JavaScript 구현체의 오류 데이터베이스

윤동준

2024/01/30 ERC 겨울 정기 워크샵 - 오류 데이터 소개

PLRG
Programming Language Research Group

KAIST

JavaScript

<명세>

JavaScript

GraalVM™

<엔진>

<명세>

PLRG
Programming Language Research Group

KAIST

JavaScript

GraalVM™

<엔진>

<명세>

BABEL

terser

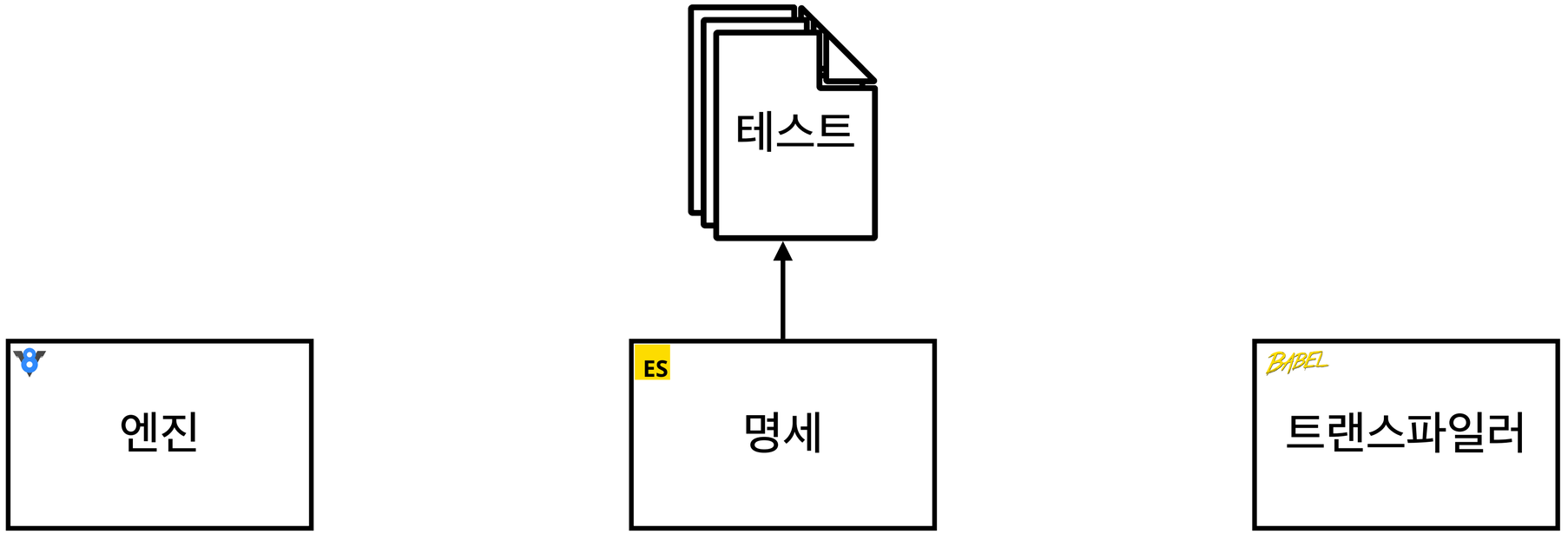
<트랜스파일러>

PLRG
Programming Language Research Group

KAIST

JavaScript - Conformance Test

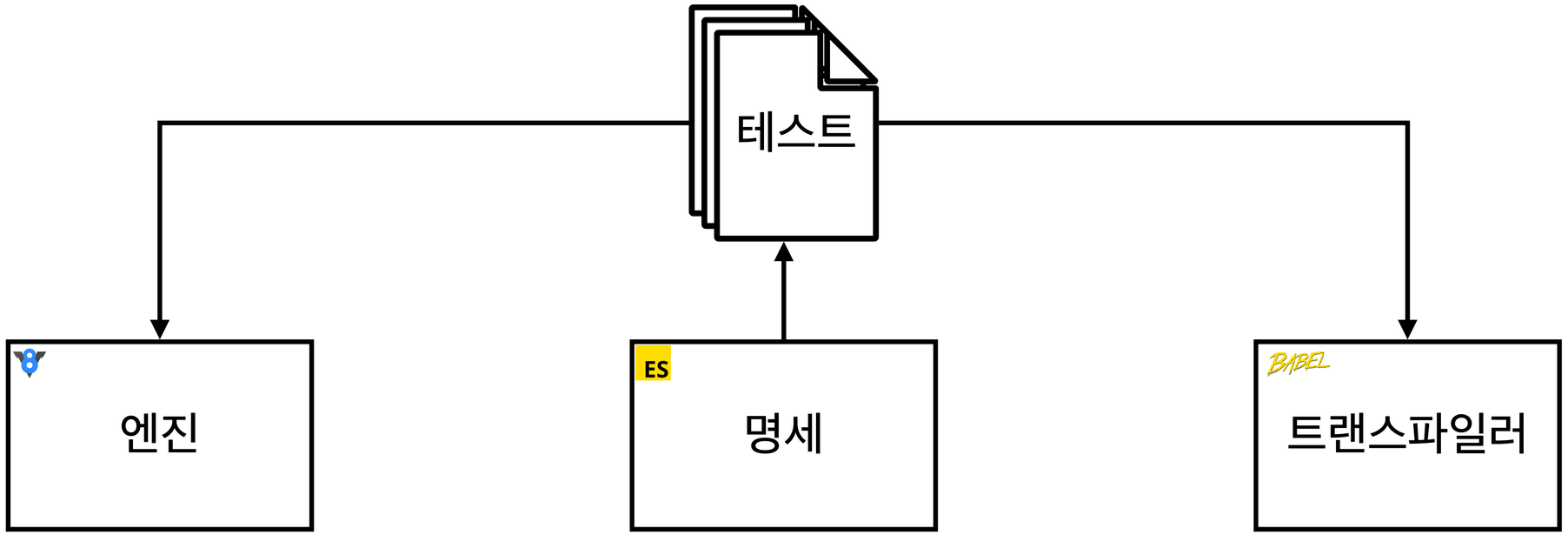
JavaScript - Conformance Test



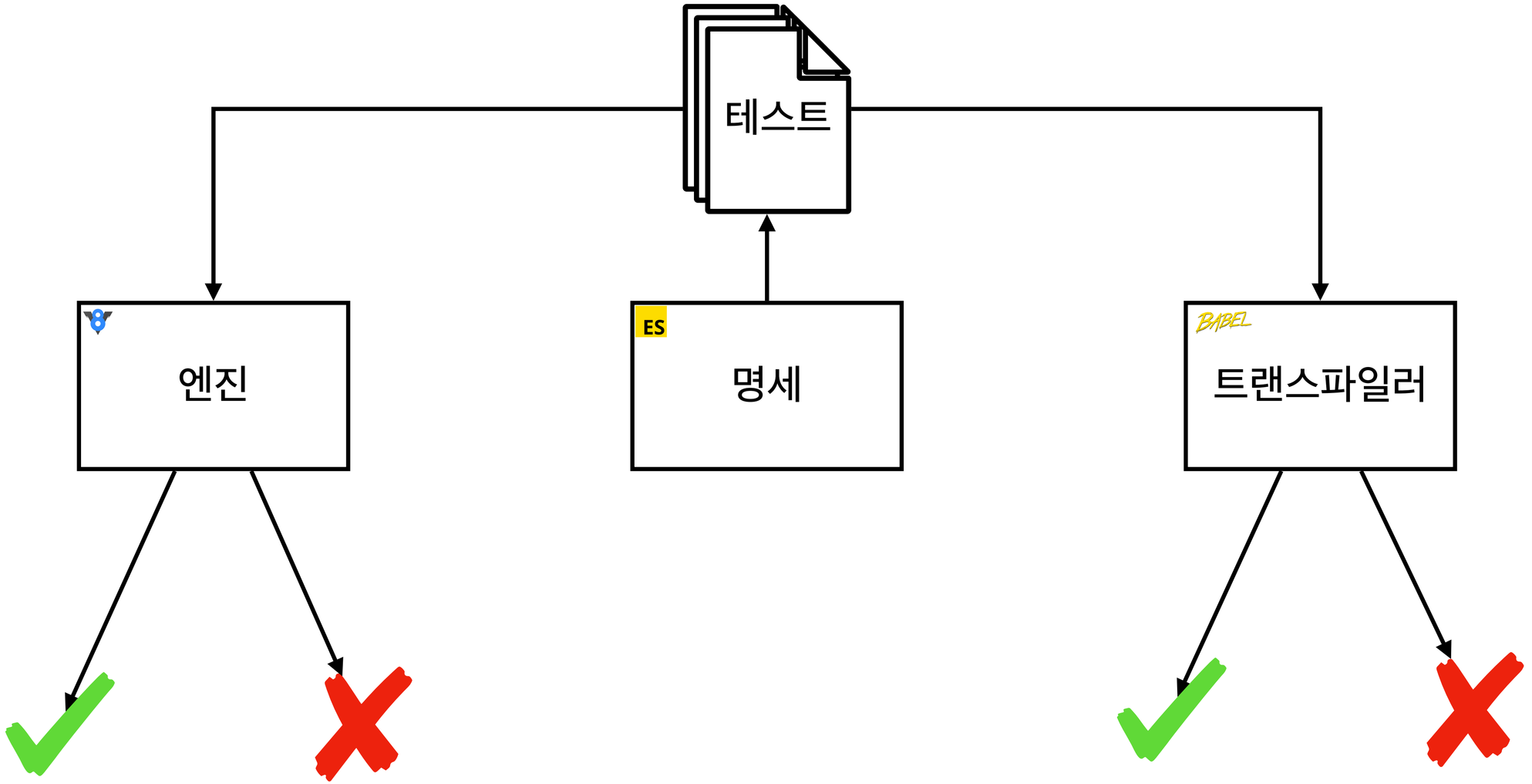
PLRG
Programming Language Research Group

KAIST

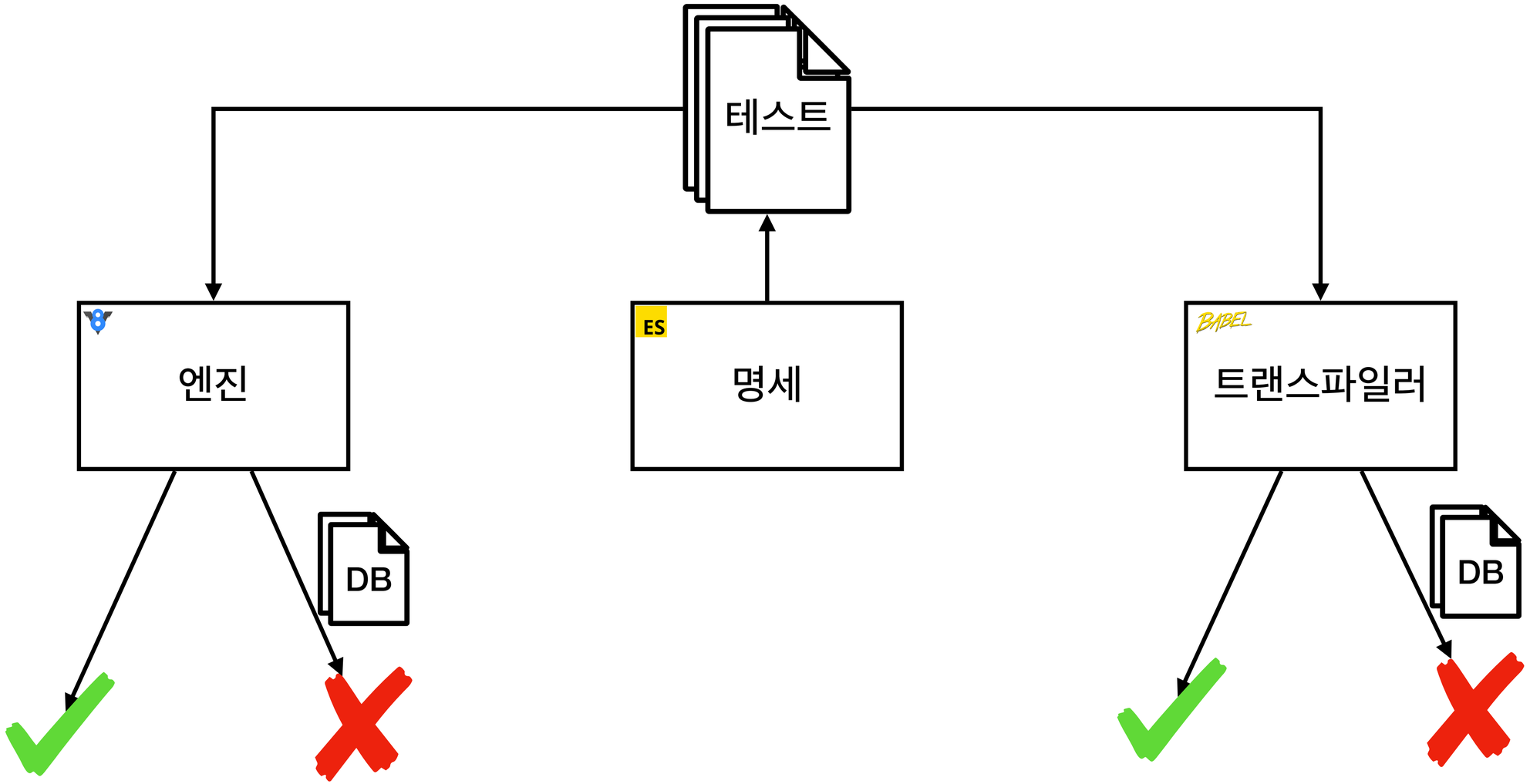
JavaScript - Conformance Test



JavaScript - Conformance Test



JavaScript - Conformance Test



PLRG
Programming Language Research Group

KAIST

본 발표에서는..

PLRG
Programming Language
Research Group

KAIST

본 발표에서는..

1. 효과적인 테스트 생성 방법 (PLDI'23: Feature-Sensitive Coverage)

본 발표에서는..

1. 효과적인 테스트 생성 방법 (PLDI'23: Feature-Sensitive Coverage)

2. JavaScript 구현체 오류 DB 및 예시

본 발표에서는..

1. 효과적인 테스트 생성 방법 (PLDI'23: Feature-Sensitive Coverage)

2. JavaScript 구현체 오류 DB 및 예시

3. 배울 수 있는 점들

1. 효과적인 테스트 생성 방법
(PLDI'23: Feature-Sensitive Coverage)

커버리지 기반 퍼징

13.15.4 EvaluateStringOrNumericBinaryExpression

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

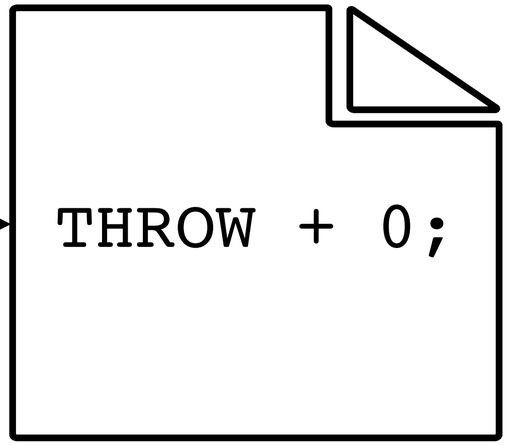
PLRG Programming Language Research Group

KAIST

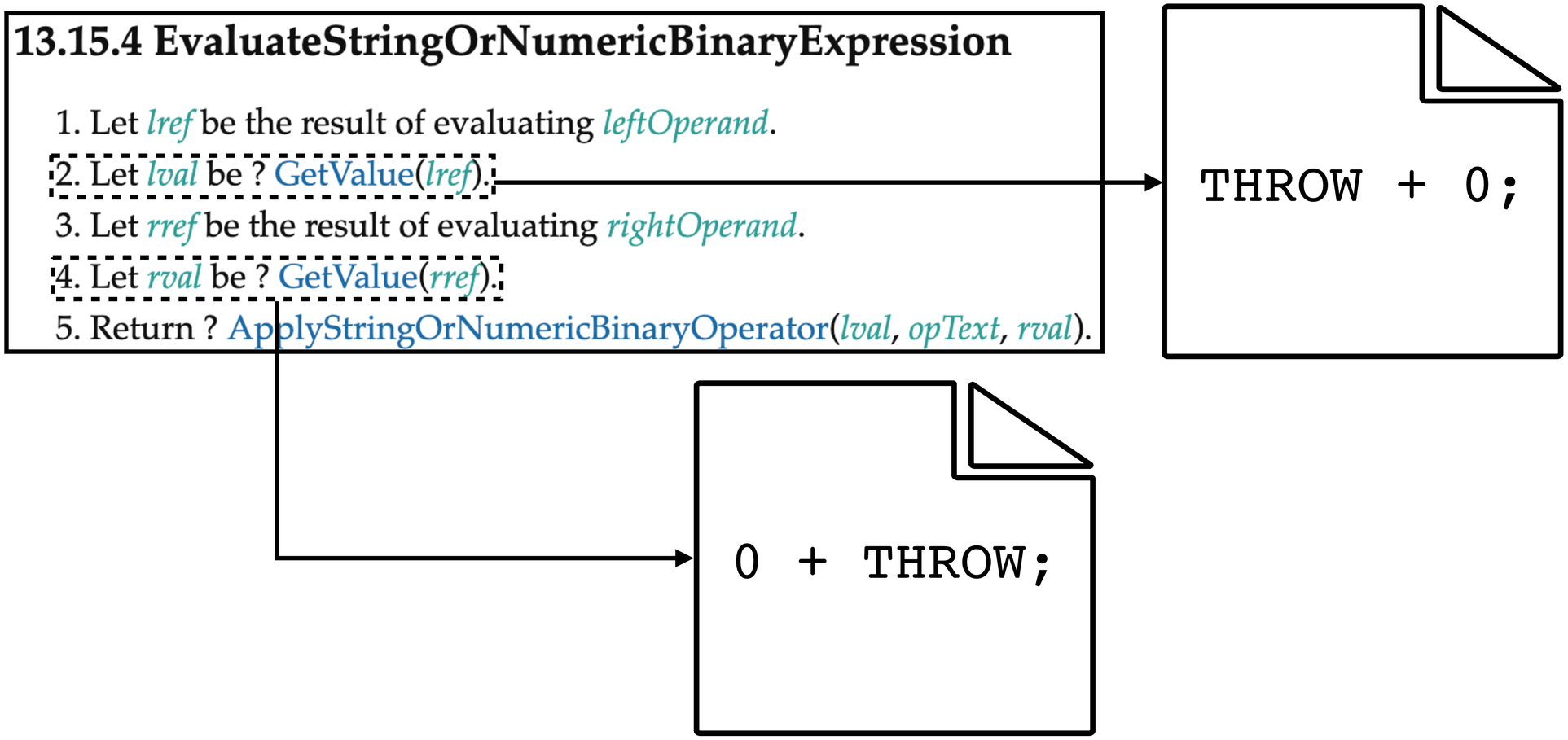
커버리지 기반 퍼징

13.15.4 EvaluateStringOrNumericBinaryExpression

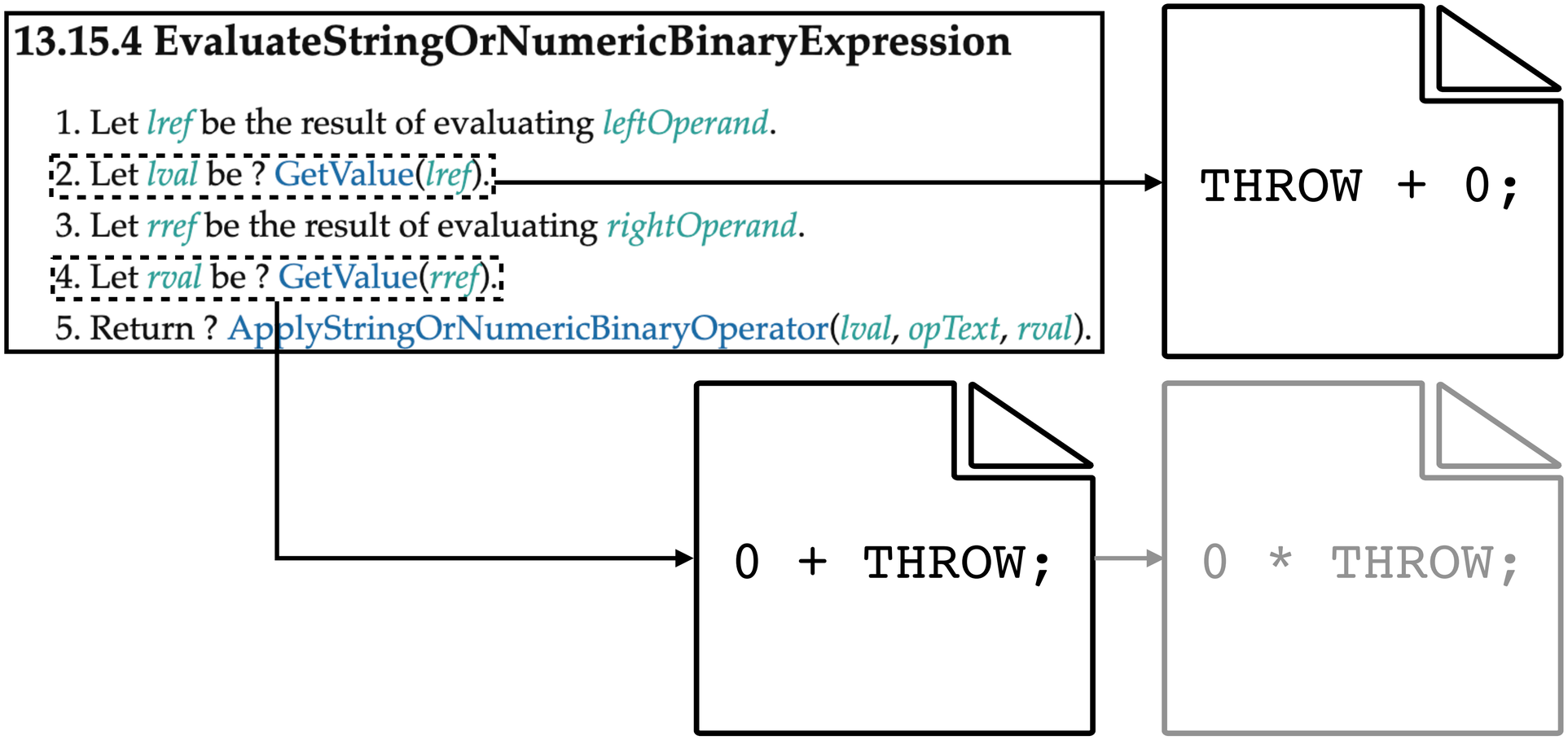
1. Let *lref* be the result of evaluating *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be the result of evaluating *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).



커버리지 기반 퍼징


13.15.4 EvaluateStringOrNumericBinaryExpression
1. Let lref be the result of evaluating leftOperand.
2. Let lval be ? GetValue(lref).
3. Let rref be the result of evaluating rightOperand.
4. Let rval be ? GetValue(rref).
5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
THROW + 0;
0 + THROW;

커버리지 기반 퍼징 - 한계


13.15.4 EvaluateStringOrNumericBinaryExpression
1. Let lref be the result of evaluating leftOperand.
2. Let lval be ? GetValue(lref).
3. Let rref be the result of evaluating rightOperand.
4. Let rval be ? GetValue(rref).
5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
THROW + 0;
0 + THROW;
0 * THROW;

Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations

JIHYEOK PARK, Korea University, South Korea
DONGJUN YOUN, KAIST, South Korea
KANGUK LEE, KAIST, South Korea
SUKYOUNG RYU, KAIST, South Korea

The *conformance testing* of programming language implementations is crucial to support correct and consistent execution environments. Because manually maintaining conformance tests for real-world programming languages is cumbersome and labor-intensive, researchers have presented various ways to make conformance tests effective and efficient. One such approach is to use graph coverage, one of the most widely-used coverage criteria, to generate tests that reach different parts of a *mechanized language specification*. Since mechanized specifications use functions or inductive definitions to describe the semantics of language features, traditional graph coverage criteria for software work as they are. However, they may not produce high-quality conformance tests because language implementations often have specialized execution paths for different features, even when their semantics descriptions use the same functions. Traditional graph coverage may not distinguish test requirements of such language features, which degrades the quality of conformance testing. Similarly, it may not distinguish test requirements of different parts of the same language feature when their semantics descriptions use the same functions.

We present *feature-sensitive (FS) coverage* as a novel coverage criterion to generate high-quality conformance tests for language implementations. It is a general extension of graph coverage, refining conventional test requirements using the innermost enclosing language features. We also introduce *feature-call-path-sensitive (FCPS) coverage*, a variant of FS coverage, and extend both coverage criteria using the k-limiting approach. To evaluate the effectiveness of the new coverage criteria for language implementations, we apply them to a mechanized specification of JavaScript. We extend JEST, the state-of-the-art JavaScript conformance test synthesizer using coverage-guided mutational fuzzing, with various FS and FCPS coverage criteria. For the latest JavaScript language specification (ES13, 2022), our tool automatically synthesizes 237,981 conformance tests in 50 hours with five coverage criteria. We evaluated the conformance of eight mainstream JavaScript implementations (four engines and four transpilers) with the synthesized conformance tests and discovered bugs in all of them. The tool detected 143 distinct conformance bugs (42 in engines and 101 in transpilers), 85 of which were confirmed by the developers and 83 of which were newly discovered bugs.

PLRG Programming Language Research Group

KAIST

Feature

Feature

14.6.2 Runtime Semantics: Evaluation

IfStatement : **`if (`** *Expression* **`)`** *Statement* **`else`** *Statement*

1. Let *exprRef* be the result of evaluating *Expression*.
2. Let *exprValue* be ToBoolean(? GetValue(*exprRef*)).
3. If *exprValue* is **true**, then
    a. Let *stmtCompletion* be the result of evaluating the first *Statement*.
4. Else,
    a. Let *stmtCompletion* be the result of evaluating the second *Statement*.
5. Return ? UpdateEmpty(*stmtCompletion*, **undefined**).

<Syntactic feature>

Feature

14.6.2 Runtime Semantics: Evaluation

IfStatement : `if` (*Expression*) *Statement* `else` *Statement*

1. Let *exprRef* be the result of evaluating *Expression*.
2. Let *exprValue* be ToBoolean(? GetValue(*exprRef*)).
3. If *exprValue* is **true**, then
    a. Let *stmtCompletion* be the result of evaluating the first *Statement*.
4. Else,
    a. Let *stmtCompletion* be the result of evaluating the second *Statement*.
5. Return ? UpdateEmpty(*stmtCompletion*, **undefined**).

<Syntactic feature>

23.1.3.14 Array.prototype.includes (*searchElement* [, *fromIndex*])

1. Let *O* be ? ToObject(**this** value).
2. Let *len* be ? LengthOfArrayLike(*O*).
3. If *len* is 0, return **false**.
4. Let *n* be ? ToIntegerOrInfinity(*fromIndex*).
5. Assert: If *fromIndex* is **undefined**, then *n* is 0.
6. If *n* is $+\infty$, return **false**.
7. Else if *n* is $-\infty$, set *n* to 0.
8. If $n \geq 0$, then
    a. Let *k* be *n*.
9. Else,
    a. Let *k* be *len* + *n*.
    b. If $k < 0$, set *k* to 0.
10. Repeat, while $k < len$,
    a. Let *elementK* be ? Get(*O*, ! ToString(ℝ(*k*))).
    b. If SameValueZero(*searchElement*, *elementK*) is **true**, return **true**.
    c. Set *k* to $k + 1$.
11. Return **false**.

<Built-in API feature>

PLRG Programming Language Research Group

KAIST

Feature-Sensitive Coverage

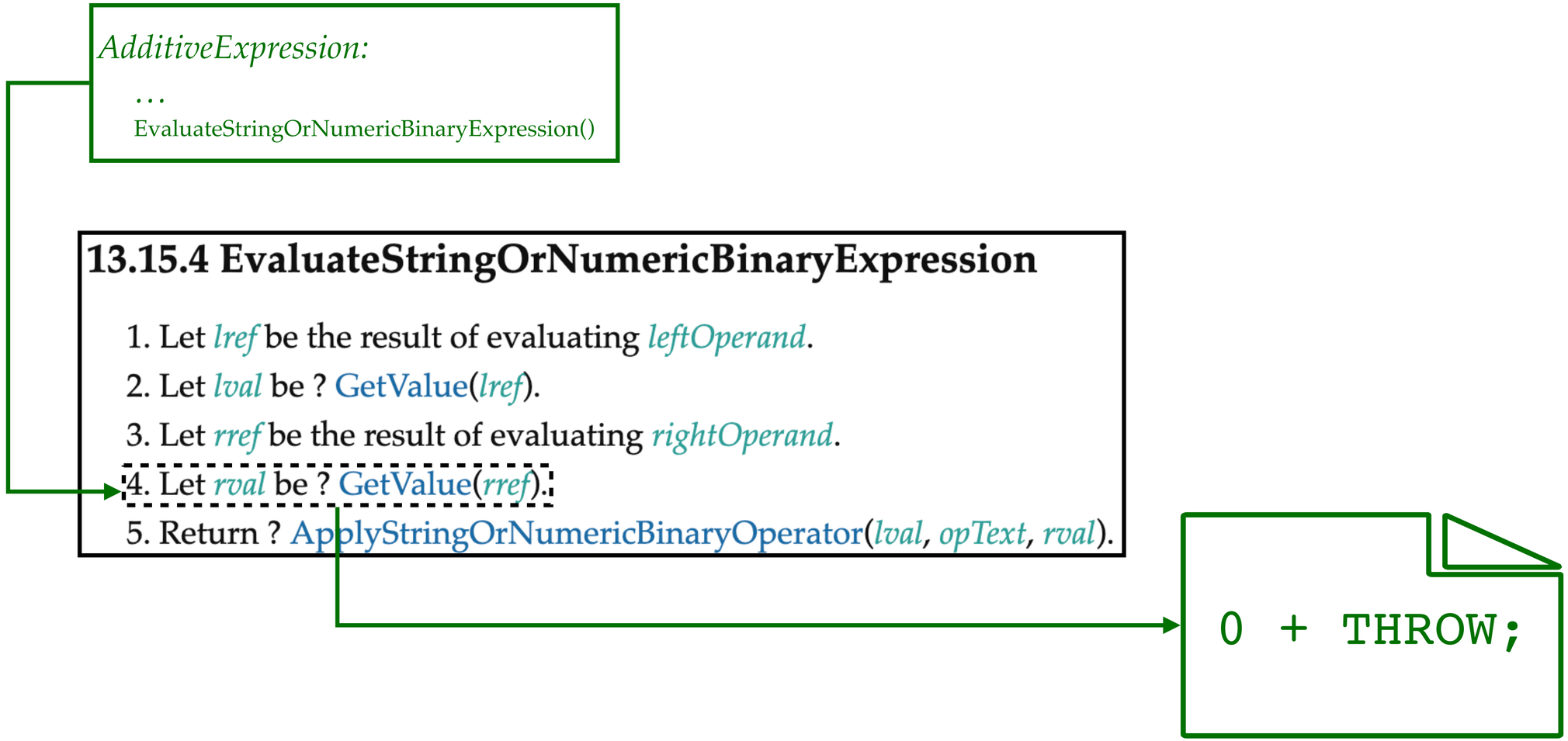
13.15.4 EvaluateStringOrNumericBinaryExpression

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

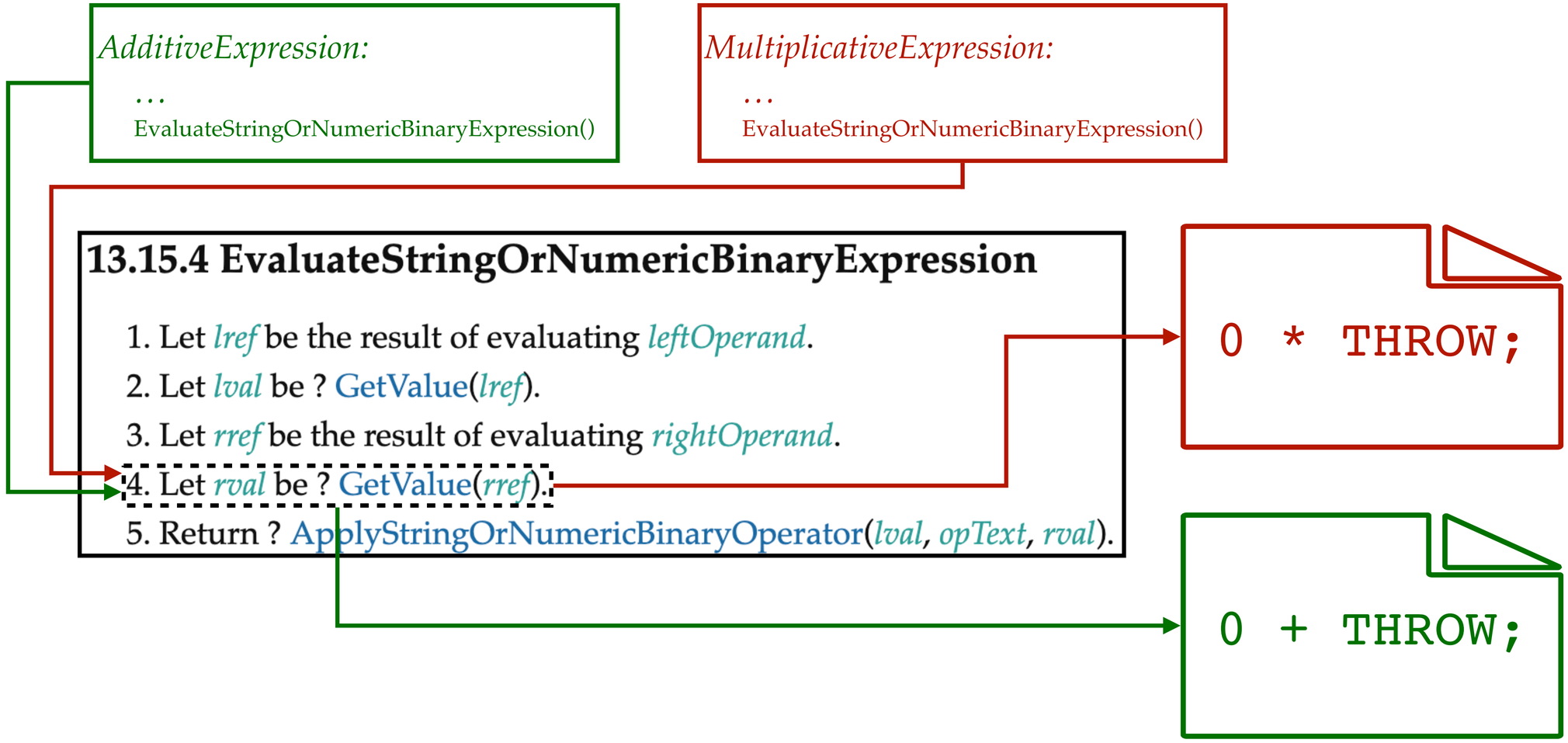
PLRG Programming Language Research Group

KAIST

Feature-Sensitive Coverage



Feature-Sensitive Coverage



Feature-Sensitive Coverage - k-fs

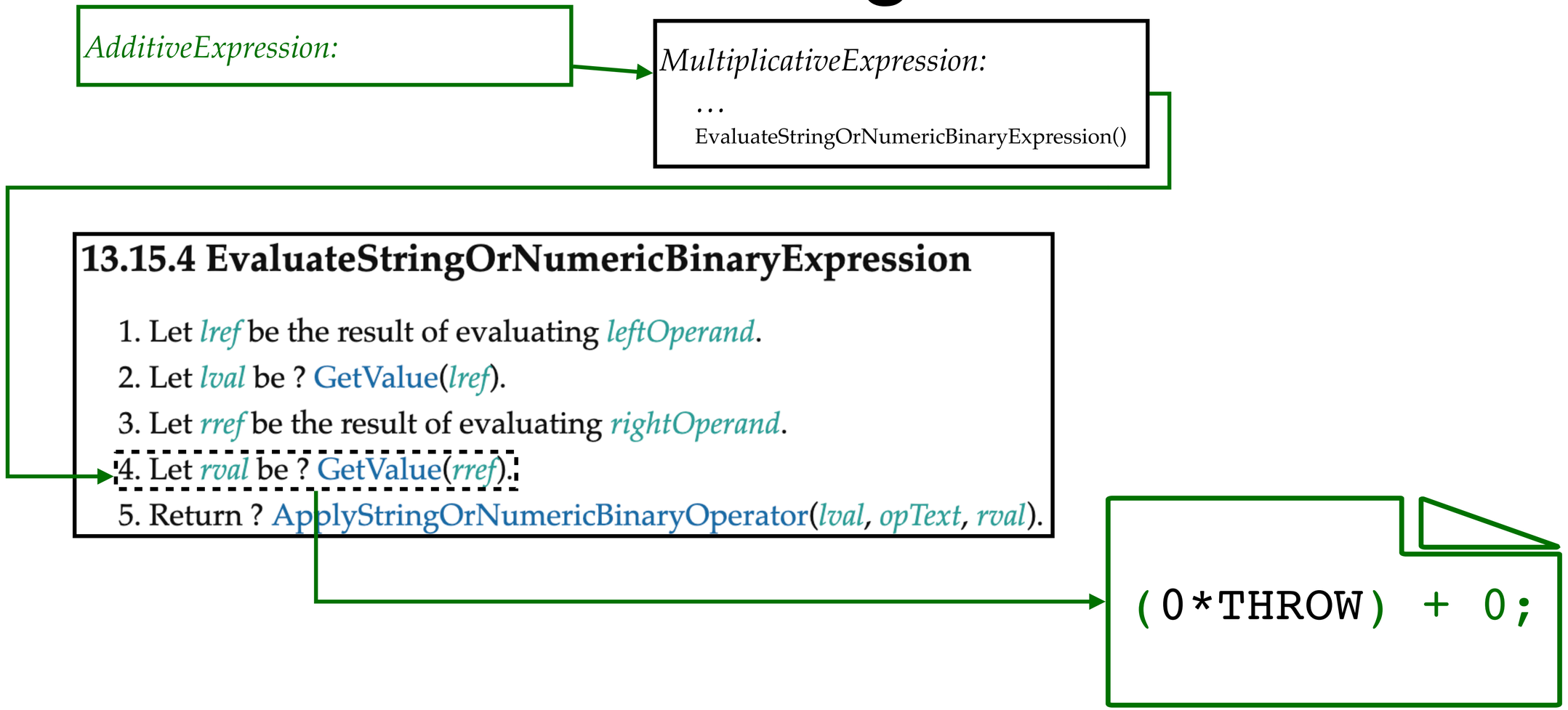
13.15.4 EvaluateStringOrNumericBinaryExpression

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

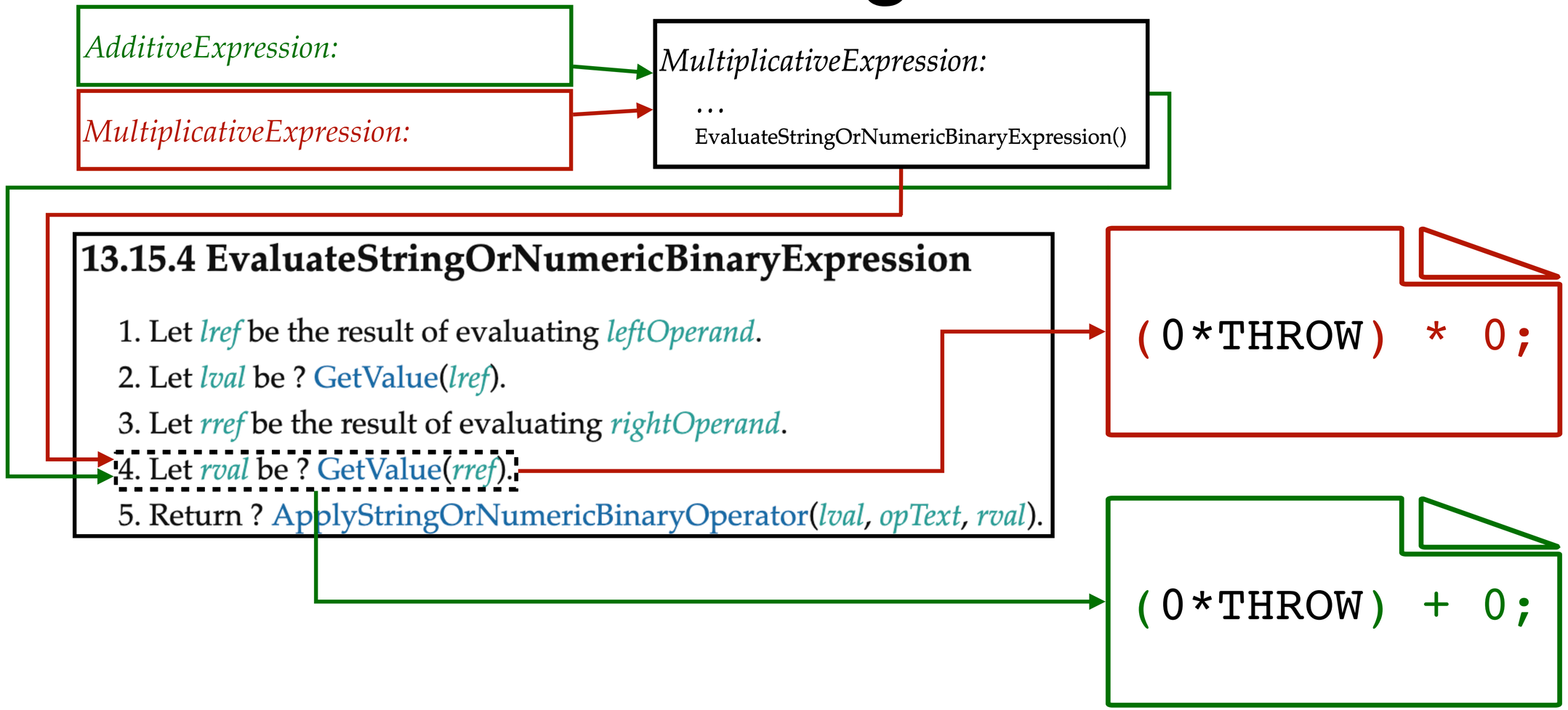
PLRG Programming Language Research Group

KAIST

Feature-Sensitive Coverage - k-fs



Feature-Sensitive Coverage - k-fs



Feature-Sensitive Coverage - fcps

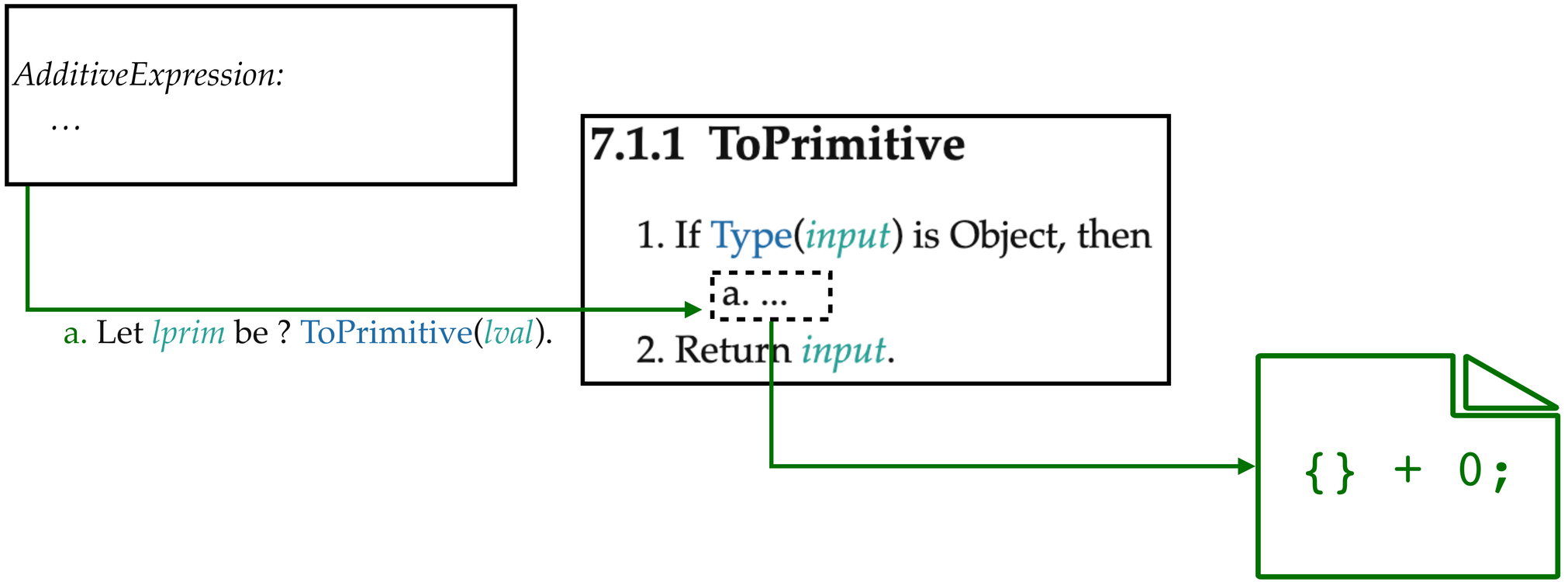
AdditiveExpression:

…

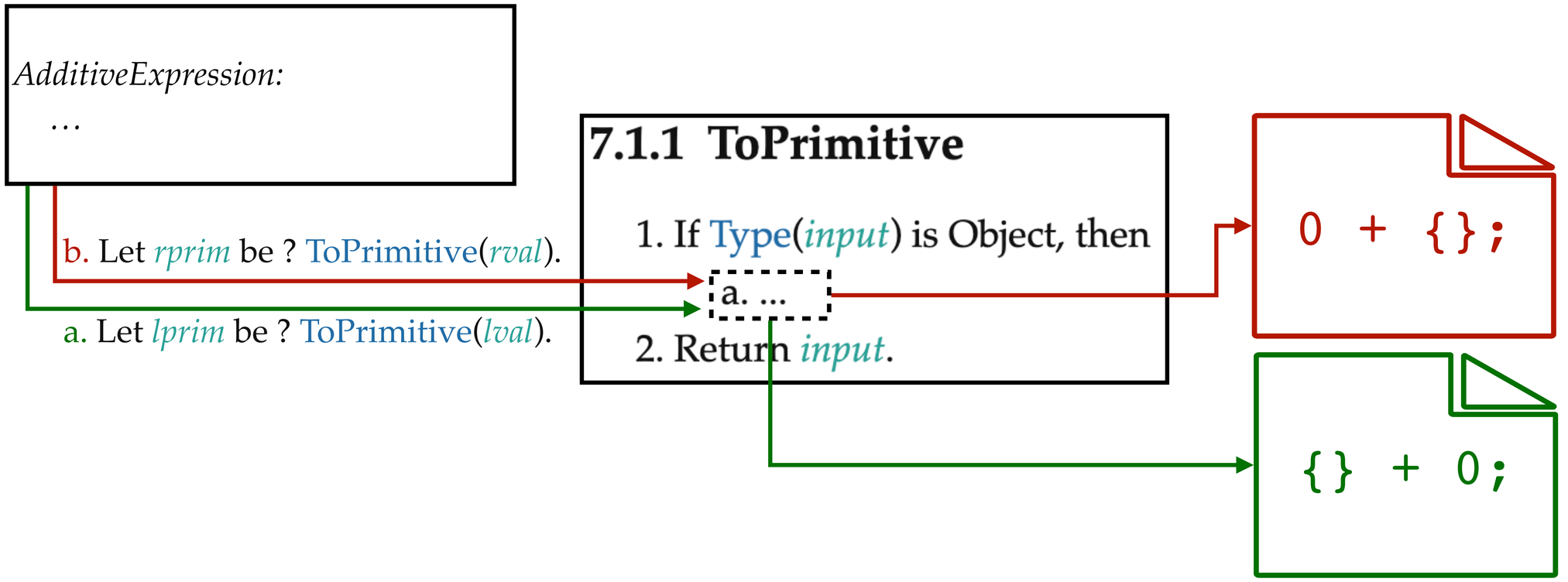
7.1.1 ToPrimitive

1. If Type(*input*) is Object, then
   a. ...
2. Return *input*.

Feature-Sensitive Coverage - fcps



Feature-Sensitive Coverage - fcps



2. JavaScript 구현체 오류 DB 및 예시

JavaScript 구현체 오류 DB

	Id	Title	Represetative Example	Issue Link
1	BBL-01	[Babel] Destructuring null/undefined with rest	var { ... x } = null ;	https://github.com/babel/babel/issues/14982
2	BBL-02	[Babel] [Crash] Duplicated name in class static ...	class x { static { var x = 42; } }	https://github.com/babel/babel/issues/15099
3	BBL-03	[Babel] Re-assign to class during static field init...	class C { static x = C = 0 ; } ;	https://github.com/babel/babel/issues/15000
4	BBL-04	[Babel] Generator with array pattern parameter	function* f([]){}; f();	https://github.com/babel/babel/issues/15012
5	BBL-05	[Babel] Name property - Unnamed to wrong na...	var [x] = [function ( ) { }] ;	https://github.com/babel/babel/issues/14986

https://github.com/orgs/kaist-plrg/projects/3

PLRG Programming Language Research Group

KAIST

엔진 버그 예시 1: GRL-05

```
false && delete f();
```

Expected: f is not called.
Actual: f is called.

PLRG
Programming Language Research Group

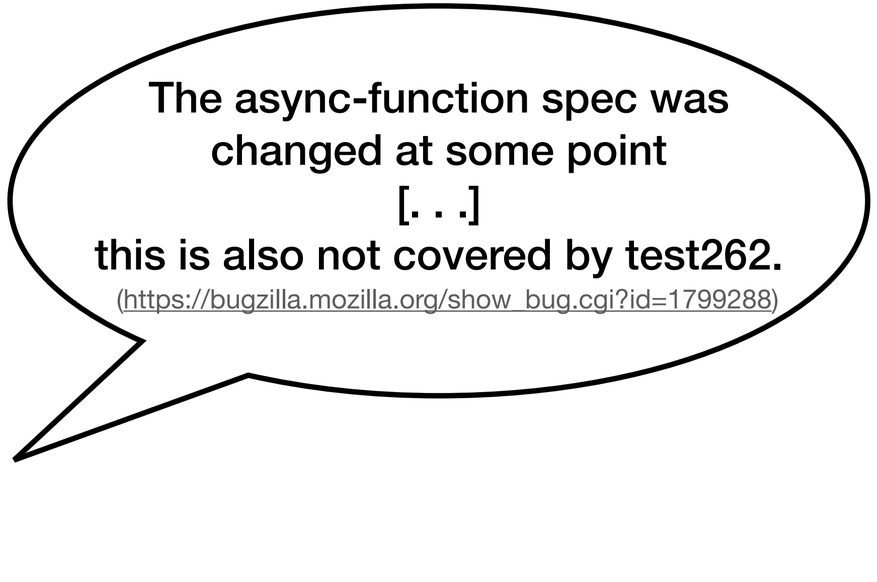
KAIST

엔진 버그 예시 2: SM-01

```
async function f([]){}
f();
```

Expected: Rejected Promise
Actual: TypeError

엔진 버그 예시 2: SM-01




```
async function f([]){}
f();
```

Expected: Rejected Promise
Actual: TypeError

PLRG
Programming Language
Research Group

KAIST

트랜스파일러 버그 예시: BBL-00

```
for (let {} = 0; 0;);
```

Expected: Normal
Actual: Crash

트랜스파일러 버그 예시: BBL-00

```
for (let {} = 0; 0;);
```

Expected: Normal
Actual: Normal

-> Effect of Feature-Sensitivity

PLRG
Programming Language Research Group

KAIST

3. 배울 수 있는 점들

배울 수 있는 점들

배울 수 있는 점들

1. 오류 유발 입력
    1. Expected behavior가 Throw Exception인 경우
    2. 신규 Feature 관련 (ES 2022)

배울 수 있는 점들

1. 오류 유발 입력
    1. Expected behavior가 Throw Exception인 경우
    2. 신규 Feature 관련 (ES 2022)
2. 스펙을 더욱 세분화해서 측정하는 커버리지 기준
   -> [프로그래밍 언어(JavaScript)의 구현체]라는 SW 뿐 아니라, 더 넓은 범위에 적용 가능하지 않을까?

PLRG
Programming Language Research Group

KAIST