Filling the gap between the JavaScript language specification and tools using the JISET family

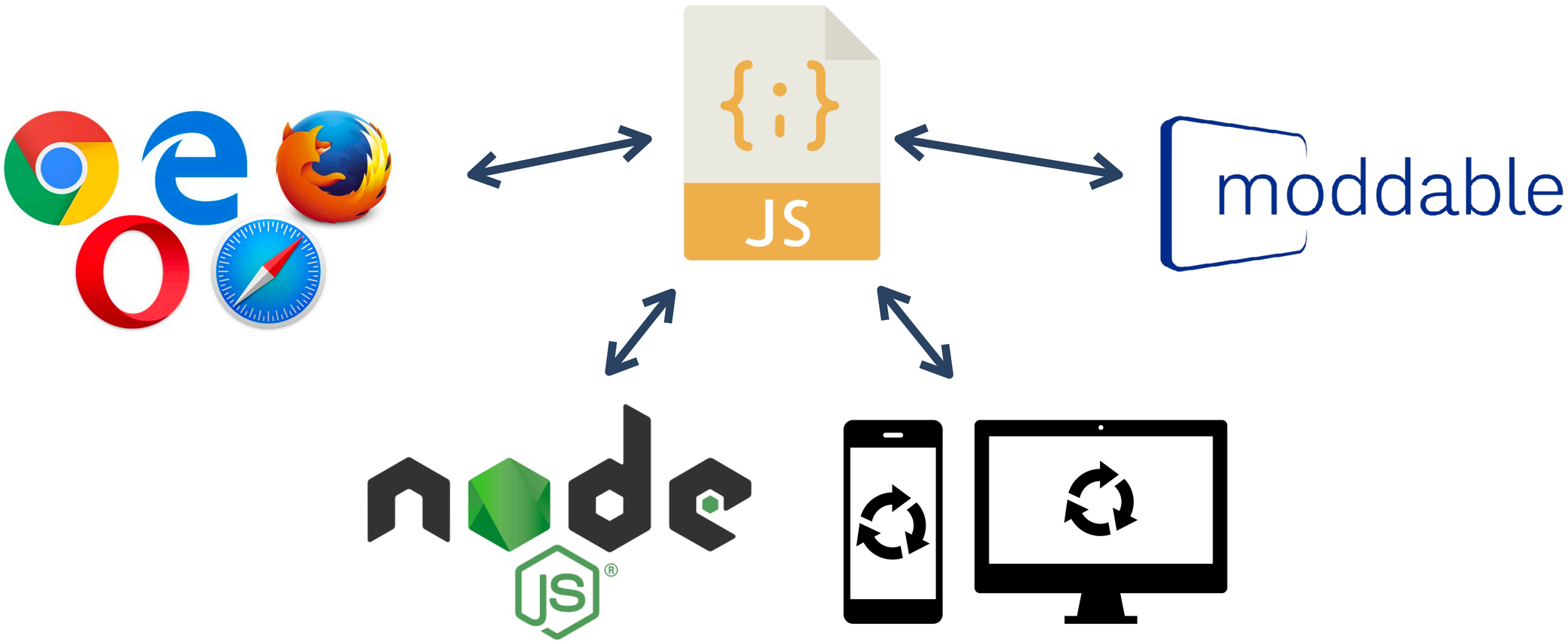
Tutorial @ PLDI 2022

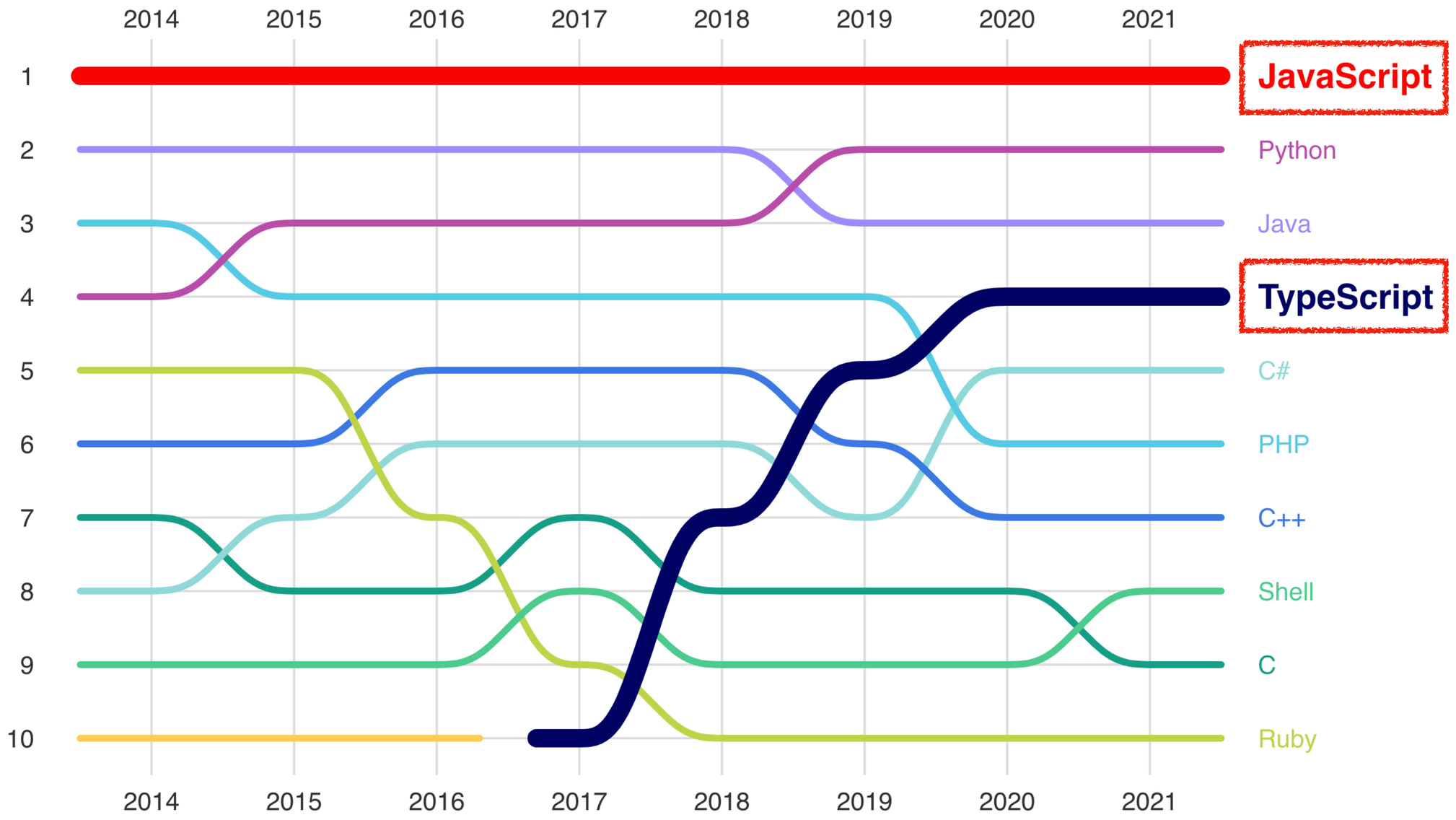
Sukyoung Ryu [1], **Jihyeok Park** [2], Seungmin An [1]

[1] KAIST, South Korea
[2] Oracle Labs, Australia

June 13, 2022

JavaScript Is Everywhere





https://octoverse.github.com/

JavaScript Complex Semantics

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

Always return `false`?

JavaScript Complex Semantics

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

Always return `false`?

NO!!

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

ECMA-262: JavaScript Language Specification

ecma INTERNATIONAL maintained by TC 39

Syntax

ArrayLiteral[Yield, Await] :

[*Elision*opt]

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

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

The production of *ArrayLiteral* in ES12

Semantics

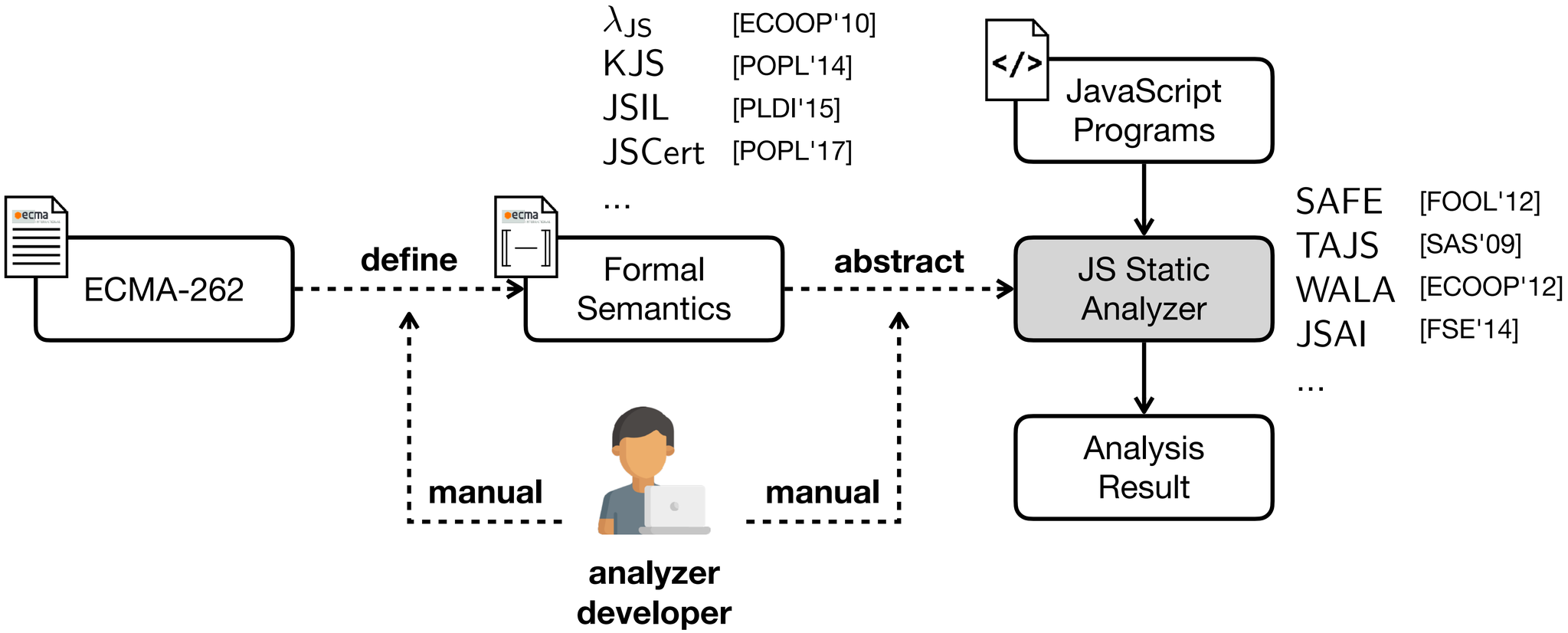
13.2.5.2 Runtime Semantics: Evaluation

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

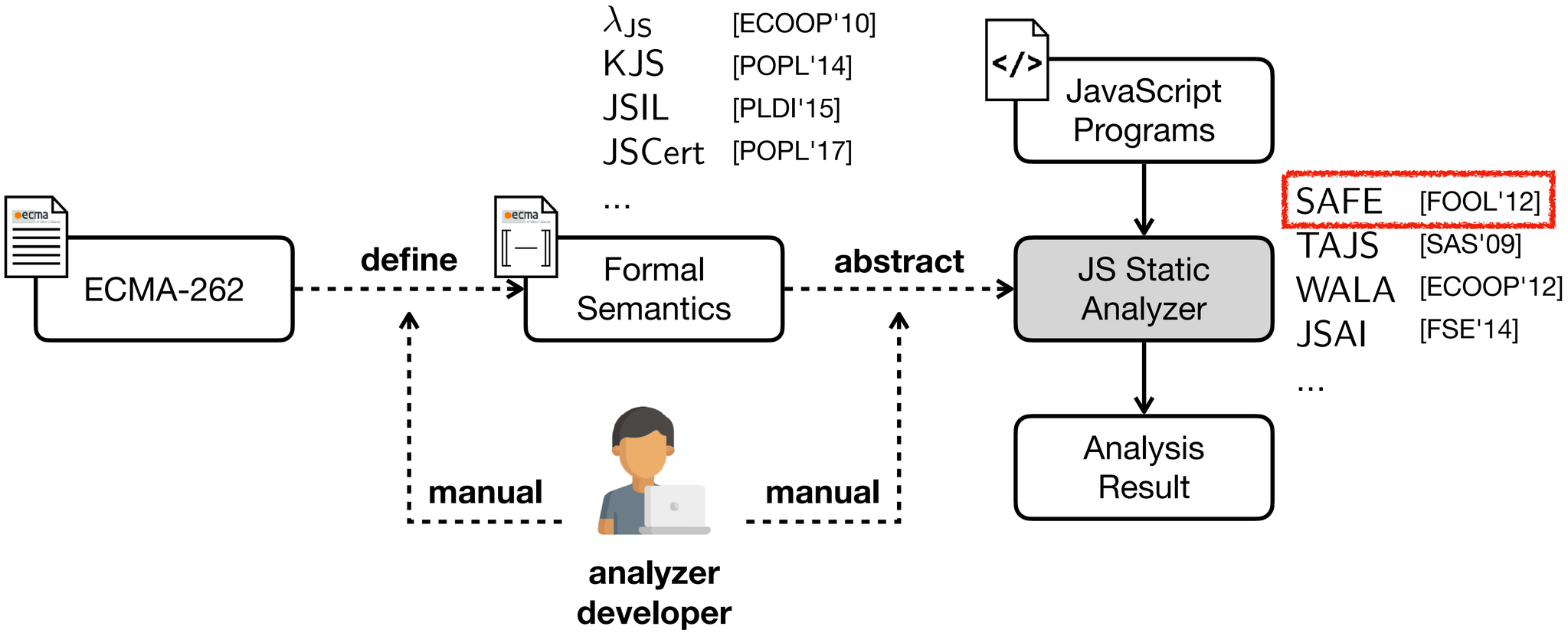
1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be the result of performing ArrayAccumulation for *ElementList* with arguments *array* and 0.
3. ReturnIfAbrupt(*nextIndex*).
4. If *Elision* is present, then
   a. Let *len* be the result of performing ArrayAccumulation for *Elision* with arguments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

The Evaluation algorithm for the third alternative of *ArrayLiteral* in ES12

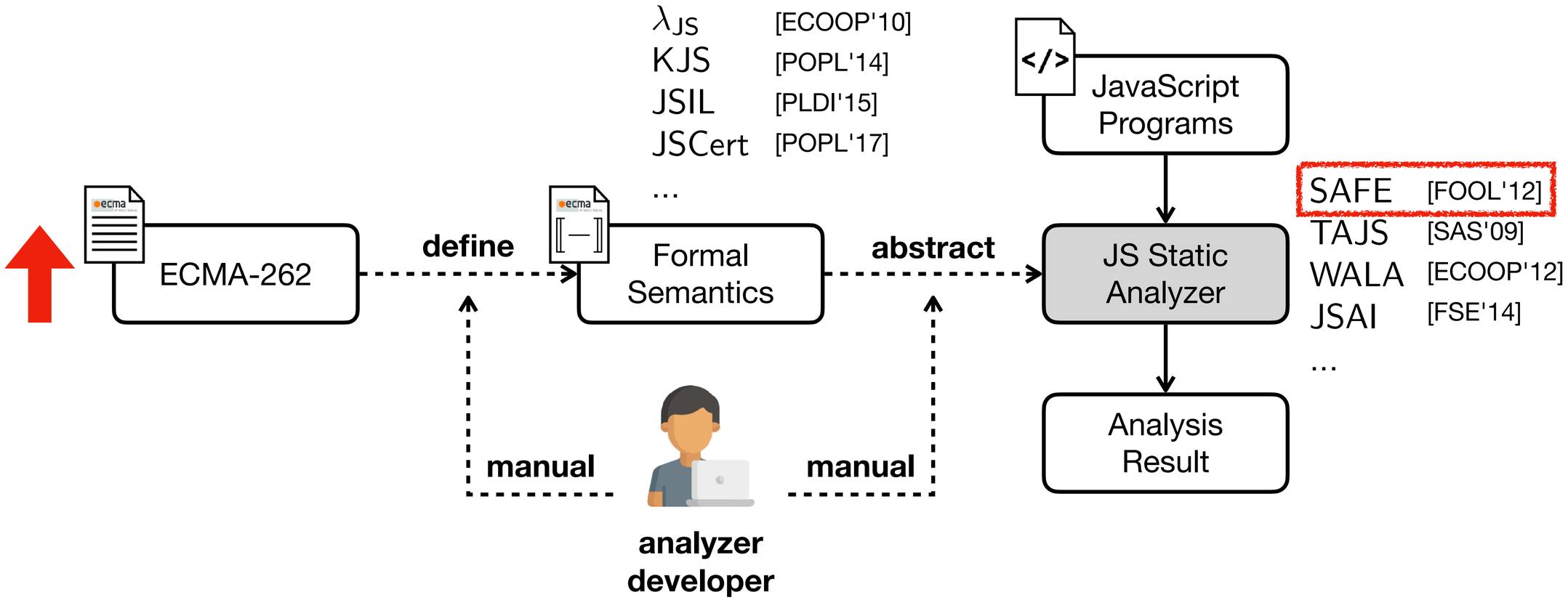
Problem: Hand-Written JavaScript Tools



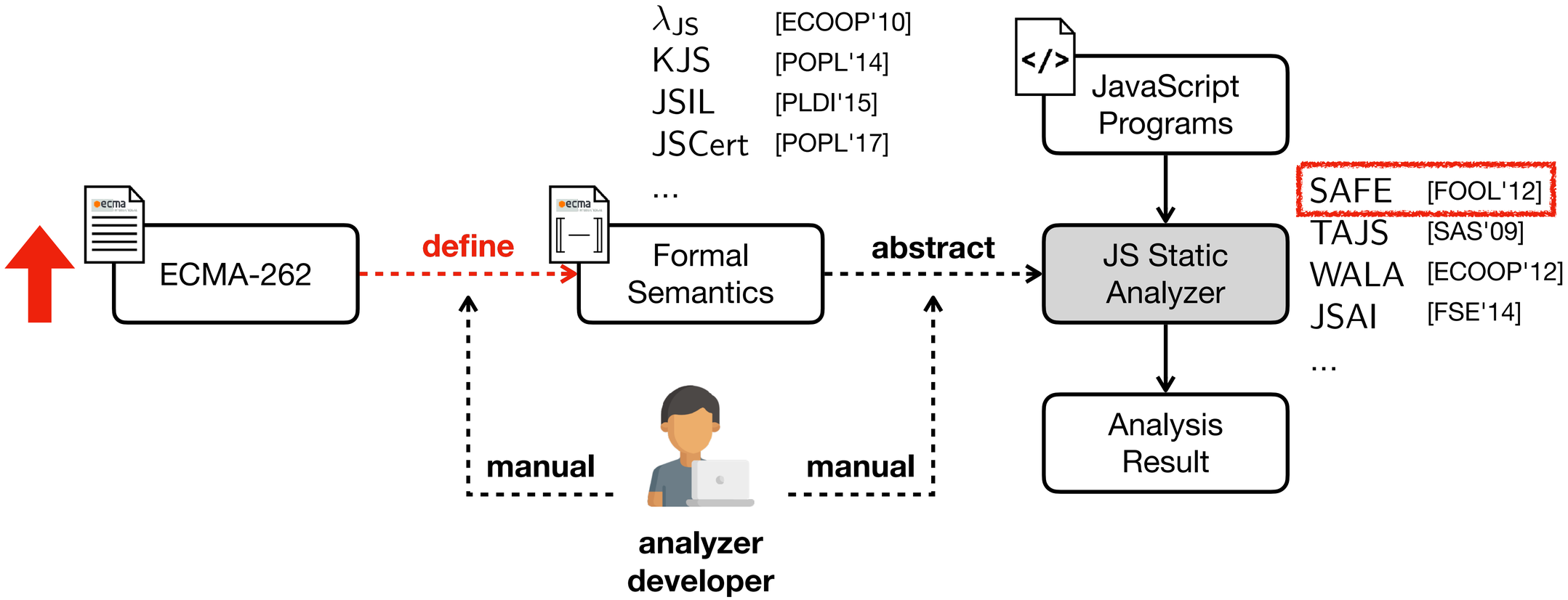
Problem: Hand-Written JavaScript Tools



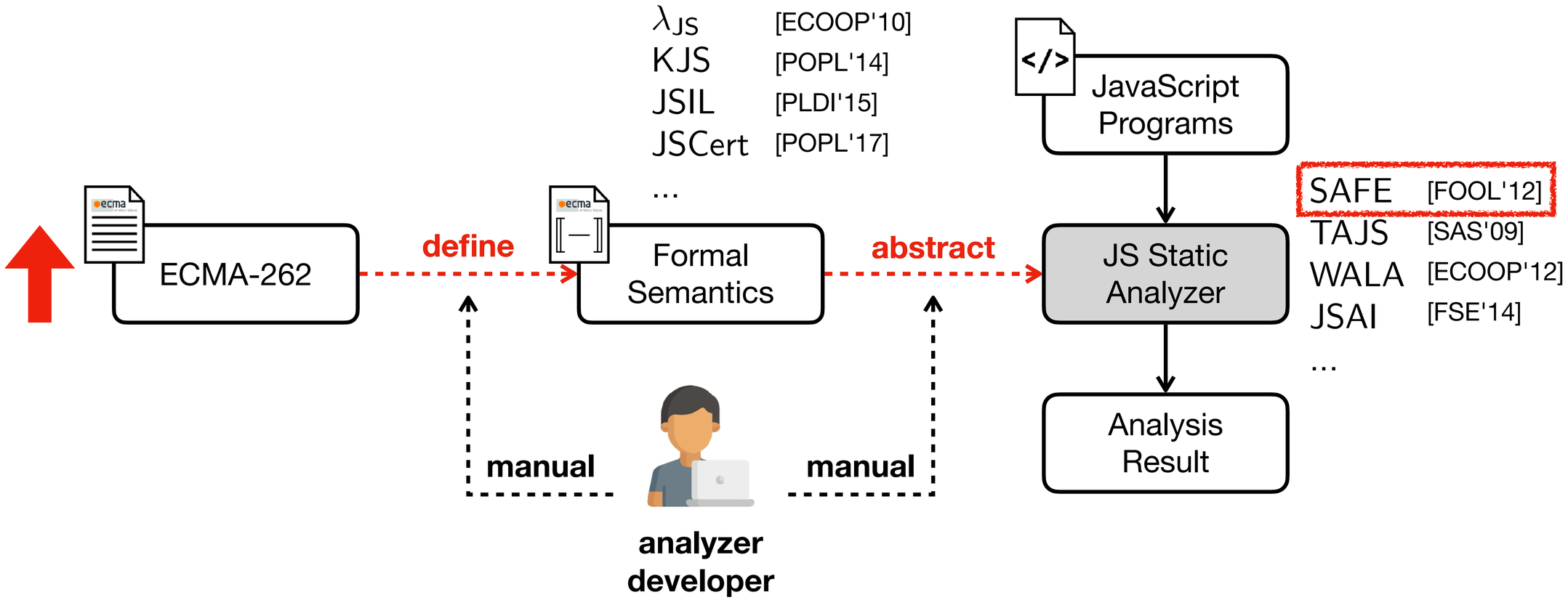
Problem: Hand-Written JavaScript Tools



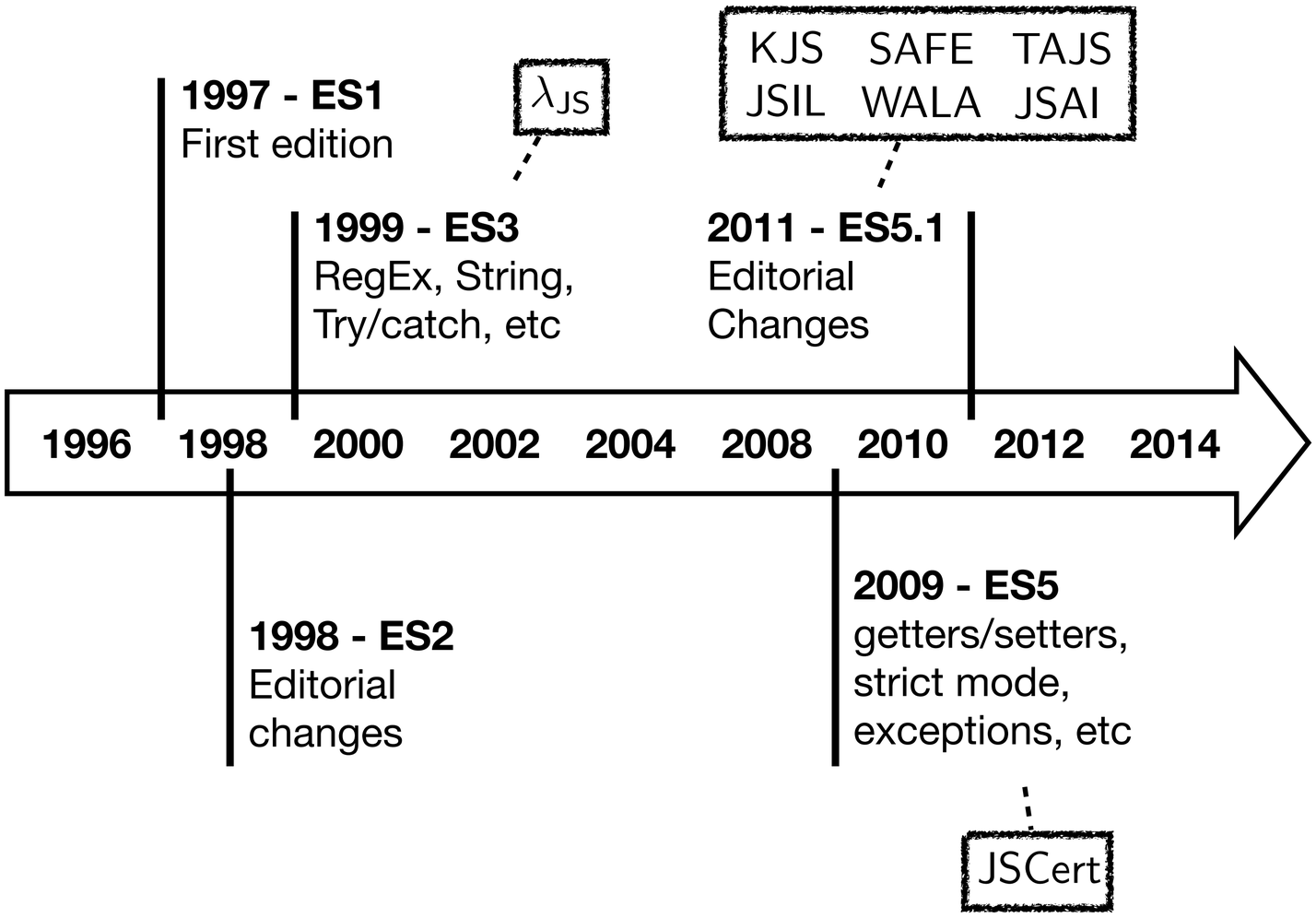
Problem: Hand-Written JavaScript Tools



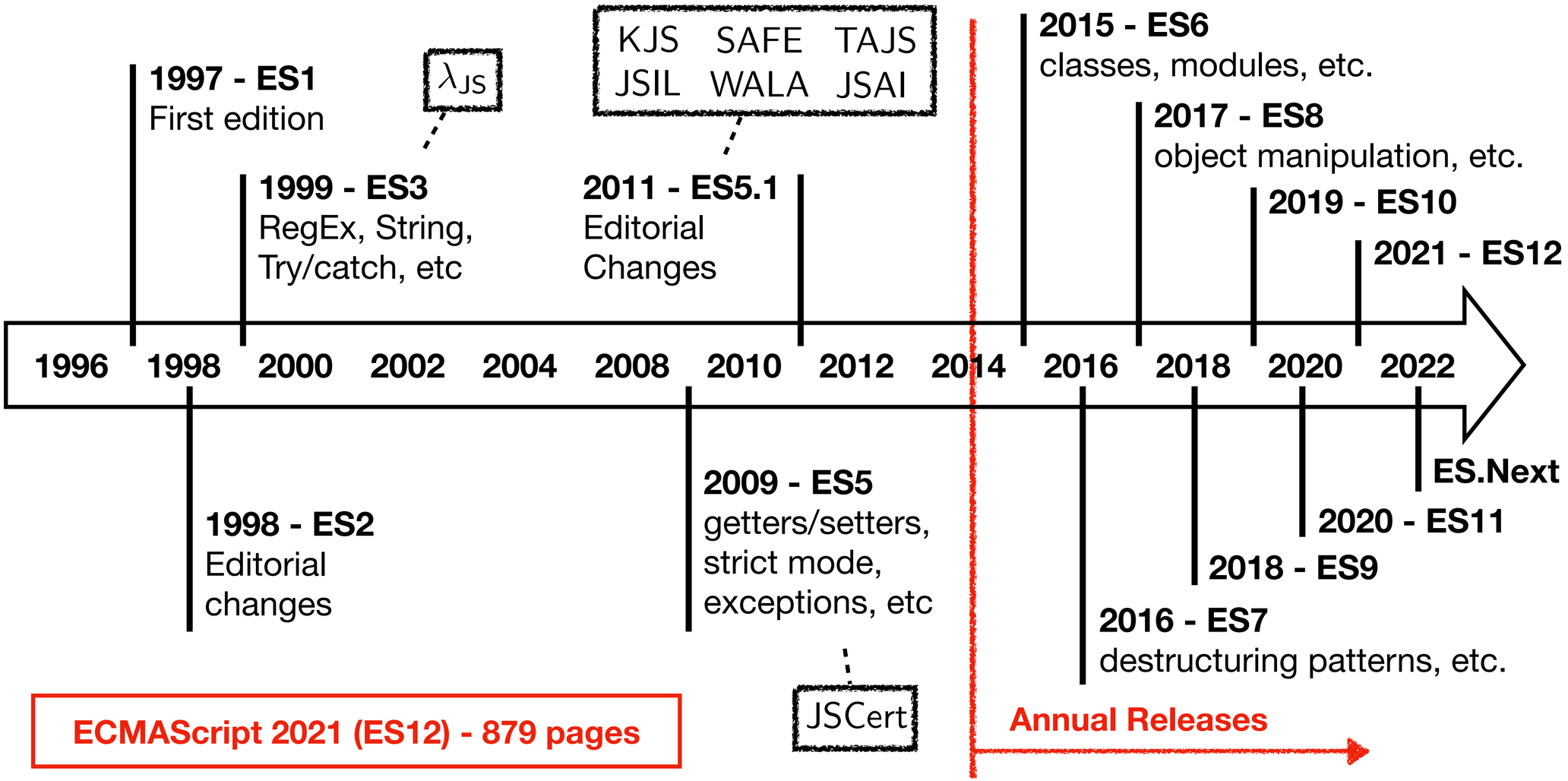
Problem: Hand-Written JavaScript Tools



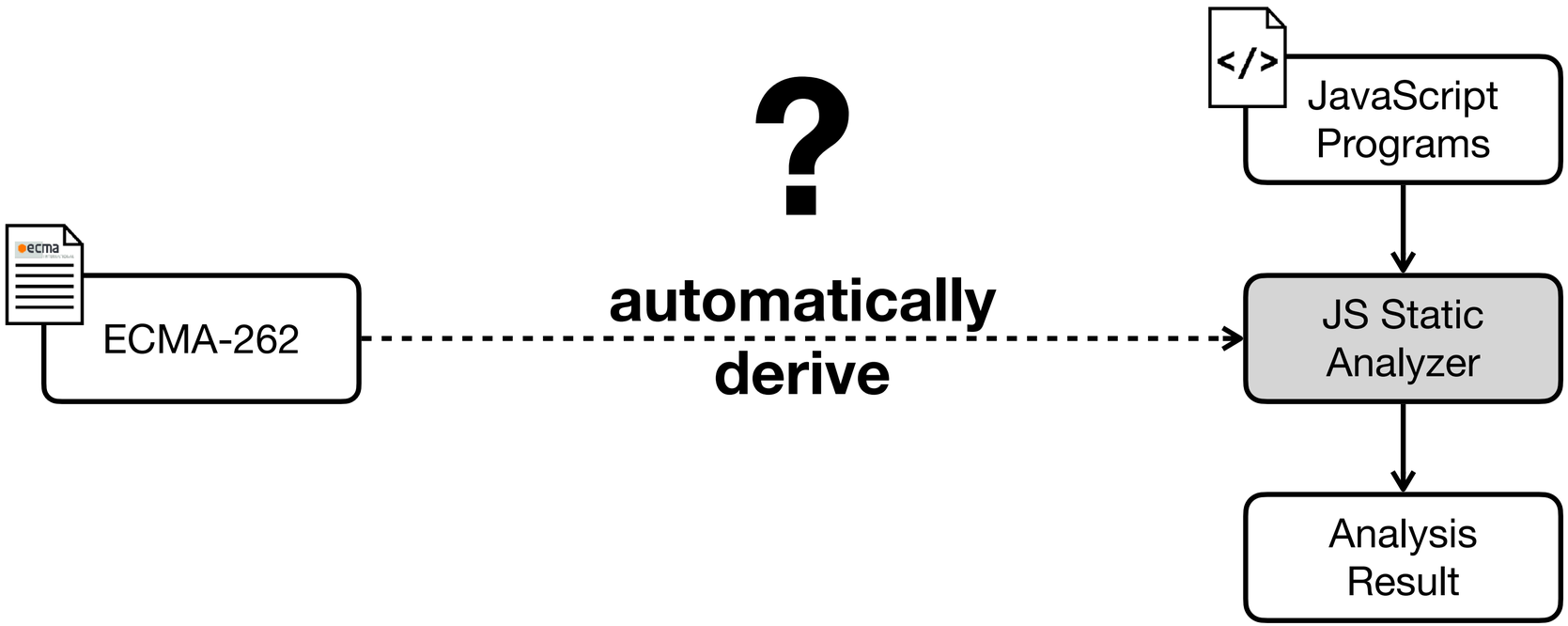
Problem: Fast Evolving JavaScript



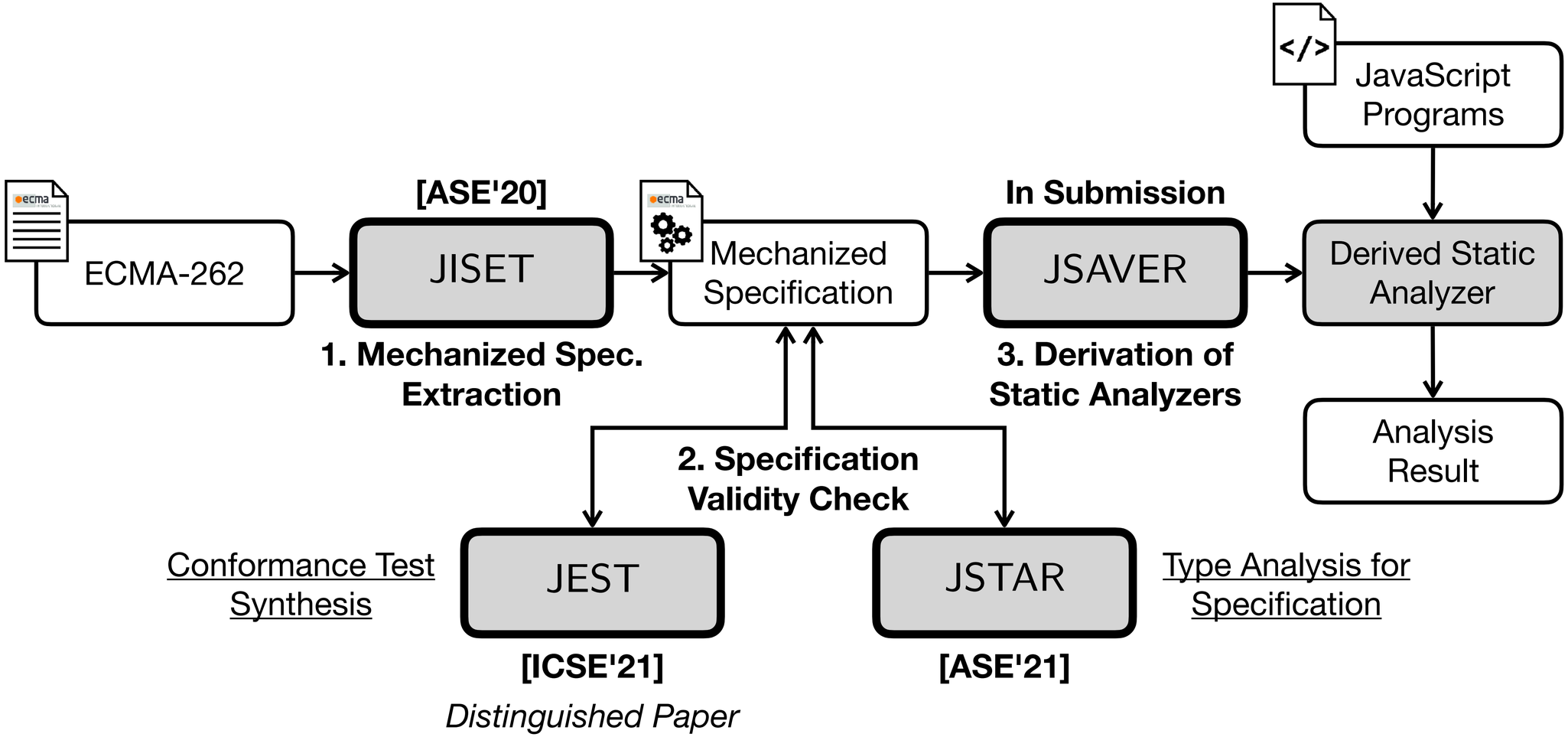
Problem: Fast Evolving JavaScript



Main Idea: Deriving Static Analyzer from Spec.



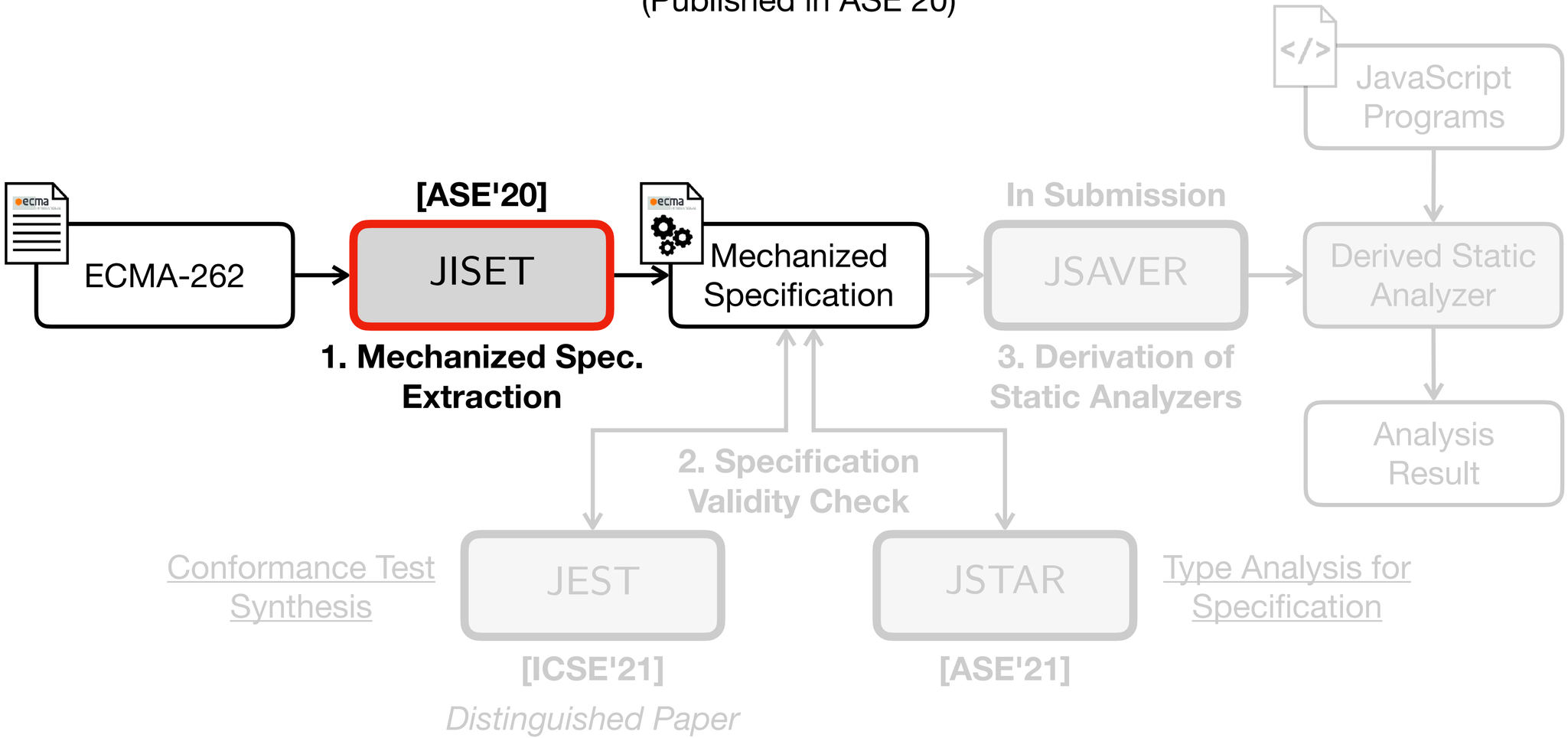
Overall Structure



JISET: JavaScript IR-based Semantics Extraction Toolchain

Jihyeok Park, Jihee Park, Seungmin An, and Sukyoung Ryu

(Published in ASE'20)



Motivation: Writing Style of ECMA-262

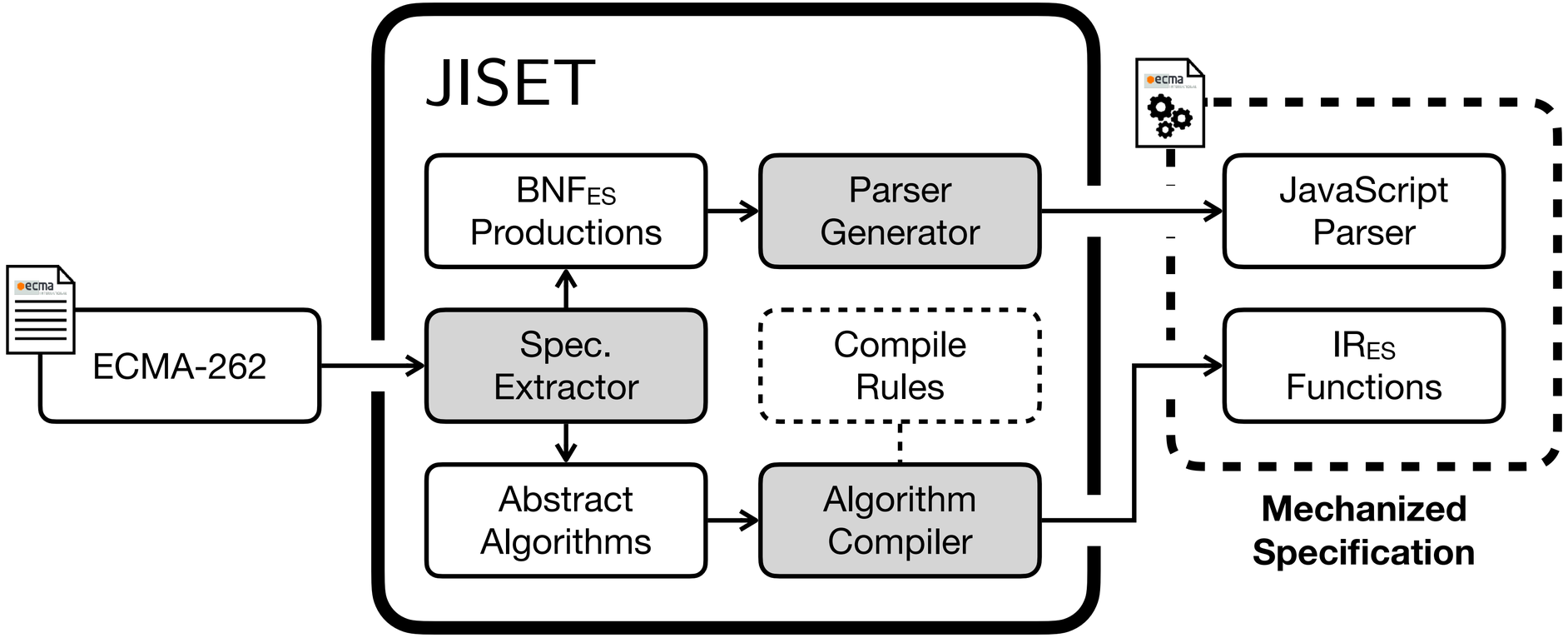
13.2.5.2 Runtime Semantics: Evaluation

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

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

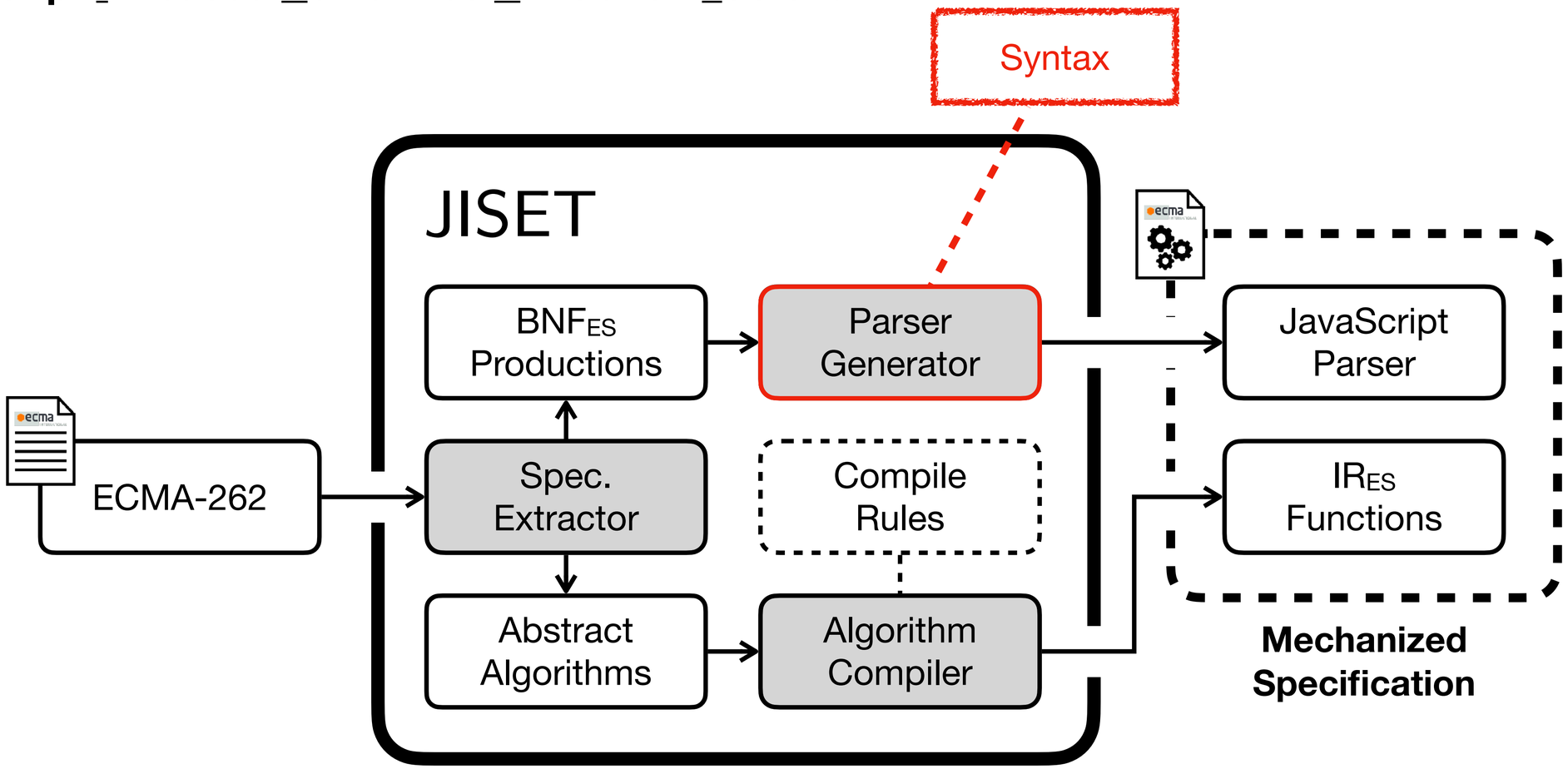
JISET [ASE'20]

JavaScript IR-based Semantics Extraction Toolchain



JISET [ASE'20]

JavaScript IR-based Semantics Extraction Toolchain



JISET - Parser Generator (Syntax)

ArrayLiteral[Yield, Await] :

[*Elision*opt]

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

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

Parsing Expression Grammar (PEG)

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

(POPL'04) Bryan Ford, "Parsing Expression Grammars: A Recognition-based Syntactic Foundation"

JISET - Parser Generator (Syntax)

- **Context-Free Grammar (CFG)**
  - Unordered Choices

$$A ::= B\texttt{;} \mid B \texttt{+} B\texttt{;}$$
$$B ::= \texttt{x} \mid \texttt{xy}$$

$\texttt{xy;}$ ✓

$\texttt{x+x;}$ ✓

- **Parsing Expression Grammar (PEG)**
  - Ordered Choices

$$A ::= B\texttt{;} \;/\; B \texttt{+} B\texttt{;}$$
$$B ::= \texttt{x} \;/\; \boxed{\texttt{xy}}$$

always ignored

$\texttt{xy;}$ ✗

$\texttt{x+x;}$ ✓

- **PEG with Lookahead Parsing**
  - Ordered Choices with ***Lookahead Tokens***

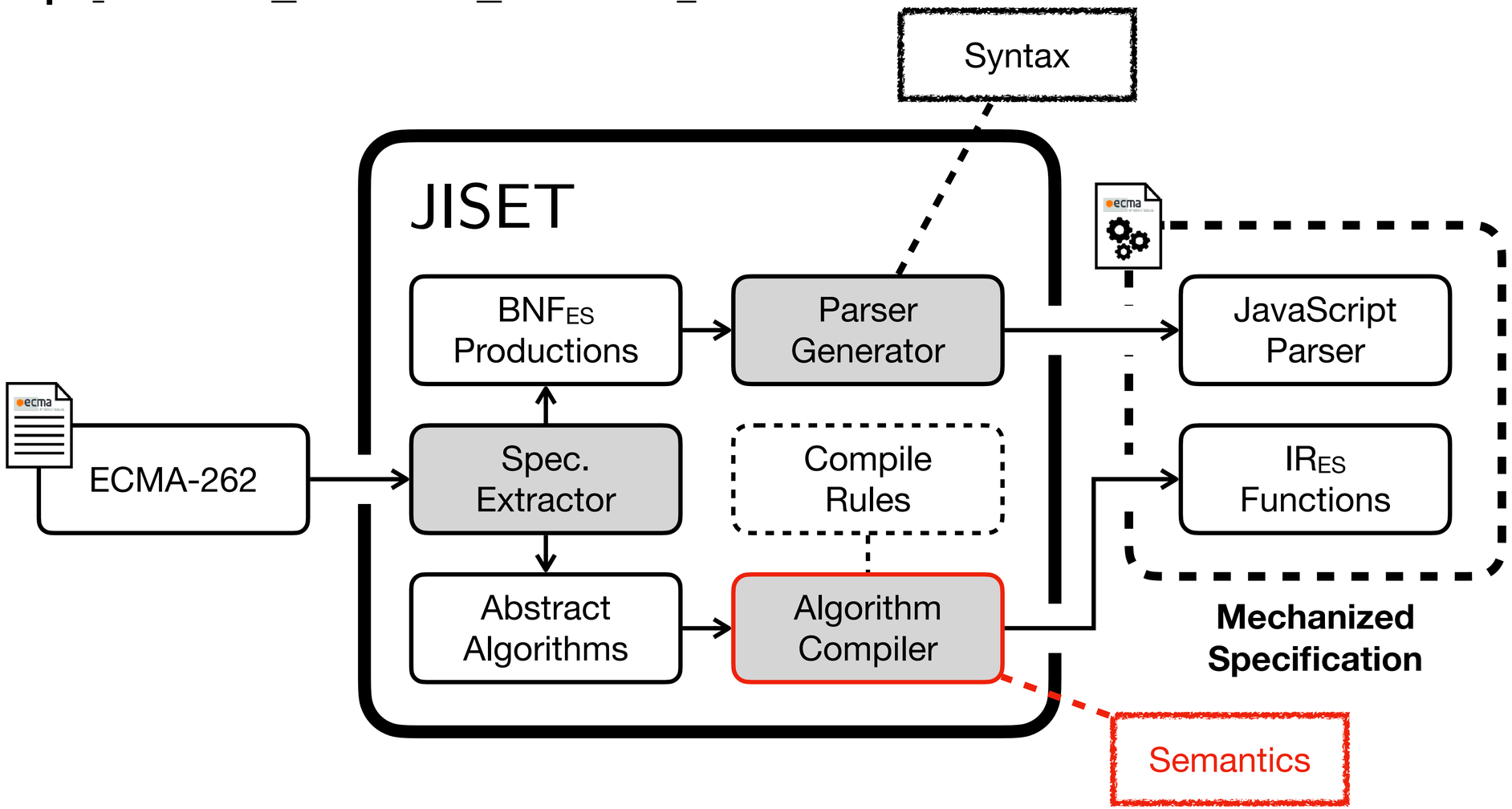
$$A ::= B\texttt{;} \;/\; B \texttt{+} B\texttt{;}$$
$$B ::= \texttt{x} \;/\; \texttt{xy}$$

$\texttt{xy;}$ ✓

$\texttt{x+x;}$ ✓

JISET [ASE'20]

JavaScript IR-based Semantics Extraction Toolchain



JISET - Metalanguage for ECMA-262

- **IR$_{ES}$** - **I**ntermediate **R**epresentation for **E**CMA**S**cript

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

States	$\sigma \in \mathbb{S}$	$= \mathcal{L} \times \mathbb{E} \times \mathbb{C}^* \times \mathbb{H}$
Environments	$\rho \in \mathbb{E}$	$= \mathcal{X} \xrightarrow{\text{fin}} \mathbb{V}$
Calling Contexts	$c \in \mathbb{C}$	$= \mathcal{L} \times \mathbb{E}$
Heaps	$h \in \mathbb{H}$	$= \mathbb{A} \xrightarrow{\text{fin}} \mathcal{L} \times \mathbb{M} \times \mathbb{M}_{\mathrm{js}}$
Internal Field Maps	$m \in \mathbb{M}$	$= \mathbb{V}_{\mathrm{str}} \xrightarrow{\text{fin}} \mathbb{V}$
External Field Maps	$m_{\mathrm{js}} \in \mathbb{M}_{\mathrm{js}}$	$= \mathbb{V}_{\mathrm{str}} \xrightarrow{\text{fin}} \mathbb{V}$
Values	$v \in \mathbb{V}$	$= \mathbb{A} \uplus \mathbb{V}^{\mathrm{p}} \uplus \mathbb{T} \uplus \mathcal{F}$
Primitive Values	$v^{\mathrm{p}} \in \mathbb{V}^{\mathrm{p}}$	$= \mathbb{V}_{\mathrm{bool}} \uplus \mathbb{V}_{\mathrm{int}} \uplus \mathbb{V}_{\mathrm{str}} \uplus \cdots$
JS ASTs	$t \in \mathbb{T}$	

JISET - Metalanguage for ECMA-262

- **IR$_{ES}$** - **I**ntermediate **R**epresentation for **E**CMA**S**cript

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

States	$\sigma \in \mathbb{S}$	$= \mathcal{L} \times \mathbb{E} \times \mathbb{C}^* \times \mathbb{H}$
Environments	$\rho \in \mathbb{E}$	$= \mathcal{X} \xrightarrow{\text{fin}} \mathbb{V}$
Calling Contexts	$c \in \mathbb{C}$	$= \mathcal{L} \times \mathbb{E}$
Heaps	$h \in \mathbb{H}$	$= \mathbb{A} \xrightarrow{\text{fin}} \mathcal{L} \times \mathbb{M} \times \mathbb{M}_{\mathrm{js}}$
Internal Field Maps	$m \in \mathbb{M}$	$= \mathbb{V}_{\mathrm{str}} \xrightarrow{\text{fin}} \mathbb{V}$
External Field Maps	$m_{\mathrm{js}} \in \mathbb{M}_{\mathrm{js}}$	$= \mathbb{V}_{\mathrm{str}} \xrightarrow{\text{fin}} \mathbb{V}$
Values	$v \in \mathbb{V}$	$= \mathbb{A} \uplus \mathbb{V}^{\mathrm{p}} \uplus \boxed{\mathbb{T}} \uplus \mathcal{F}$
Primitive Values	$v^{\mathrm{p}} \in \mathbb{V}^{\mathrm{p}}$	$= \mathbb{V}_{\mathrm{bool}} \uplus \mathbb{V}_{\mathrm{int}} \uplus \mathbb{V}_{\mathrm{str}} \uplus \cdots$
JS ASTs	$t \in \mathbb{T}$	

JISET - Algorithm Compiler (Semantics)

13.2.5.2 Runtime Semantics: Evaluation

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

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

118 Compile Rules for Steps in Abstract Algorithms

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

JISET - Algorithm Compiler (Semantics)

13.2.5.2 Runtime Semantics: Evaluation

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

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

118 Compile Rules for Steps in Abstract Algorithms

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

Parsing rules | Conversion Rules

```
S = // statements
  Let ~ V ~ be ~ E ~ . ^^ ILet

E = // expressions
  ! E                  ^^ EAbruptCheck |
  str ~ ( ~ E ~ )      ^^ ECall        |
  num                  ^^ _.toDouble
```

Simplified compile rules

Let *array* be ! ArrayCreate (0) .

Parsing rules | Conversion Rules

```
S = // statements
  Let ~ V ~ be ~ E ~ . ^^ ILet

E = // expressions
  ! E                  ^^ EAbruptCheck |
  str ~ ( ~ E ~ )      ^^ ECall        |
  num                  ^^ _.toDouble
```

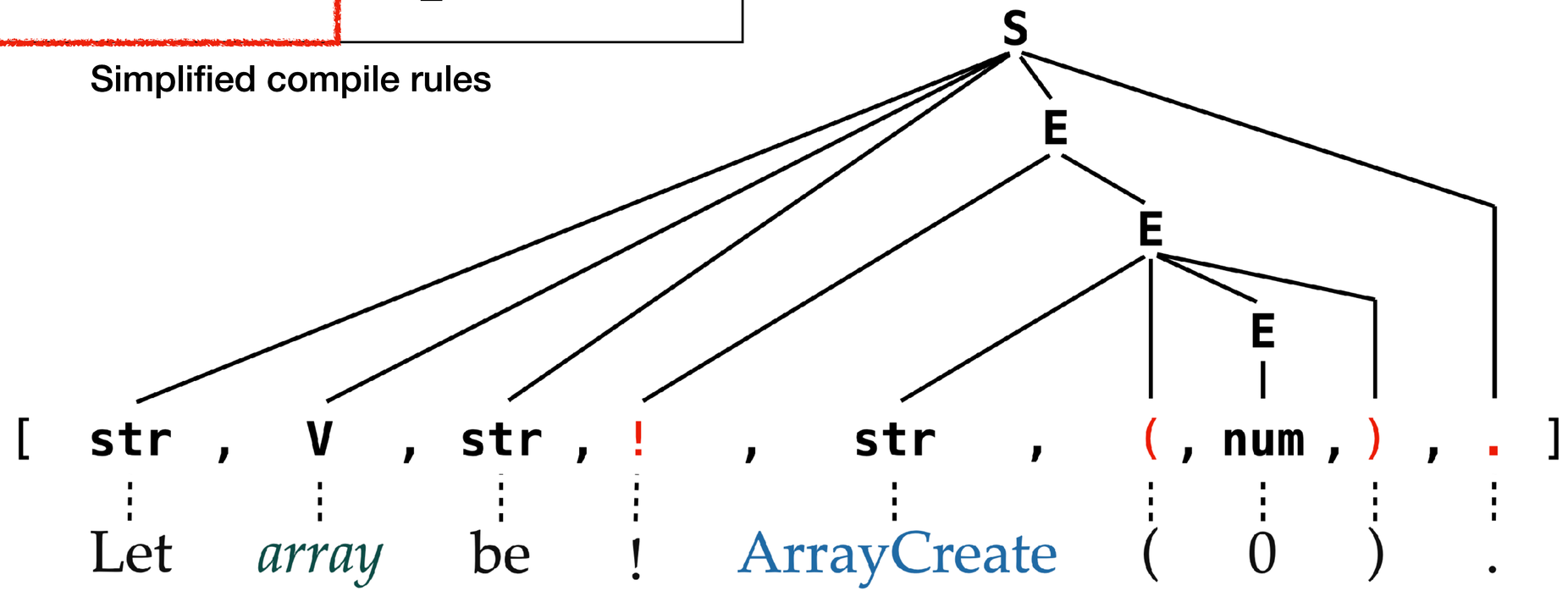
Simplified compile rules

Parsing rules | Conversion Rules

```
S = // statements
  Let ~ V ~ be ~ E ~ .   ^^ ILet

E = // expressions
  ! E                    ^^ EAbruptCheck |
  str ~ ( ~ E ~ )        ^^ ECall        |
  num                    ^^ _.toDouble
```

Simplified compile rules

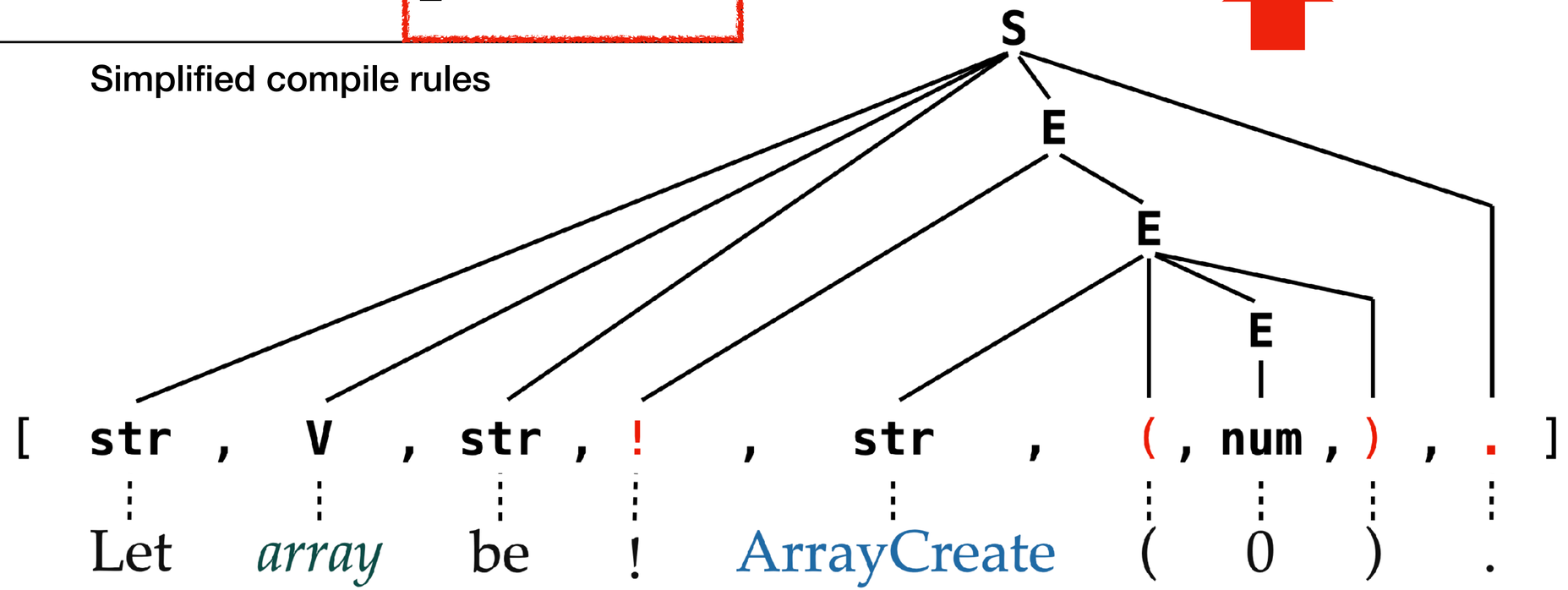


Parsing rules	Conversion Rules
`S = // statements`	
`  Let ~ V ~ be ~ E ~ .`	`^^ ILet`
`E = // expressions`	
`  ! E`	`^^ EAbruptCheck |`
`  str ~ ( ~ E ~ )`	`^^ ECall |`
`  num`	`^^ _.toDouble`

Simplified compile rules

```
let array = ! (ArrayCreate 0)
```

```
ILet(array, EAbruptCheck(
  ECall("ArrayCreate", 0)))
```



JISET - Evaluation

≈ 96% Compiled

auto manual

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

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

JISET - Evaluation

≈ 96% Compiled

auto manual

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

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

Complete Missing Parts

JISET - Evaluation

≈ 96% Compiled

Passed All Tests

auto manual

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

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

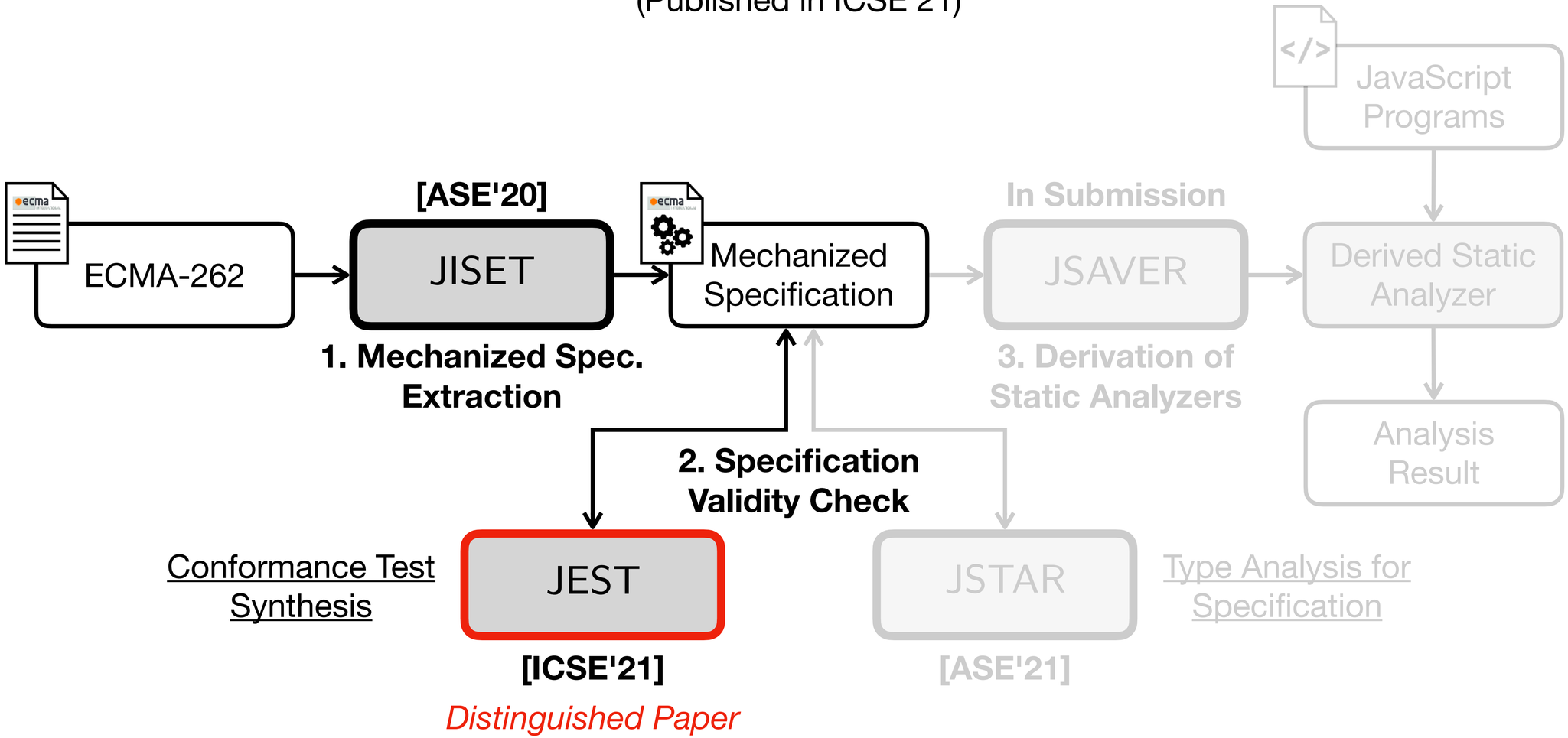
Complete Missing Parts

- **Test262**
  (Official Conformance Tests)
  - 18,556 applicable tests
- **Parsing tests**
  - Passed all 18,556 tests
- **Evaluation Tests**
  - Passed all 18,556 tests

JEST: N+1-version Differential Testing of Both JavaScript Engines

Jihyeok Park, Seungmin An, Dongjun Youn, Gyeongwon Kim, and Sukyoung Ryu

(Published in ICSE'21)



JEST - Conformance with Engines

ECMA-262

?

Conform

GraalVM

QuickJS

moddable

JavaScript
Engines

JEST - N+1-version Differential Testing

ECMA-262

Synthesize

Test

test

test

test

test

GraalVM

QuickJS

moddable

JavaScript Engines

JEST - N+1-version Differential Testing

ECMA-262

Synthesize

Test

test

test

test

test

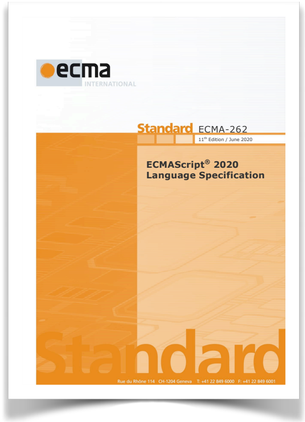
GraalVM

QuickJS

moddable

JavaScript Engines

JEST - N+1-version Differential Testing



ECMA-262

Synthesize → Test → test, test, test, test

GraalVM

QuickJS

moddable

JavaScript Engines

An engine bug in

JEST - N+1-version Differential Testing

ECMA-262

Synthesize

Test

test

test

test

test

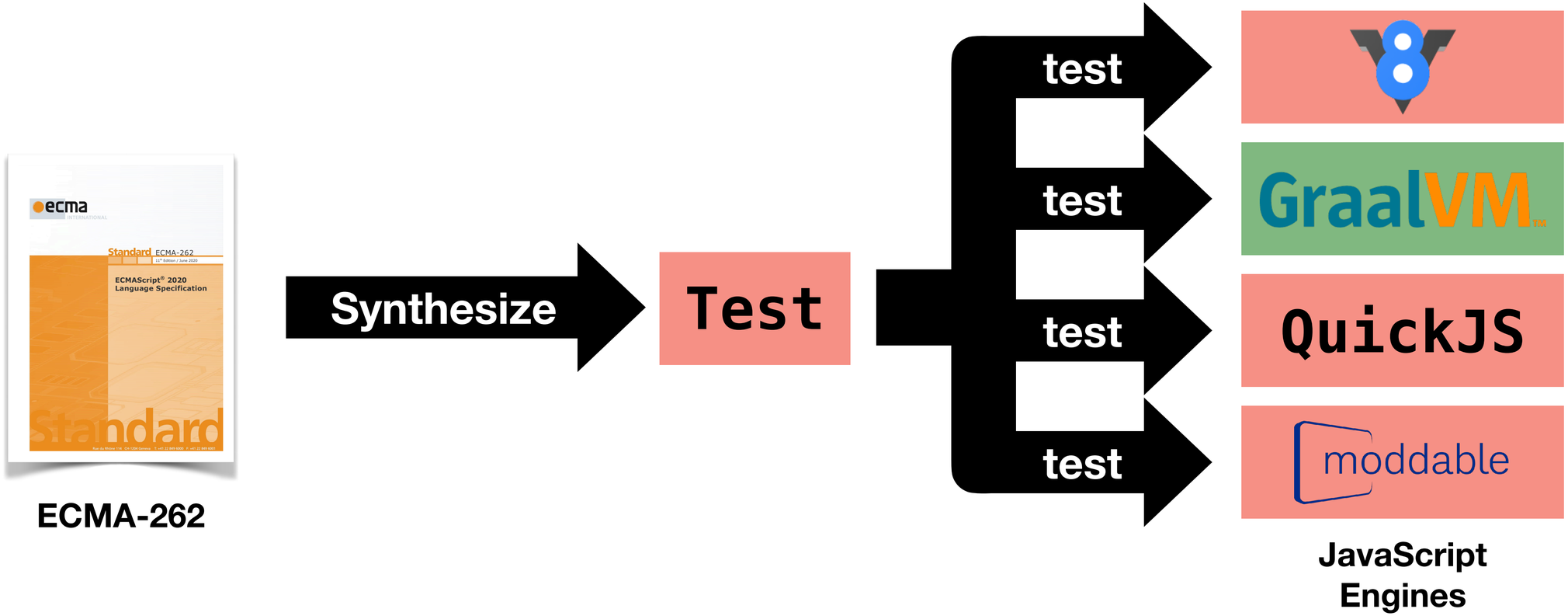
GraalVM

QuickJS

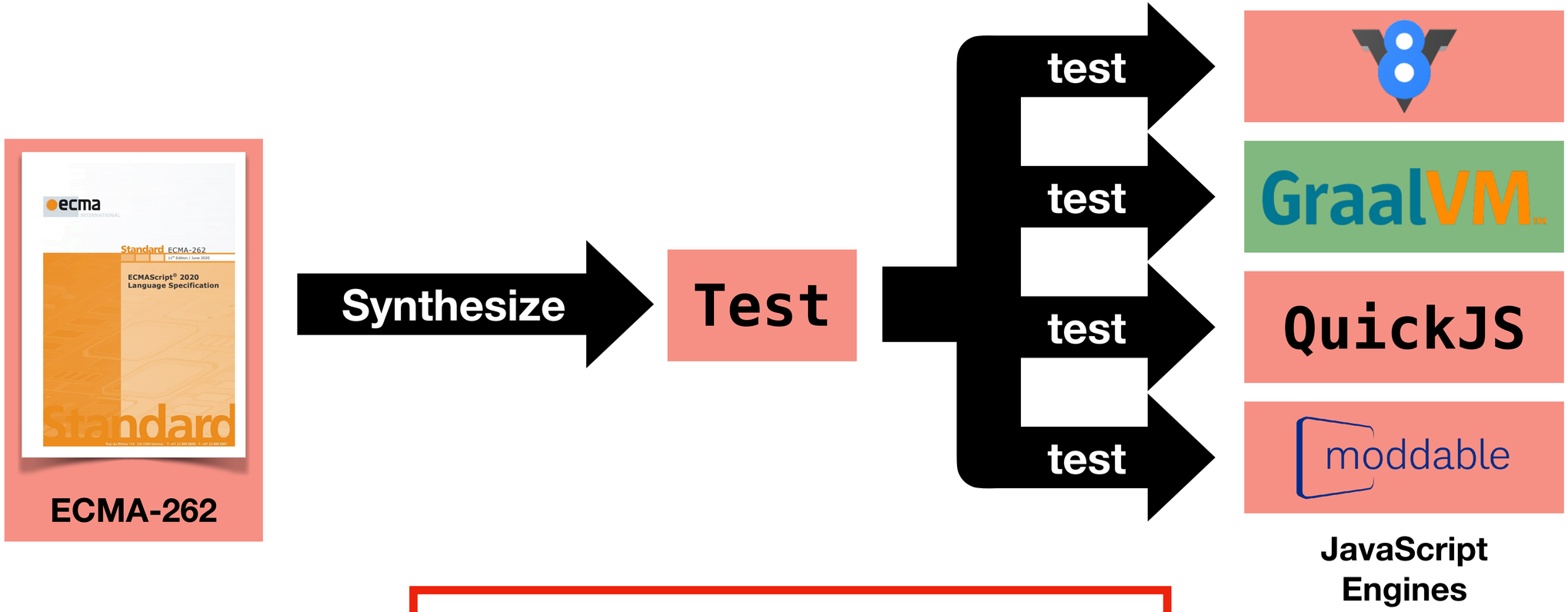
moddable

JavaScript Engines

JEST - N+1-version Differential Testing

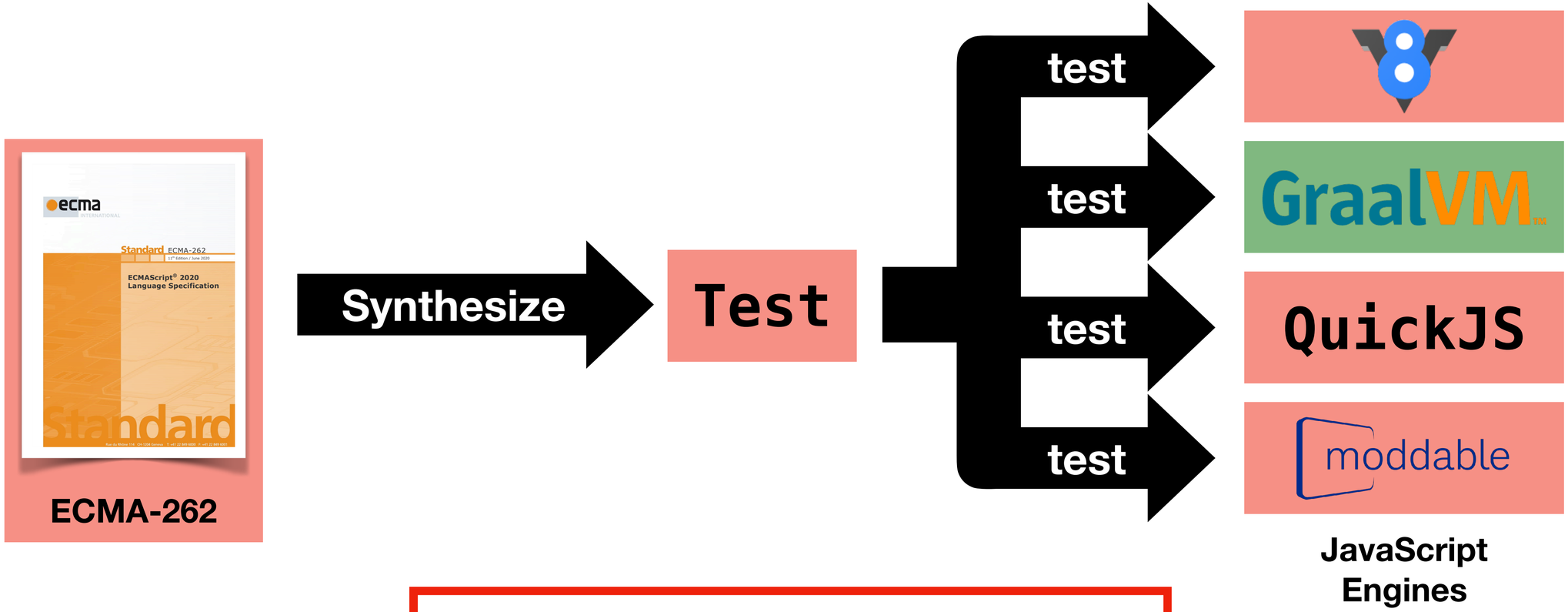


JEST - N+1-version Differential Testing



A <u>specification</u> bug in ECMA-262

JEST - N+1-version Differential Testing

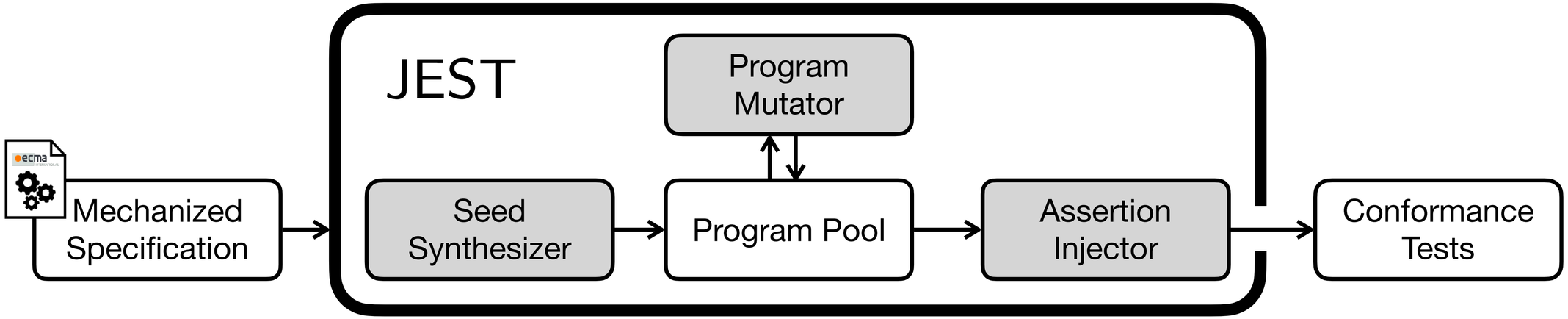


A <u>specification</u> bug in ECMA-262

An <u>engine</u> bug in GraalVM

JEST [ICSE'21]

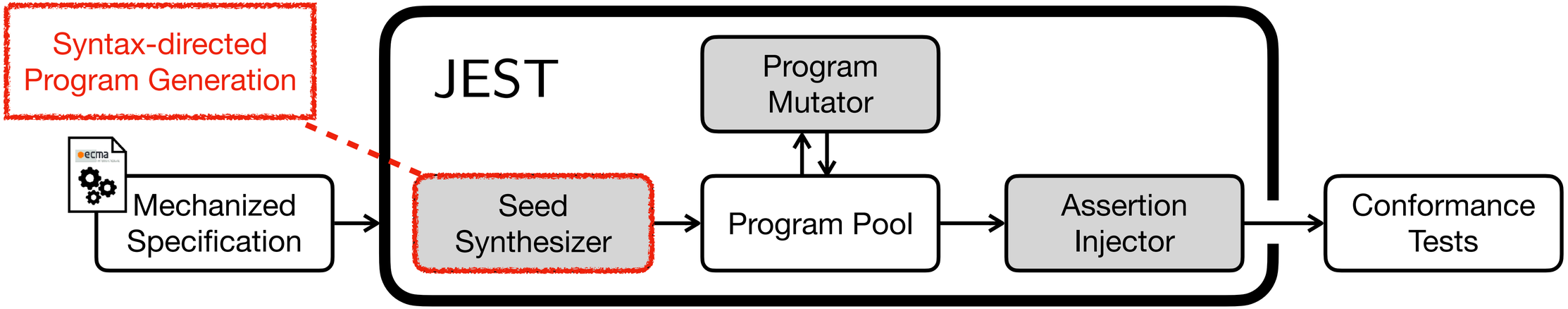
JavaScript Engines and Specification Tester



Program Pool

JEST [ICSE'21]

JavaScript Engines and Specification Tester

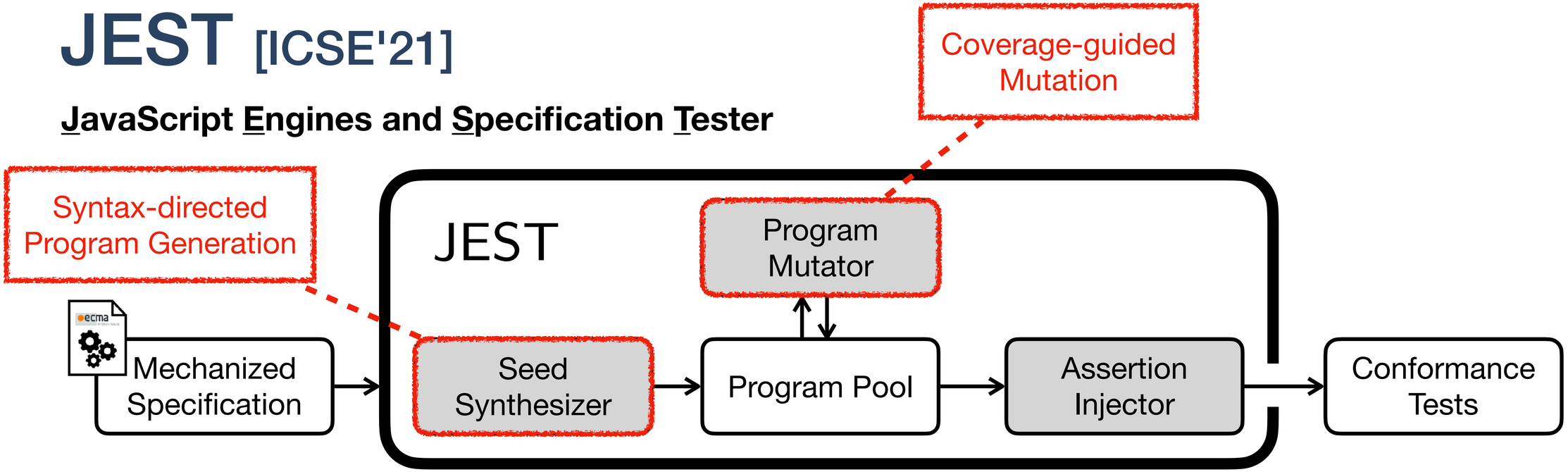


Program Pool

```
let x = 42;
```

JEST [ICSE'21]

JavaScript Engines and Specification Tester



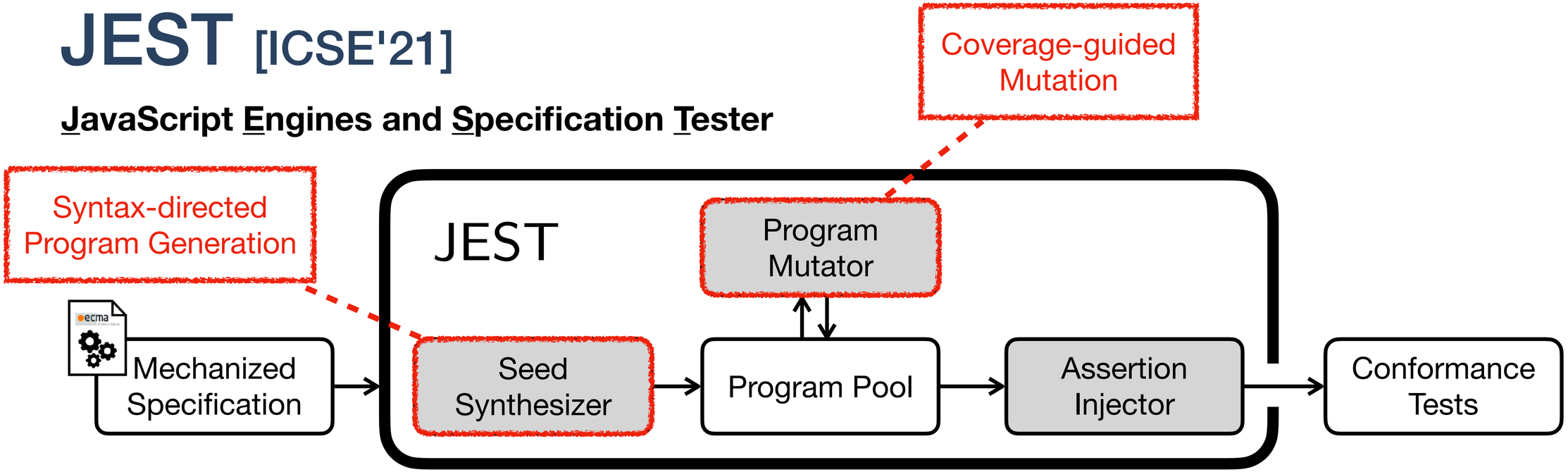
Program Pool

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

```
let x = 42;
```

JEST [ICSE'21]

JavaScript Engines and Specification Tester

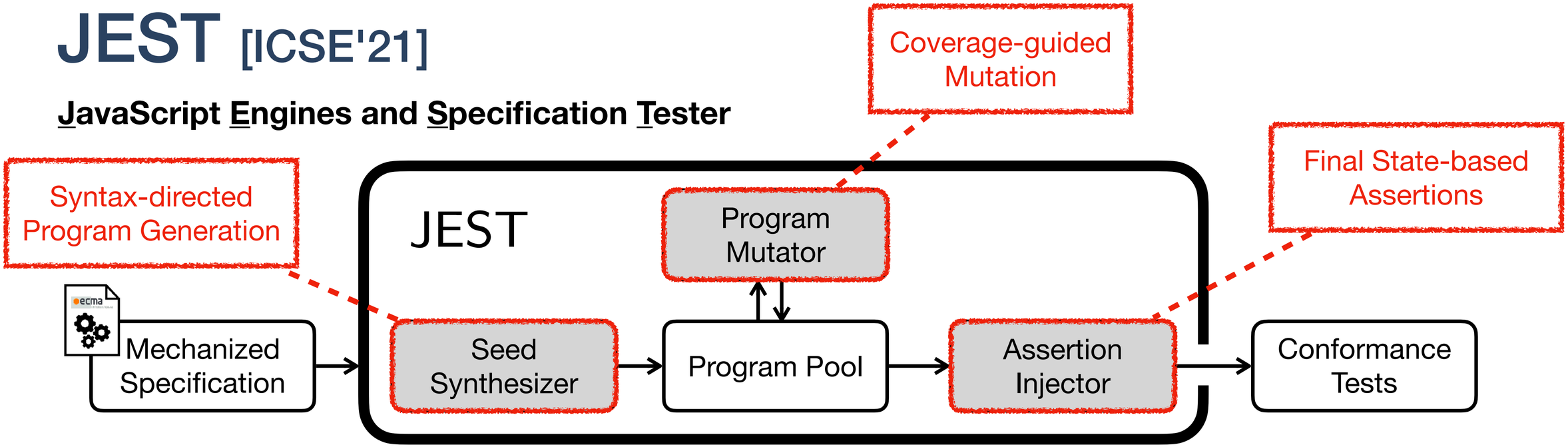


Program Pool

```
let x = 1 + 2;
let x = 42;
let x = ![];
```

JEST [ICSE'21]

JavaScript Engines and Specification Tester



Program Pool

```
let x = 1 + 2;
assert(x == 3);
```

```
let x = 42;
assert(x == 42);
```

```
let x = ![];
assert(x == false);
```

JEST - Assertion Injector (7 Kinds)

1. **Exceptions (`Exc`)**

```
+ // Throw
  let x = 42;
  function x() {};
```

2. **Aborts (`Abort`)**

```
+ // Abort
  var x = 42; x++;
```

3. **Variable Values (`Var`)**

```
  var x = 1 + 2;
+ $assert.sameValue(x, 3);
```

4. **Object Values (`Obj`)**

```
  var x = {}, y = {}, z = { p: x, q: y };
+ $assert.sameValue(z.p, x);
+ $assert.sameValue(z.q, y);
```

JEST - Assertion Injector (7 Kinds)

5. **Object Properties (`Desc`)**

```
  var x = { p: 42 };
+ $verifyProperty(x, "p", {
+   value: 42.0, writable: true,
+   enumerable: true, configurable: true
+ });
```

- -

6. **Property Keys (`Key`)**

```
  var x = {[Symbol.match]: 0, p: 0, 3: 0, q: 0, 1: 0}
+ $assert.compareArray(
+   Reflect.ownKeys(x),
+   ["1", "3", "p", "q", Symbol.match]
+ );
```

- -

7. **Internal Methods and Slots (`In`)**

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

JEST - Evaluation

- **JEST successfully synthesized 1,700 conformance tests from ES11**

44 Bugs in Engines

TABLE II: The number of engine bugs detected by JEST

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

27 Bugs in Spec.

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

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

JEST - Example in GraalVM


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

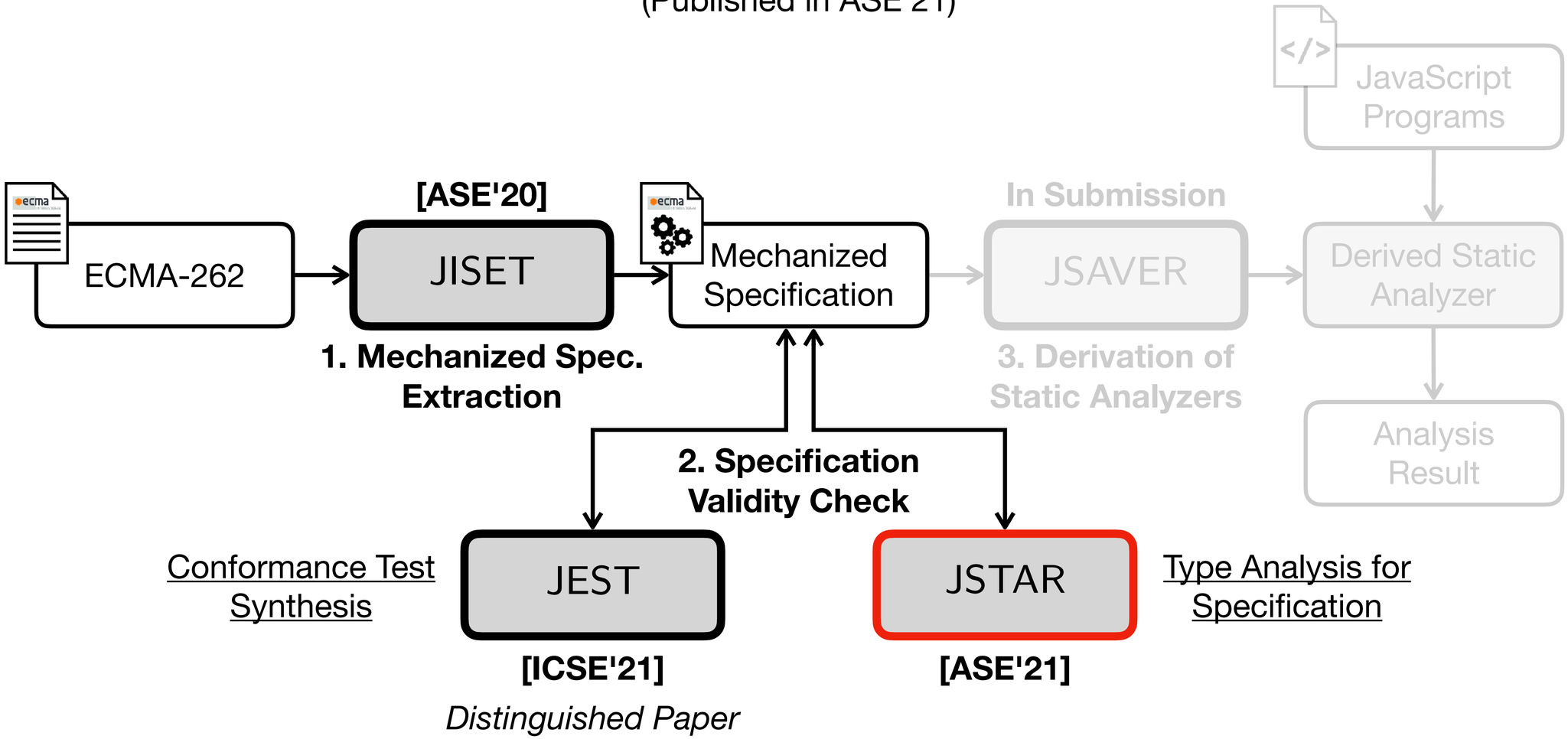
"Right now, we are running Test262 and the V8 and Nashorn unit test suites in our CI for every change, it might make sense to add your suite as well."

- A Developer of GraalVM

JSTAR: JavaScript Specification Type Analyzer using Refinement

Jihyeok Park, Seungmin An, Wonho Shin, Yusung Sim, and Sukyoung Ryu

(Published in ASE'21)



JSTAR - Types in Specification

20.3.2.28 Math.round (*x*)

1. Let *n* be ? ToNumber(*x*).
2. If *n* is an integral Number, return *n*.
3. If *x* < 0.5 and *x* > 0, return **+0**.
4. If *x* < 0 and *x* ≥ -0.5, return **-0**.

...

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

JSTAR - Types in Specification

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

1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

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

JSTAR - Types in Specification

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

1. Let *n* be ? ToNumber(*x*). `ToNumber(x): (Number v Exception)`
2. If *n* is an integral Number, return *n*.
3. If *x* < 0.5 and *x* > 0, return **+0**.
4. If *x* < 0 and *x* ≥ -0.5, return **-0**.

...

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

JSTAR - Types in Specification

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

1. Let n be ? ToNumber(x). `ToNumber(x): (Number v Exception)`
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

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

JSTAR - Types in Specification

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

1. Let n be ? ToNumber(x). `ToNumber(x): (Number v Exception) ∧ n: (Number)`
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

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

JSTAR - Types in Specification

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

1. Let n be ? ToNumber(x). `ToNumber(x): (Number v Exception) ∧ n: (Number)`
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

Type Mismatch for numeric operator `>`

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

JSTAR - Types in Specification

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

1. Let n be ? ToNumber(x). `ToNumber(x): (Number v Exception) ∧ n: (Number)`
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

Type Mismatch for numeric operator `>`

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

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

JSTAR - Types in Specification

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

1. Let *n* be ? ToNumber(*x*). `ToNumber(x): (Number v Exception) ∧ n: (Number)`
2. If *n* is an integral Number, return *n*.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

...

Type Mismatch for numeric operator `>`

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

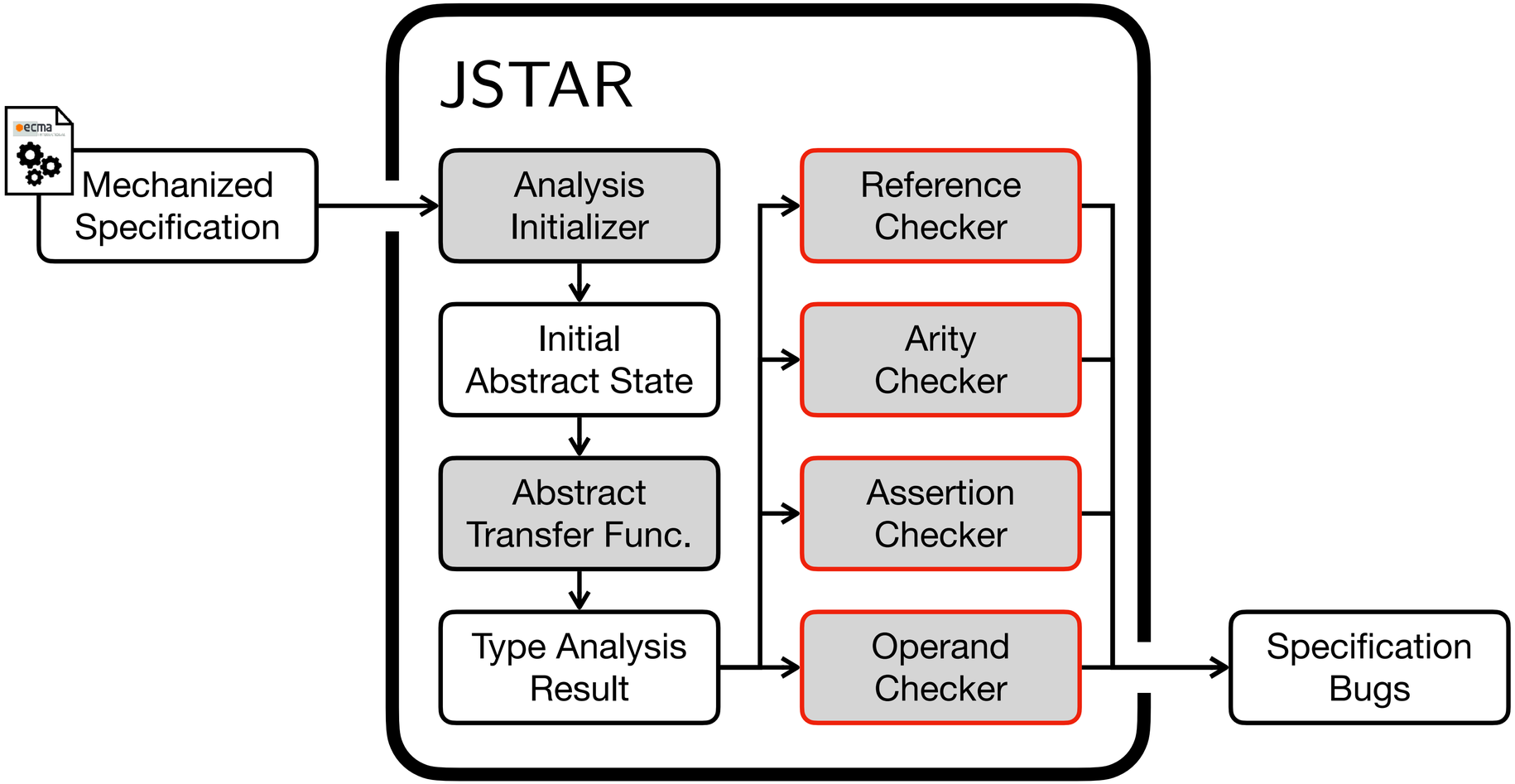
3. If $n < 0.5$ and $n > 0$, return **+0**.
4. If $n < 0$ and $n \geq -0.5$, return **-0**.

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

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

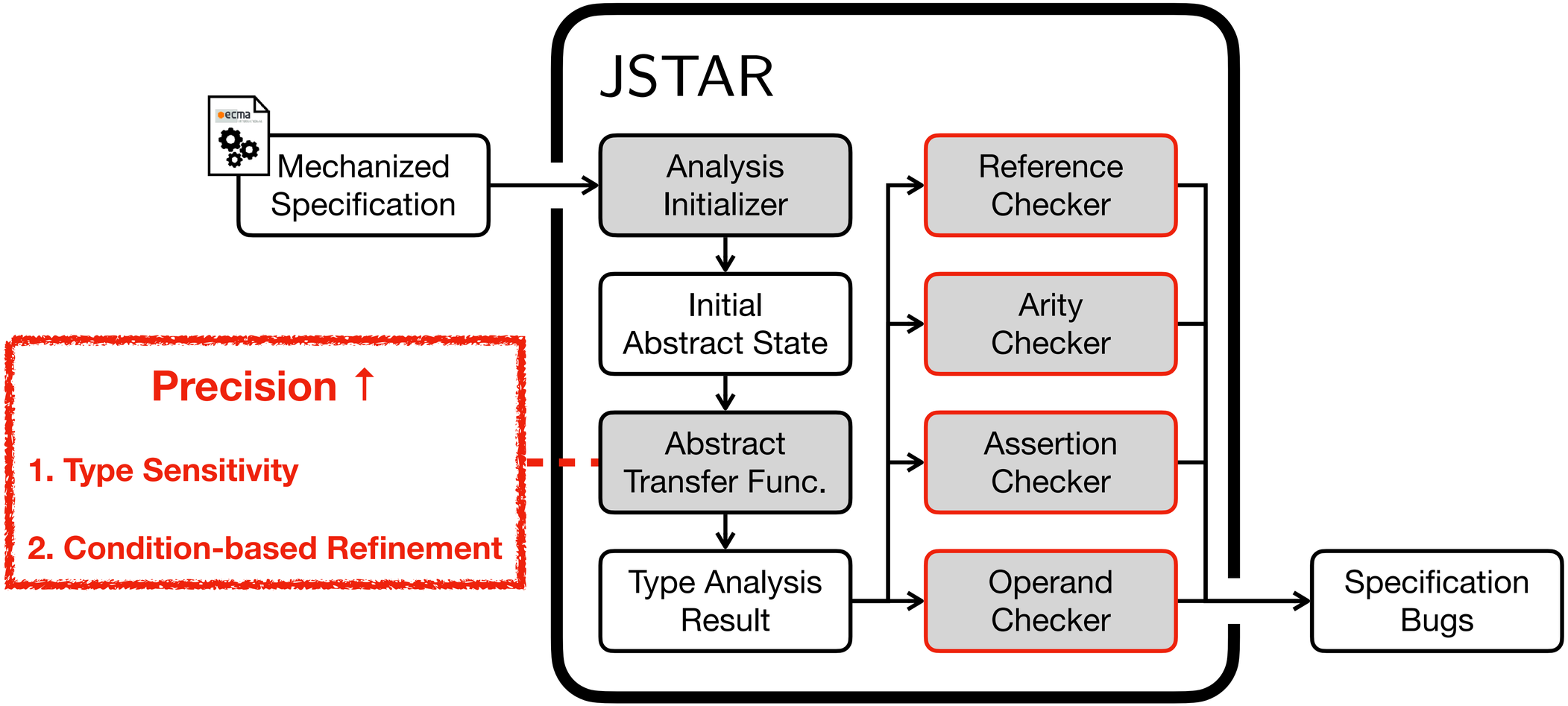
JSTAR [ASE'21]

JavaScript Specification Type Analyzer using Refinement

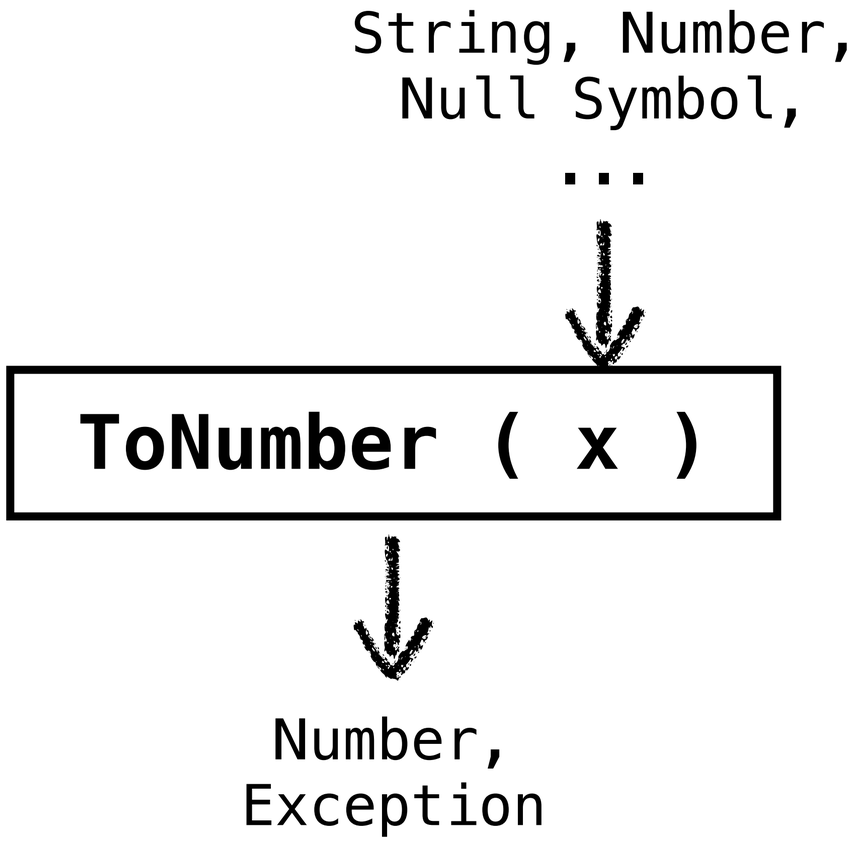


JSTAR [ASE'21]

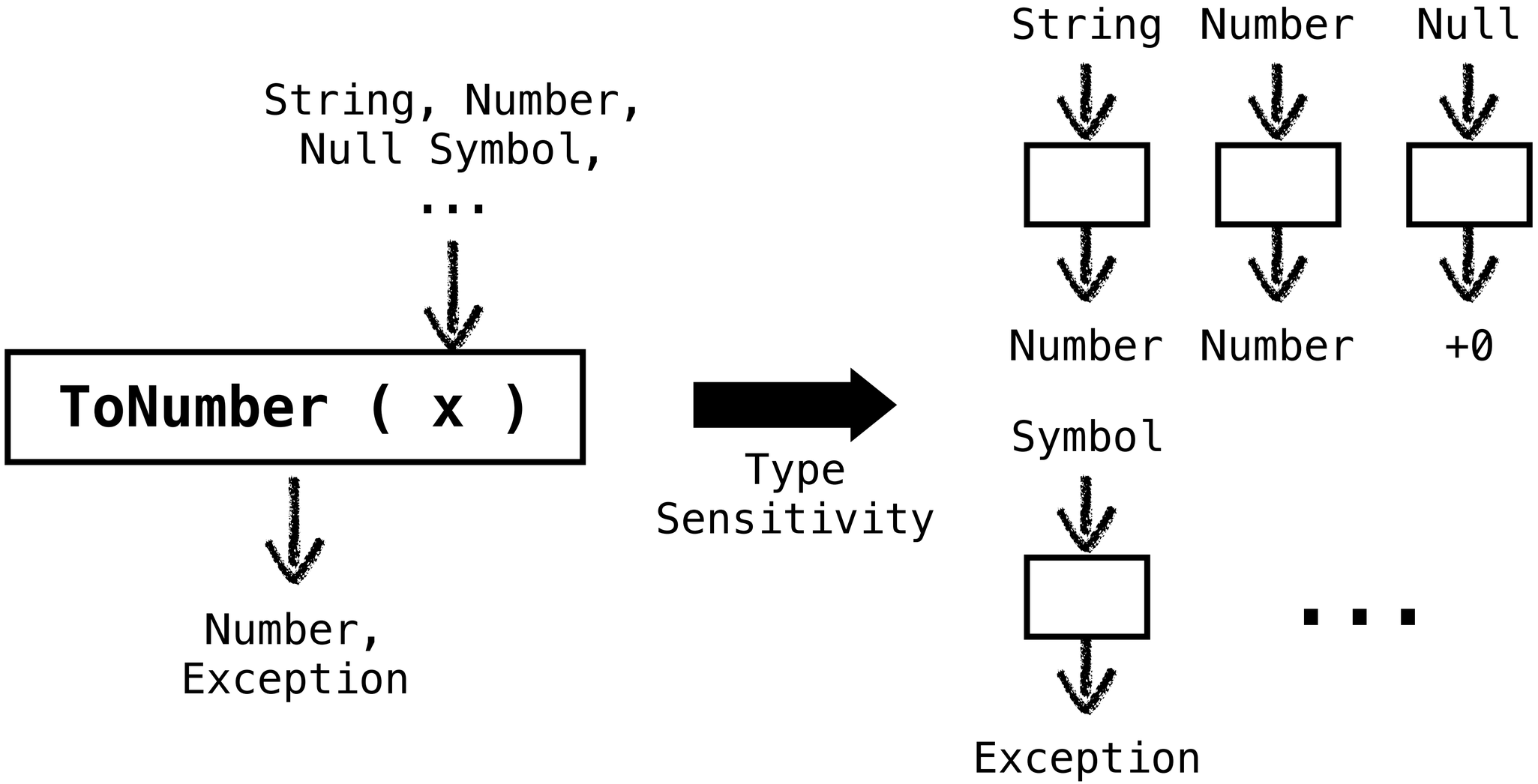
JavaScript Specification Type Analyzer using Refinement



JSTAR - Type Sensitivity



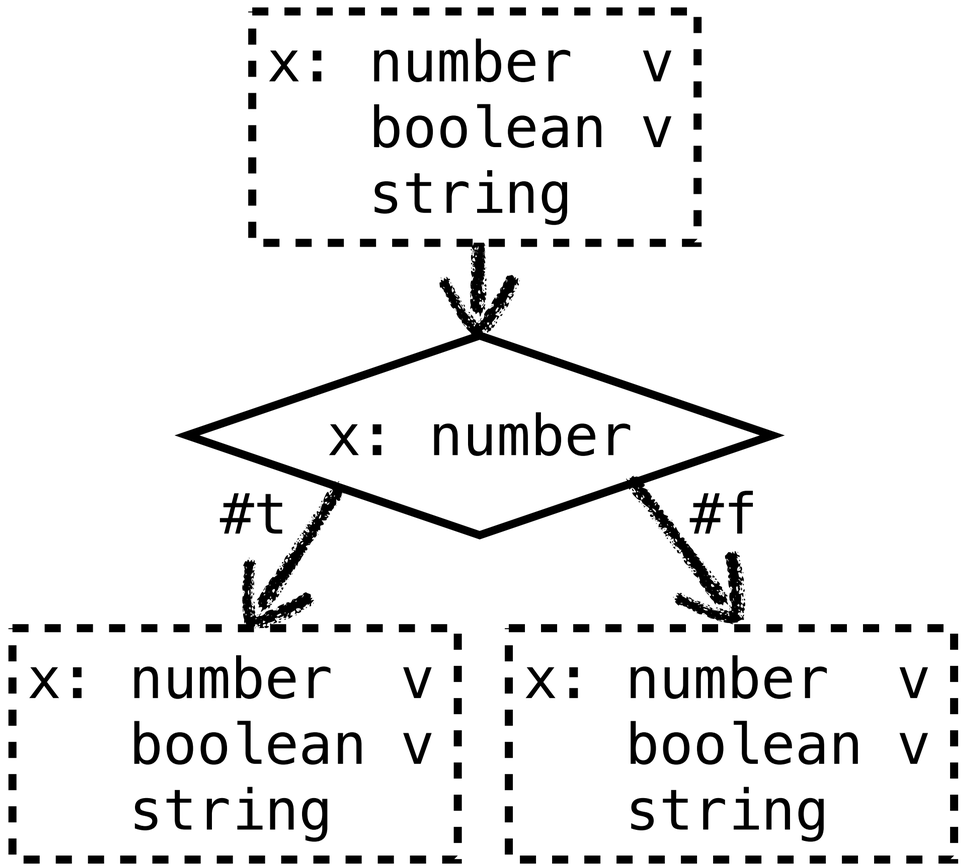
JSTAR - Type Sensitivity



JSTAR - Condition-based Refinement

$$
\begin{aligned}
\texttt{refine}(!e, b)(\sigma^\sharp) &= \texttt{refine}(e, \neg b)(\sigma^\sharp) \\
\texttt{refine}(e_0 \texttt{ || } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(e_0 \texttt{ \&\& } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \texttt{normal}(\mathbb{T})] \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\texttt{abrupt}\}] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \tau_e^\sharp] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \lfloor \tau_e^\sharp \rfloor] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\tau\}] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \{\tau' \mid \tau' <: \tau\}] \\
\texttt{refine}(e, b)(\sigma^\sharp) &= \sigma^\sharp
\end{aligned}
$$

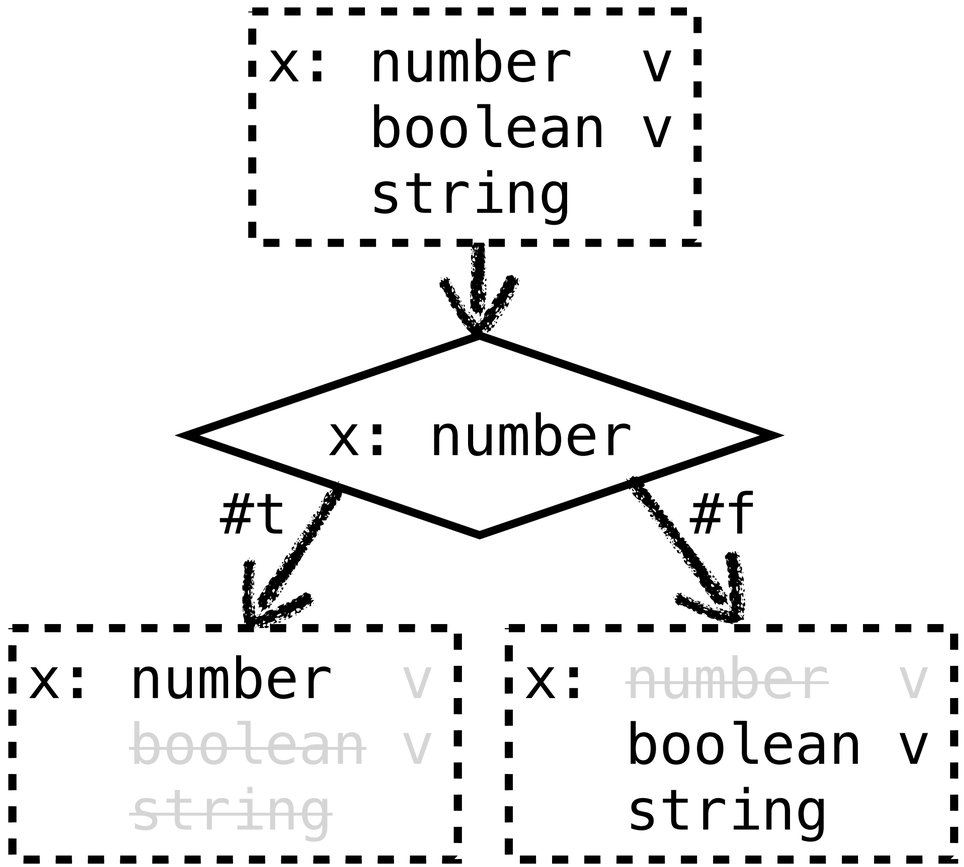
where $\sigma_j^\sharp = \texttt{refine}(e_j, b)(\sigma^\sharp)$ for $j = 0, 1$, $\tau_e^\sharp = [\![e]\!]_e^\sharp(\sigma^\sharp)$, and $\lfloor \tau^\sharp \rfloor$ returns $\{\tau\}$ if $\tau^\sharp$ denotes a singleton type τ, or returns $\varnothing$, otherwise.



JSTAR - Condition-based Refinement

$$
\begin{aligned}
\texttt{refine}(!e, b)(\sigma^\sharp) &= \texttt{refine}(e, \neg b)(\sigma^\sharp) \\
\texttt{refine}(e_0 \texttt{ || } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(e_0 \texttt{ \&\& } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \texttt{normal}(\mathbb{T})] \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\texttt{abrupt}\}] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \tau_e^\sharp] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \lfloor \tau_e^\sharp \rfloor] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\tau\}] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \{\tau' \mid \tau' <: \tau\}] \\
\texttt{refine}(e, b)(\sigma^\sharp) &= \sigma^\sharp
\end{aligned}
$$

where $\sigma_j^\sharp = \texttt{refine}(e_j, b)(\sigma^\sharp)$ for $j = 0, 1$, $\tau_e^\sharp = [\![e]\!]_e^\sharp(\sigma^\sharp)$, and $\lfloor \tau^\sharp \rfloor$ returns $\{\tau\}$ if $\tau^\sharp$ denotes a singleton type τ, or returns $\varnothing$, otherwise.



JSTAR - Evaluation

59.2% Precision

93 Bugs Detected

- **Type Analysis for 864 versions of ECMA-262 in 3 years**

Checker	Bug Kind	Precision = (# True Bugs) / (# Detected Bugs)					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25	/ -25
Operand	NoNumber	22 / 113	2 / 65	22 / 44	2 / 6	/ -69	/ -59
	Abrupt		20 / 48		20 / 38		/ -10
Total		92 / 279 (33.0%)		93 / 157 (59.2%)		+1 / -122 (+26.3%)	

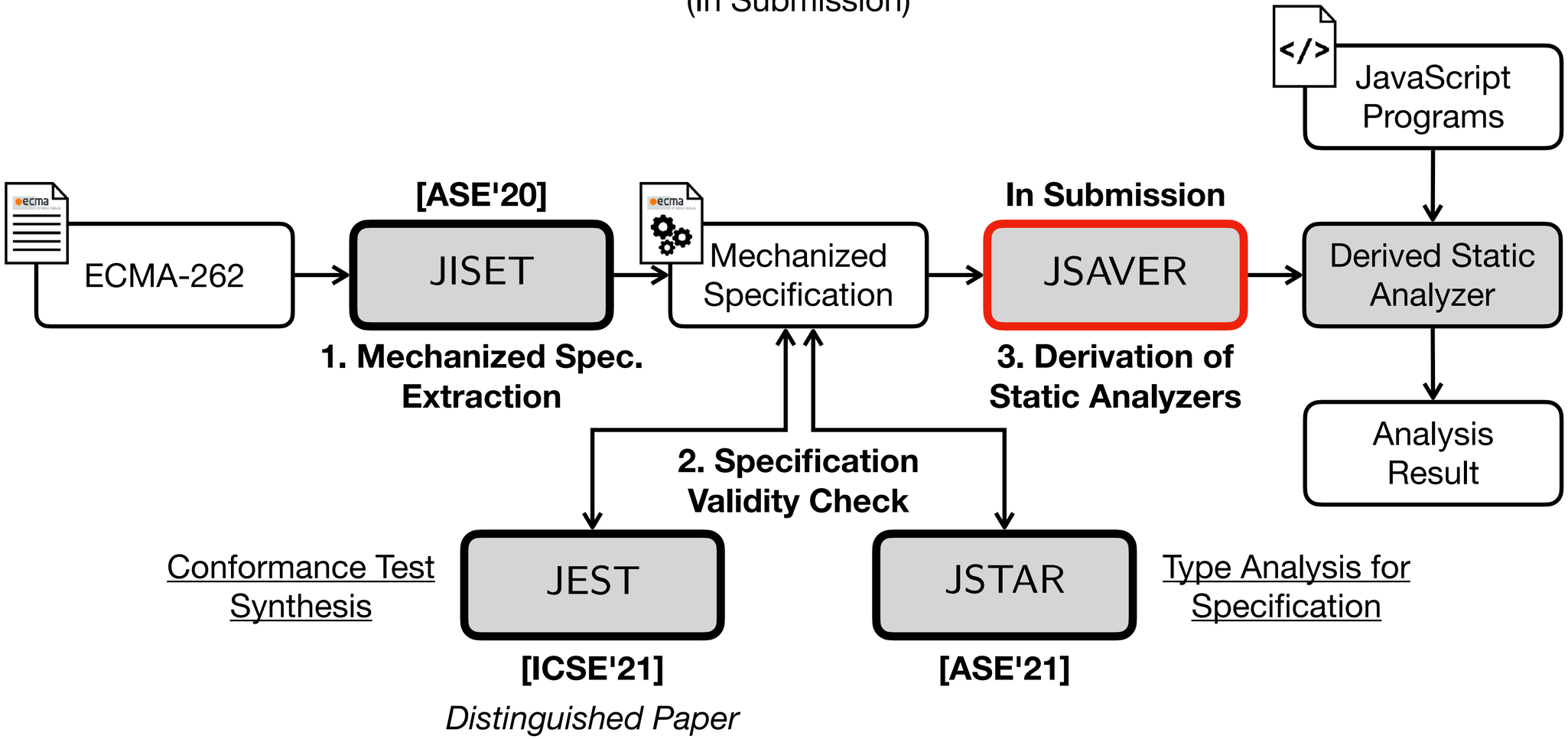
Name	Feature	#	Checker	Created	Life Span
ES12-1	Switch	3	Reference	2015-09-22	1,996 days
ES12-2	Try	3	Reference	2015-09-22	1,996 days
ES12-3	Arguments	1	Reference	2015-09-22	1,996 days
ES12-4	Array	2	Reference	2015-09-22	1,996 days
ES12-5	Async	1	Reference	2015-09-22	1,996 days
ES12-6	Class	1	Reference	2015-09-22	1,996 days
ES12-7	Branch	1	Reference	2015-09-22	1,996 days
ES12-8	Arguments	2	Operand	2015-12-16	1,910 days

14 Bugs in ES12

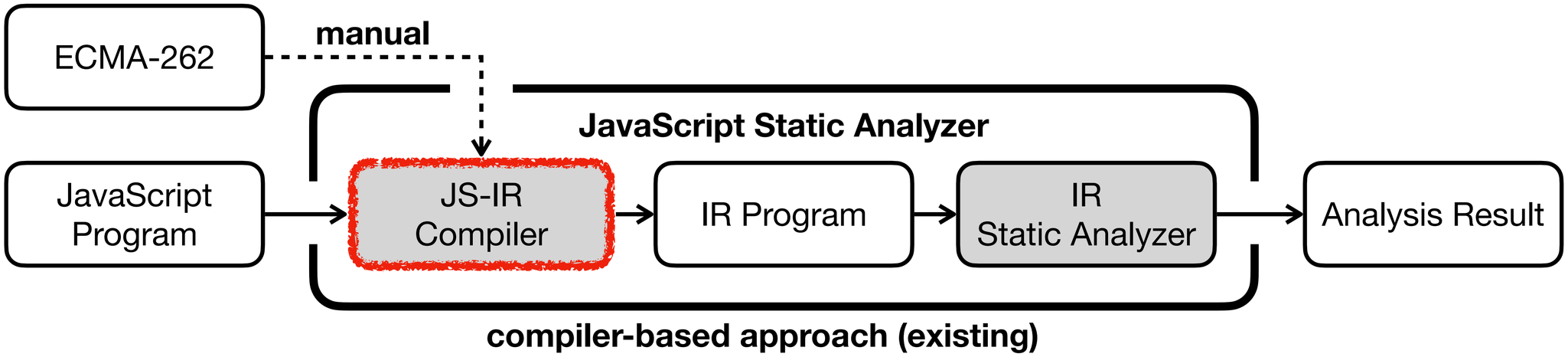
Automatically Deriving JavaScript Static Analyzers from Language Specifications

Jihyeok Park, Seungmin An, and Sukyoung Ryu

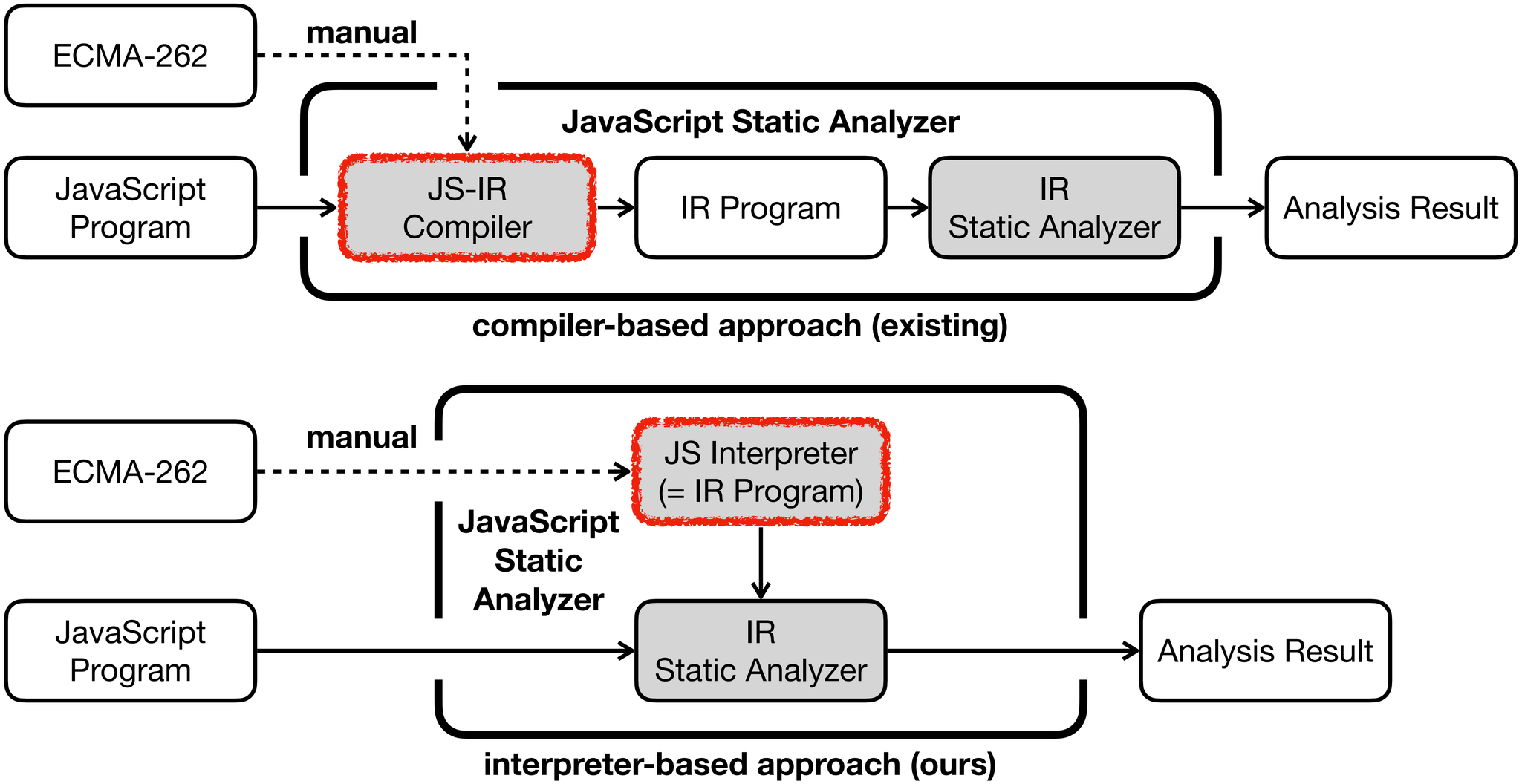
(In Submission)



JSAVER - Meta-Level Static Analysis



JSAVER - Meta-Level Static Analysis



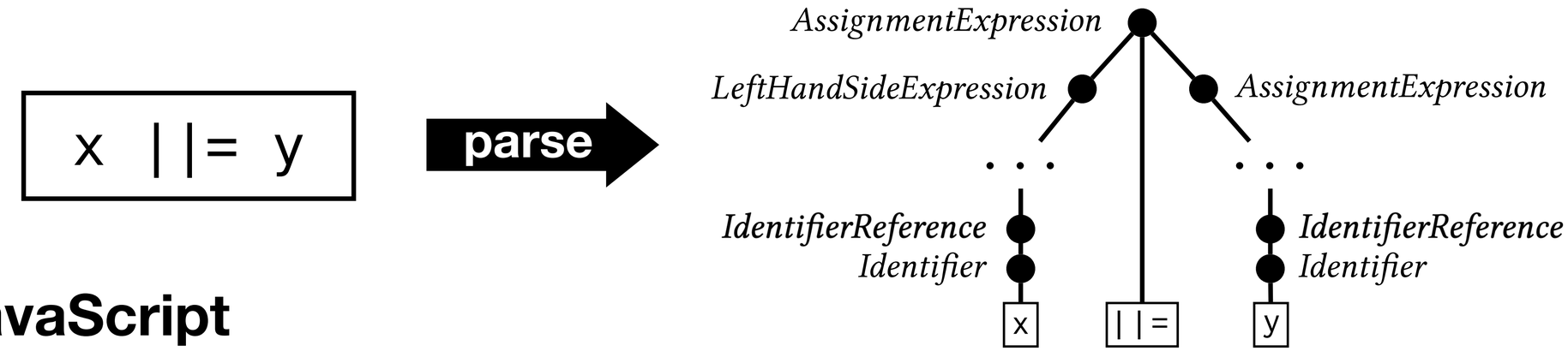
JSAVER - Meta-Level Static Analysis

```
x ||= y
```

JavaScript

IR_{ES}

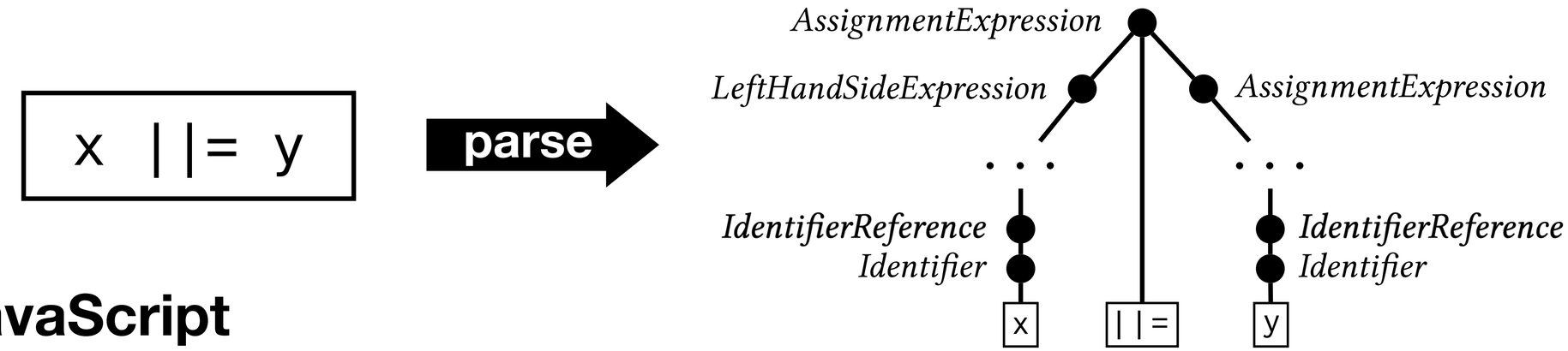
JSAVER - Meta-Level Static Analysis



JavaScript

IR_{ES}

JSAVER - Meta-Level Static Analysis

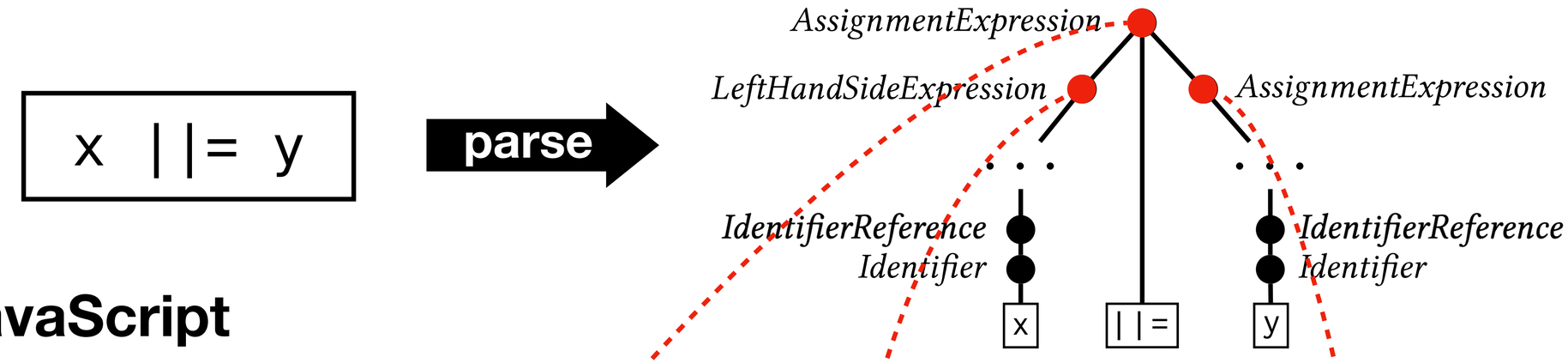


JavaScript

IR$_{ES}$

```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

JSAVER - Meta-Level Static Analysis

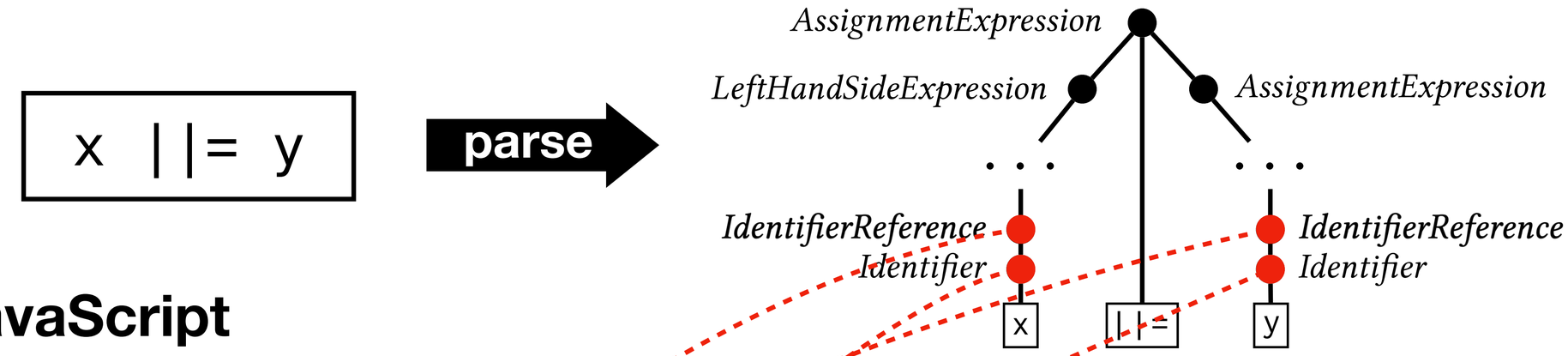


JavaScript

IR$_{ES}$

```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

JSAVER - AST Sensitivity

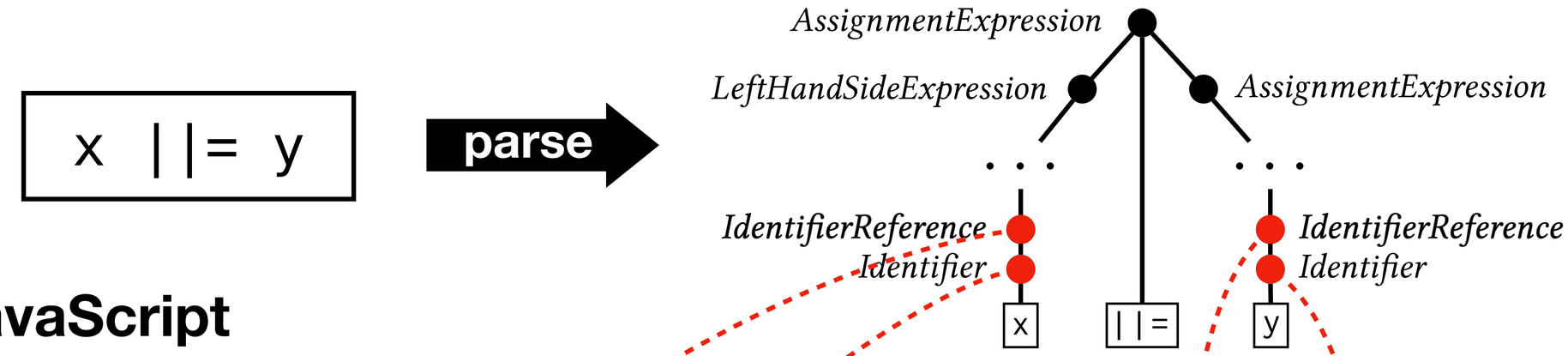


JavaScript

IR$_{ES}$

```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

JSAVER - AST Sensitivity



JavaScript

IR$_{ES}$

```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

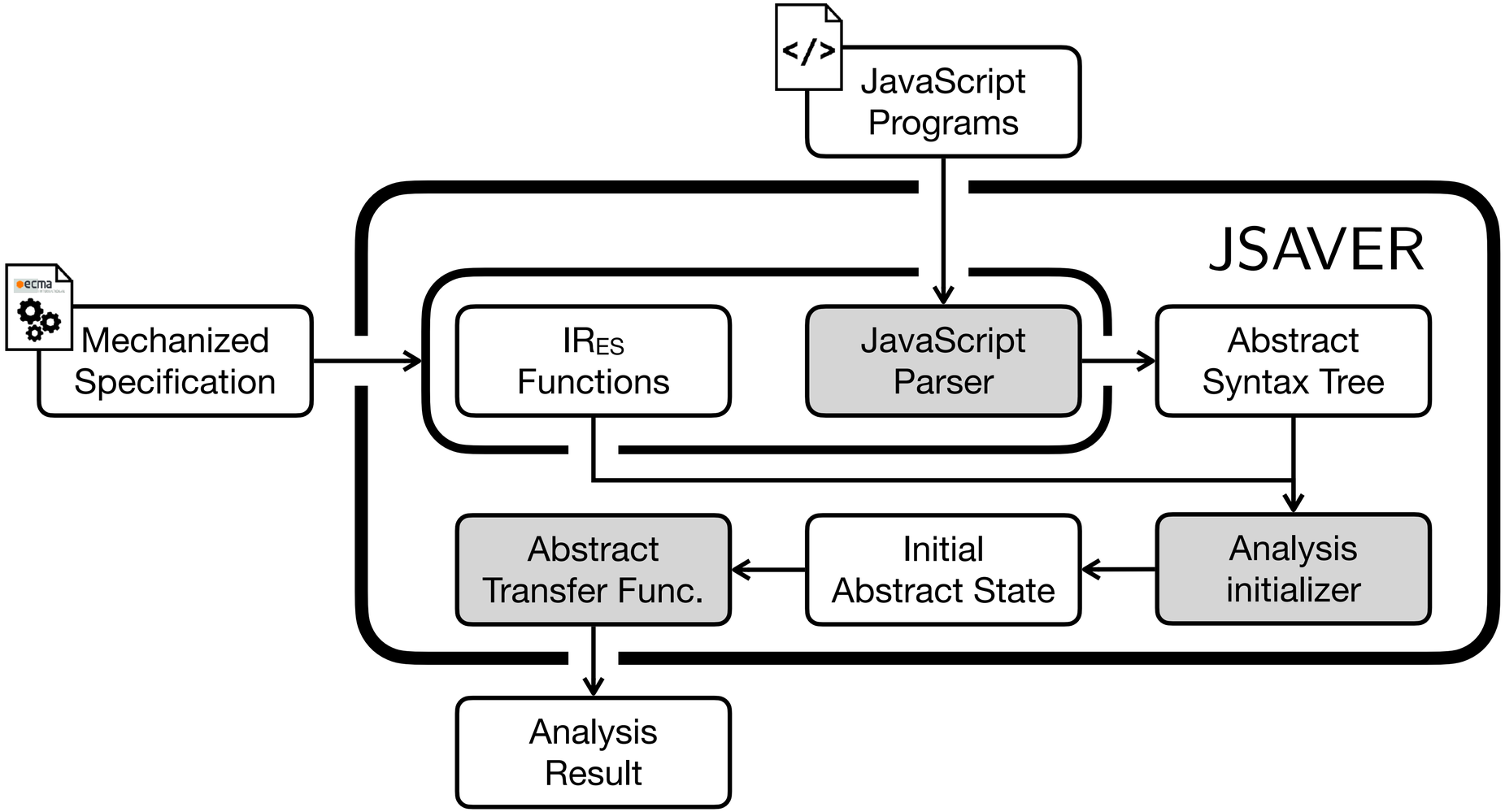
```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

JSAVER - AST Sensitivity

JavaScript	IR_{ES}
flow-sensitivity	$\delta^{\text{flow}}(t \in \mathbb{T}) = \{\sigma = (\_, c) \in \mathbb{S} \mid \text{syntax-ctxt}(c) = c' \wedge c'(\texttt{this}) = t\}$
k-callsite sensitivity	$\delta^{k\text{-cfa}}(\bar{t} \in \mathbb{T}^{\leq k}) = \{\sigma = (\_, c) \in \mathbb{S} \mid$ $\bar{t} = [t_1, \cdots, t_n] \wedge n \leq k \wedge$ $(\nexists(\text{syntax-ctxt} \circ \text{js-call-ctxt})^{n+1}(c) \vee n = k) \wedge$ $\forall 1 \leq i \leq n.$ $((\text{syntax-ctxt} \circ \text{js-call-ctxt})^{i}(c) = c_i \wedge c_i(\texttt{this}) = t_i)\}$

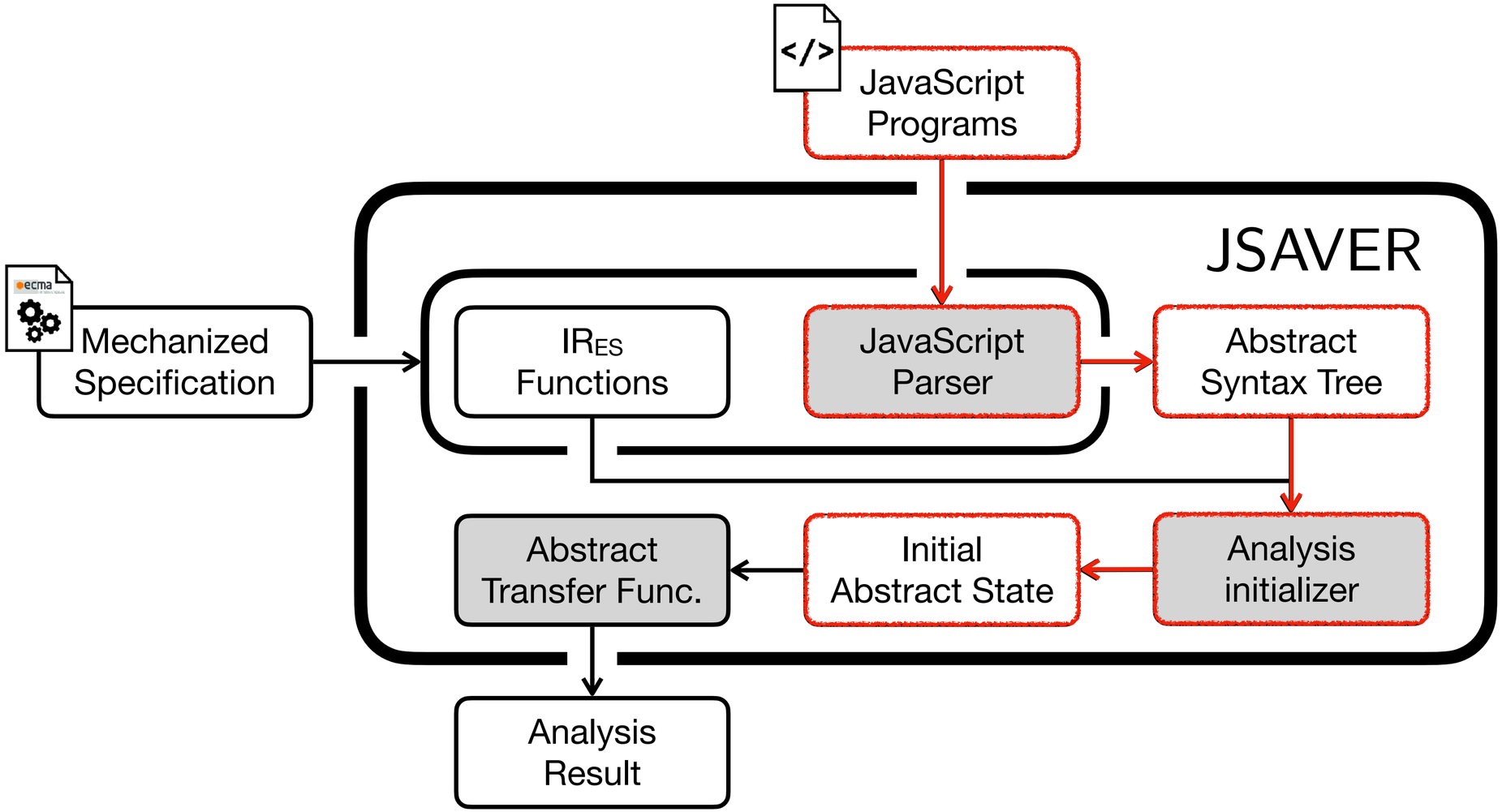
JSAVER [In Submission]

JavaScript Static Analyzer via ECMAScript Representation

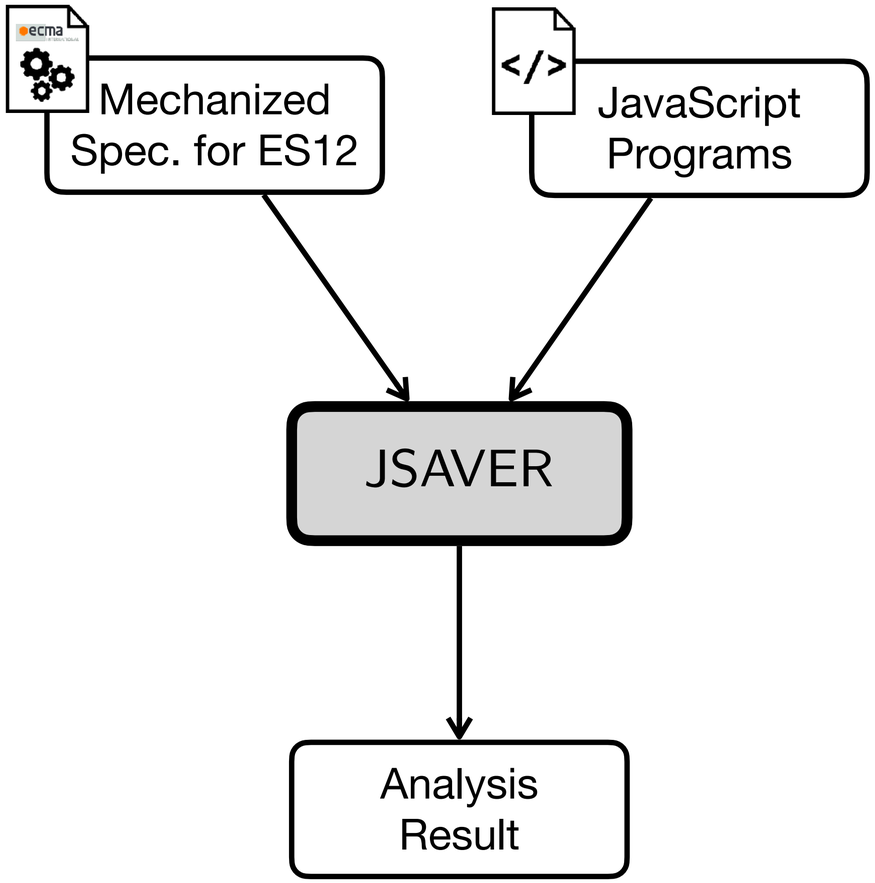


JSAVER [In Submission]

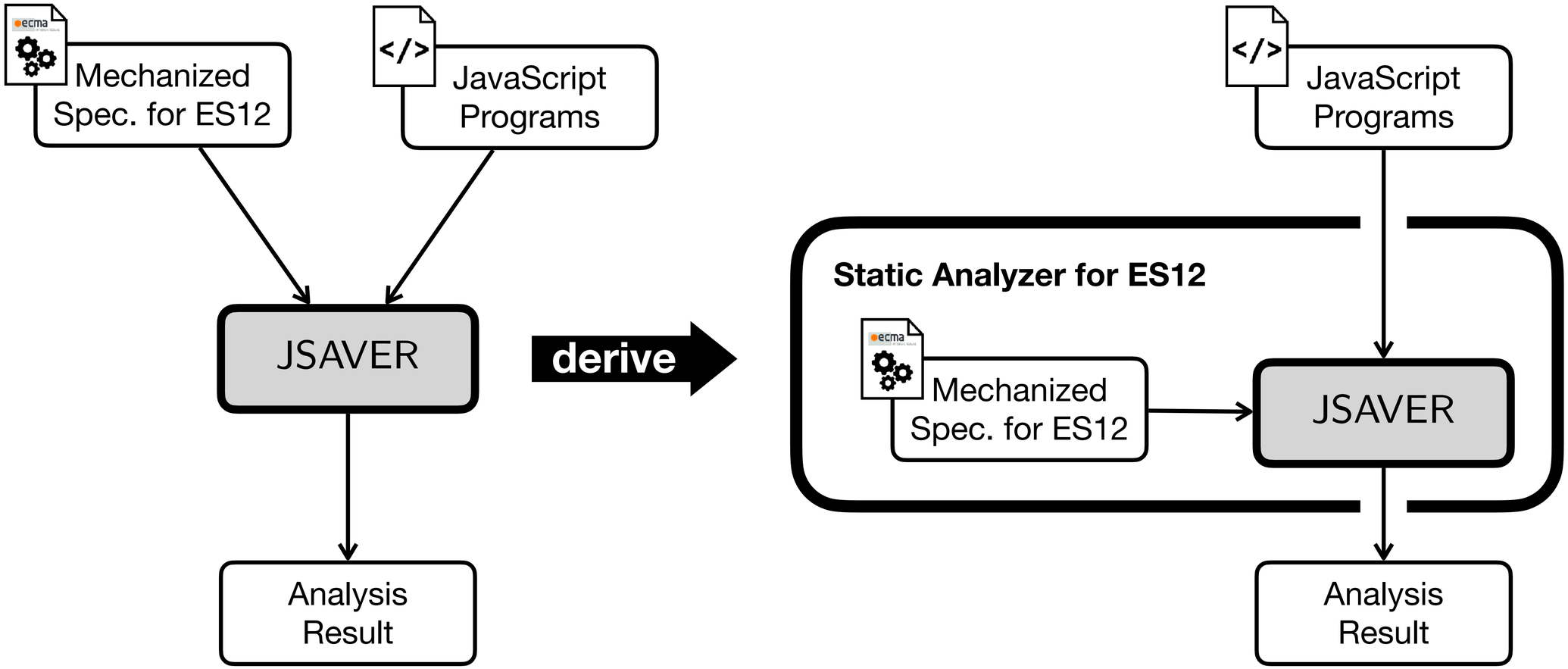
JavaScript Static Analyzer via ECMAScript Representation



JSAVER - Static Analyzer Derivation

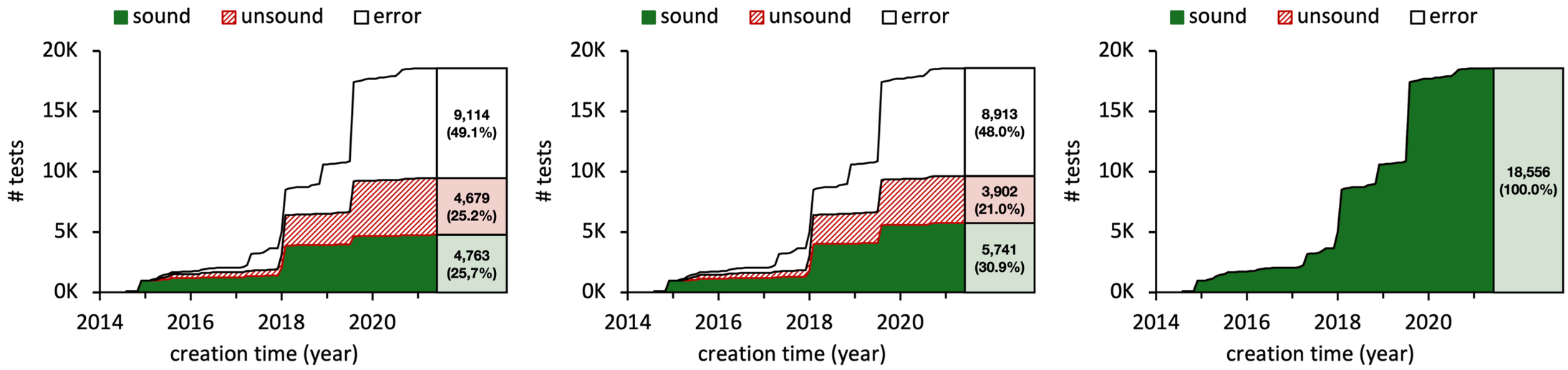


JSAVER - Static Analyzer Derivation



JSAVER - Evaluation

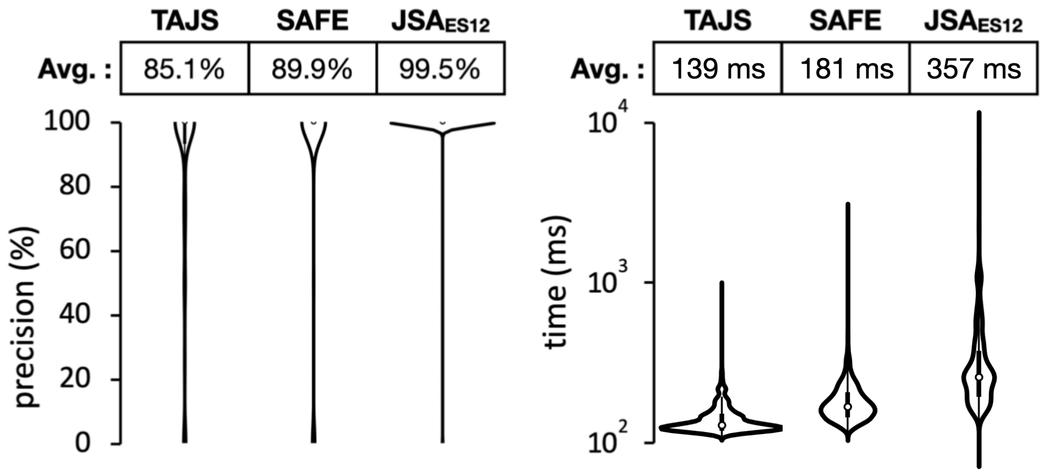
- **JSA_{ES12} - Derived Analyzer for ES12**
- **Soundness / Precision / Performance**
  - 18,556 applicable tests in Test262
  - 3,903 tests analyzable by all the three analyzers



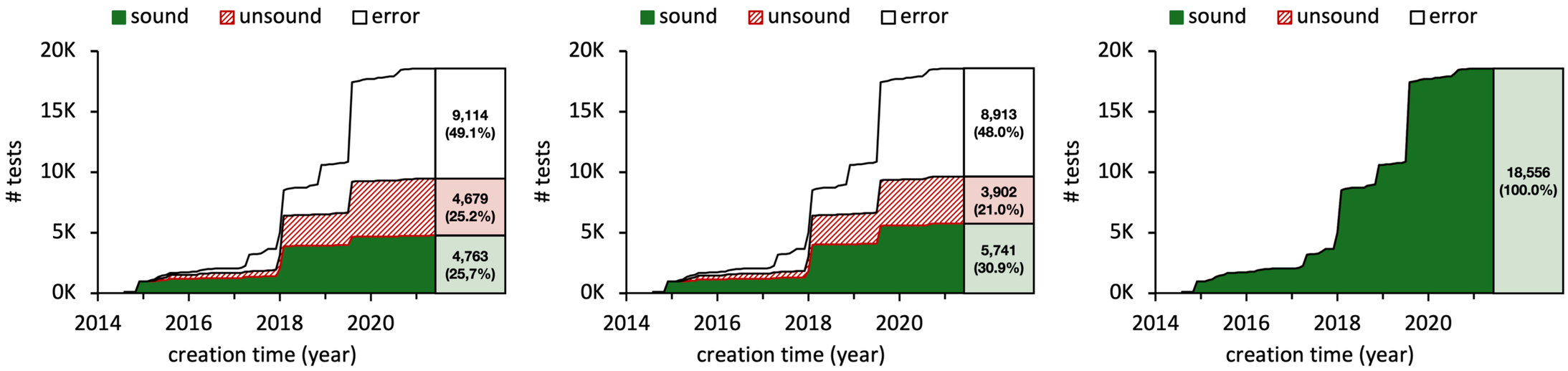
(a) Analysis results of TAJS **(b)** Analysis results of SAFE **(c)** Analysis results of JSA_{ES12}

JSAVER - Evaluation

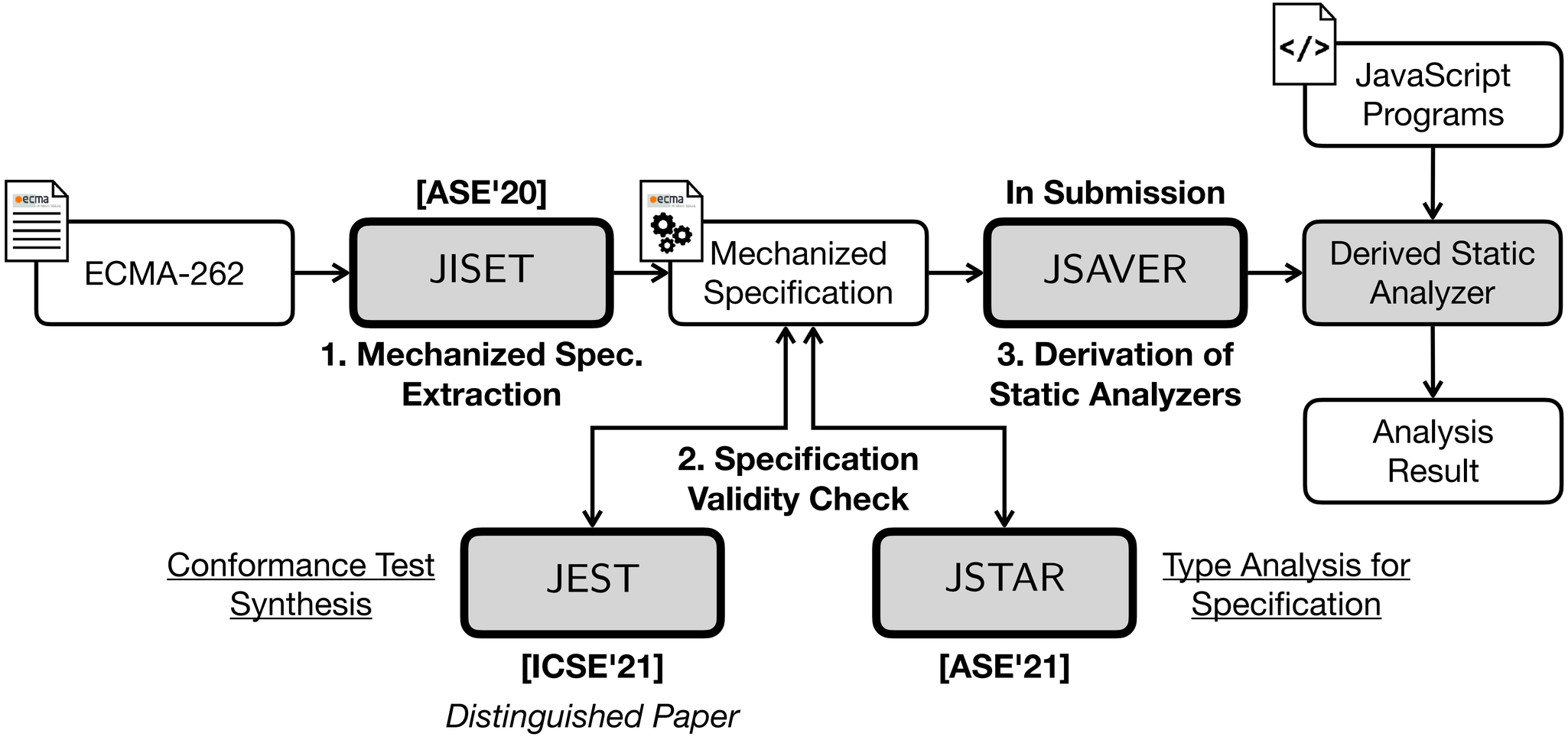
- **JSA_{ES12} - Derived Analyzer for ES12**
- **Soundness / Precision / Performance**
  - 18,556 applicable tests in Test262
  - 3,903 tests analyzable by all the three analyzers



(a) The analysis precision **(b)** The analysis performance



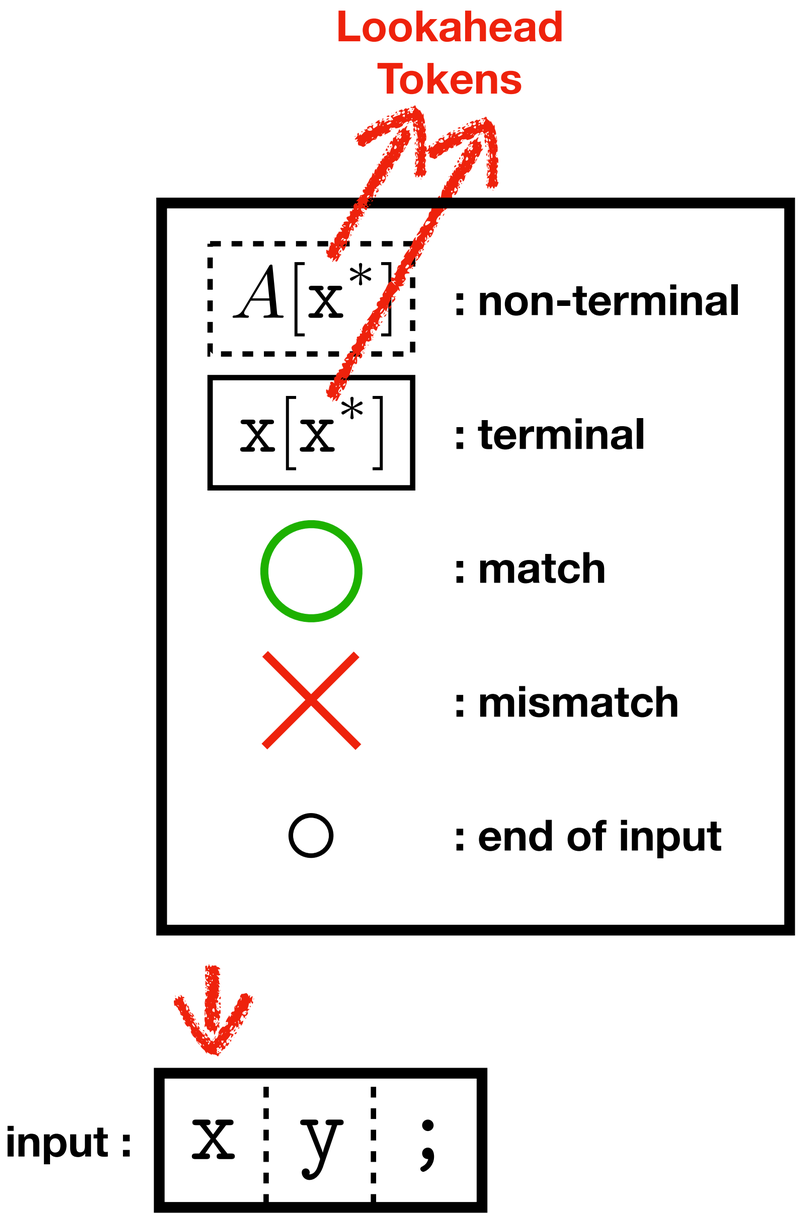
(a) Analysis results of TAJS **(b)** Analysis results of SAFE **(c)** Analysis results of JSA_{ES12}

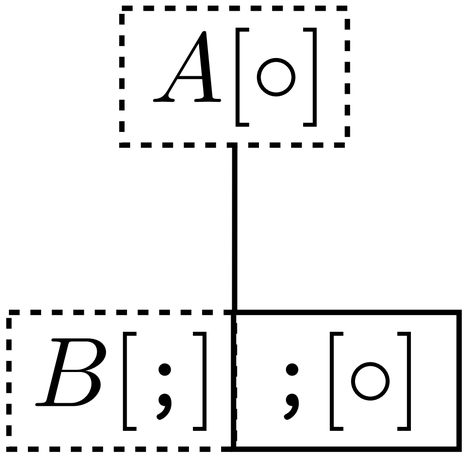

JavaScript Programs
ECMA-262
[ASE'20]
JISET
Mechanized Specification
In Submission
JSAVER
Derived Static Analyzer
1. Mechanized Spec. Extraction
3. Derivation of Static Analyzers
Analysis Result
2. Specification Validity Check
Conformance Test Synthesis
JEST
JSTAR
Type Analysis for Specification
[ICSE'21]
[ASE'21]
Distinguished Paper

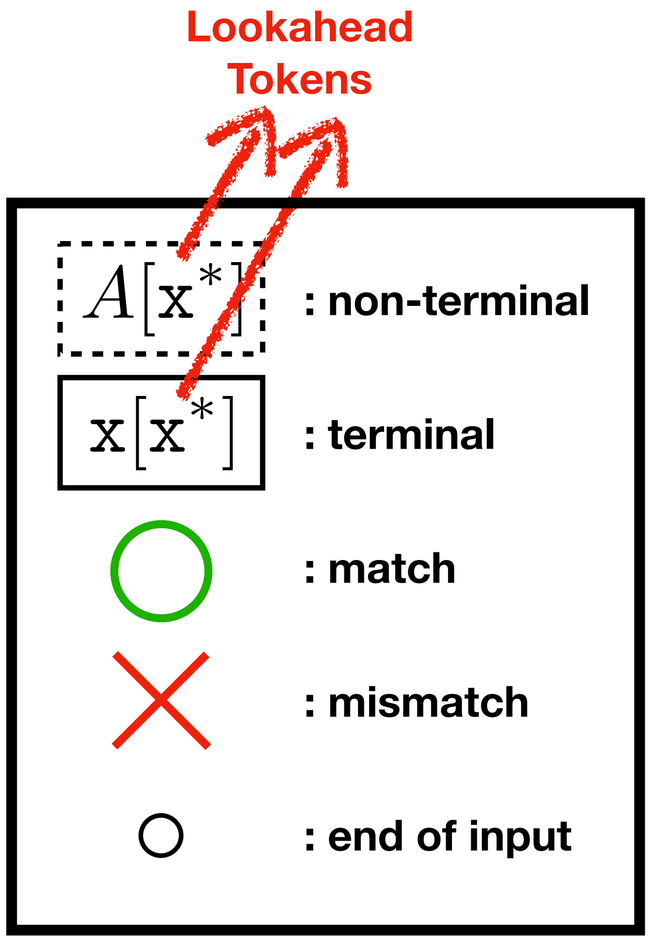
Backup Slides

$A[\circ]$

$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$

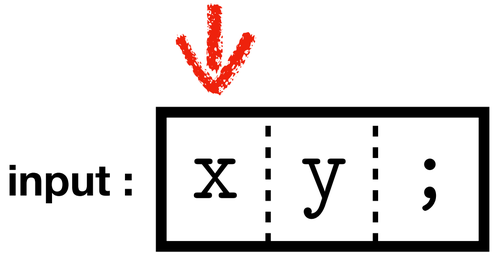


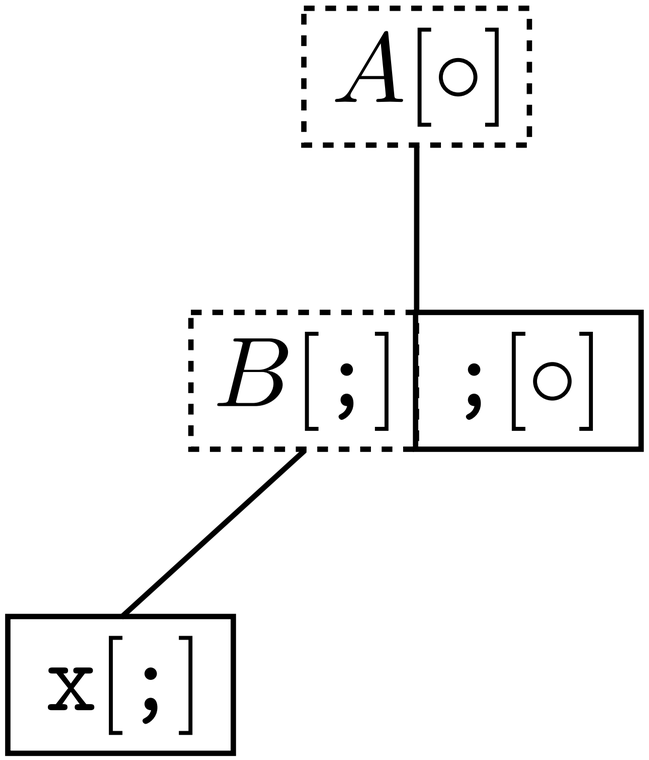




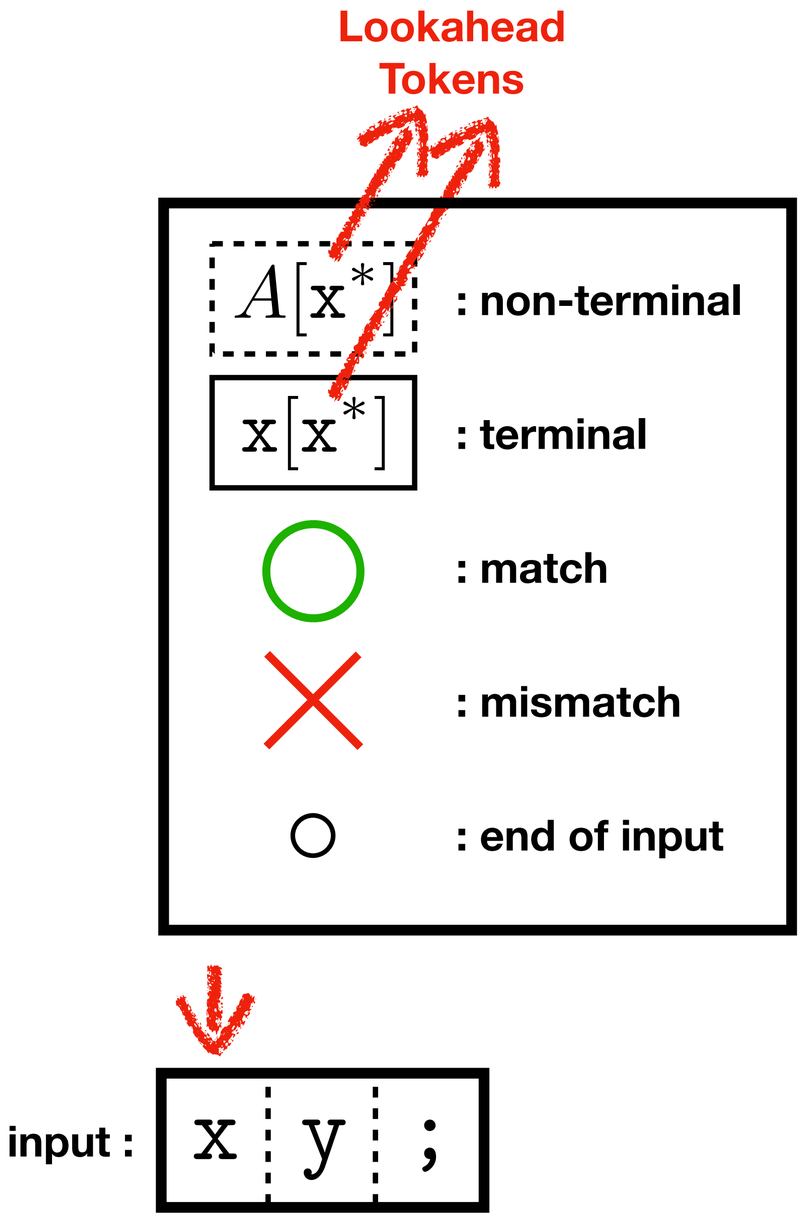
$A ::= B\,; \;/\; B + B\,;$

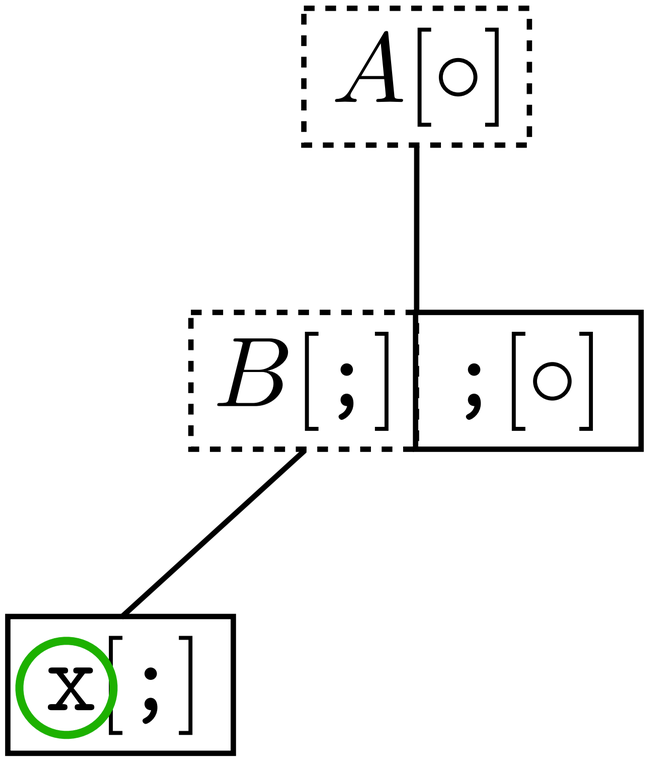
$B ::= \mathtt{x} \;/\; \mathtt{xy}$



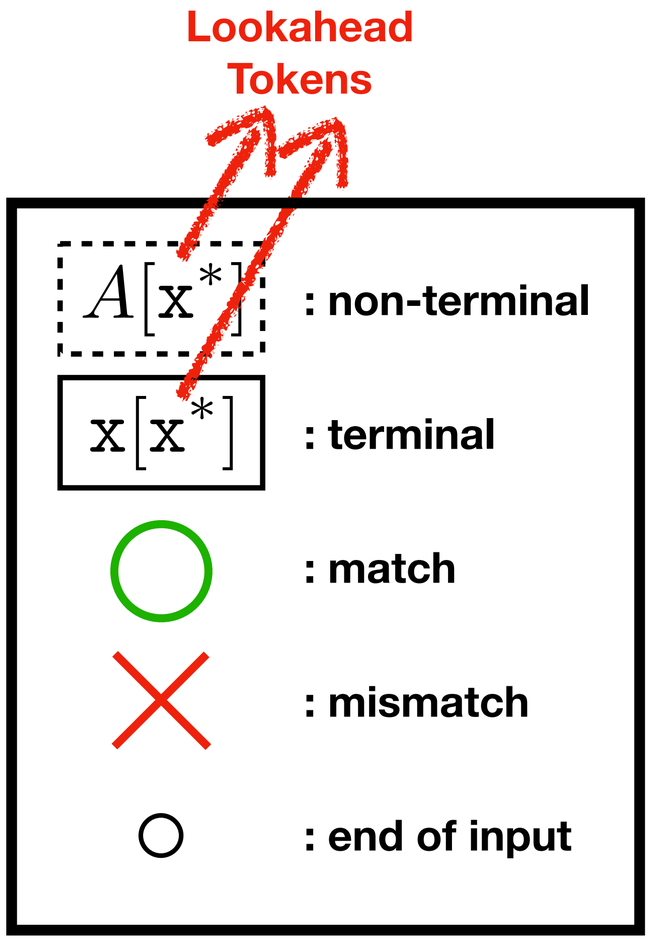


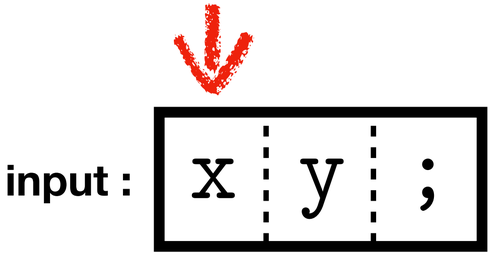
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$

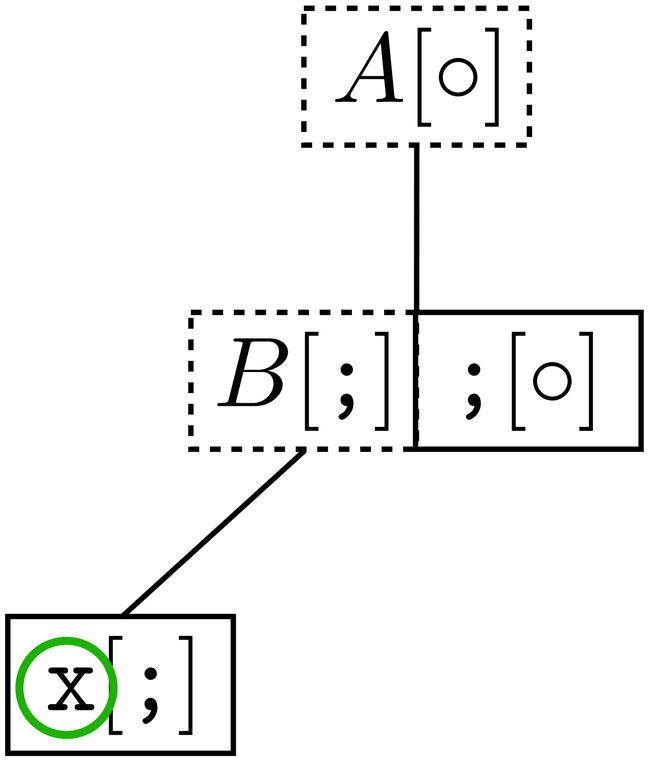




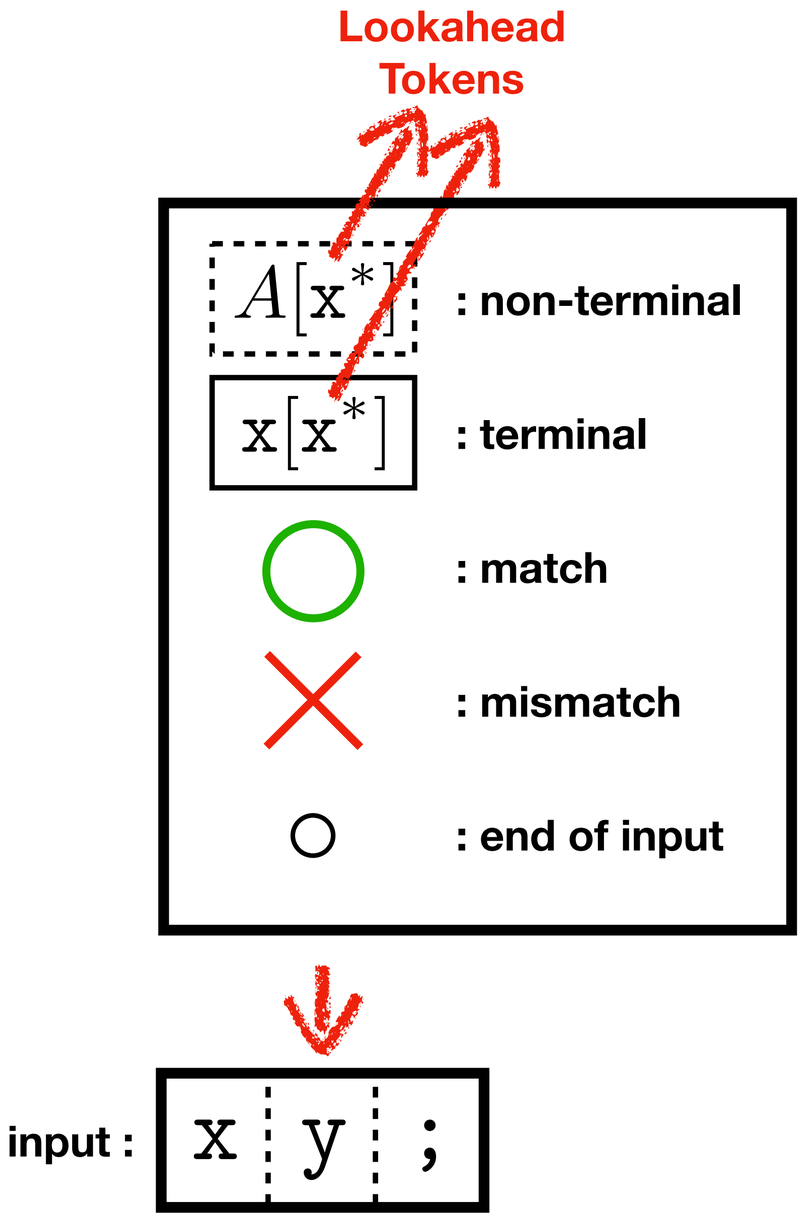
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$

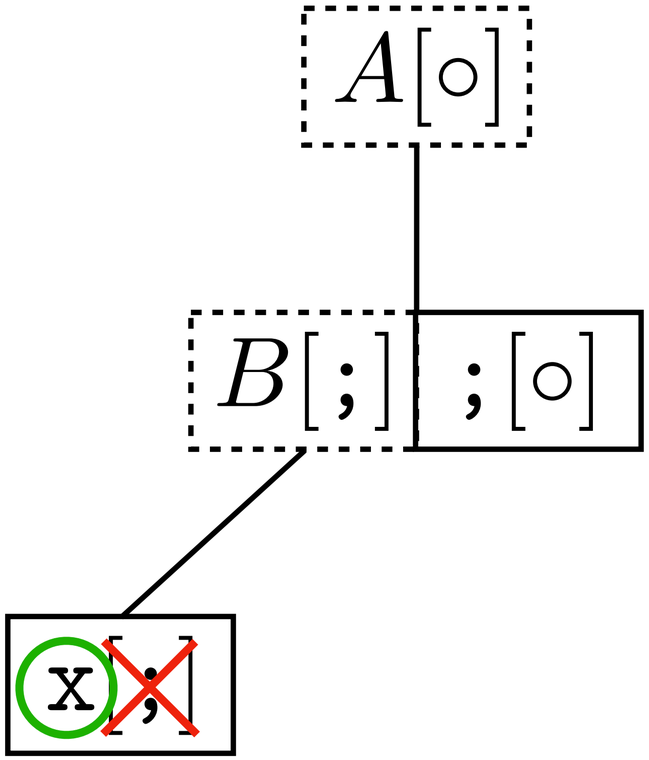




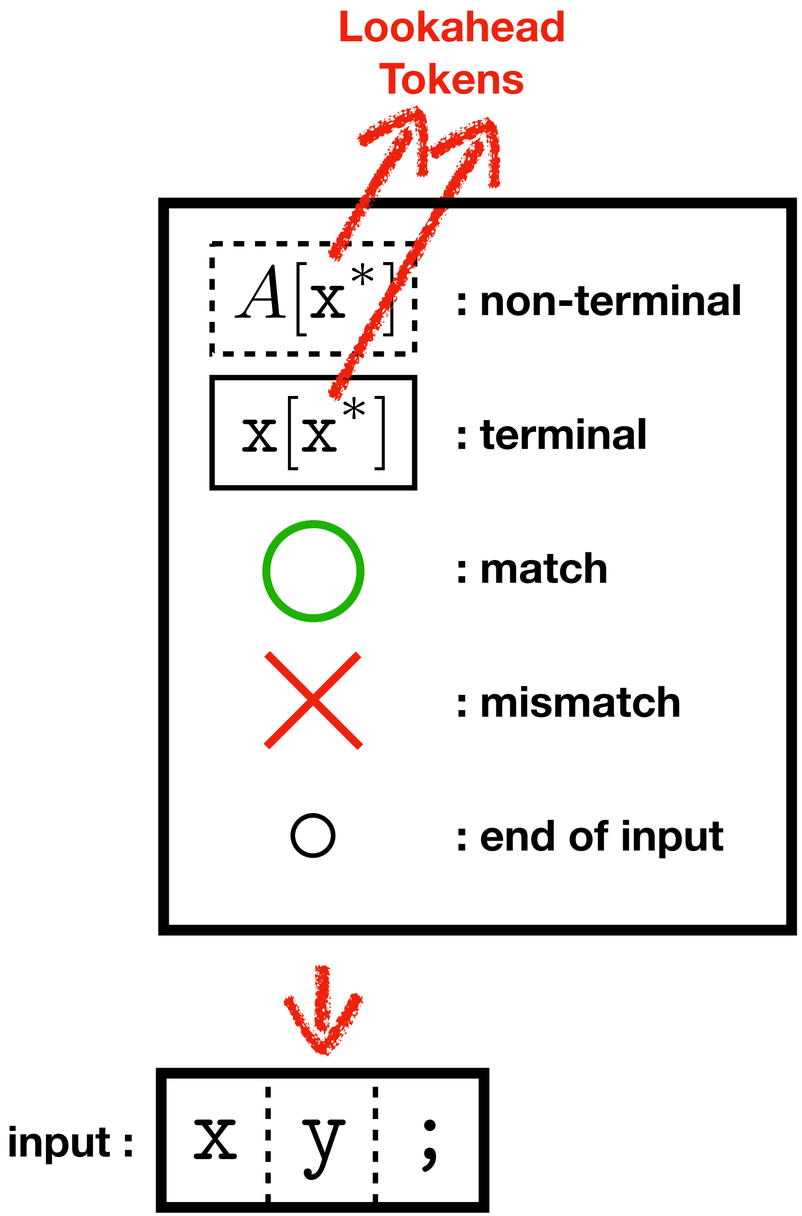


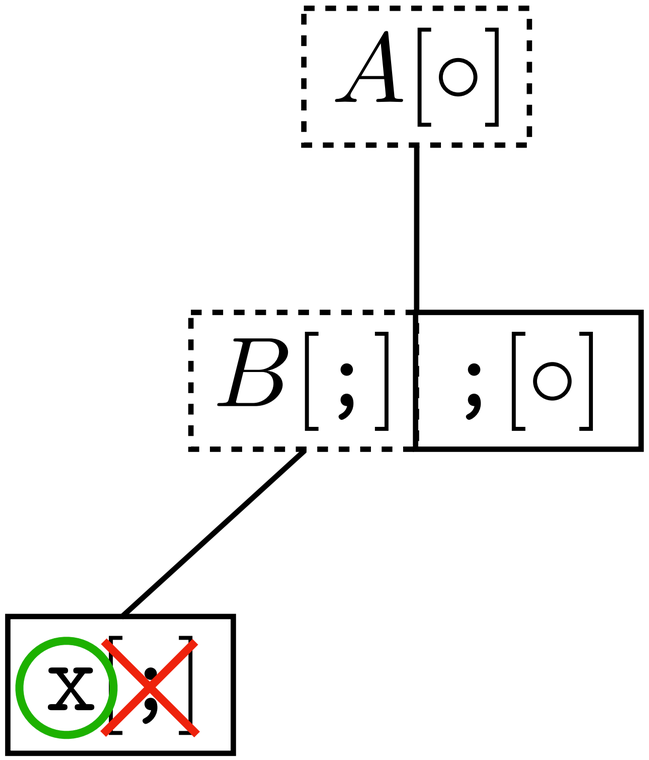
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$



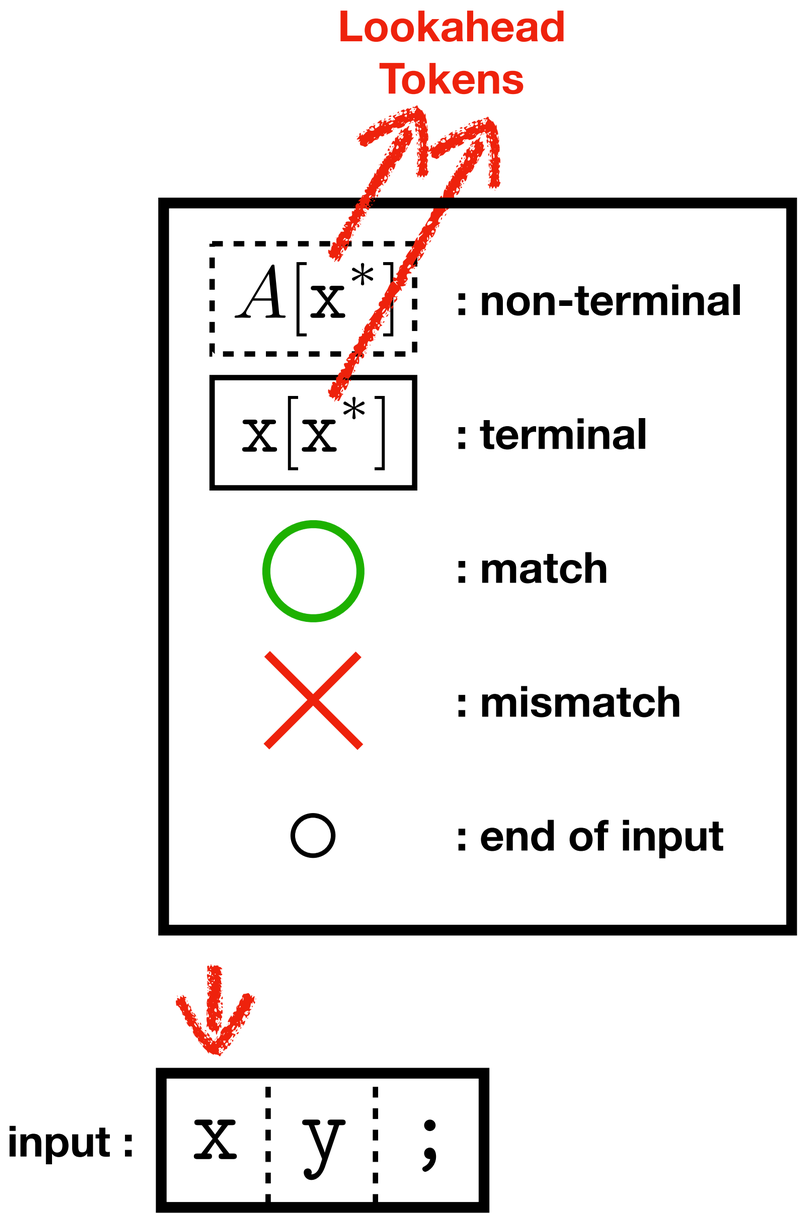


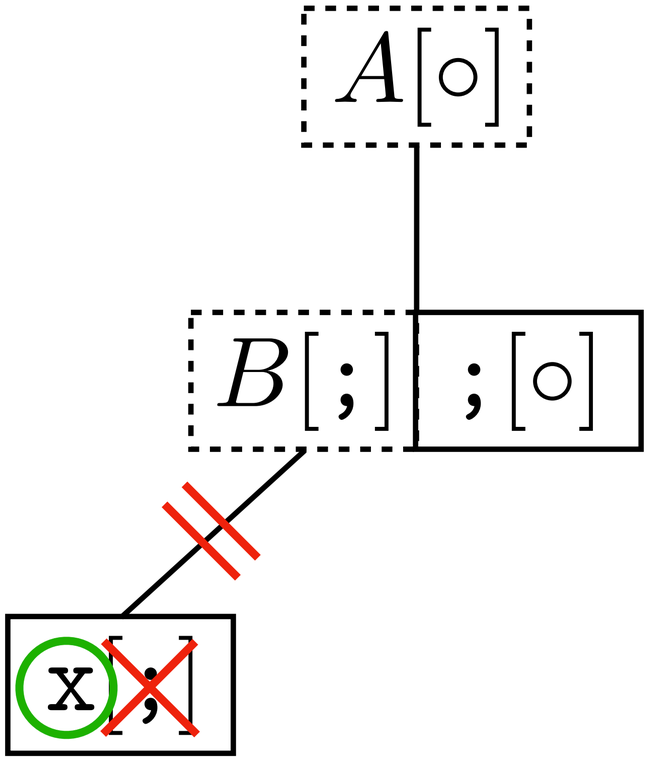
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$



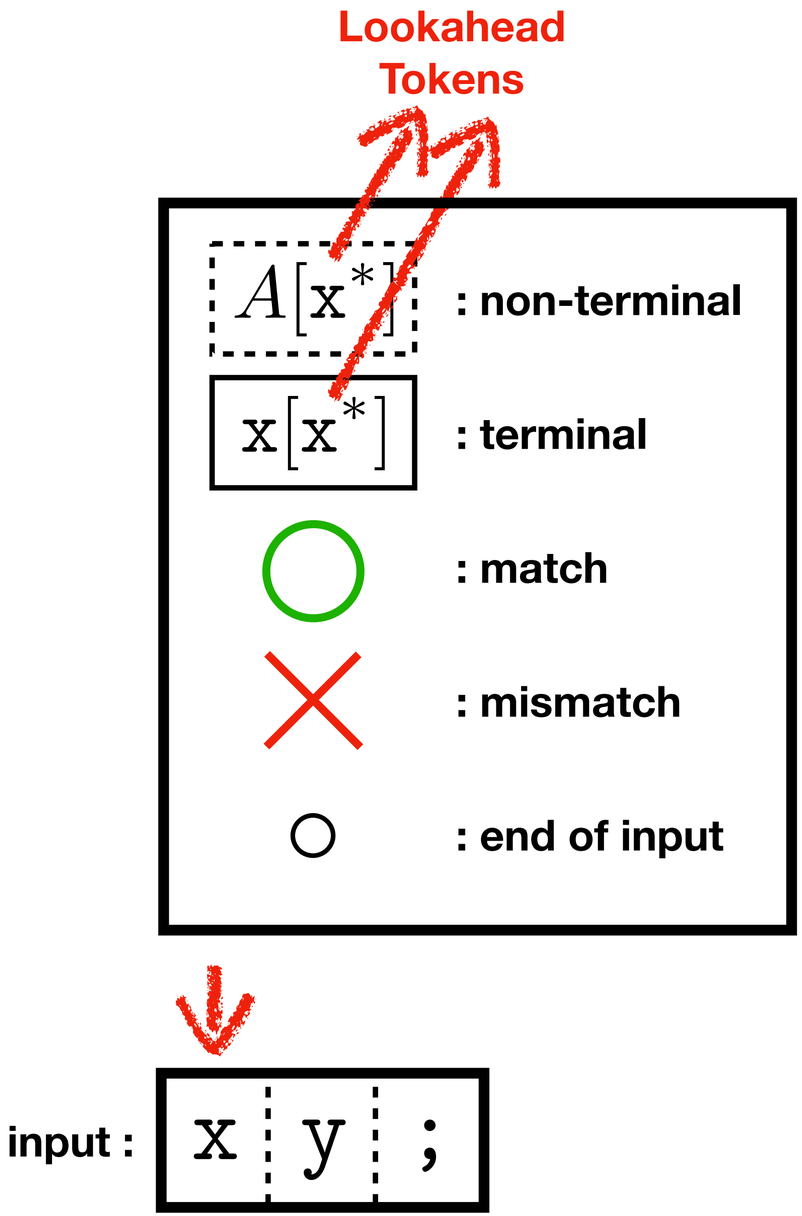


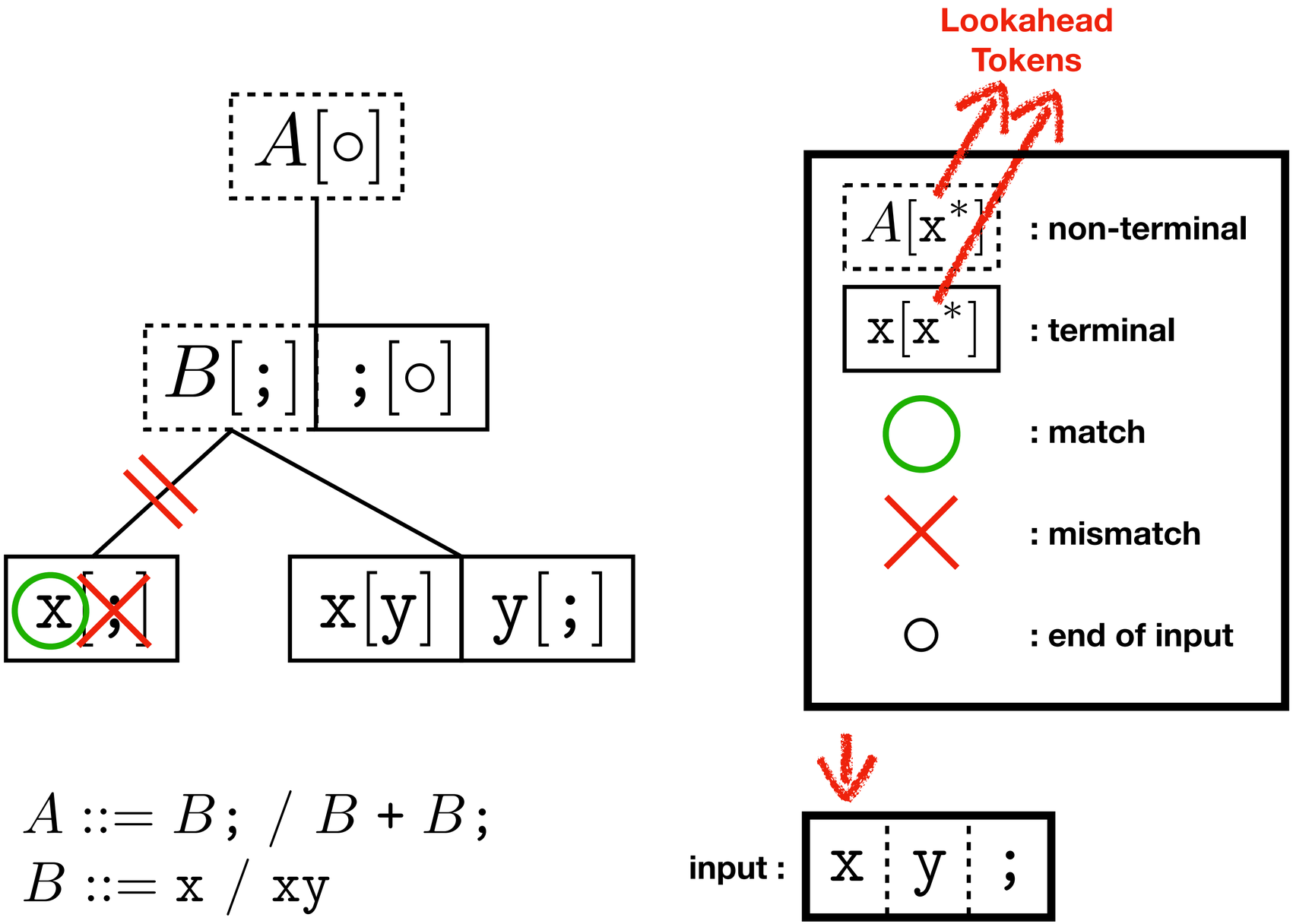
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$





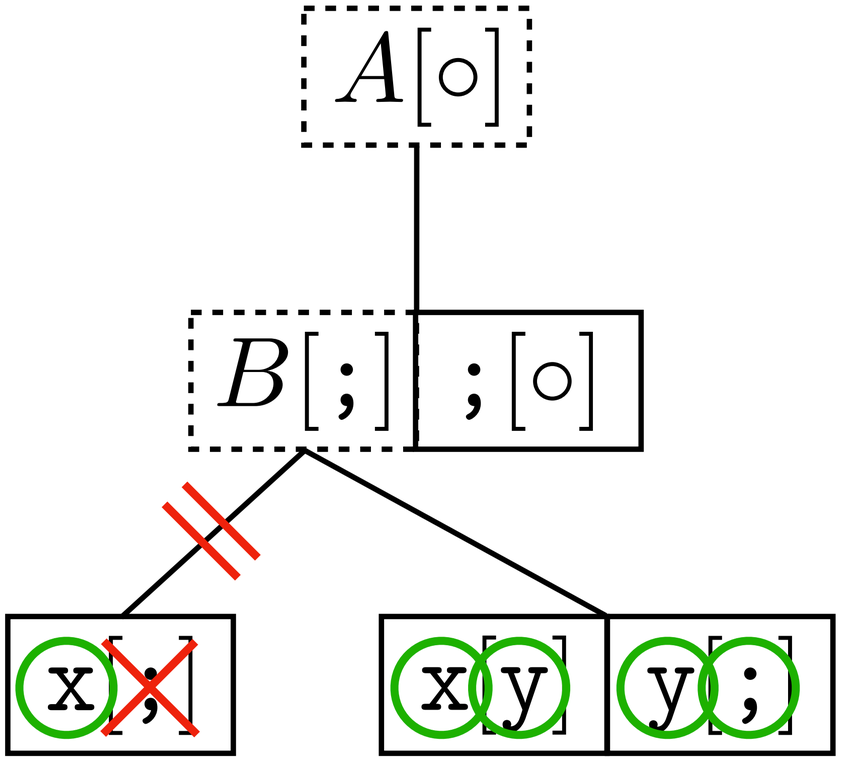
$A ::= B\,; \;/\; B + B\,;$
$B ::= \mathtt{x} \;/\; \mathtt{xy}$





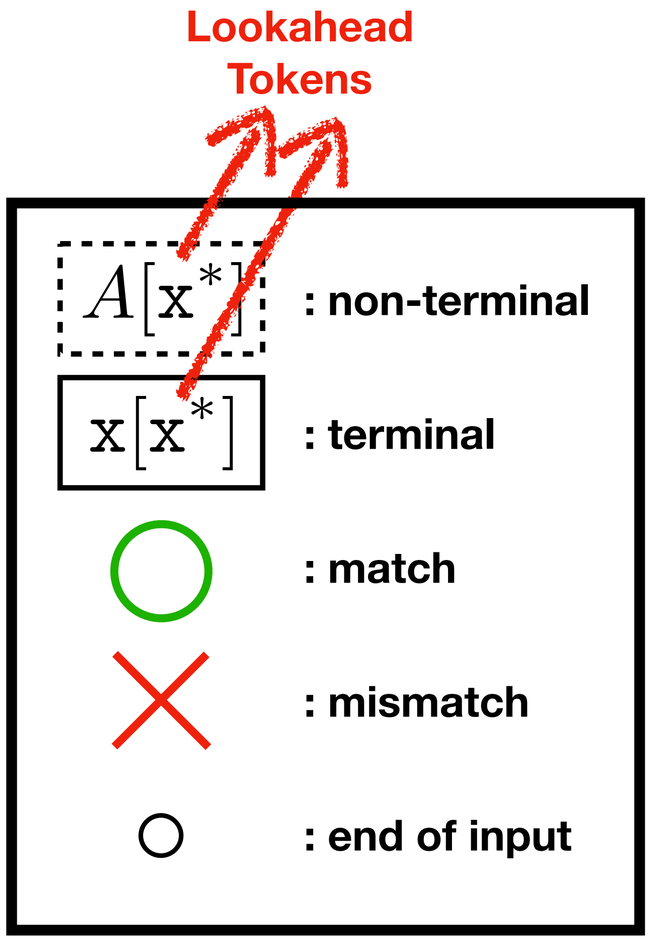
$A ::= B\,; \;/\; B + B\,;$

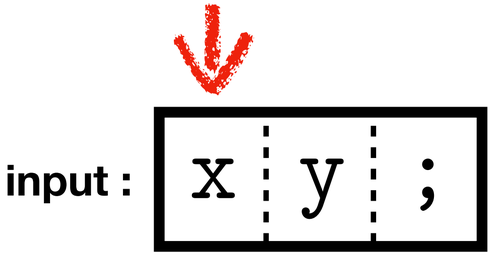
$B ::= \mathtt{x} \;/\; \mathtt{xy}$

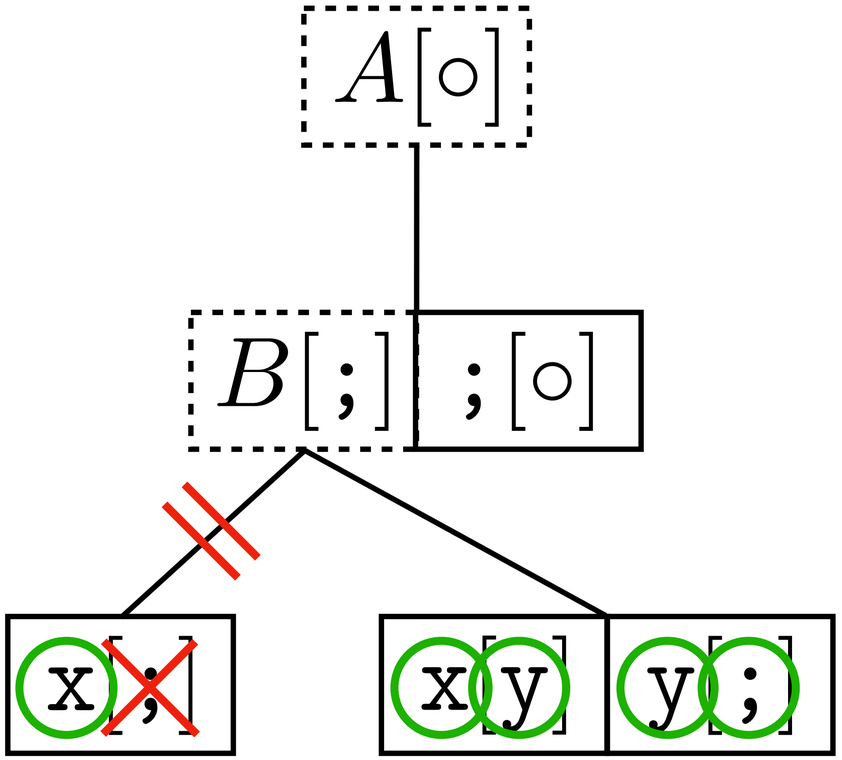


$A ::= B\,; \;/\; B + B\,;$

$B ::= \mathtt{x} \;/\; \mathtt{xy}$

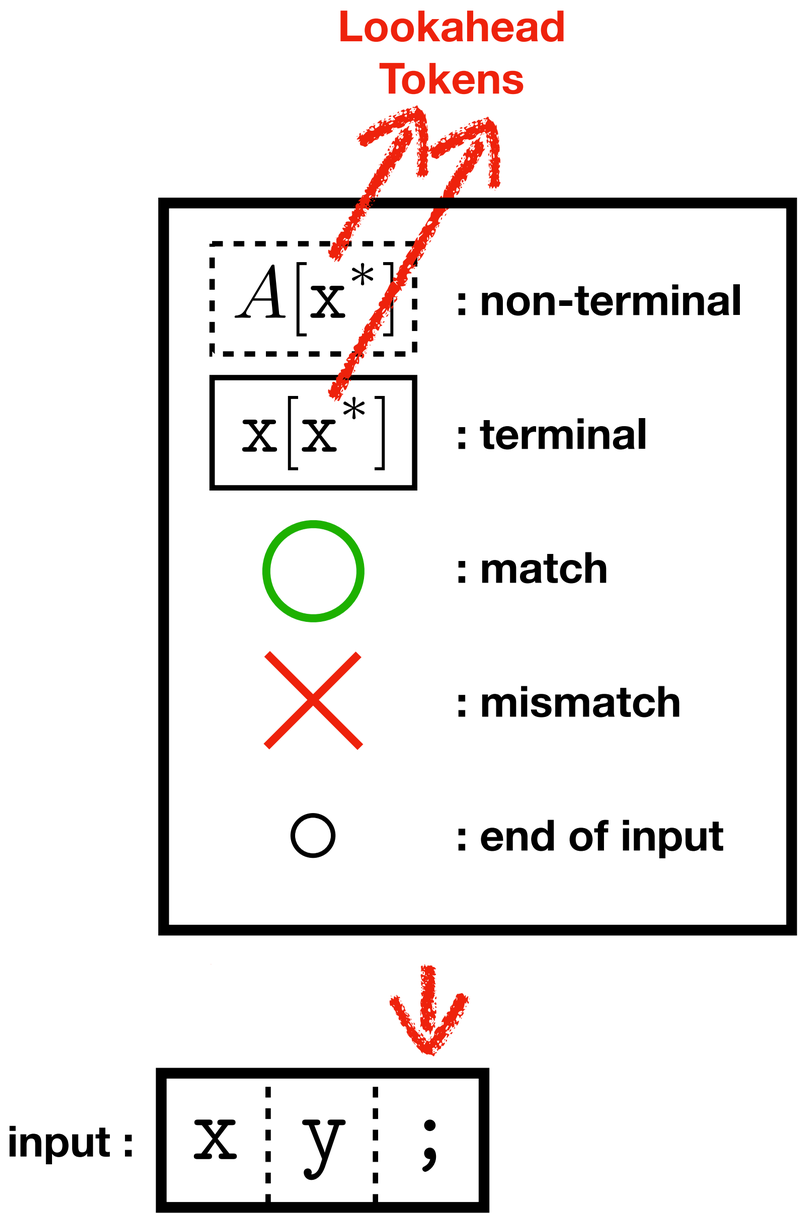


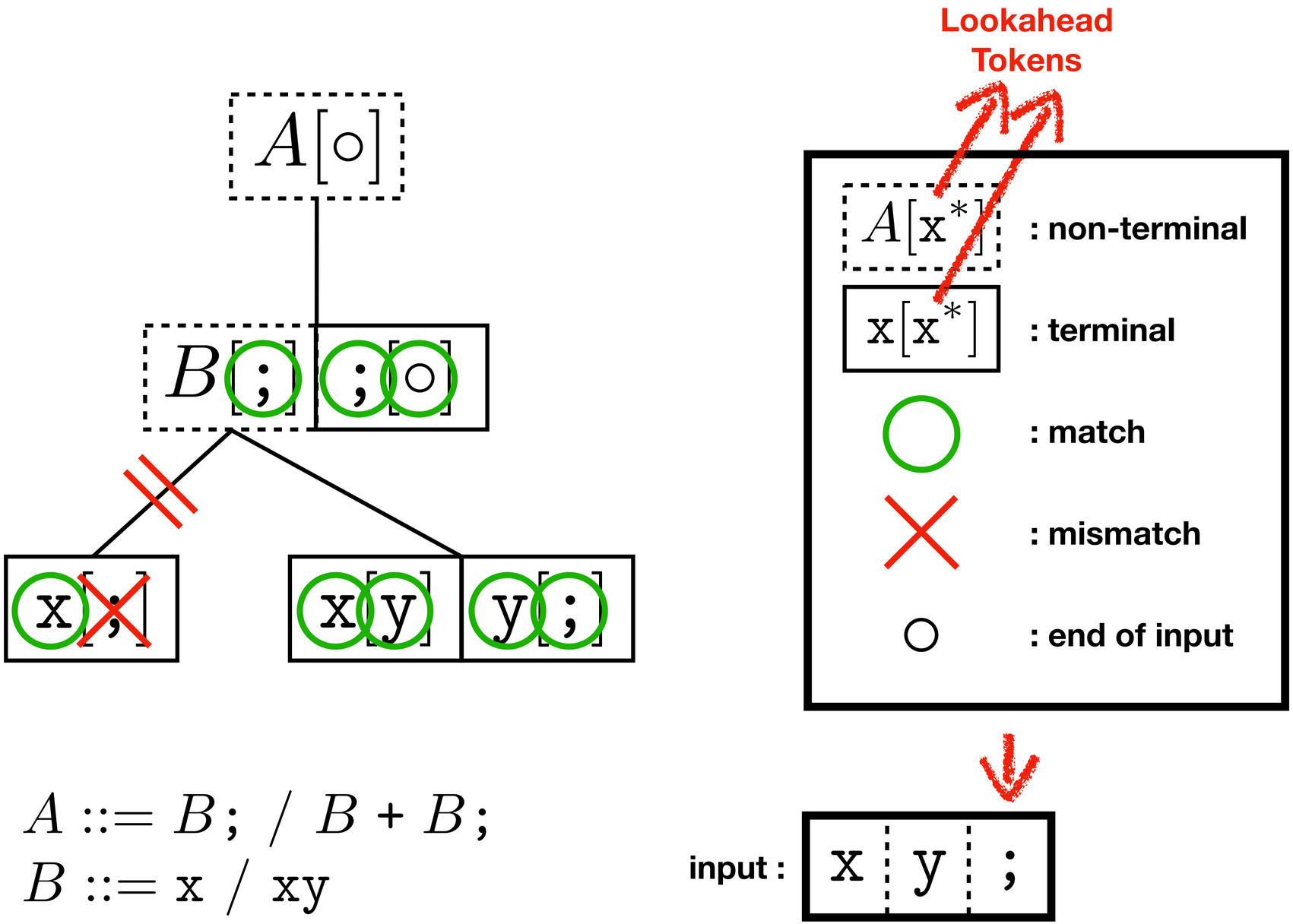




$A ::= B\,; \;/\; B + B\,;$

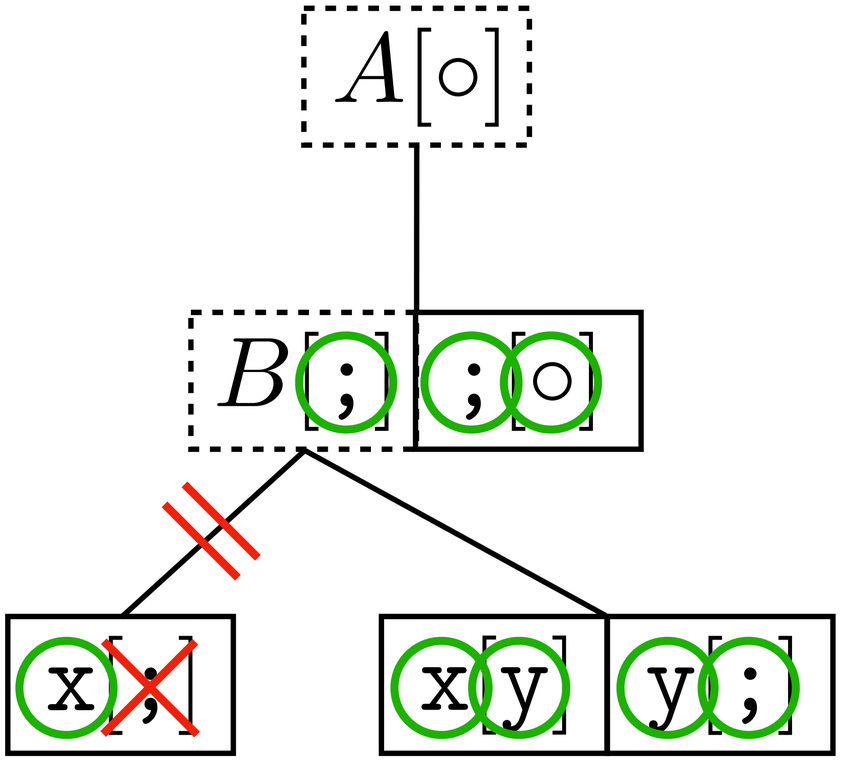
$B ::= \mathtt{x} \;/\; \mathtt{xy}$

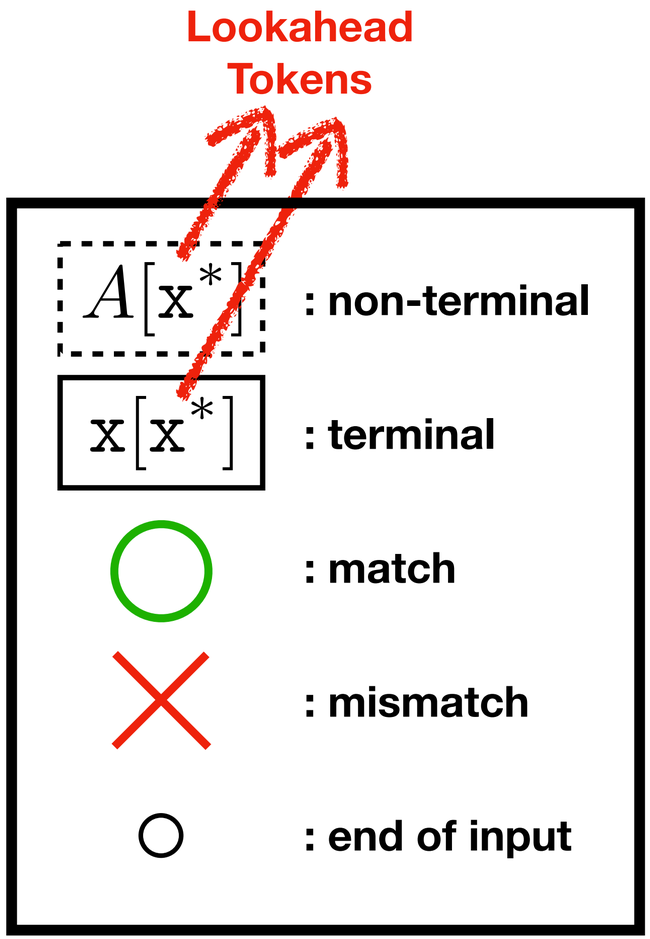




$A ::= B\,; \;/\; B + B\,;$

$B ::= \mathtt{x} \;/\; \mathtt{xy}$

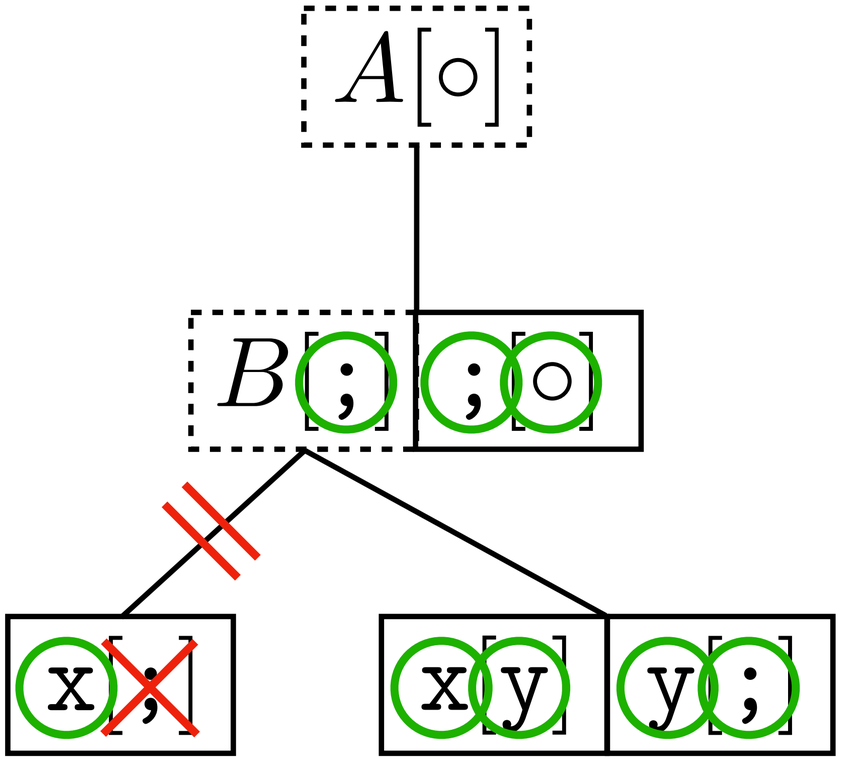


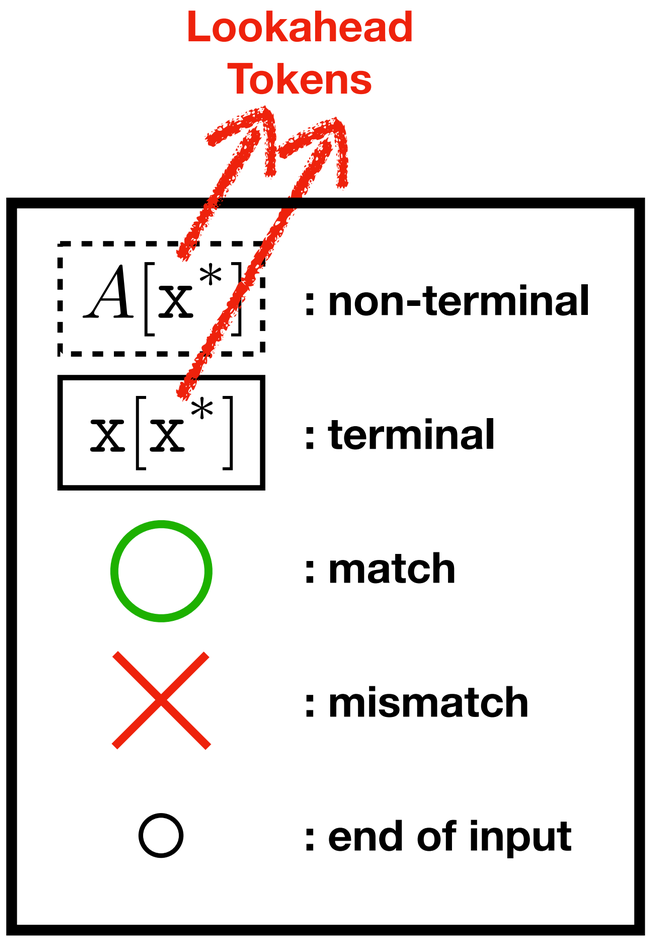


$A ::= B\,; \;/\; B + B\,;$

$B ::= \mathtt{x} \;/\; \mathtt{xy}$

input : x | y | ;





$A ::= B\,; \;/\; B + B\,;$

$B ::= \mathtt{x} \;/\; \mathtt{xy}$

input : x | y | ;

$$
\begin{aligned}
\mathbf{first}_\alpha(s_1 \cdots s_n) &= \mathbf{first}_s(s_1) :+ \mathbf{first}_s(s_2 \cdots s_n) \\
&\quad \text{where } x :+ y = \begin{cases} x \cup y & \text{if } \circ \in x \\ x & \text{otherwise} \end{cases} \\
\mathbf{first}_s(\epsilon) &= \{\circ\} \\
\mathbf{first}_s(\mathsf{a}) &= \{\mathsf{a}\} \\
\mathbf{first}_s(A(a_1, \cdots, a_k)) &= \mathbf{first}_\alpha(\alpha_1) \cup \cdots \cup \mathbf{first}_\alpha(\alpha_n) \\
&\quad \text{where } A(a_1, \cdots, a_k) = \alpha_1 \mid \cdots \mid \alpha_n \\
\mathbf{first}_s(s?) &= \mathbf{first}_s(s) \cup \{\circ\} \\
\mathbf{first}_s(+s) &= \mathbf{first}_s(s) \\
\mathbf{first}_s(-s) &= \{\circ\} \\
\mathbf{first}_s(s \setminus s') &= \mathbf{first}_s(s) \\
\mathbf{first}_s(\langle \neg \mathsf{LT} \rangle) &= \{\circ\}
\end{aligned}
$$

Algorithm for first tokens of BNF$_{\text{ES}}$

Algorithm for lookahead parsing

$$
\begin{aligned}
(s_1 \cdots s_n)[L] &= s_1[\mathbf{first}_s(s_2 \cdots s_n) :+ L]\ (s_1 \cdots s_n)[L] \\
\epsilon[L] &= +\mathbf{get}_s(L) \\
\mathsf{a}[L] &= \mathsf{a}\ + \mathbf{get}_s(L) \\
A(a_1, \cdots, a_k)[L] &= \alpha_1[L] \mid \cdots \mid \alpha_n[L] \\
&\quad \text{where } A(a_1, \cdots, a_k) = \alpha_1 \mid \cdots \mid \alpha_n \\
s?[L] &= s[L] \mid \epsilon[L] \\
(\pm s)[L] &= \pm(s[L]) \\
(s \setminus s')[L] &= s[L] \setminus s' \\
\langle \neg \mathsf{LT} \rangle &= \langle \neg \mathsf{LT} \rangle\ + \mathbf{get}_s(L)
\end{aligned}
$$

?

“Can I ask, are you using some automated tooling to find these, or just checking manually?”

- Kevin Gibbons, An Editor of ECMA-262

ecma INTERNATIONAL

?

"Can I ask, are you using some automated tooling to find these, or just checking manually?"

- Kevin Gibbons, An Editor of ECMA-262

!

"Would you and your collaborators be able to (virtually) attend a TC39 meeting to present your work to the committee? We can reserve a session for you as invited experts."

- Michael Ficarra, An Editor of ECMA-262

TC 39

"Yeah, first of all, I want to, I can hardly express how amazing this work is, this is really impressive. I sat through the presentation with my mouth open the whole time. So thank you very much."

"First, this is truly amazing work. My mind is blown. I tried to get screenshots, just to remember the slides and then was just taking screenshots of every slide. So I stopped."

"I think this was an excellent presentation. In terms of committee feedback, what you're hearing here, this is the committee in ecstatic mode. This is, this is the maximum that I've heard in terms of positive feedback for a presentation. So, so thank you very much."