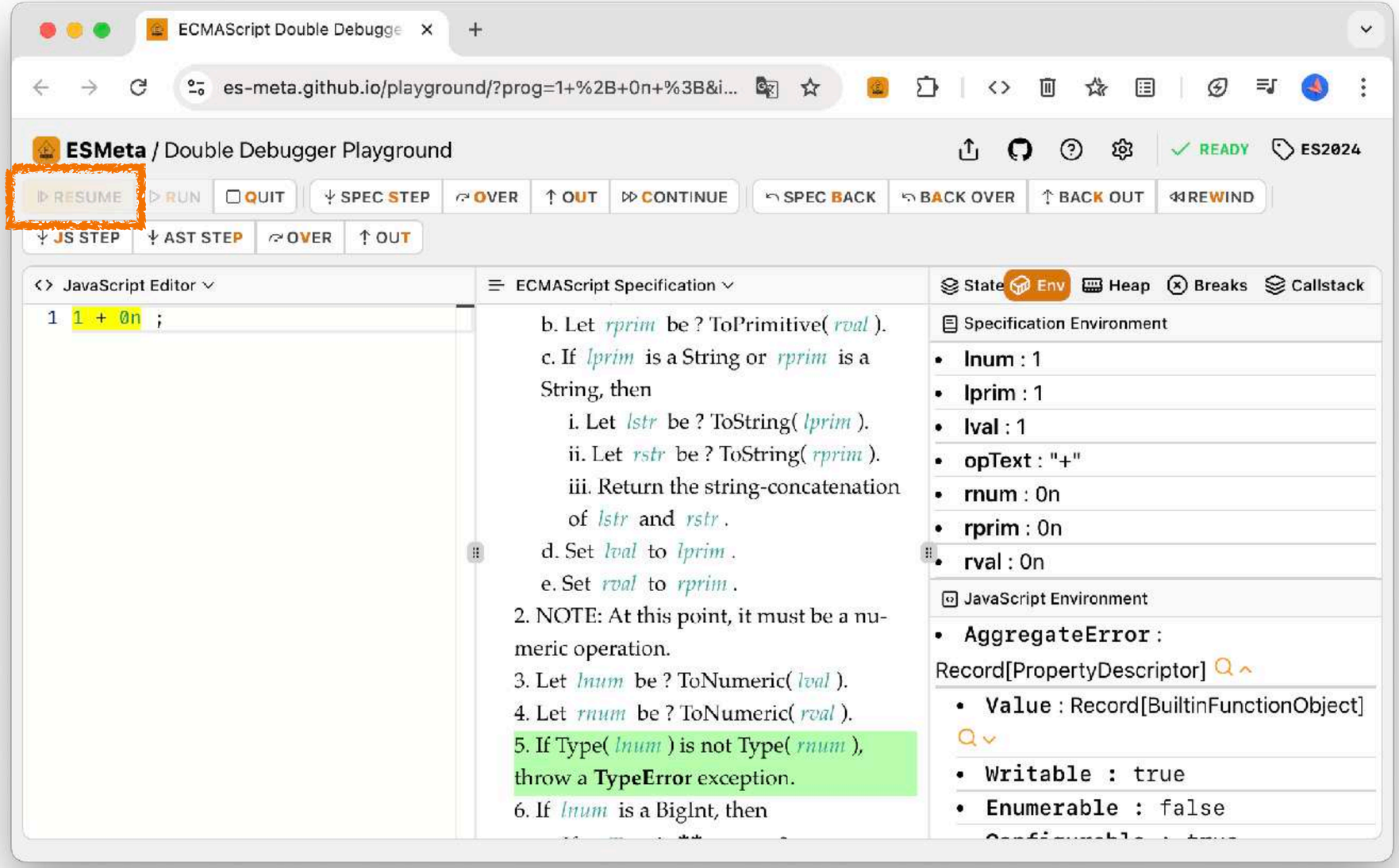

ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-lan...
5.3 ApplyStringOrNumericBinaryOperator (lval, opText, rval)
The abstract operation ApplyStringOrNumericBinaryOperator takes arguments lval (an ECMAScript language value), opText (**, *, /, %, +, -, <<, >>, >>>, &, ^, or |), and rval (an ECMAScript language value) and returns either a normal completion containing either a String, a BigInt, or a Number, or a throw completion. It performs the following steps when called:
1. If opText is +, then
a. Let lprim be ? itive(lval).
b. Let rprim be ? itive(rval).
c. If lprim is a St g rim is a String, then
i. Let lstr be ? ToS
ii. Let rstr be ? ToString(
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).
Option (alt) + 클릭
ECMA Visualizer
ESMeta ECMA Visualizer
Program
Run on Double Debugger
1 1 + 1 ;
Test262
8660 found
Download All
built-ins/Array/S15.4_A1.1_T10.js
built-ins/Array/from/calling-from-valid-1-onlyStrict.j
built-ins/Array/from/calling-from-valid-2.js
built-ins/Array/from/elements-added-after.js
CallStack
Start by pressing a function link with Left Click

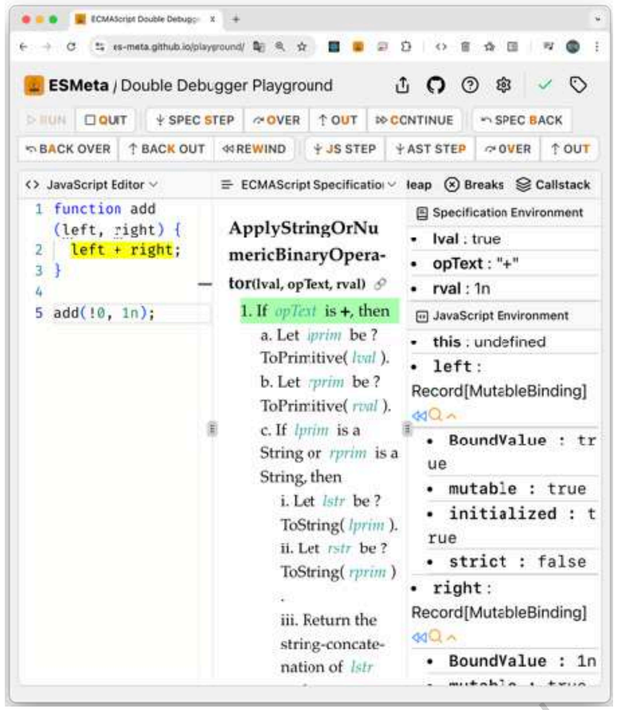

ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-lan...
a normal completion containing either a String, a BigInt, or a
ber, or a throw completion. It performs the following steps when
called:
1. If opText is +, then
a. Let lprim be ? ToPrimitive(lval).
b. Let rprim be ? ToPrimitive(rval).
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, rnum).
b. If opText is /, return ? BigInt::divide(lnum, rnum).
c. If opText is %, return ? BigInt::remainder(lnum, rnum).
ECMA Visualizer
ESMeta ECMA Visualizer
Program
Run on Double Debugger
1 0 * { [Symbol . toPrimitive] : 0 } ;
Test262
43 found
Download All
language/expressions/addition/bigint-errors.js
language/expressions/addition/coerce-symbol-to-p
language/expressions/addition/order-of-evaluation.j
language/expressions/bitwise-and/S11.10.1_A2.2_T1
CallStack
Start by pressing a function link with Left Click


ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-lan...
a normal completion containing either a String, a BigInt, or a
ber, or a throw completion. It performs the following steps when
called:
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, rnum).
b. If opText is /, return ? BigInt::divide(lnum, rnum).
c. If opText is %, return ? BigInt::remainder(lnum, rnum).
ECMA Visualizer
ESMeta ECMA Visualizer
Program
Run on Double Debugger
1
1 + 0n ;
Test262
12 found
Download All
language/expressions/addition/bigint-and-number.j
language/expressions/bitwise-and/bigint-and-numb
language/expressions/bitwise-or/bigint-and-numbe
language/expressions/bitwise-xor/bigint-and-numb
CallStack
Start by pressing a function link with Left Click


ECMAScript® 2024 Language
tc39.es/ecma262/2024/multipage/ecmascript-lan...
a normal completion containing either a String, a BigInt, or a
ber, or a throw completion. It performs the following steps when
called:
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, rnum).
b. If opText is /, return ? BigInt::divide(lnum, rnum).
c. If opText is %, return ? BigInt::remainder(lnum, rnum).
ECMA Visualizer
ESMeta ECMA Visualizer
Program
Run on Double Debugger
1 1 + 0n ;
재개(Resume)
Test262
12 found
Download All
language/expressions/addition/bigint-and-number.j
language/expressions/bitwise-and/bigint-and-numb
language/expressions/bitwise-or/bigint-and-numbe
language/expressions/bitwise-xor/bigint-and-numb
CallStack
Start by pressing a function link with Left Click


ECMAScript Double Debugge
es-meta.github.io/playground/?prog=1+%2B+0n+%3B&i...
ESMeta / Double Debugger Playground
READY
ES2024
RESUME
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 1 + 0n ;
ECMAScript Specification
Context Not Found
State
Env
Heap
Breaks
Callstack
Disabled. Start debugger to use.


ECMAScript Double Debugge
es-meta.github.io/playground/?prog=1+%2B+0n+%3B&i...
ESMeta / Double Debugger Playground
READY
ES2024
RESUME
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
1 1 + 0n ;
ECMAScript Specification
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
State
Env
Heap
Breaks
Callstack
Specification Environment
lnum : 1
lprim : 1
lval : 1
opText : "+"
rnum : 0n
rprim : 0n
rval : 0n
JavaScript Environment
AggregateError : Record[PropertyDescriptor]
Value : Record[BuiltinFunctionObject]
Writable : true
Enumerable : false



명세
(ECMA-262)

Test
테스트

프로그램 수집기

JS
합성된
프로그램

Test
선별된
테스트

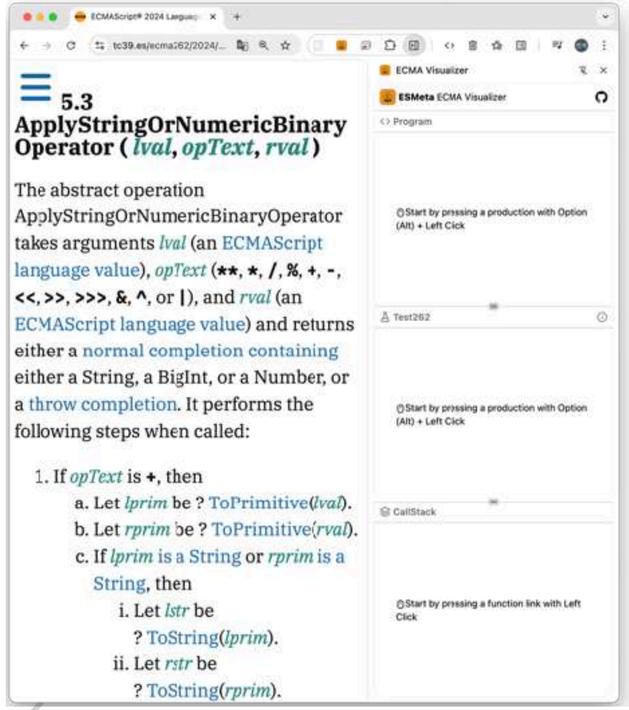
알고리즘 스텝마다 수집

JS
프로그램

1
이중 디버거

재개
(Resume)

2
프로그램
시각화기



5.3 ApplyStringOrNumericBinary Operator (*lval*, *opText*, *rval*)

The abstract operation ApplyStringOrNumericBinaryOperator takes arguments *lval* (an ECMAScript language value), *opText* (**, *, /, %, +, -, <<, >>, >>>, &, ^, or |), and *rval* (an ECMAScript language value) and returns either a normal completion containing either a String, a BigInt, or a Number, or a throw completion. It performs the following steps when called:

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*).

JavaScript 명세에 Occurrence Typing 적용하기

김준겸, 박지혁

2025. 08. 21 @ SIGPL 여름학교 2025

PLRG

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

`Undefined, Null, Boolean, String,`
`Symbol, Number, BigInt, Object`

ECMAScript language value (**`ESValue`**)

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

ECMAScript language value (**ESValue**)

그러나, 명세에서는 **더 많은 타입**을 사용

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

ECMAScript language value (**ESValue**)

그러나, 명세에서는 **더 많은 타입**을 사용

7.2.2 IsArray (*argument*)

The abstract operation IsArray takes argument *argument* (an ECMAScript language value) and returns either a normal completion containing a Boolean or a throw completion.

1. If *argument* is not an Object, return **false**.
2. If *argument* is an Array exotic object, return **true**.
3. If *argument* is a Proxy exotic object, then
    a. Perform ? ValidateNonRevokedProxy(*argument*).
    b. Let *proxyTarget* be *argument*.[[ProxyTarget]].
    c. Return ? IsArray(*proxyTarget*).
4. Return **false**.

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

ECMAScript language value (**`ESValue`**)

그러나, 명세에서는 **더 많은 타입**을 사용

7.2.2 IsArray (*argument*)

`ESValue`

The abstract operation IsArray takes argument *argument* (an ECMAScript language value) and returns either a normal completion containing a Boolean or a throw completion.

1. If *argument* is not an Object, return **false**.
2. If *argument* is an Array exotic object, return **true**.
3. If *argument* is a Proxy exotic object, then
    a. Perform ? ValidateNonRevokedProxy(*argument*).
    b. Let *proxyTarget* be *argument*.[[ProxyTarget]].
    c. Return ? IsArray(*proxyTarget*).
4. Return **false**.

JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

ECMAScript language value (**`ESValue`**)

그러나, 명세에서는 **더 많은 타입**을 사용

7.2.2 IsArray (*argument*)

`Normal[Boolean]`

`ESValue`

The abstract operation IsArray takes argument *argument* (an ECMAScript language value) and returns either a normal completion containing a Boolean or a throw completion.

1. If *argument* is not an Object, return **false**.
2. If *argument* is an Array exotic object, return **true**.
3. If *argument* is a Proxy exotic object, then
    a. Perform ? ValidateNonRevokedProxy(*argument*).
    b. Let *proxyTarget* be *argument*.[[ProxyTarget]].
    c. Return ? IsArray(*proxyTarget*).
4. Return **false**.

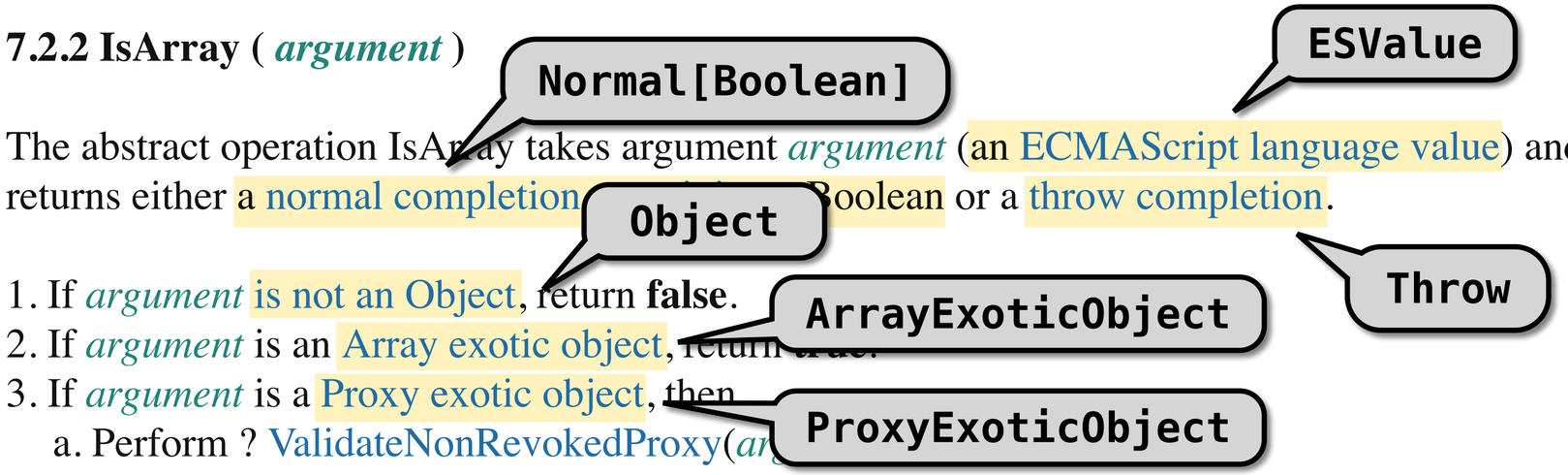
JavaScript 기계화 명세(ECMA-262)의 타입

JavaScript 타입은 단 8종류

```
Undefined, Null, Boolean, String,
Symbol, Number, BigInt, Object
```

ECMAScript language value (**`ESValue`**)

그러나, 명세에서는 **더 많은 타입**을 사용



ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)
The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters.
This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that **`new`** was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters. This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that `new` was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters. This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

```
Constructor | Undefined
```

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that `new` was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters. This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

`Constructor | Undefined`

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, ...

`Requires: Constructor`

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that `new` was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters. This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

`Constructor | Undefined`

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

`Requires: Constructor`

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that `new` was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters. This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

`Constructor | Undefined`

Type mismatch

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

`Requires: Constructor`

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that `new` was initially applied to. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor*. It performs the following steps when called:

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters.
This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

`Constructor | Undefined`

Type mismatch

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

`Requires: Constructor`

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that was originally called to perform the creation. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor* in order to perform this kind of creation. When called:

newTarget (a constructor)

↓

newTarget (a constructor or **undefined**)

Fixed in PR #3380 (2024.07.25)

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters.
This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor function that is performing this action. *newTarget* is the constructor that was originally called to perform the creation. *parameterArgs* and *bodyArg* reflect the argument values that were passed to *constructor* by its caller. *kind* is used to:

`Constructor | Undefined`

Typechecks

`Requires: Constructor | Undefined`

newTarget (a constructor)

newTarget (a constructor or **undefined**)

Fixed in PR #3380 (2024.07.25)

ECMA-262의 타입 분석

20.2.1.1 Function (...*parameterArgs*, *bodyArg*)

The last argument (if any) specifies the body (executable code) of a function; any preceding arguments specify formal parameters.
This function performs the following steps when called:

1. Let *C* be the active function object.
2. If *bodyArg* is not present, set *bodyArg* to the empty String.
3. Return ? CreateDynamicFunction(*C*, NewTarget, NORMAL, *parameterArgs*, *bodyArg*).

`Constructor | Undefined`

Typechecks

20.2.1.1.1 CreateDynamicFunction (*constructor*, *newTarget*, *kind*, *parameterArgs*, *bodyArg*)

`Requires: Constructor | Undefined`

The abstract operation CreateDynamicFunction takes arguments *constructor* (a constructor), *newTarget* (a constructor), *kind* (NORMAL, GENERATOR, ASYNC, or ASYNC-GENERATOR), *parameterArgs* (a List of ECMAScript language values), and *bodyArg* (an ECMAScript language value) and returns either a normal completion containing an ECMAScript function object or a throw completion. *constructor* is the constructor to which *newTarget* is the originally applied to. *parameterArgs* and *bodyArg* are the arguments that were passed to *constructor* when it was called:

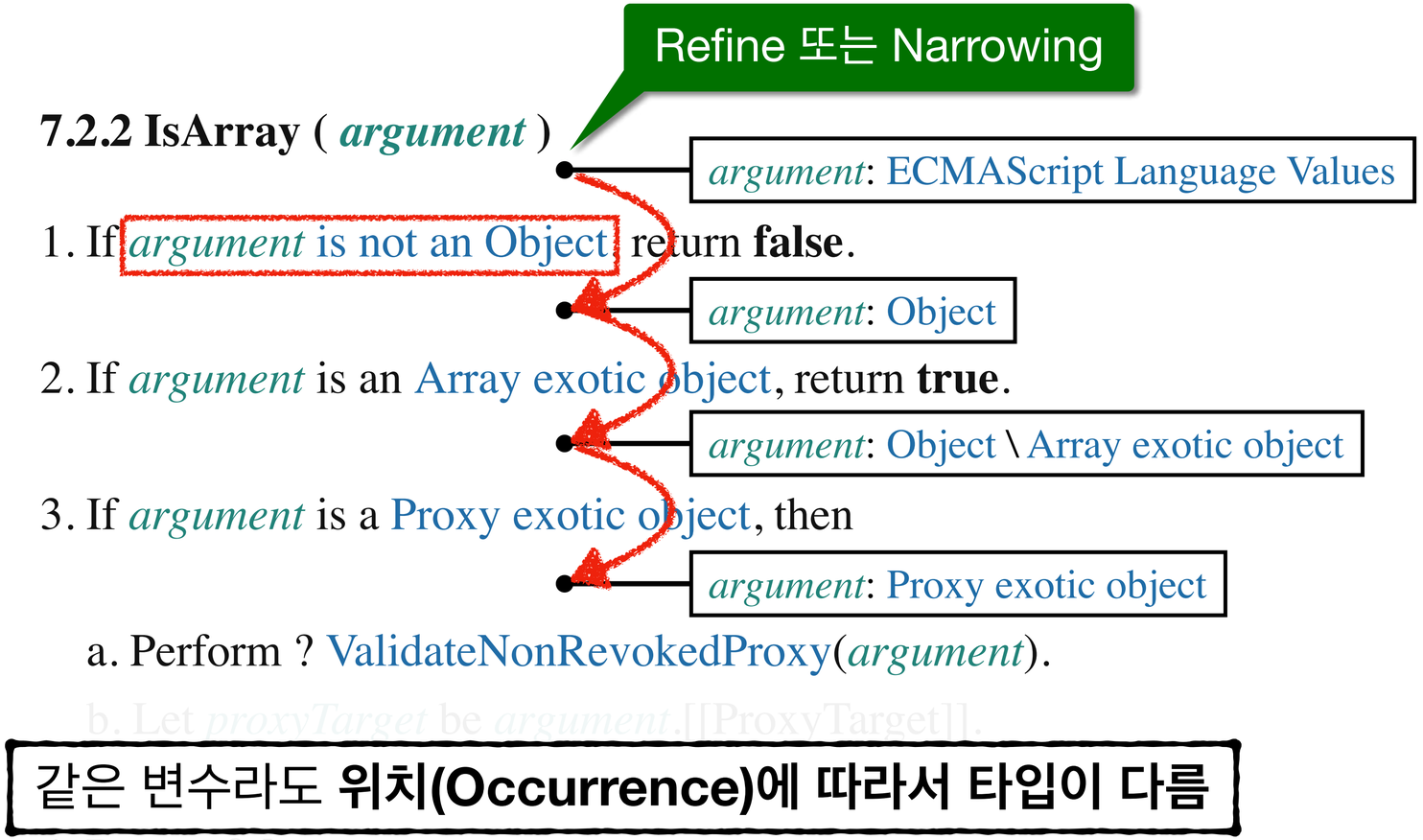
newTarget (a constructor or **undefined**)

Fixed in PR #3380 (2024.07.25)

타입 분석기 ESMeta가

ecma 262 의 CI/CD에 편입

타입 분석의 걸림돌: Occurrence Typing



같은 변수라도 **위치(Occurrence)에 따라서 타입이 다름**

기존의 Occurrence Typing 방법들

- **Syntax-based refinement** (JSTAR, TypeScript)

조건문의 expression만 보고, 여기에 등장하는 변수만 사전에 정의된 규칙 기반으로 refine함

```
ex) if (typeof x == "Object") {  ... // x: Object  }
```

- **Proposition-based Typing** (Typed Racket, Typed Clojure)

Type에 추가적으로 '값이 참이면 성립하는 명제'와 '값이 거짓이면 성립하는 명제'를 붙여서 강화

```
ex) Boolean { True => x: Object | False => x: number }
```

그 외에도 **Type variable** 기반 (Flow), **Set-theoretic type** 등의 다양한 타입 시스템 존재

기존 기술을 그대로 사용할 수 있을까?

JavaScript 명세를 잘 분석하기 위해서는,

1. Refinement의 조건을 **값으로서 전파**할 수 있어야 하고
2. True와 False 뿐만 아니라 **일반적인 타입에 대한 proposition**을 달 수 있으며
3. **Mutation**이 있더라도 최소한의 Precision을 보장해야 한다.

기존 기술을 그대로 사용할 수 있을까?

JavaScript 명세를 잘 분석하기 위해서는,

1. Refinement의 조건을 **값으로서 전파**할 수 있어야 하고
2. True와 False 뿐만 아니라 **일반적인 타입에 대한 proposition**을 달 수 있으며
3. **Mutation**이 있더라도 최소한의 Precision을 보장해야 한다.

	(1) 값으로서 전파	(2) 일반적인 타입 지원	(3) Mutation 지원
Typed Racket	O	X	△
Flow	X	X	O
Set-Theoretic Type	O	O	X

기존 기술을 그대로 사용할 수 있을까?

JavaScript 명세를 잘 분석하기 위해서는,

1. Refinement의 조건을 **값으로서 전파**할 수 있어야 하고
2. True와 False 뿐만 아니라 **일반적인 타입에 대한 proposition**을 달 수 있으며
3. **Mutation**이 있더라도 최소한의 Precision을 보장해야 한다.

오늘의 비교군

	(1) 값으로서 전파	(2) 일반적인 타입 지원	(3) Mutation 지원
Typed Racket	O	X	△
Flow	X	X	O
Set-Theoretic Type	O	O	X

Our solution: "Type Guard"

- 기존의 타입 시스템을 **Abstract Interpetation** 위로 옮김
- 기존의 **Proposition을 같이 담는** 타입 구조를 유지하되, 다음과 같이 formalization함
  - **(a)** Proposition을 True/False뿐만이 아니라 **제한 없이 담을 수 있게**
  - **(b)** Mutation이 일어난 경우에도, **살릴 수 있는 정보는 최대한 살릴 수 있도록**

$\hat{v} =$ `Any { Object => x: BigInt, Object[prop: Number] => y: Bool }`

`Object`라면 `x`가 `BigInt`임을 보장할 수 있고,

`Object`면서 `Number` 타입인 `"prop"`을 가지고 있다면 **추가로** `y`가 `Bool`임을 보장 가능

How Typed Racket works?

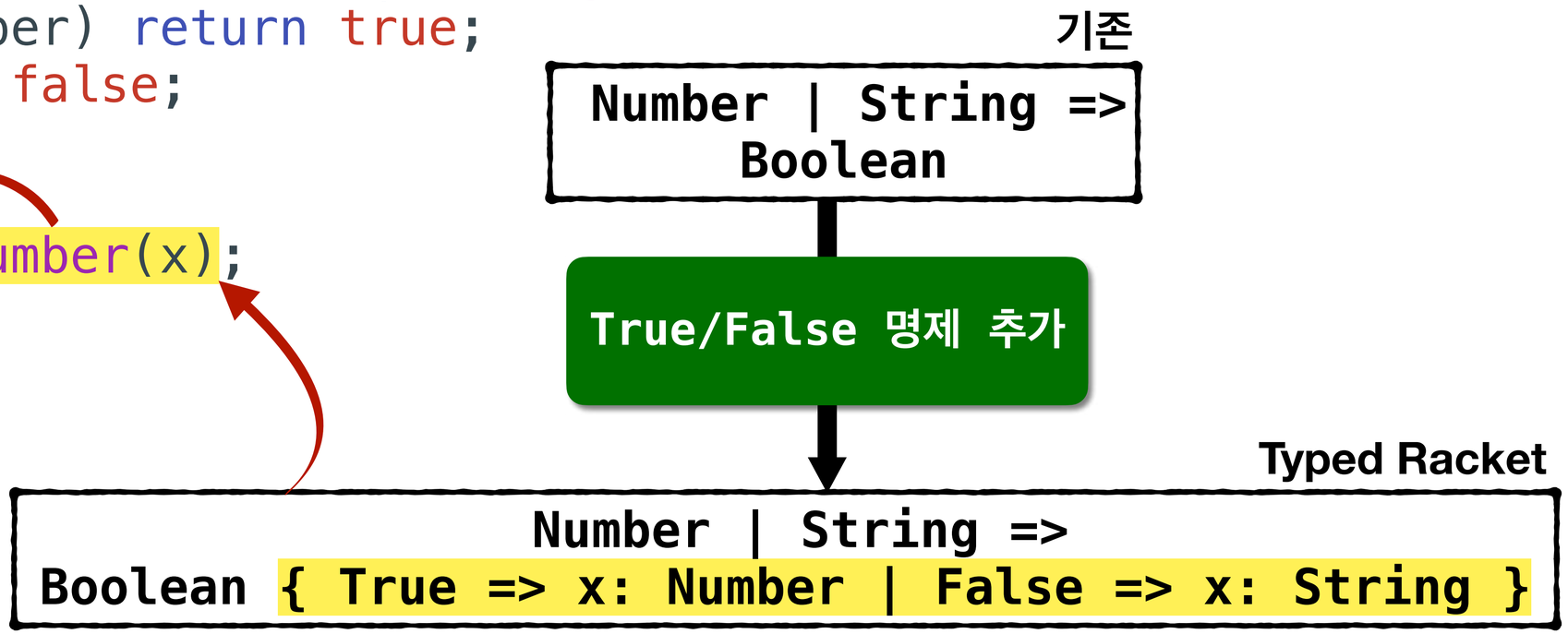

```
function isNumber(x: Number | String) {
    if(x is Number) return true;
    else return false;
}

let check = isNumber(x);
if(check) {
  // x: Number
}
```

How Typed Racket works?

```
function isNumber(x: Number | String) {
    if(x is Number) return true;
    else return false;
}

let check = isNumber(x);
if(check) {
  // x: Number
}
```



- Typed Racket에서는 이미 **(1) 값으로서의 전파**를 잘 처리하고 있음

남은 문제: (2) 일반적인 타입에 대한 Proposition

- Typed Racket은 다음과 같은 문제점이 존재

```
function isNumber(x: Number | String) {
    if(x is Number) return {value: true};
    else return {value: false};
}

let check = isNumber(x);
if(check.value) {
  // x: Number
}
```

True/False가 아님

```
Number | String =>
{ value: Boolean } { True => ??? | False => ??? }
```

아무것도 넣을 수 없음

해결: (2) 일반적인 타입에 대한 Proposition

- 아이디어: 사전에 정의한 **임의의 타입 집합**에 대해서 proposition을 가질 수 있도록 하기

Demanded Type

```
function isNumber(x: Number | String) {
    if(x is Number) return {value: true};
    else return {value: false};
}

let check = isNumber(x);
if(check.value) {
  // x: Number
}
```

Demanded Type: { value: True }, { value: False }

isNumber

```
Number | String => { value: Boolean } {
  { value: True } => x: Number |
  { value: False } => x: String
}
```

해결: (2) 일반적인 타입에 대한 Proposition

- 아이디어: 사전에 정의한 **임의의 타입 집합**에 대해서 proposition을 가질 수 있도록 하기

Demanded Type

```
function isNumber(x: Number | String) {
    if(x is Number) return {value: true};
    else return {value: false};
}

let check = isNumber(x);
if(check.value) {
  // x: Number
}
```

check

```
{ value: Boolean } {
  { value: True } => x: Number |
  { value: False } => x: String
}
```

check.value

```
Boolean  {
  True => x: Number |
  False => x: String
}
```

남은 문제: (3) Mutation

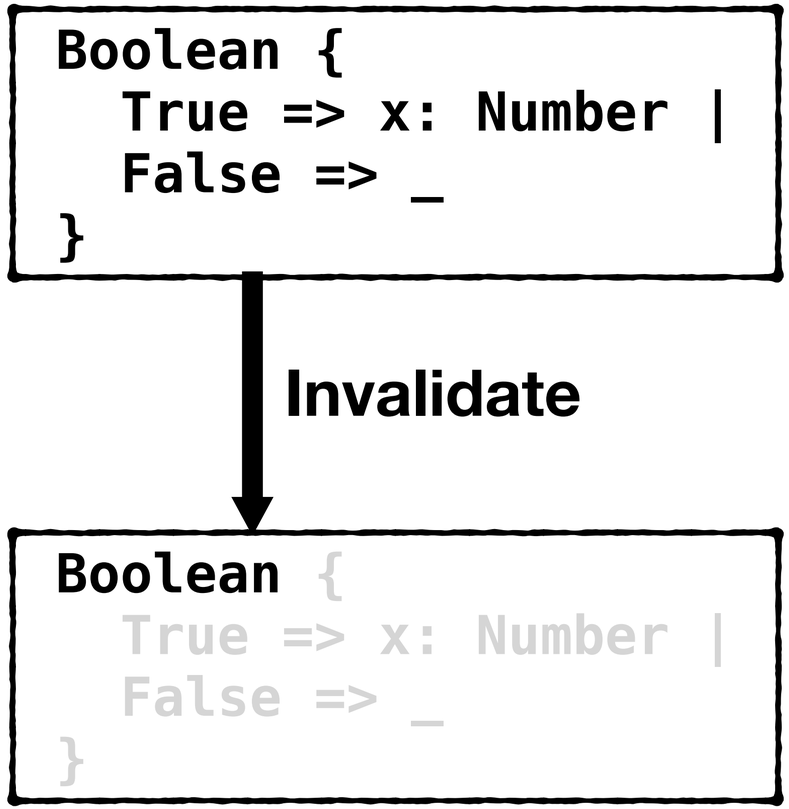
- 한편, Typed Racket에서는...

```
let doJob = isNumber();
if (doJob) {
    // ...
    doJob = false;
}
doJob = isNumber();
if (doJob) {
    // ...
}
```

Mutation 발생

전략:

Syntax 기반 사전 분석으로 동일한 이름을 가진 변수를 모두 invalidate



해결: (3) Mutation

- On-the-fly로 선택적으로 Type Guard를 약화시키기

```
// x: Boolean { true => y: String }
x = "abc"
// x: String
```

Variable Invalidating

```
function foo(x) {
  x.f = "abc";
}

function main() {
  // x: { f: Number | Boolean } { { f: Number } => y: String }
  foo(x)
  // x: { f: Number | Boolean | String }
}
```

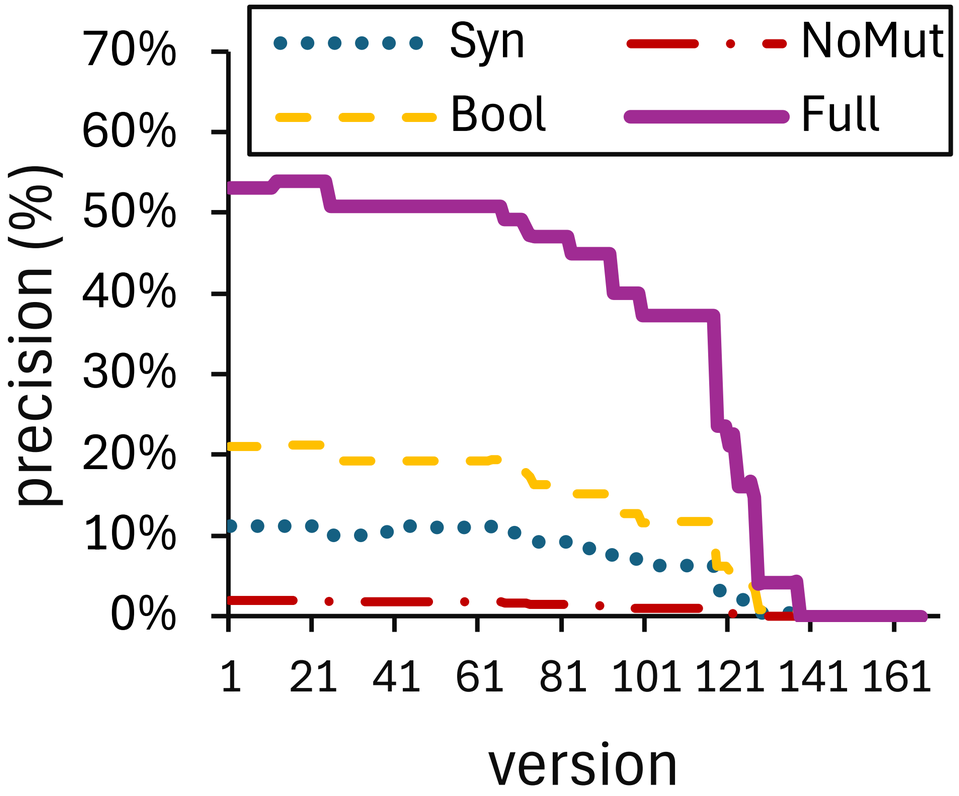
Field-based Invalidating

Evaluation: Precision

ES2024부터 지금까지...

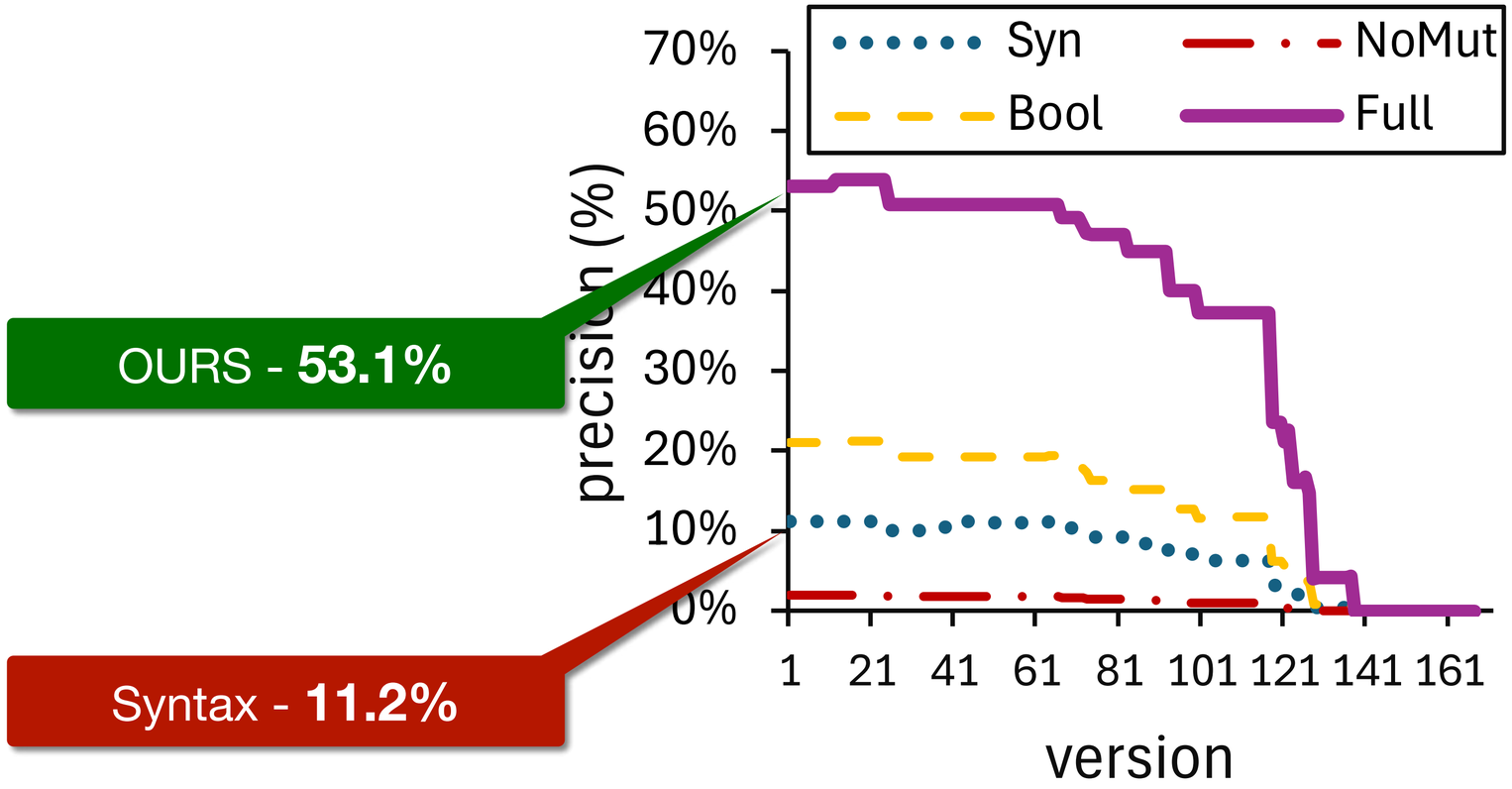
Evaluation: Precision

ES2024부터 지금까지...



Evaluation: Precision

ES2024부터 지금까지...

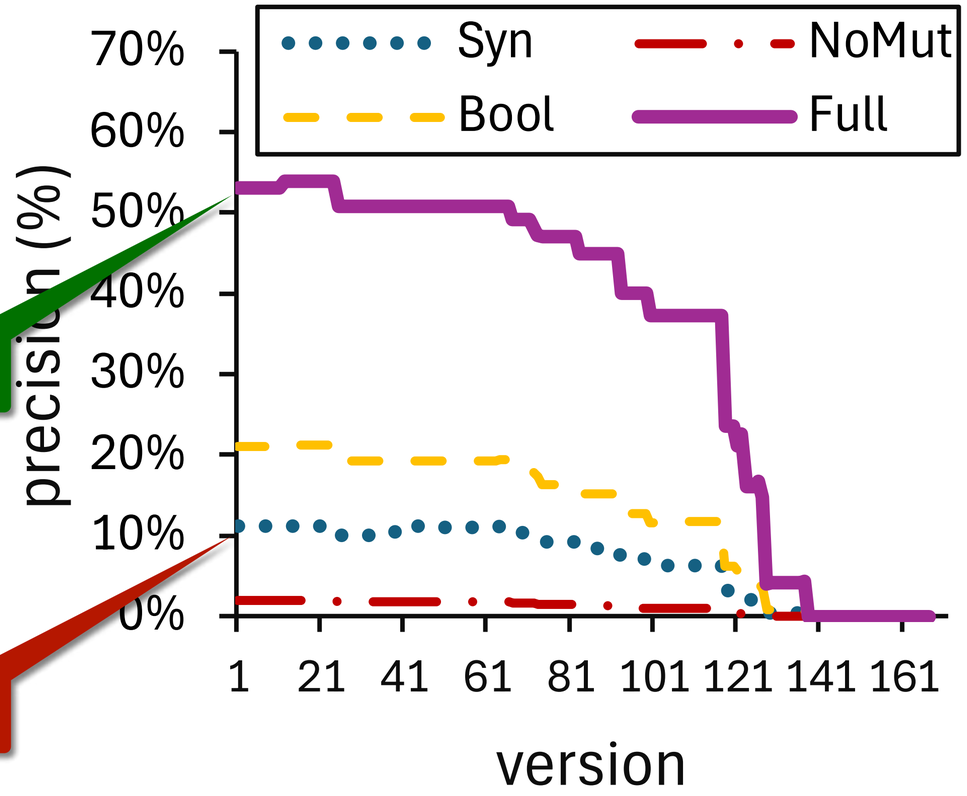


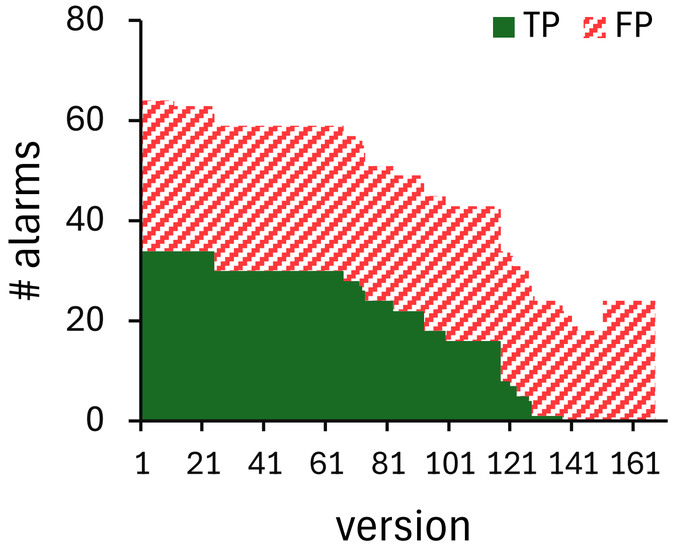
Evaluation: Precision

ES2024부터 지금까지...

OURS - **53.1%**

Syntax - **11.2%**



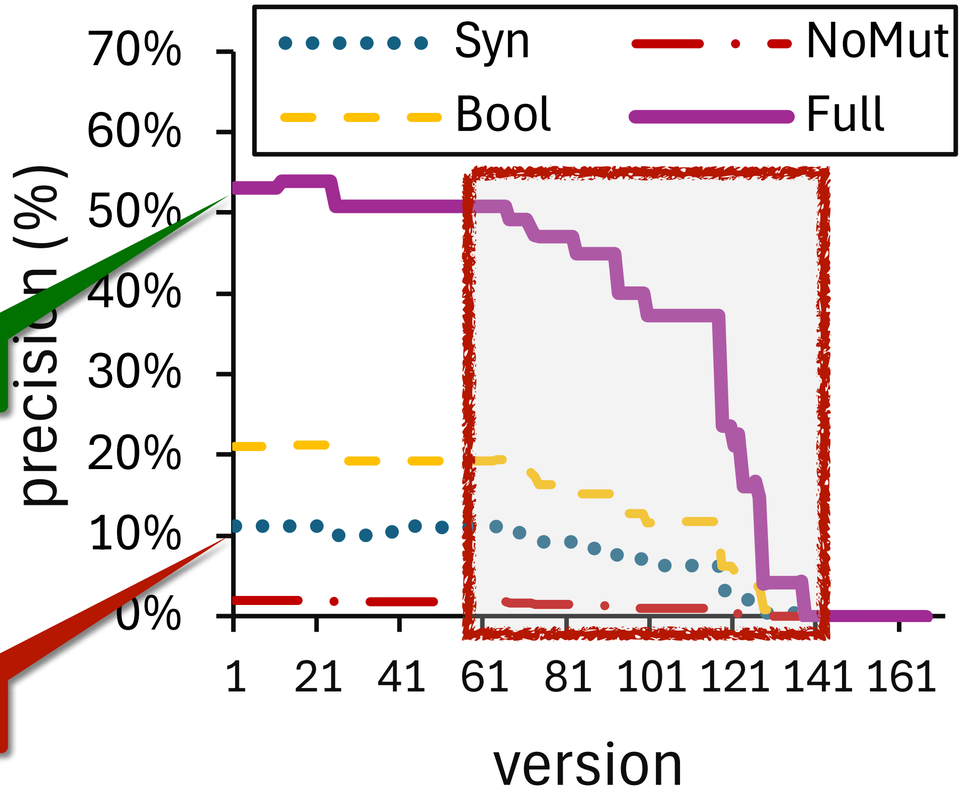


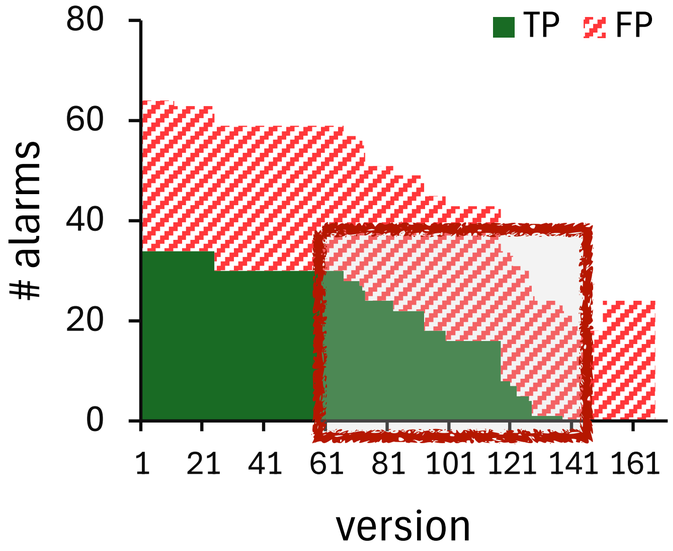
Evaluation: Precision

ES2024부터 지금까지...

OURS - **53.1%**

Syntax - **11.2%**



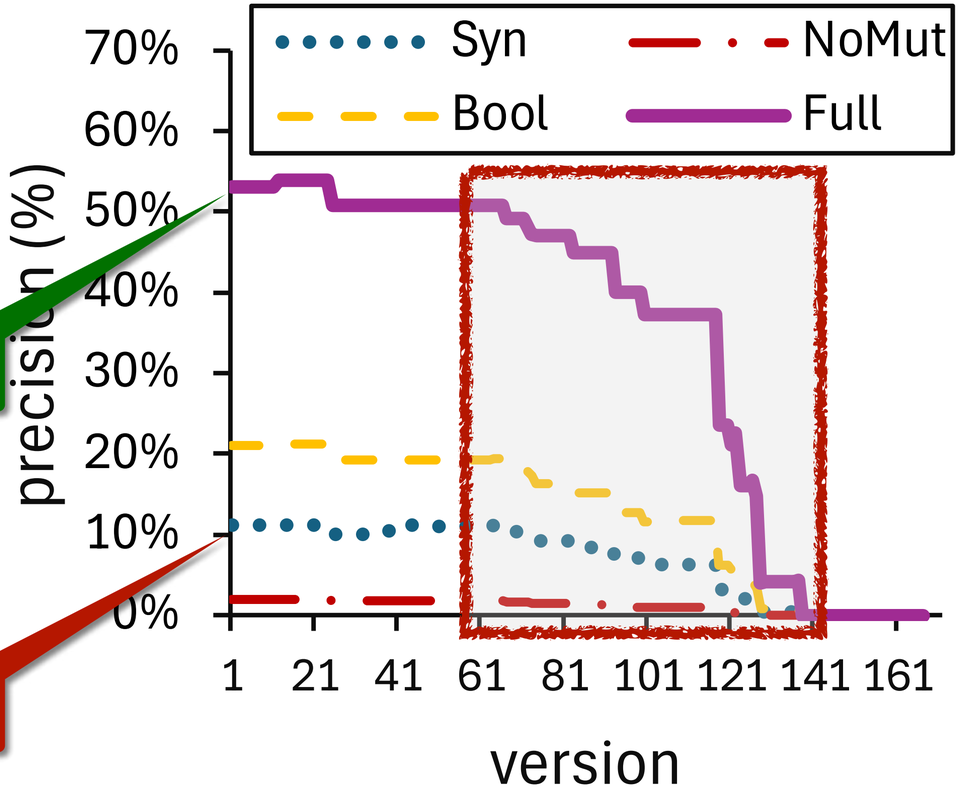


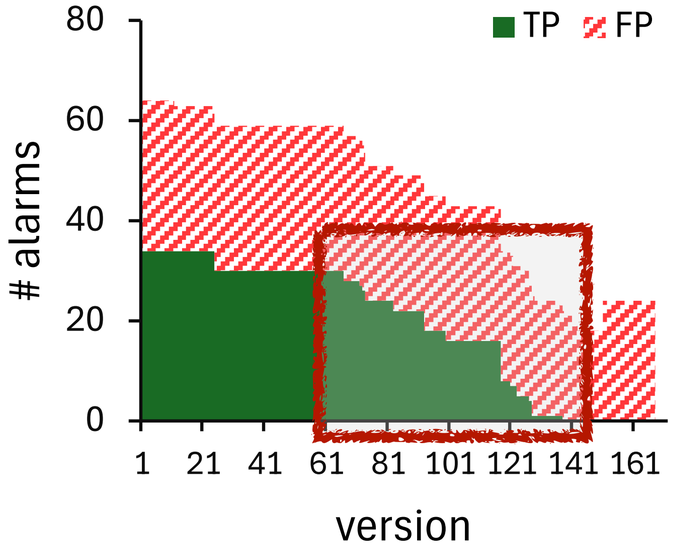
Evaluation: Precision

ES2024부터 지금까지...

OURS - **53.1%**

Syntax - **11.2%**





True Alarm 34개 중 25개 제보

[ASE'25] Debun: Detecting Bundled JavaScript Libraries on Web using Property-Order Graphs

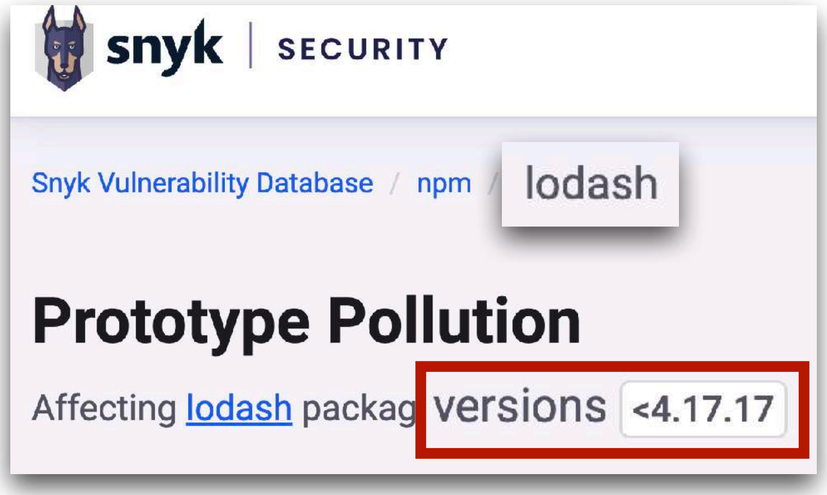
속성 순서 그래프를 활용하여 웹에 배포된 자바스크립트 라이브러리 탐지하기

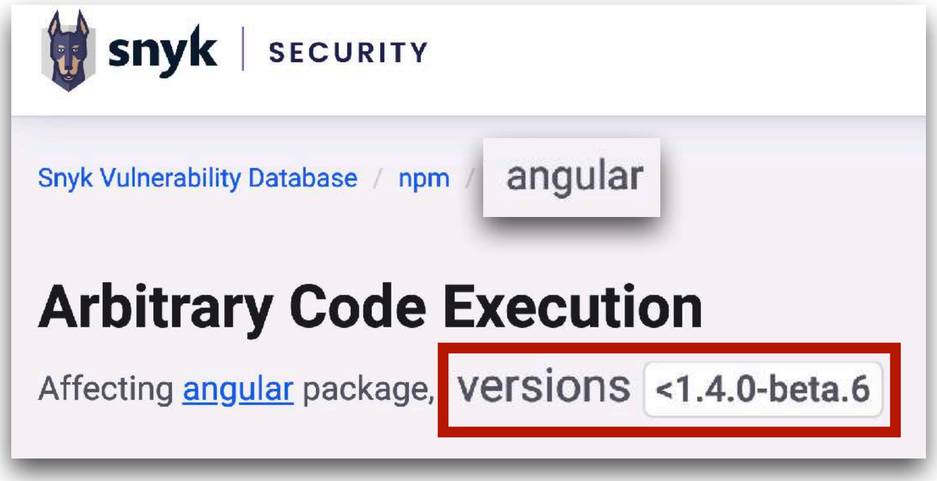
김서진*, **박성민***, 박지혁

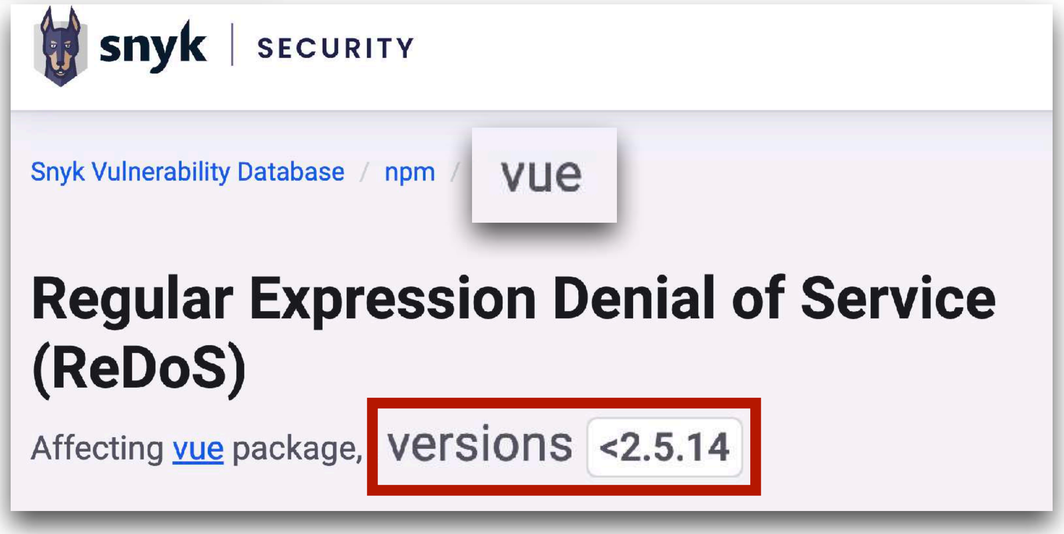
KOREA UNIVERSITY

PLRG

웹사이트 자바스크립트 라이브러리 취약점







웹사이트 배포 과정

- **배포 과정**에서 라이브러리 코드가 **변환 및 압축**이 되어 **검출이 어려움**

배포 전 라이브러리 파일들

다양한 트랜스파일러 및 번들러

배포 후 병합된 파일

기존 연구 - 속성 패턴으로 런타임에 검사하기

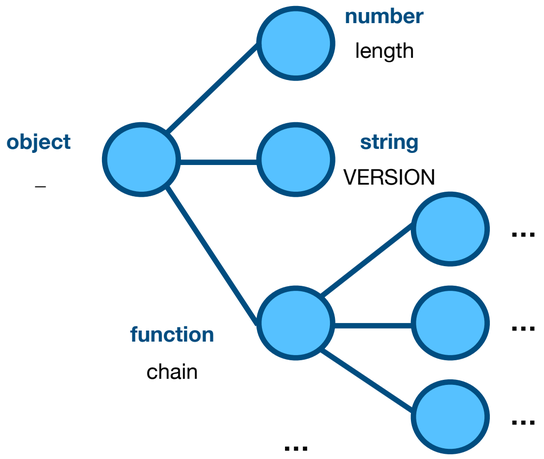
- **LDC** (Library Detector for Chrome)
  - **수동**으로 작성한 **속성 패턴**으로 **런타임에 검사**하기
  - 라이브러리 탐지

```
typeof (_ = window._) == 'function'
typeof (chain = _ && _.chain) == ‘function'
```

  - 버전 탐지

```
return { version: _.VERSION || UNKNOWN_VERSION };
```

- **PTDetector** (ASE’23)
  - **속성 패턴**을 **트리** 형태로 **자동** 추출




```
1  (function() {
2    function lodash(value) {
3      return new LodashWrapper(value);
4    }
5
6    // Define properties
7    lodash.chain = function(value) {
8      var result = lodash(value);
9      result.__wrapped__ = value;
10     return result;
11   }
12   lodash.differenceBy = ...
13   ...
14   lodash.VERSION = '4.17.21'
15   // Export lodash
16   window._ = lodash
17 }.call(this));
```

Lodash v4.17.21

배포되어 있는 코드를 활용하고 있지 않는 이유

```
function remove(array, predicate) {
  var result = [];
  if (!(array && array.length)) {
    return result;
  }
  var index = -1,
    indexes = [],
    length = array.length;

  predicate = getIteratee(predicate, 3);
  while (++index < length) {
    var value = array[index];
    if (predicate(value, index, array)) {
      result.push(value);
      indexes.push(index);
    }
  }
  basePullAt(array, indexes);
  return result;
}
```

Lodash v4.17.21의 **remove** 함수

terser

변환

```
function f(e,r){var t=[];if(!e||!e.length)
return t;var n=-1,u=[],a=e.length;
for(r=an(r,3);++n<a;){var h=e[n];
r(h,n,e)&&(t.push(h),u.push(n))}
return bf(e,u),t}
```

변환된 코드

배포과정에서 **구조적 변환**으로 인해 단순비교로는 **탐지불가**

배포되어 있는 코드를 활용하고 있지 않는 이유

```
function remove(array, predicate) {
  var result = [];
  if (!(array && array.length)) {
    return result;
  }
  var index = -1,
    indexes = [],
    length = array.length;

  predicate = getIteratee(predicate, 3);
  while (++index < length) {
    var value = array[index];
    if (predicate(value, index, array)) {
      result.push(value);
      indexes.push(index);
    }
  }
  basePullAt(array, indexes);
  return result;
}
```

Lodash v4.17.21의 **remove** 함수

배포과정에서 **구조적 변환**으로 인해 단순비교로는 **탐지불가**

변하지 않는 부분만 비교하여 **같은 함수**임을 확인

terser

변환

```
function f(e,r){var t=[];if(!e||!e.length)
return t;var n=-1,u=[],a=e.length;
for(r=an(r,3);++n<a;){var h=e[n];
r(h,n,e)&&(t.push(h),u.push(n))}
return bf(e,u),t}
```

변환된 코드

배포되기 전과 후에 무엇이 유지될까? - 왜?

- **속성명**
  - **임의의 값으로 속성 접근이 가능**하여 안전한 변환을 위해 대체로 **속성명은 변환하지 않음**

```
array['len' + 'gth']  // array.length
```

- **속성 연산자의 실행순서**
  - 연산자의 순서가 바뀌면 **부수 효과(side effect)의 순서**가 바뀔 수 있음

```
obj = {
  get p() { console.log(1); }
  set q() { console.log(2); }
}
```

```
obj.p;        // print 1
obj.q = 42;   // print 2
```

≠

```
obj.q = 42;   // print 2
obj.p;        // print 1
```

배포되기 전과 후에 무엇이 유지될까?

```
function remove(array, predicate) {
  var result = [];
  if (!(array && array.length)) {
    return result;
  }
  var index = -1,
    indexes = [],
    length = array.length;

  predicate = getIteratee(predicate, 3);
  while (++index < length) {
    var value = array[index];
    if (predicate(value, index, array)) {
      result.push(value);
      indexes.push(index);
    }
  }
  basePullAt(array, indexes);
  return result;
}
```

Lodash v4.17.21의 **remove** 함수

terser

변환

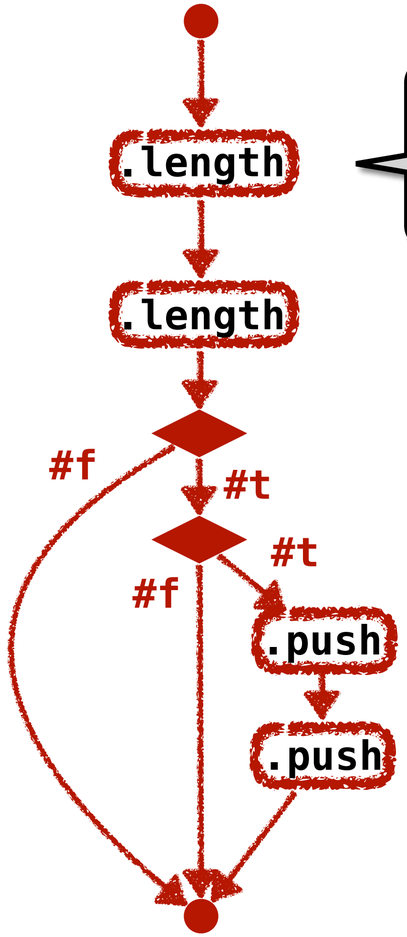
1. 속성명

2. 속성 연산 순서

```
function f(e,r){var t=[];if(!e||!a.length)
return t;var n=-1,u=[],a=e.length;
for(r=an(r,3);++n<a;){var h=e[n];
r(h,n,e)&&(t.push(h),u.push(n))}
return bf(e,u),t}
```

변환된 코드

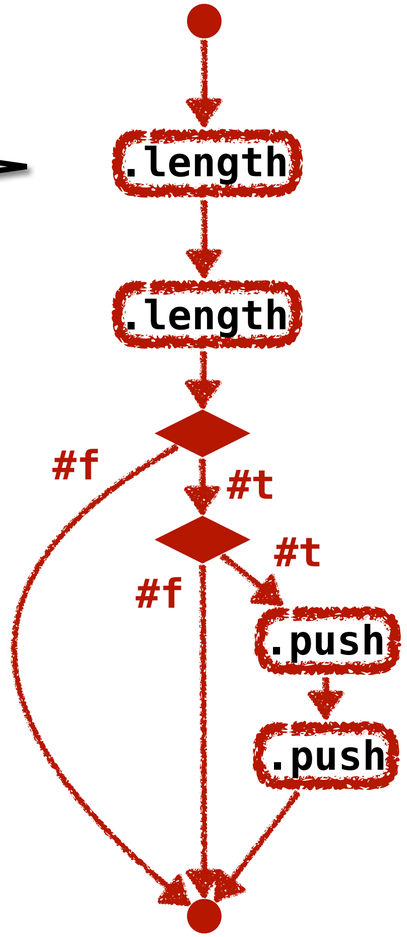
속성 순서 그래프 (Property-Order Graph, **POG**)



지문:
0x3FA345B1

=

지문:
0x3FA345B1



terser

변환

1. 속성명

2. 속성 연산 순서

Lodash v4.17.21의 **remove** 함수

변환된 코드

문제점: 속성 순서 그래프(POG)도 항상 유지 되지 않는다

1. 분기가 뒤집힌 경우

```
if (!y) x.p; else x.q;
```

변환

```
if (y) x.q; else x.p;
```

2. 분기가 합쳐지는 경우

```
while (x.p) {
  if (x.q) break;
}
```

변환

```
for (; x.p && !x.q; );
```

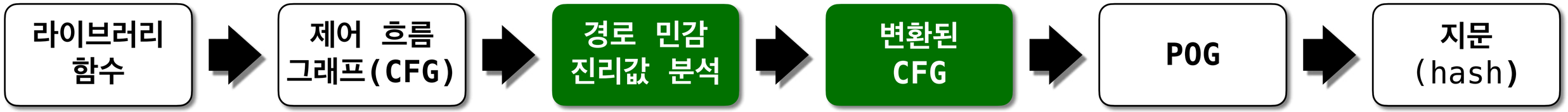
3. 중복된 부분이 합쳐지는 경우

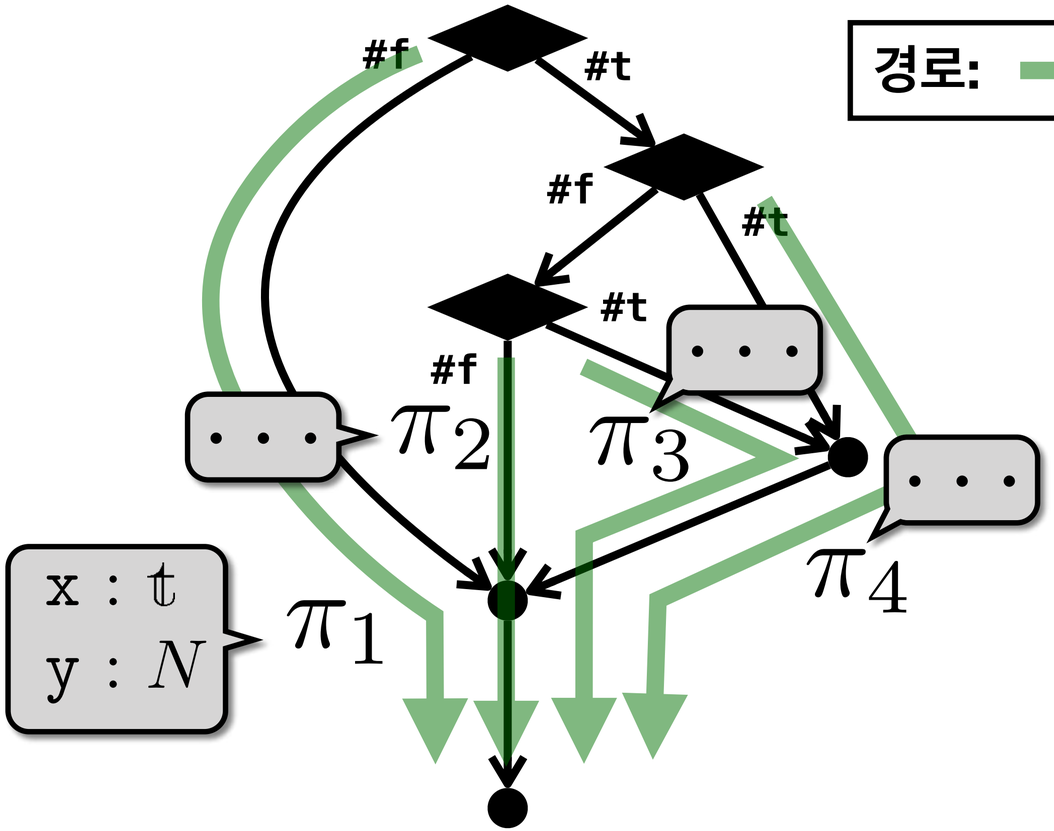
```
if (y) x.p = x.q = e1;
else   x.p = x.q = e2;
```

변환

```
x.p = x.q = y ? e1 : e2;
```

해결책: 경로 민감 진리값 분석으로 그래프를 변환하자

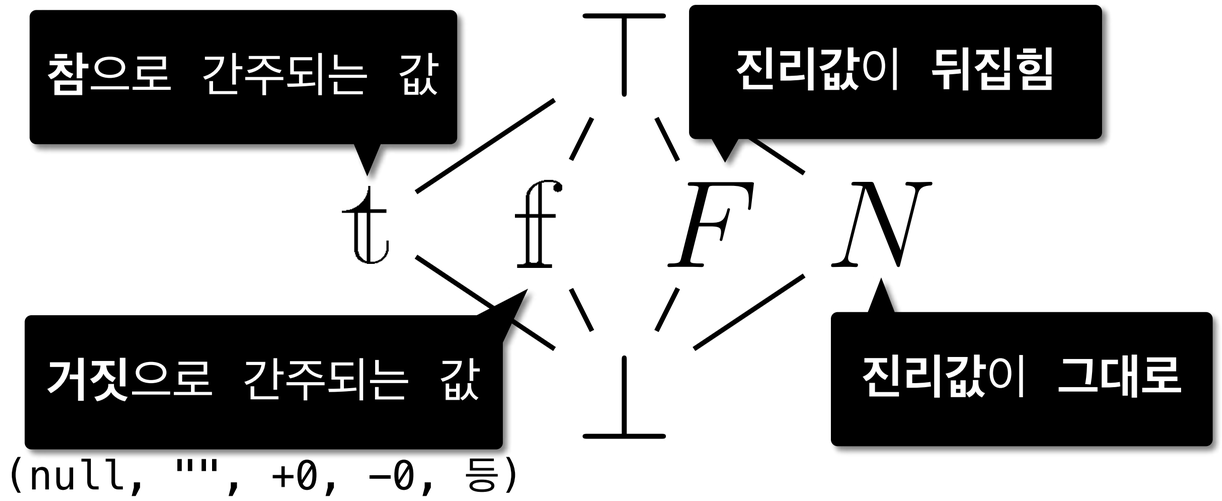




경로:

경로마다 변수의 진리값의 상태를 추적하자!

(경로란? 각 분기에서 현재 노드까지의 흐름)



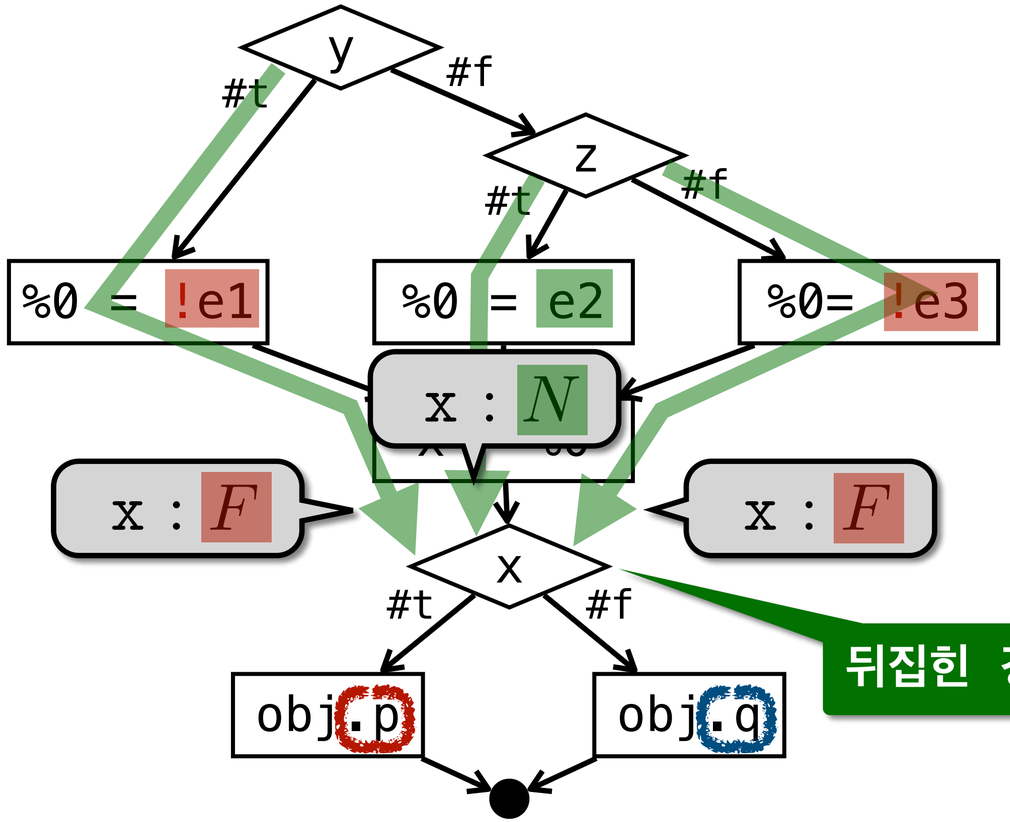
진리값 요약 도메인

(1) 분기 뒤집기 - 진리값이 **뒤집힌 경로**가 더 많은 경우

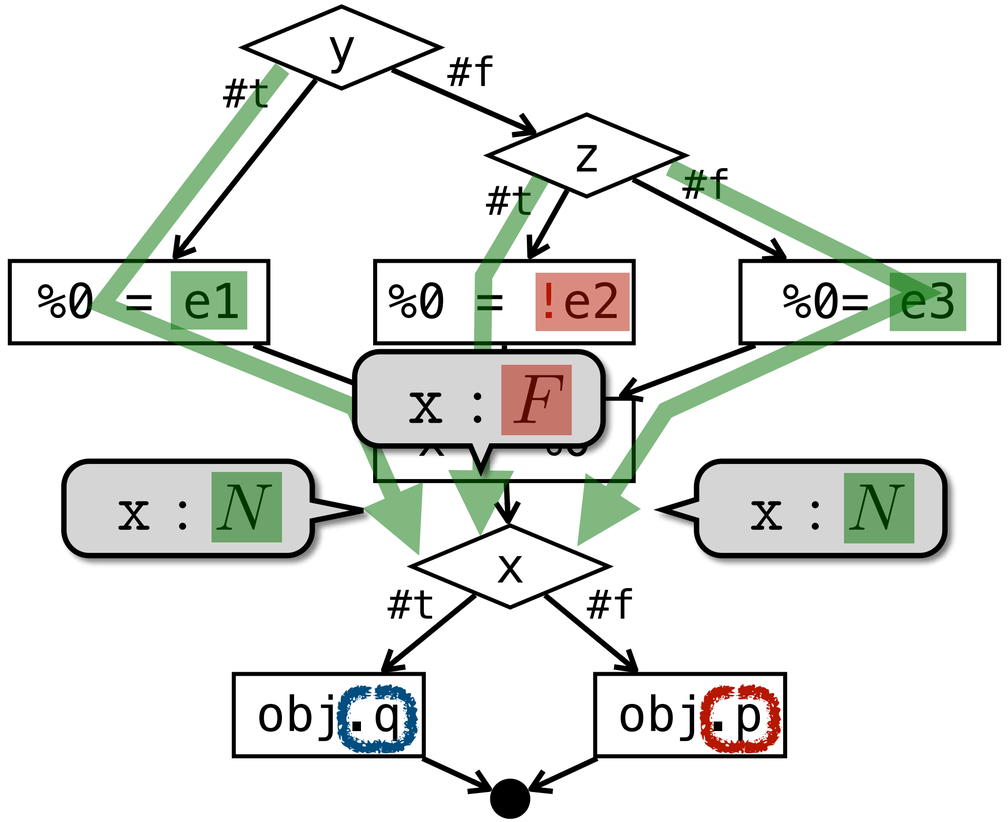
```
var x = y ? !e1 : z ? e2 : !e3;
if (x) obj.p;
else   obj.q;
```

변환

```
var x = y ? e1 : z ? !e2 : e3;
if (x) obj.q;
else   obj.p;
```



뒤집힌 경로가 더 많음

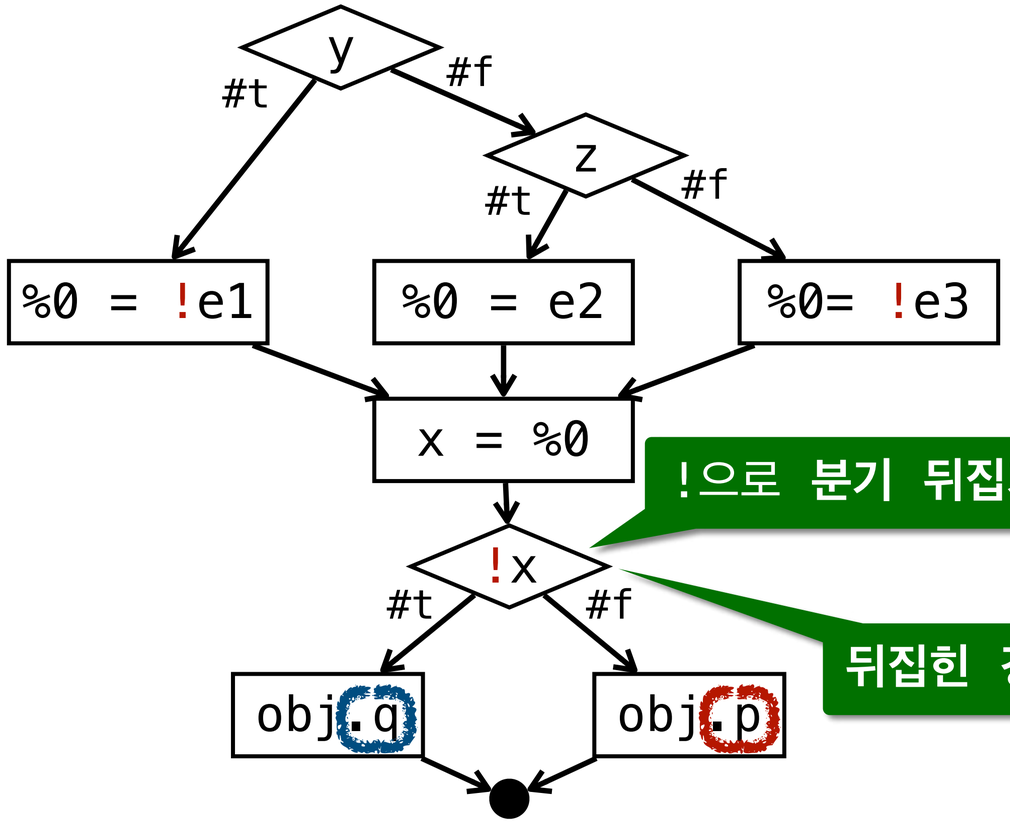


(1) 분기 뒤집기 - 진리값이 **뒤집힌 경로**가 더 많은 경우

```
var x = y ? !e1 : z ? e2 : !e3;
if (x) obj.p;
else   obj.q;
```

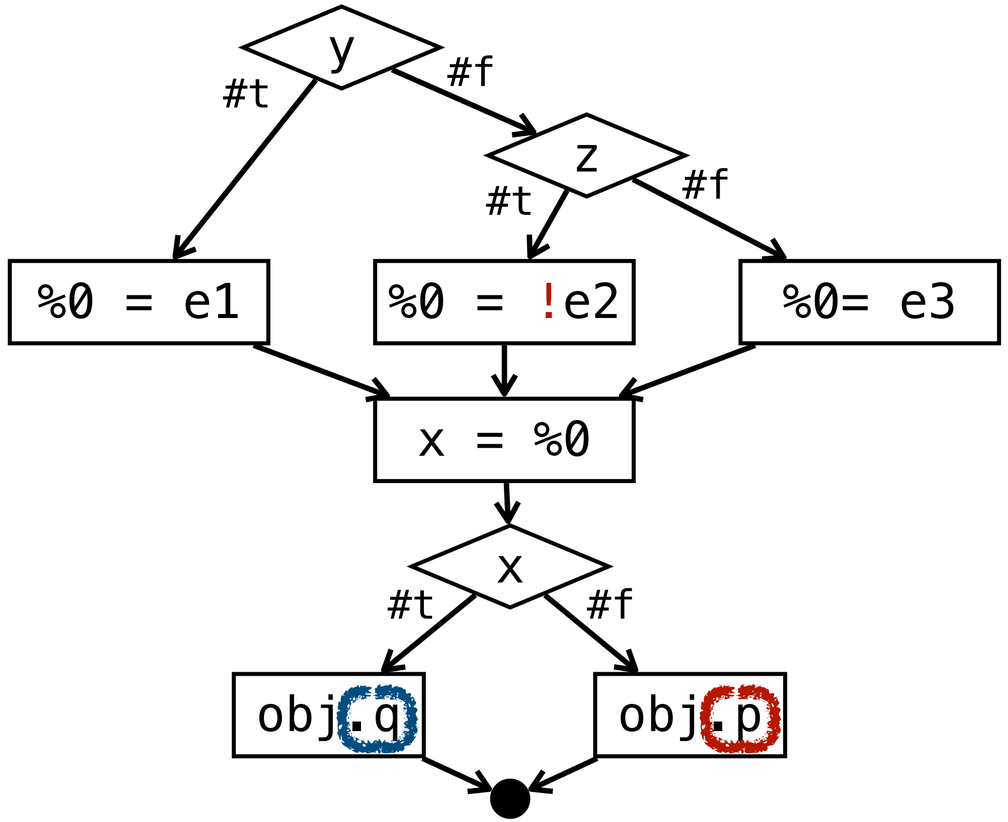
변환

```
var x = y ? e1 : z ? !e2 : e3;
if (x) obj.q;
else   obj.p;
```



!으로 분기 뒤집기

뒤집힌 경로가 더 많음

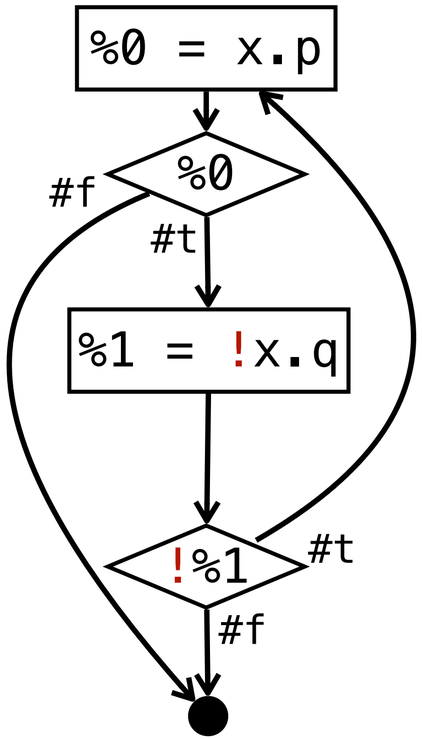


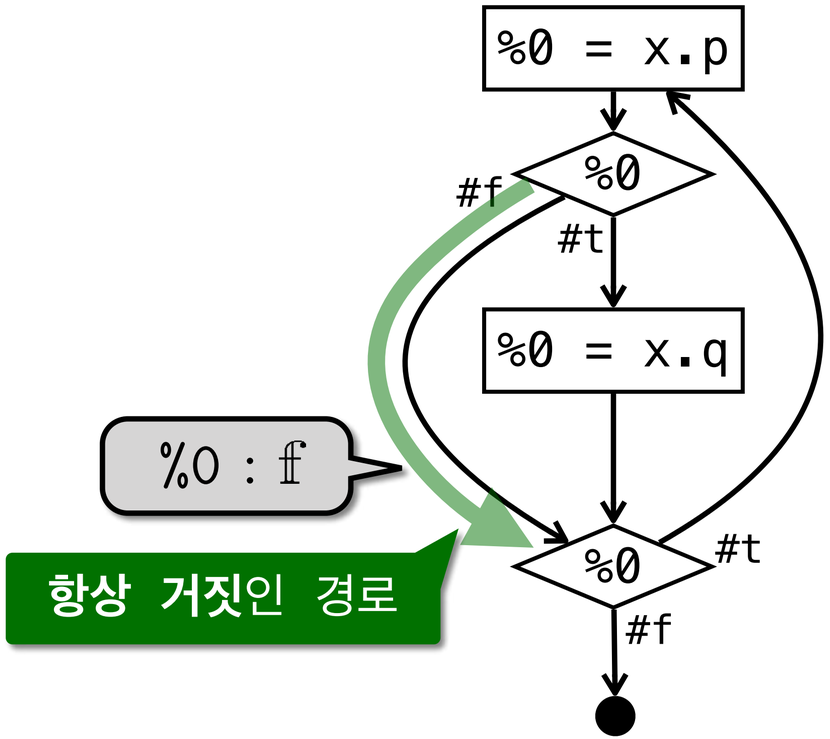
(2) 분기 우회하기 - 항상 참/거짓인 값을 가지는 경로

```
while (x.p) {
  if (!x.q) break;
}
```

변환

```
for (; x.p && x.q; );
```



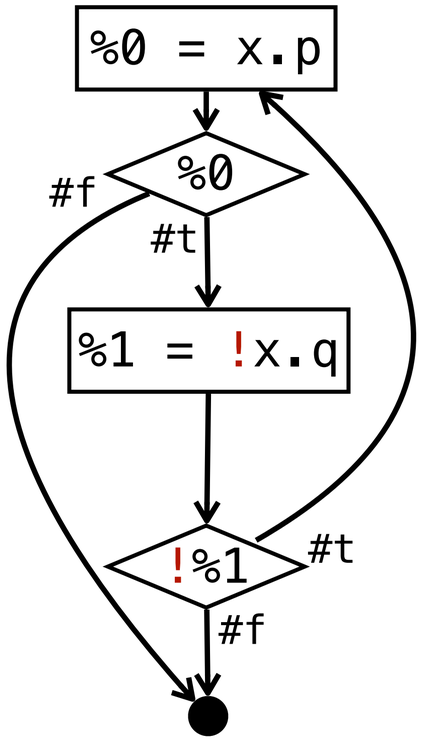


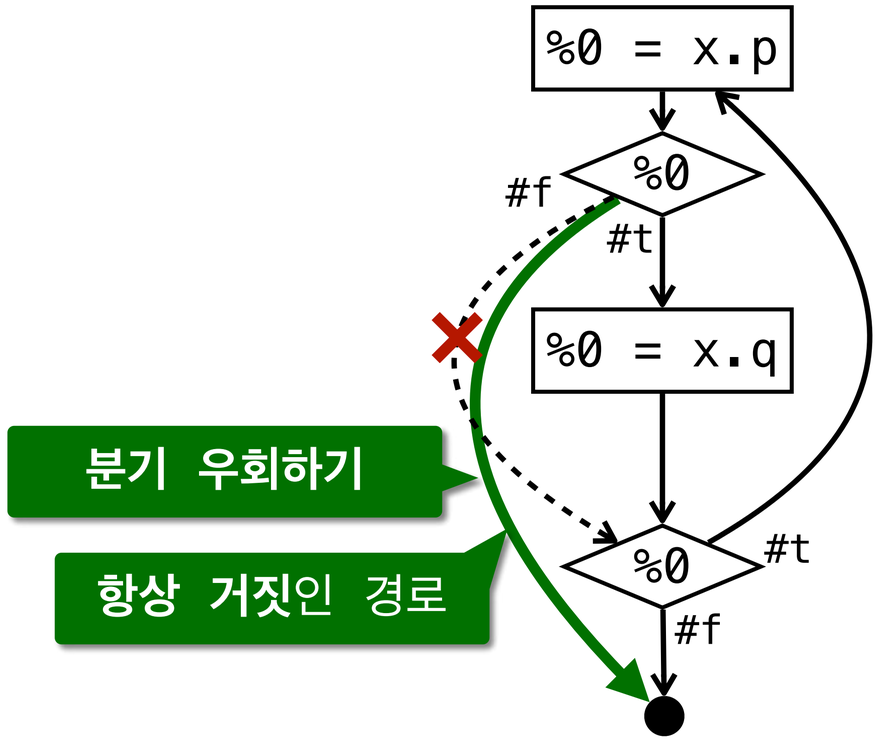
(2) 분기 우회하기 - 항상 참/거짓인 값을 가지는 경로

```
while (x.p) {
  if (!x.q) break;
}
```

변환

```
for (; x.p && x.q; );
```



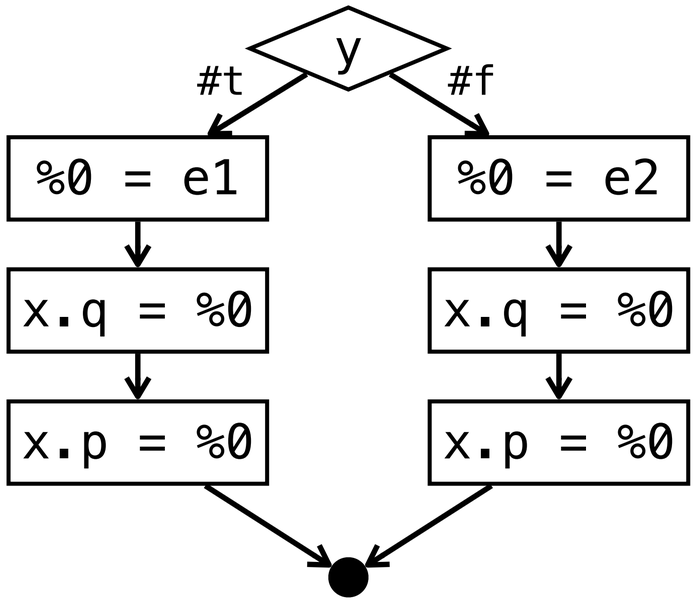


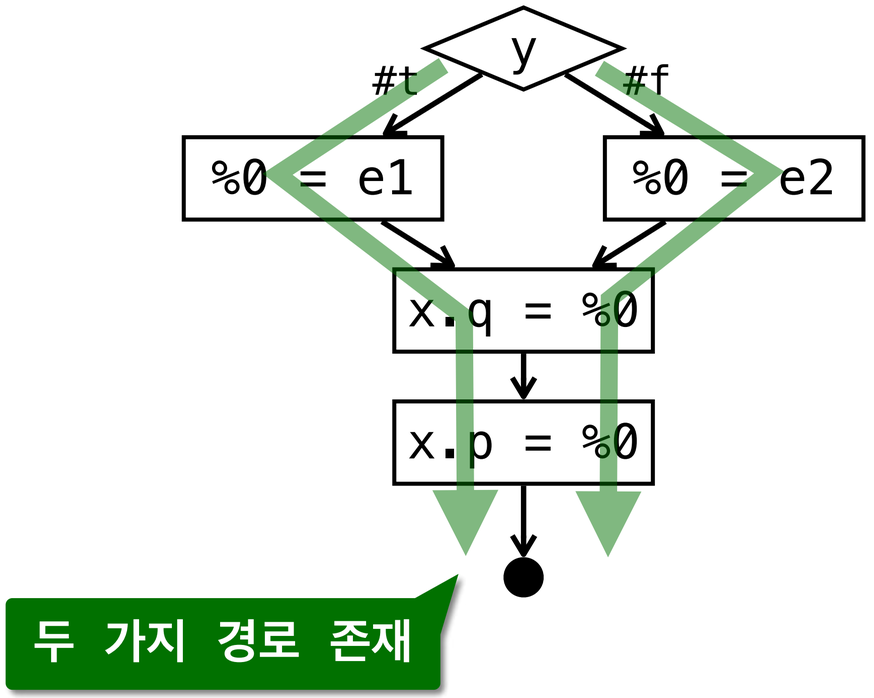
(3) 경로 복사하기

```
if (y) x.p = x.q = e1;
else   x.p = x.q = e2;
```

변환

```
x.p = x.q = y ? e1 : e2;
```



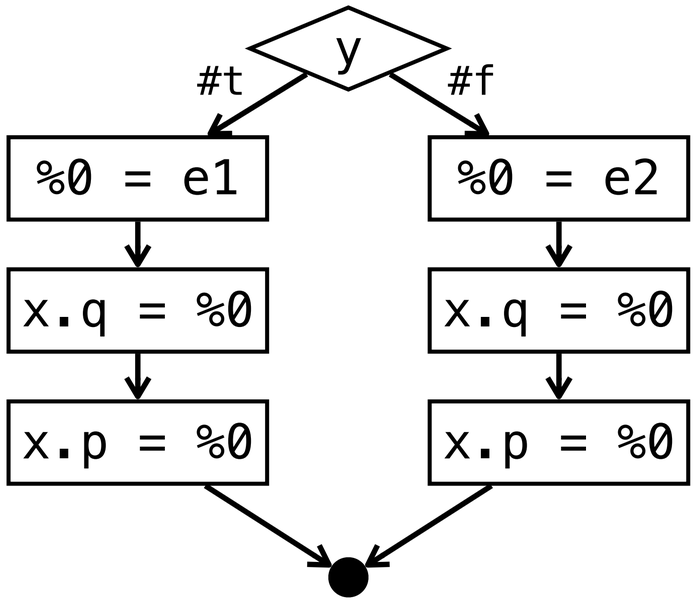


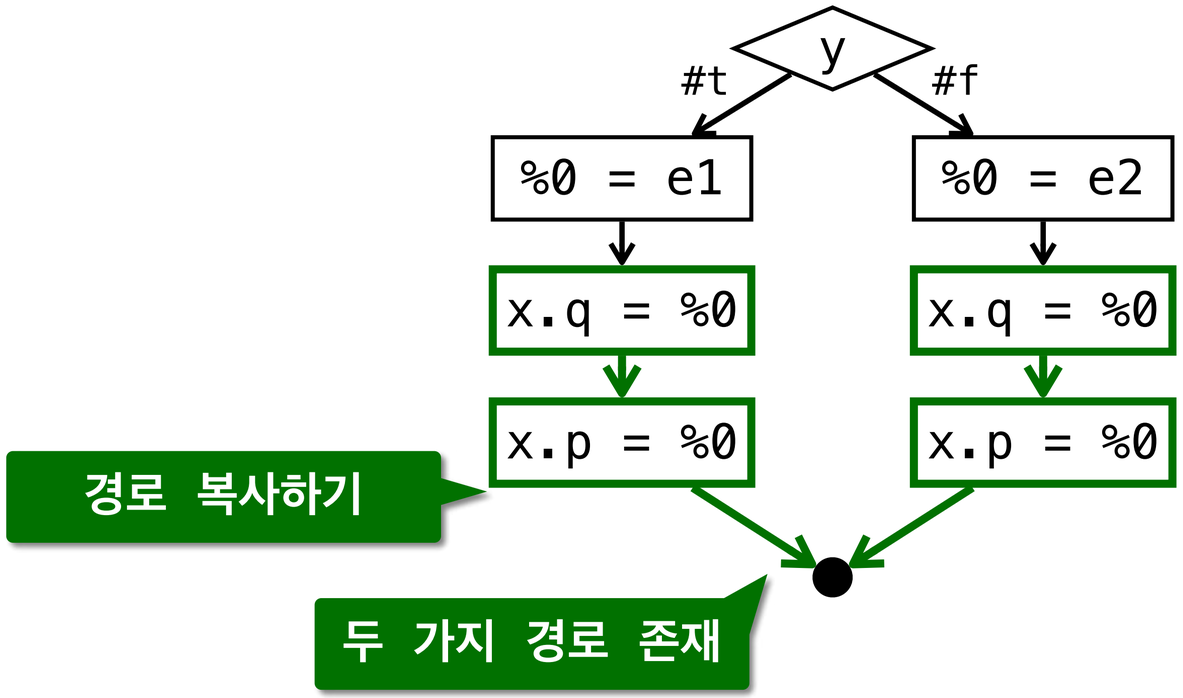
(3) 경로 복사하기

```
if (y) x.p = x.q = e1;
else   x.p = x.q = e2;
```

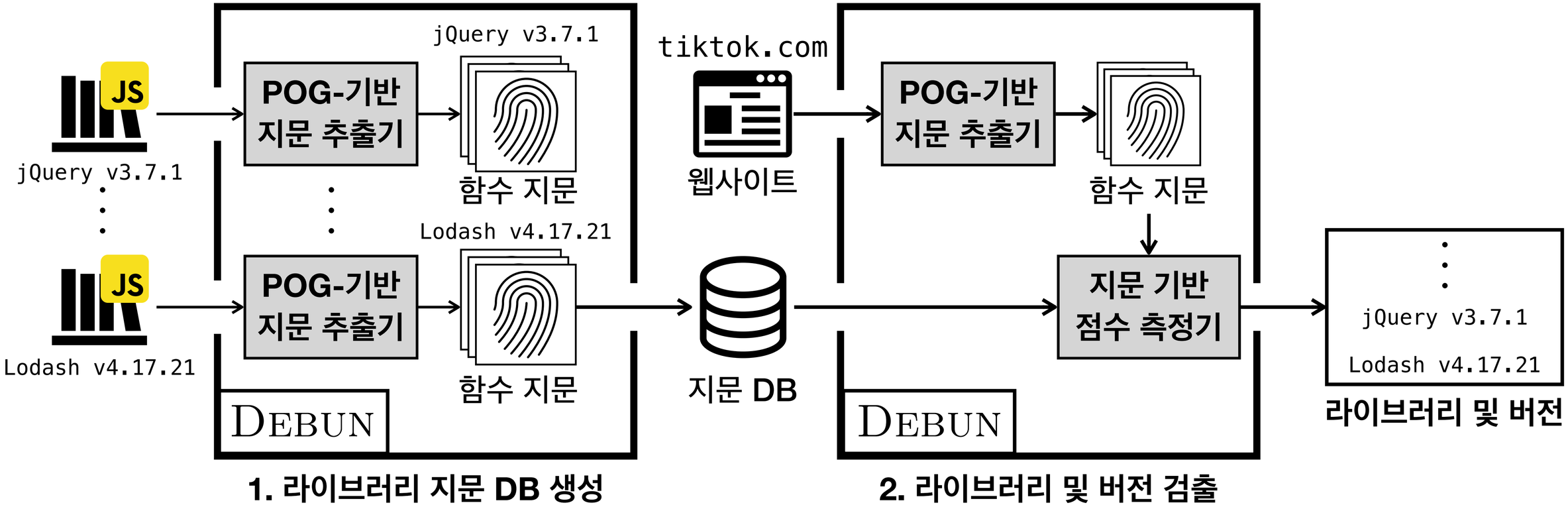
변환

```
x.p = x.q = y ? e1 : e2;
```





Debun: POG를 활용한 라이브러리 검출기



평가

- Cdnjs에 등록된 **78개 라이브러리**의 **8,256개 버전**으로 라이브러리 **지문 DB 구성**
- 트래픽 기준 상위 100개의 웹사이트에서 크롤링이 가능한 **68개의 웹사이트**를 대상으로 실험 진행
- **RQ1)** 기존 기술과의 **라이브러리 검출** 능력 비교
- **RQ2)** 기존 기술과의 라이브러리 **버전 검출** 능력 비교
- **RQ3) 소거 실험** - **경로 민감 진리값 분석** 기반 **그래프 변형**의 효과

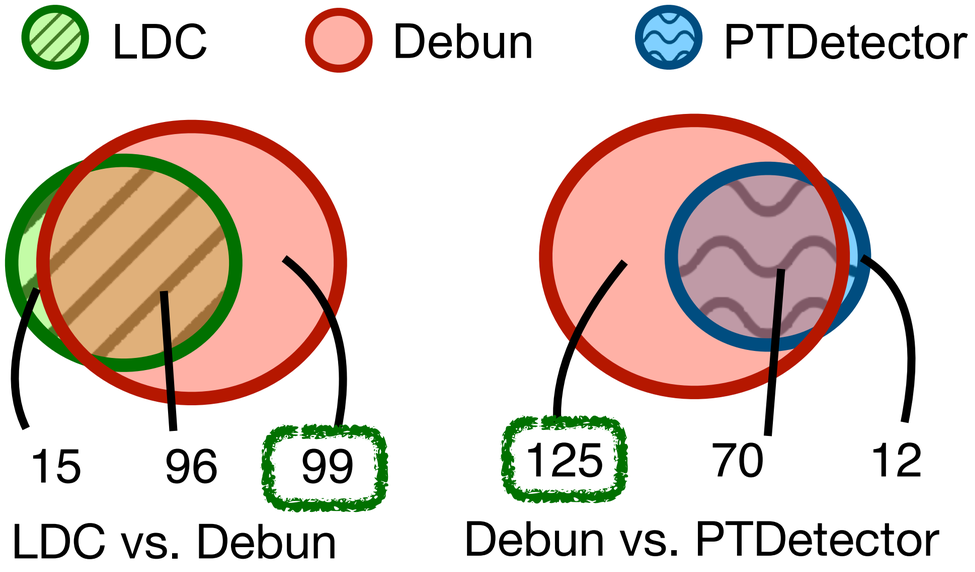
RQ1) 기존 기술과의 **라이브러리 검출** 능력 비교

- 웹 사이트 당 **평균 1,009ms**에 라이브러리 및 버전 검출

Metric	LDC	PTDETECTOR	DEBUN
TP	111	82	195
FP	3	9	7
FN	112	141	28
Precision	97.37%	90.11%	96.53%
Recall	49.78%	36.77%	87.44%
F1-score	65.88%	52.23%	91.76%

+39.53%p

+25.88%p

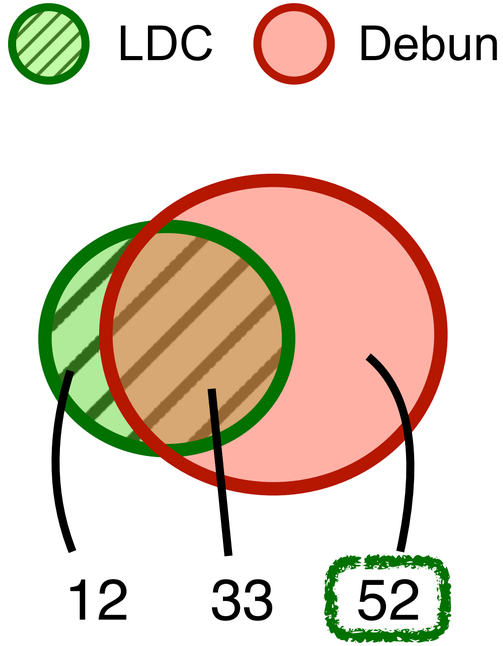


RQ2) 기존 기술과의 라이브러리 **버전 검출** 능력 비교

- 웹 사이트 당 **평균 1,009ms**에 라이브러리 및 버전 검출

Metric	LDC	DEBUN
TP	45	85
FP	0	16
FN	60	20
Precision	100.00%	84.16%
Recall	42.86%	80.95%
F1 score	60.00%	82.52%

+22.52%p



RQ3) 소거실험 - **경로 민감 진리값 분석** 기반 **그래프 변형**의 효과

유지력(Consistency)은 배포 전 후에 함수 지문이 유지되는 비율

속성 연산 순서 고려 X

Metric	Count	POG	POG+F	POG+FB	POG+FBC
# Consistent	47,385	35,370	43,358	45,404	45,522
# Functions	54,368	54,368	54,368	54,368	54,368
Consistency	87.16%	65.06%	79.75%	83.51%	83.73%

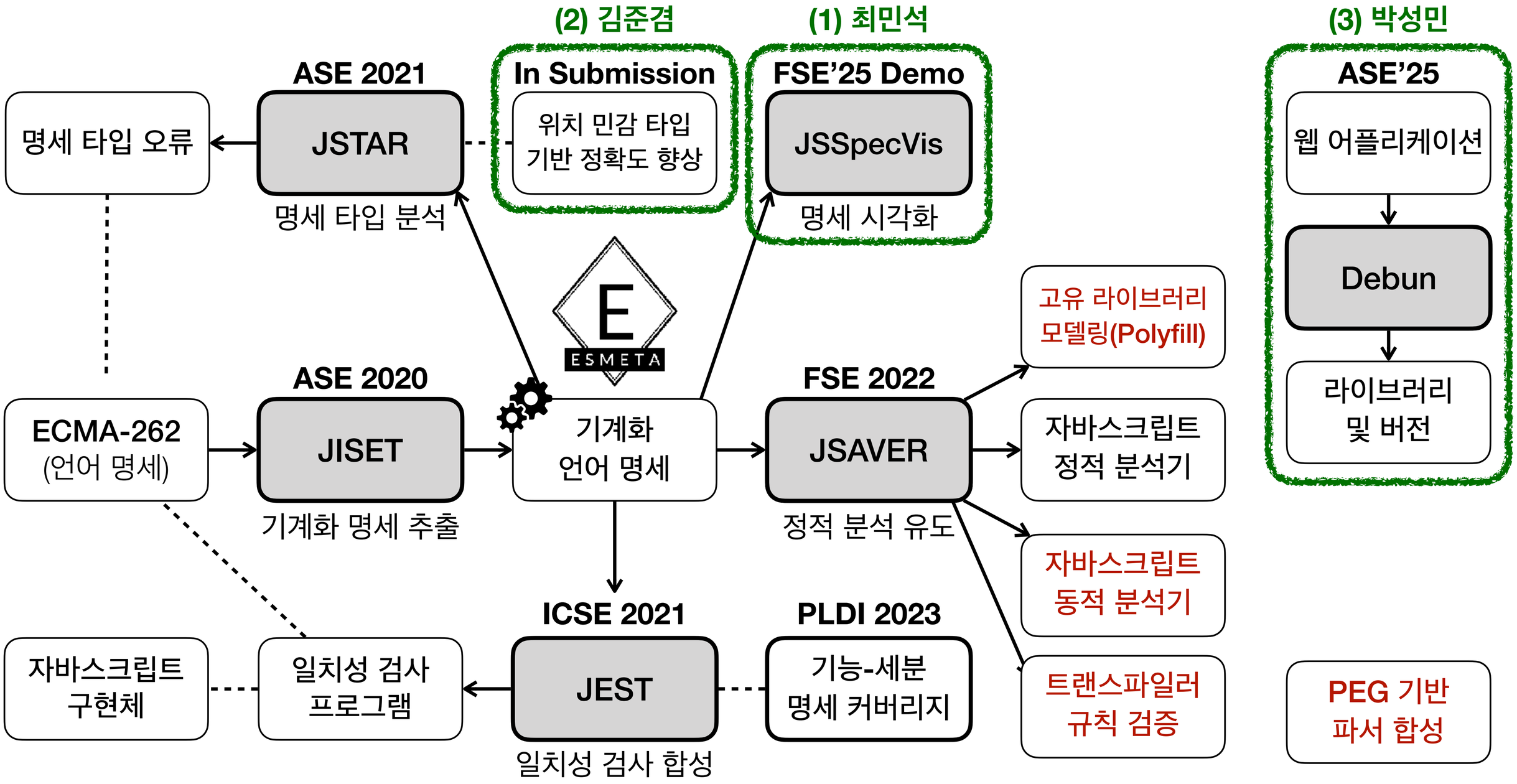
+18.67%p

정확도(Accuracy)는 다른 함수 지문들이 얼마나 다른가에 대한 지표

Metric	Count	POG	POG+F	POG+FB	POG+FBC
# Functions	55,518	55,518	55,518	55,518	55,518
# Duplicated	171,5034	274,252	273,252	273,678	273,684
Accuracy	3.28%	20.24%	20.32%	20.29%	20.29%

낮은 정확도

거의 유지


(2) 김준겸
(1) 최민석
(3) 박성민
ASE 2021
In Submission
FSE'25 Demo
ASE'25
명세 타입 오류
JSTAR
위치 민감 타입
기반 정확도 향상
JSSpecVis
웹 어플리케이션
Debun
라이브러리
및 버전
명세 타입 분석
명세 시각화
ESMETA
ASE 2020
FSE 2022
고유 라이브러리
모델링(Polyfill)
ECMA-262
(언어 명세)
JISET
기계화
언어 명세
JSAVER
자바스크립트
정적 분석기
기계화 명세 추출
정적 분석 유도
자바스크립트
동적 분석기
ICSE 2021
PLDI 2023
자바스크립트
구현체
일치성 검사
프로그램
JEST
기능-세분
명세 커버리지
트랜스파일러
규칙 검증
PEG 기반
파서 합성
일치성 검사 합성