PLRG
Programming Language
Research Group

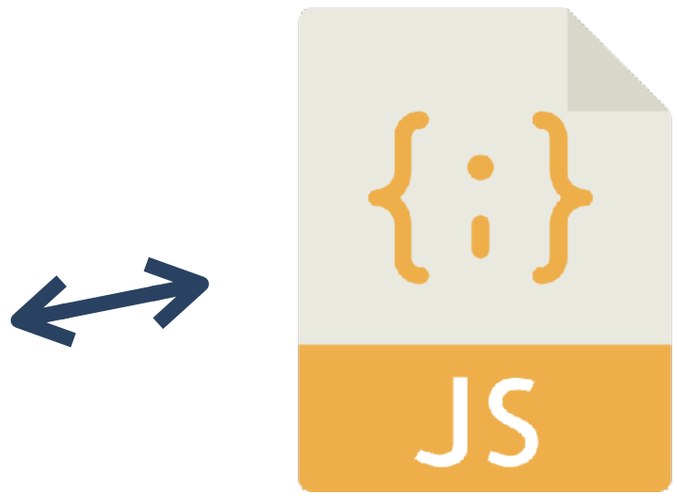
# JISET: JavaScript IR-based Semantics Extraction Toolchain

The 35th IEEE/ACM International Conference on
Automated Software Engineering (ASE'20)

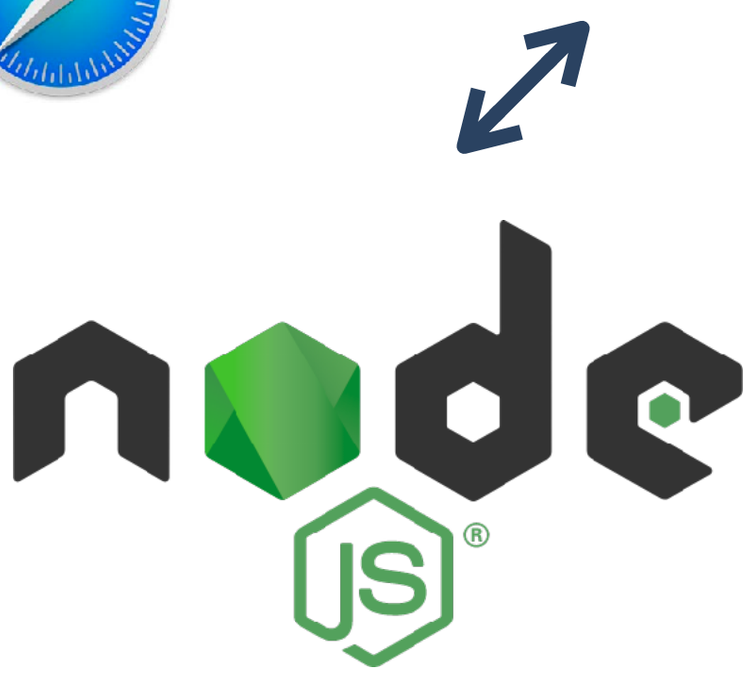
**Jihyeok Park**, Jihee Park, Seungmin An, Sukyoung Ryu

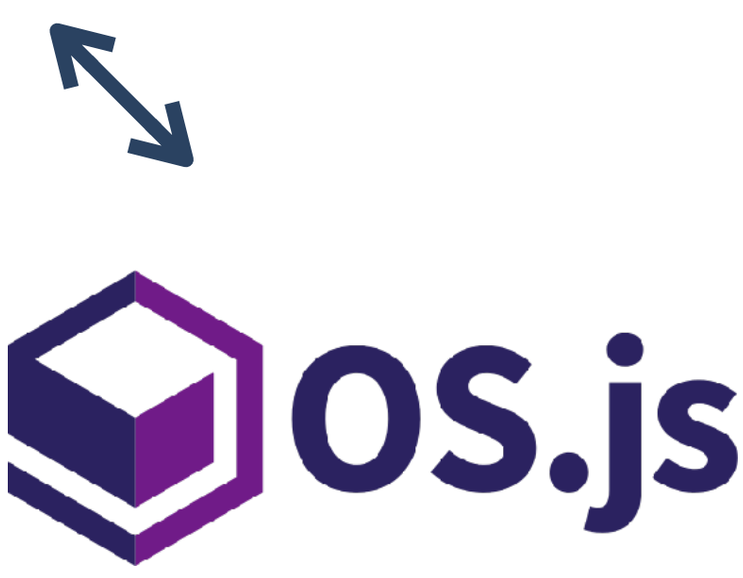
PLRG @ KAIST
September 23, 2020

# JavaScript in Broad Fields

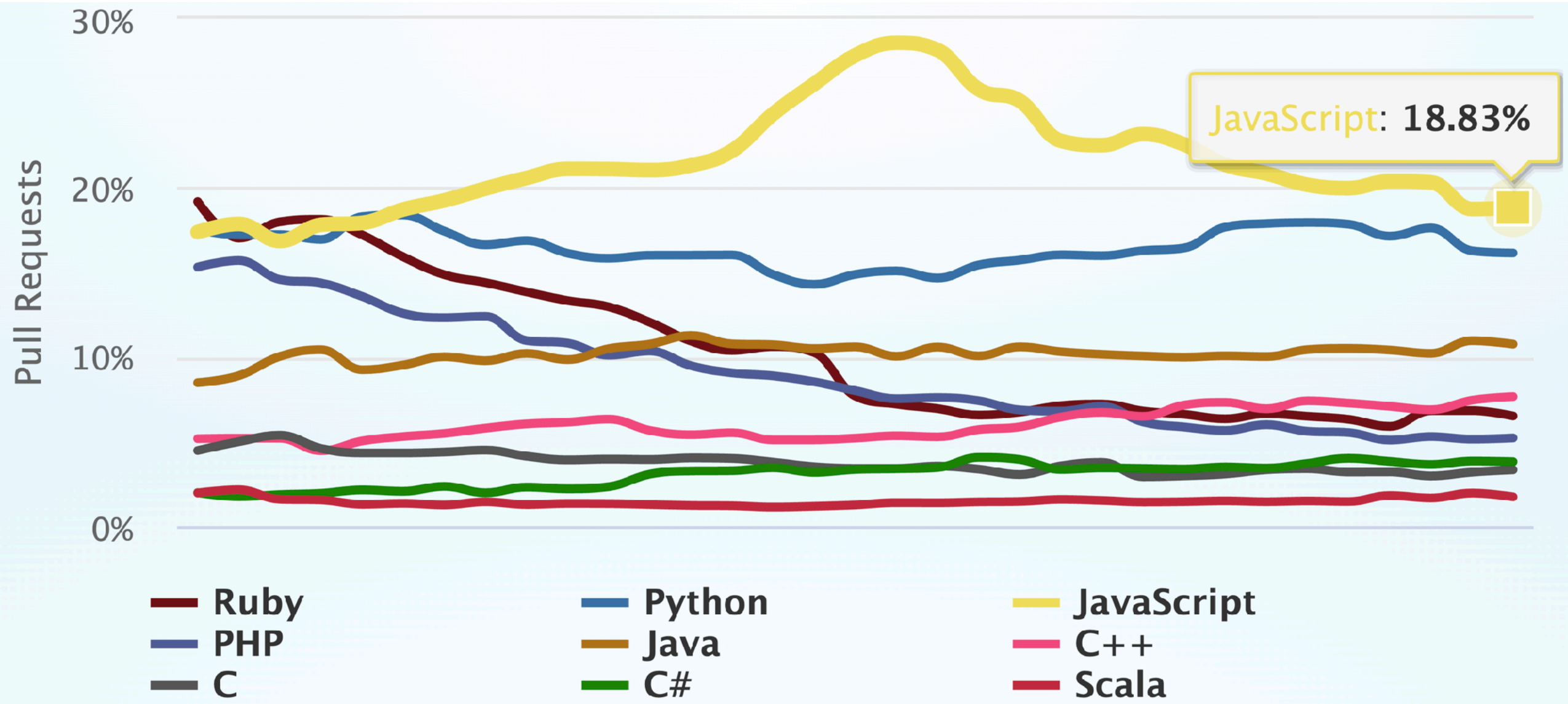






# JavaScript is Most Popular




https://madnight.github.io/githut/

# JavaScript Complex Semantics

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

Always return `false`?

## NO!!

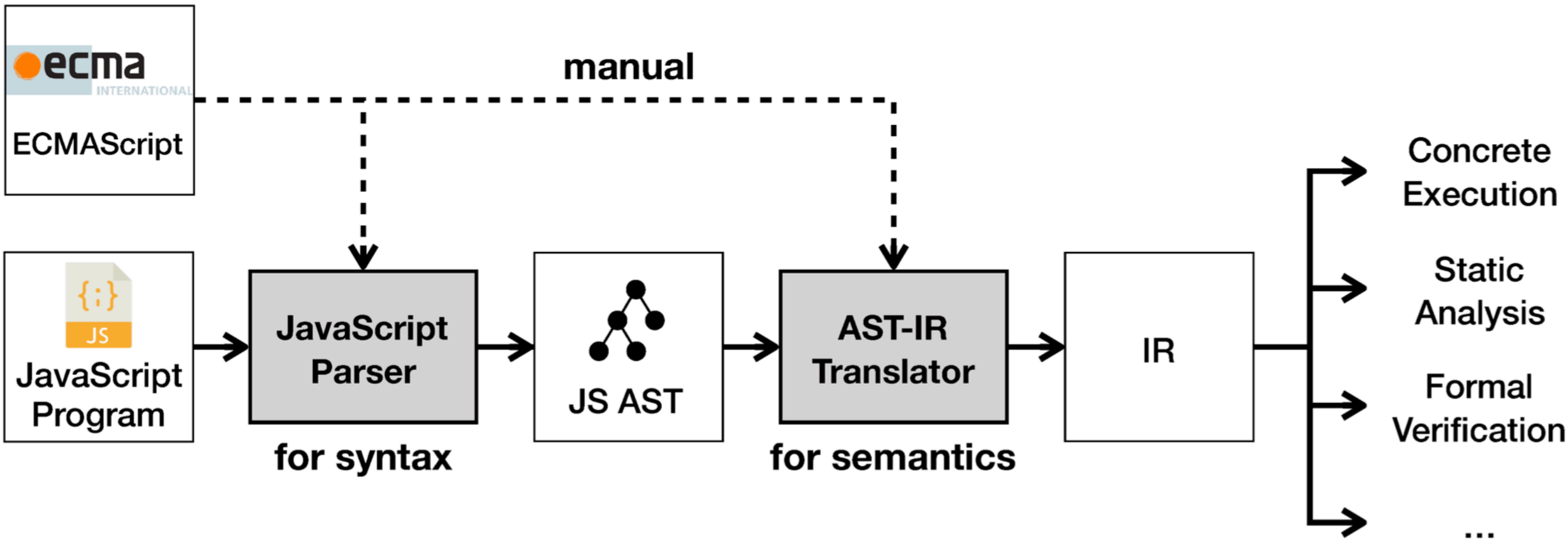
```
f([]) -> [] == ![]
      -> [] == false
      -> +[] == +false
      -> 0 == 0
      -> true
```

# ECMAScript - Spec. of JavaScript


The standard for JavaScript is ECMAScript. As of 2012, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3. On June 17, 2015, ECMA International published the sixth major version of ECMAScript, which is officially called ECMAScript 2015, and was initially referred to as ECMAScript 6 or ES6.

**https://developer.mozilla.org/en-US/docs/Web/JavaScript**

# IR-based Semantics Extraction



# IR-based Semantics Extraction

*ArrayLiteral*`[Yield, Await]` :

[ *Elision*$_{opt}$ ]

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

[ *ElementList*`[?Yield, ?Await]` , *Elision*$_{opt}$ ]

The *ArrayLiteral* production in ES10

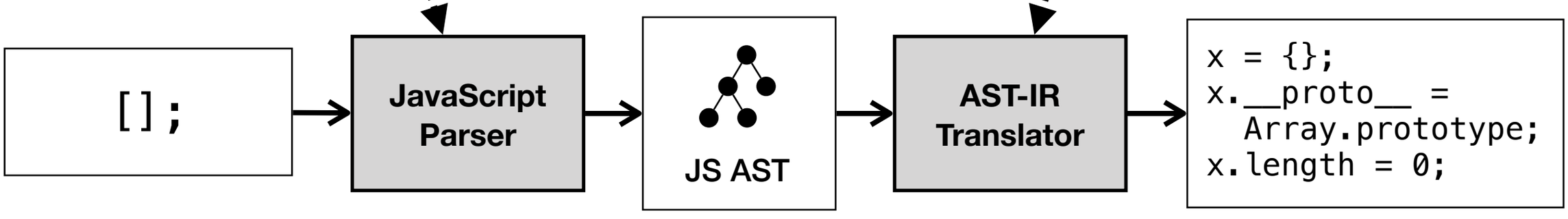
**12.2.5.3 Runtime Semantics: Evaluation**

*ArrayLiteral* : [ *Elision* ]

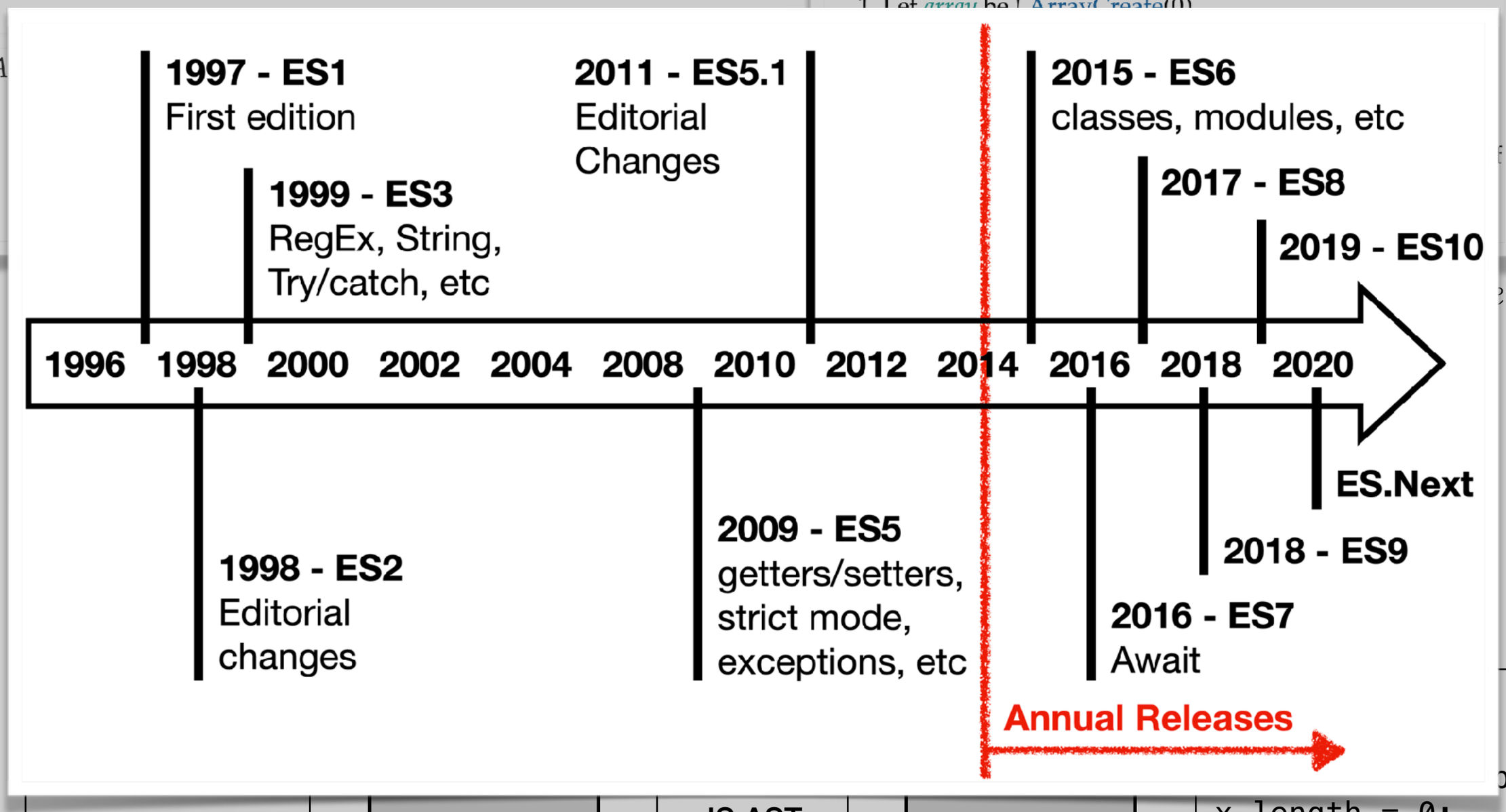
1. Let *array* be ! ArrayCreate(0).
2. Let *pad* be the ElisionWidth of *Elision*; if *Elision* is not present, use the numeric value zero.
3. Perform Set(*array*, **"length"**, ToUint32(*pad*), **false**).
4. NOTE: The above Set cannot fail because of the nature of the object returned by ArrayCreate.
5. Return *array*.

The semantics of the first alternative for *ArrayLiteral*

MANUALLY implemented



# IR-based Semantics Extraction




JS AST

# Core Idea

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

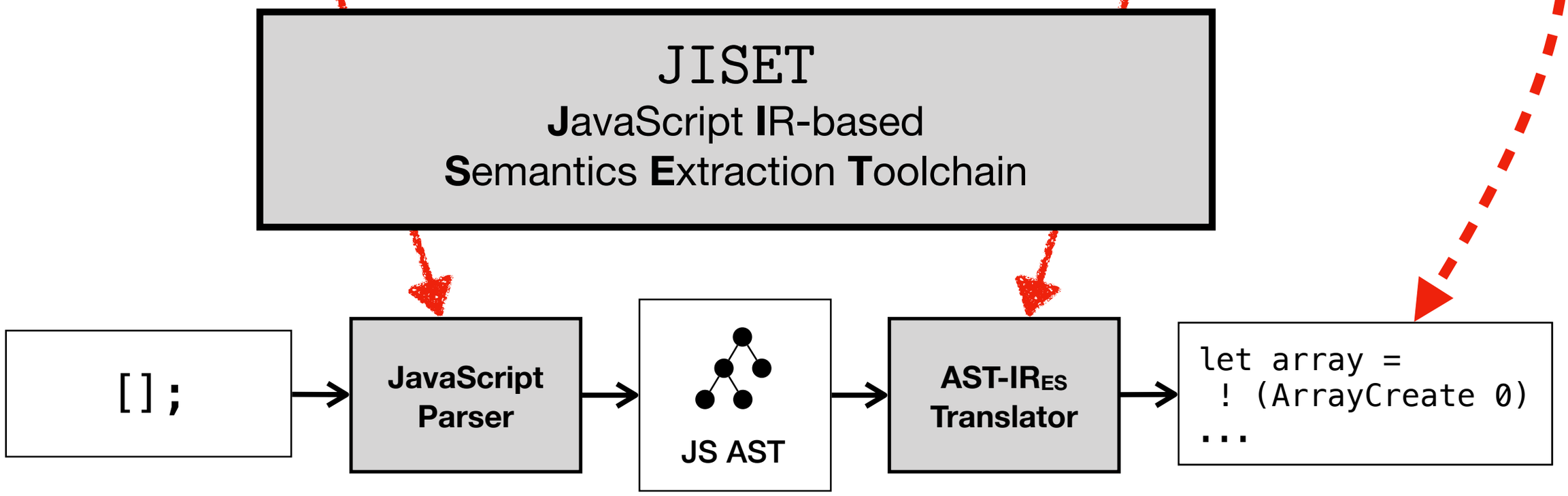
The *ArrayLiteral* production in ES10

**12.2.5.3 Runtime Semantics: Evalu**
*ArrayLiteral* : [ *Elision* ]

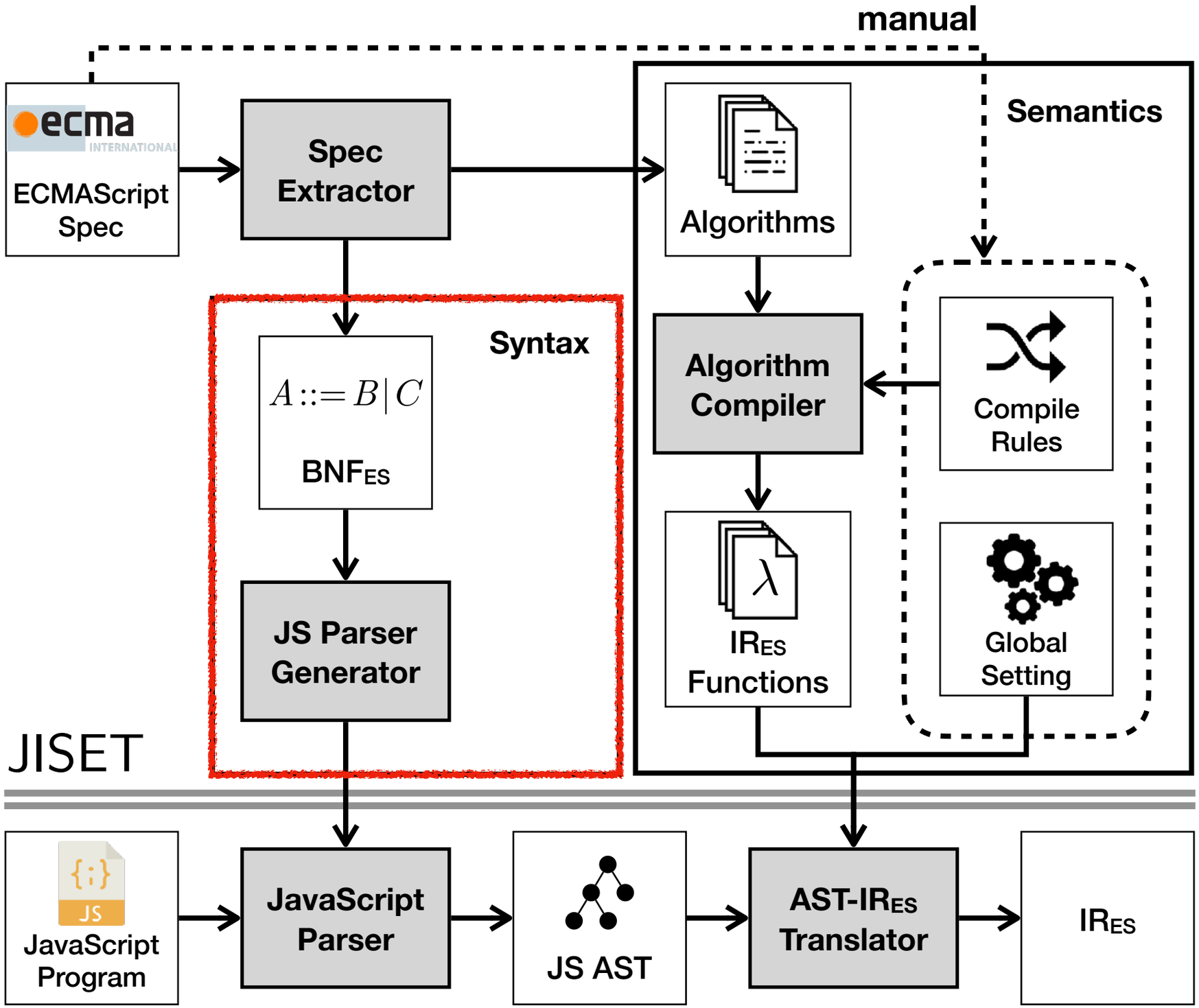
1. Let *array* be ! ArrayCreate(0).
2. Let *pad* be the ElisionWidth of *Elision*; if *Elision* is not present, use the numeric value zero.
3. Perform Set(*array*, "length", ToUint32(*pad*), **false**).
4. NOTE: The above Set cannot fail because of the nature of the object returned by ArrayCreate.
5. Return *array*.

The semantics of the first alternative for *ArrayLiteral*

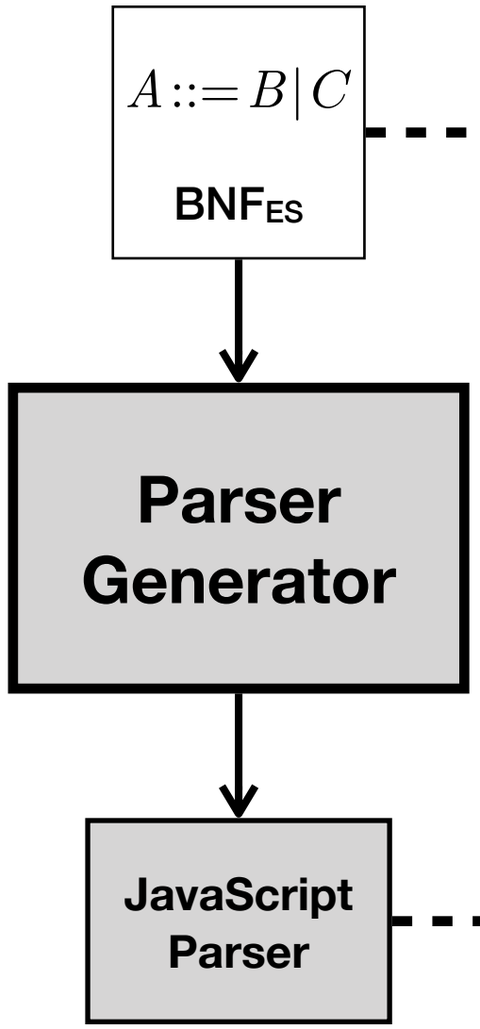
**IRES (Intermediate Rep. for ES)**



# Overall Structure of JISET



# JS Parser Synthesis




$ArrayLiteral_{[Yield,\ Await]}$ :

[ $Elision_{opt}$ ]

[ $ElementList_{[?Yield,\ ?Await]}$ ]

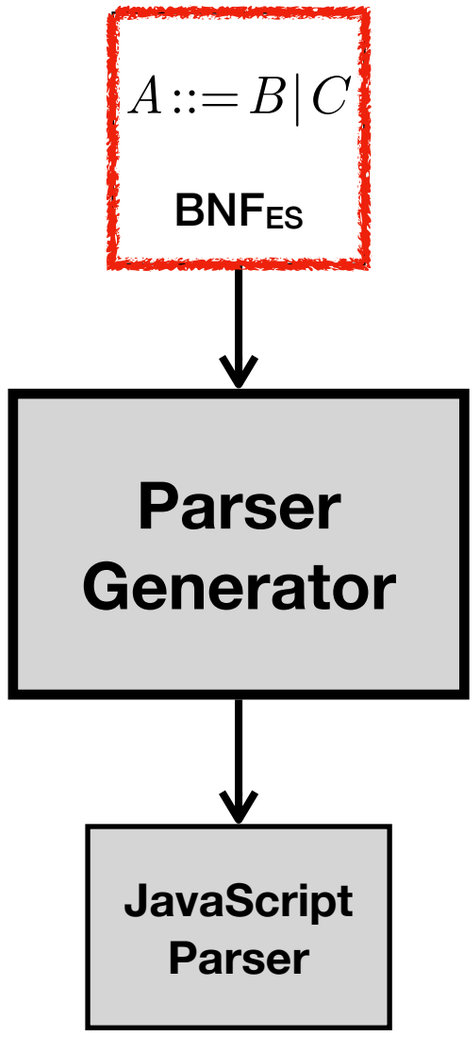
[ $ElementList_{[?Yield,\ ?Await]}$ , $Elision_{opt}$ ]

The $ArrayLiteral$ production in ES10

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

The generated parser for $ArrayLiteral$

# $\text{BNF}_{\text{ES}}$ - BNF for ECMAScript




**Productions**

$$A(p_1, \cdots, p_k) ::= (c_1 \Rightarrow)^? \alpha_1 \mid \cdots \mid (c_n \Rightarrow)^? \alpha_n$$

where $p_i$ is a boolean parameter.

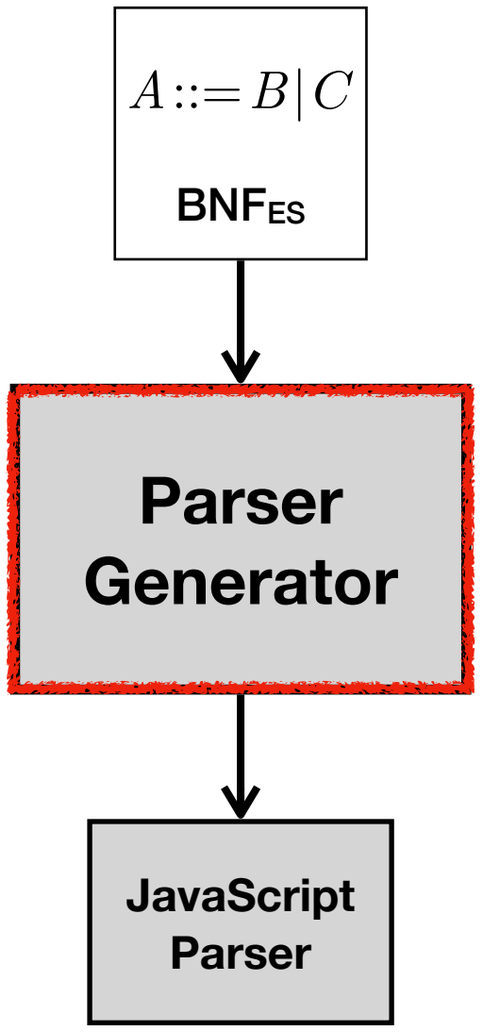
**Conditions**

$$c ::= p_i \mid !p_i$$

**Symbols**

| Symbol $s$ | Description |
|---|---|
| $\epsilon$ | empty sequnce |
| `a` | terminal |
| $A(a_1, \cdots, a_k)$ | non-terminal |
| $s^?$ | optional symbol |
| $+s$ | positive lookahead |
| $-s$ | negative lookahead |
| $s \setminus s'$ | exclusion |
| $\langle \neg \texttt{LT} \rangle$ | no line-terminator |

# Parser Generator




## Parsing Expression Grammar (PEG)

+ Human-Readable Parsers
+ Easy to Support BNF$_{ES}$ Features
+ Linear Parsing Time

~~- Different with BNF$_{ES}$ (∵ Ordered Choices)~~

+

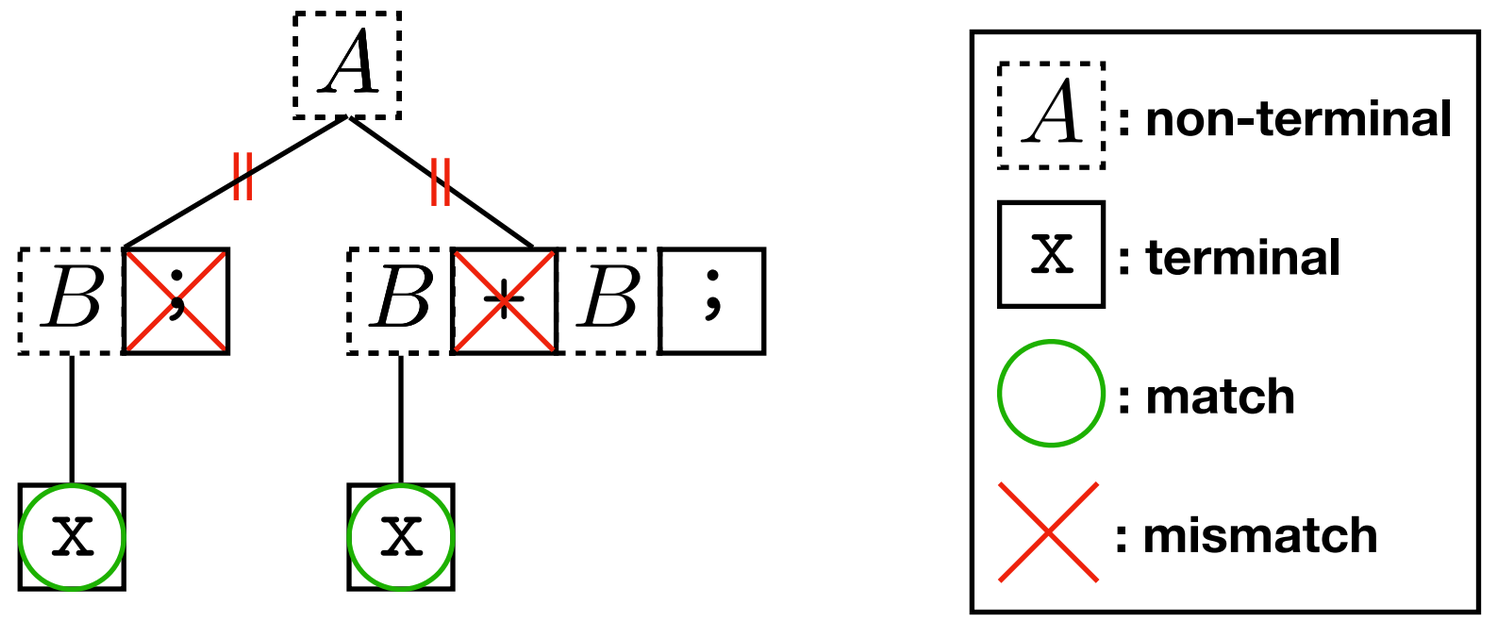
## Lookahead Parsing

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

**(ICFP'02) Bryan Ford, "Packrat parsing: simple, powerful, lazy, linear time, functional pearl"**

# Parsing Expression Grammar

- **Ordered Choices ( $A / B$ )**



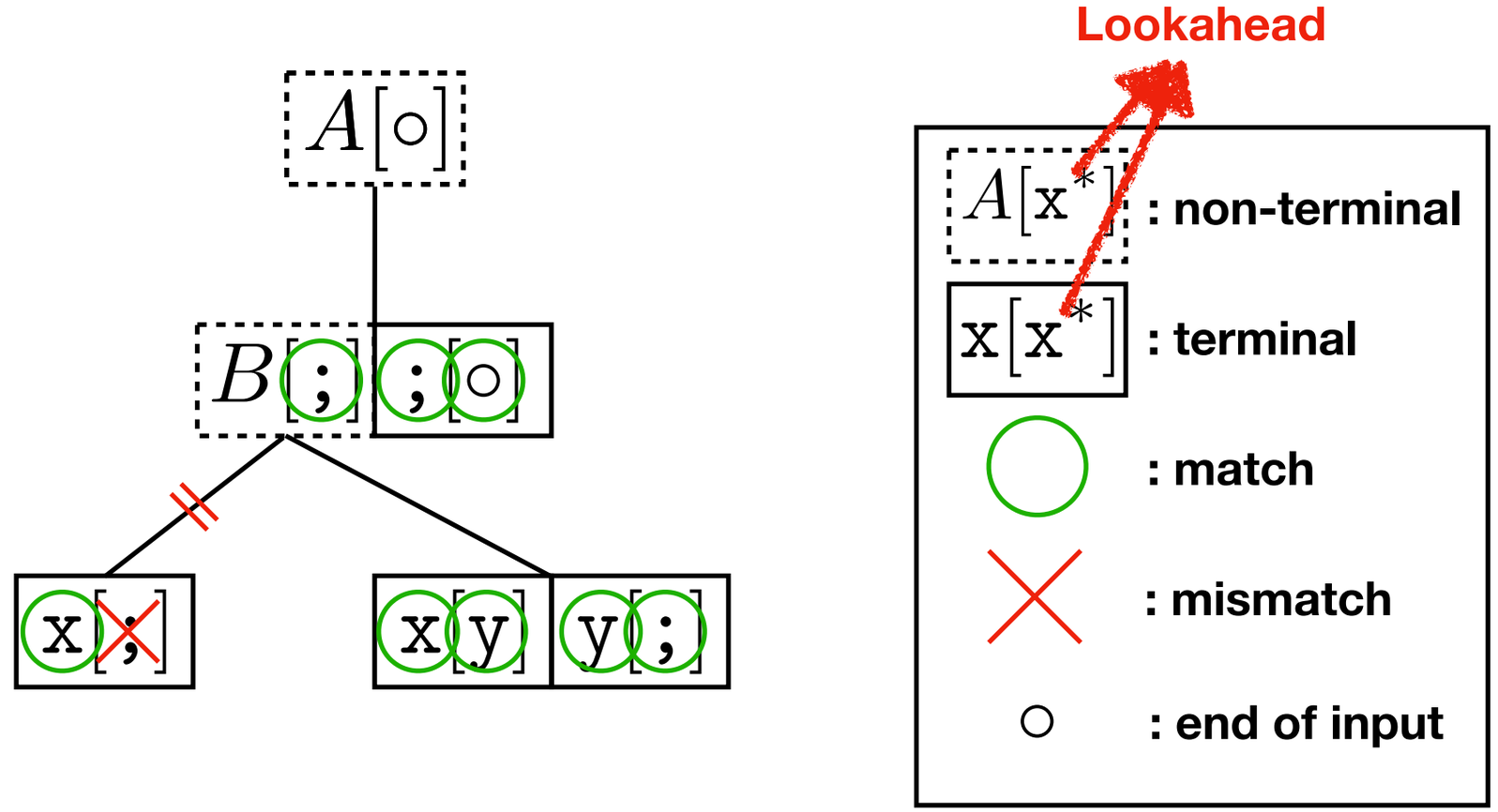

$$A ::= B\,; \;/\; B + B\,;$$
$$B ::= \mathtt{x} \;/\; \mathtt{xy}$$

**Always ignored**

**Input :** xy;

**Unable to parse**

# Lookahead Parsing




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

**Input :** `xy;`

# Lookahead Parsing

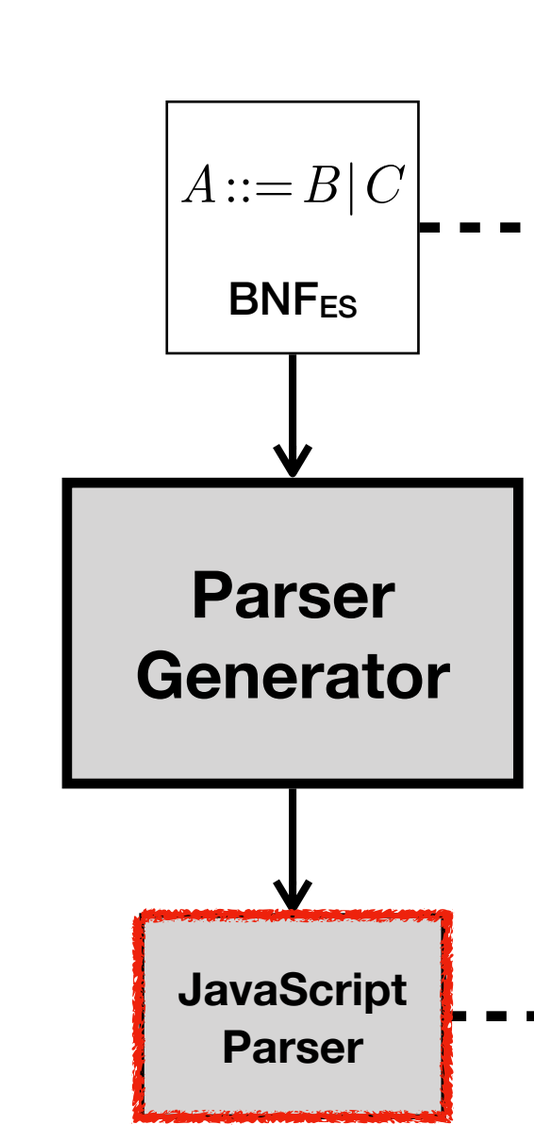
$$
\begin{aligned}
\mathbf{first}_\alpha(s_1 \cdots s_n) &= \mathbf{first}_s(s_1) :+ \mathbf{first}_s(s_2 \cdots s_n) \\
&\quad \text{where } x :+ y = \begin{cases} x \cup y & \text{if } \circ \in x \\ x & \text{otherwise} \end{cases} \\
\mathbf{first}_s(\epsilon) &= \{\circ\} \\
\mathbf{first}_s(\mathsf{a})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

**Algorithm for first tokens of BNF$_{ES}$**

**Algorithm for lookahead parsing**

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

# Implementation




$ArrayLiteral_{[Yield, Await]}$ :

[ $Elision_{opt}$ ]

[ $ElementList_{[?Yield, ?Await]}$ ]

[ $ElementList_{[?Yield, ?Await]}$ , $Elision_{opt}$ ]

The $ArrayLiteral$ production in ES10

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

The generated parser for $ArrayLiteral$

# Evaluation - Syntax

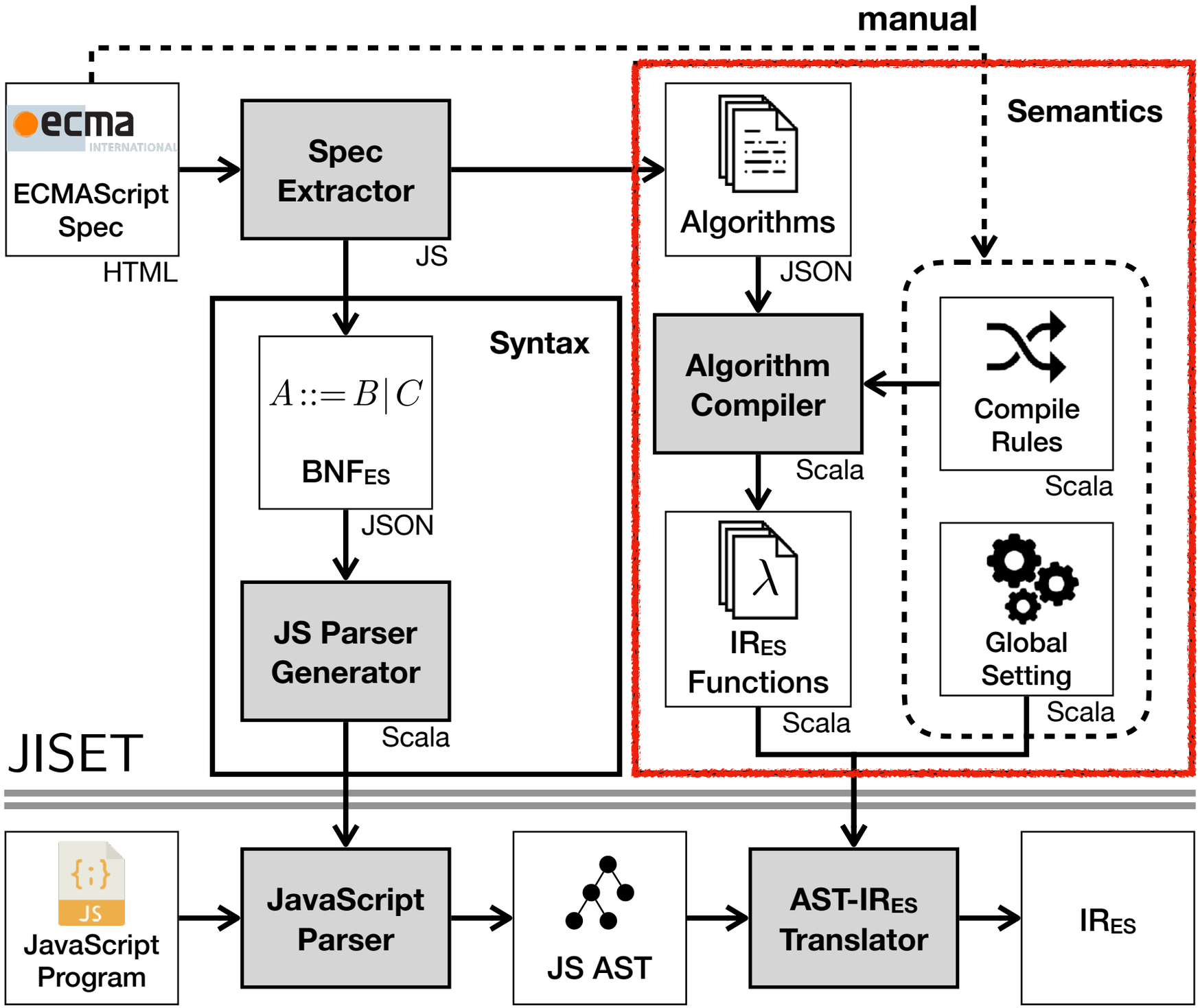
All Success!!

| Version | ES7 | ES8 | ES9 | ES10 | Average |
|---|---|---|---|---|---|
| # Lexical productions | 78 | 78 | 78 | 81 | 78.75 |
| # Syntactic productions | 157 | 167 | 167 | 174 | 166.25 |

Test with JS programs
in Test262

| Old version / New version | ES7 / ES8 | ES8 / ES9 | ES9 / ES10 | Average |
|---|---|---|---|---|
| Δ # Lexical productions | 3 | 5 | 6 | 4.67 |
| Δ # Syntactic productions | 140 | 15 | 8 | 54.33 |

# Overview of JISET



# JS Semantics Extraction

**12.2.5.3 Runtime Semantics: Evaluation**

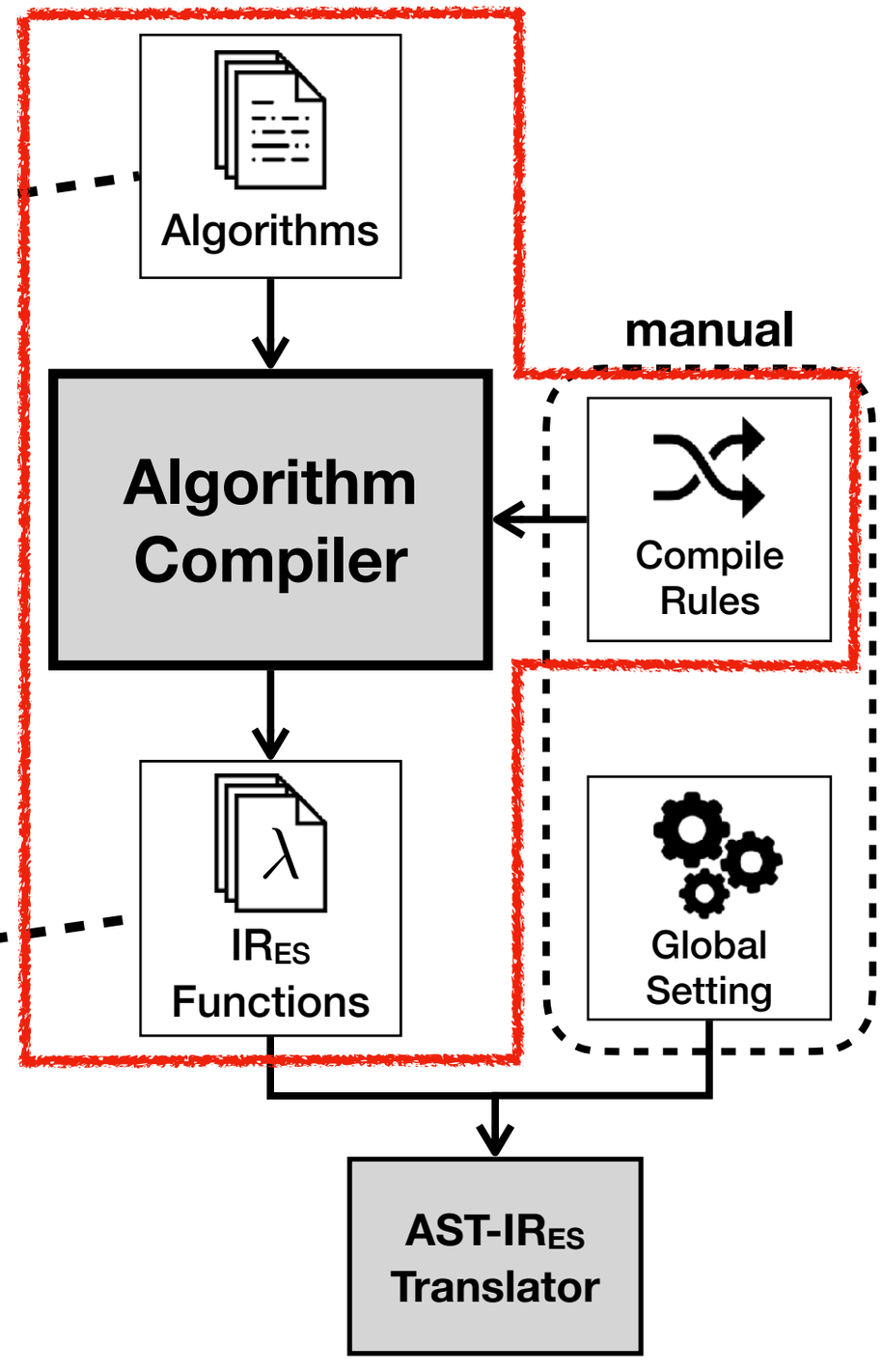
*ArrayLiteral* : [ *Elision* ]

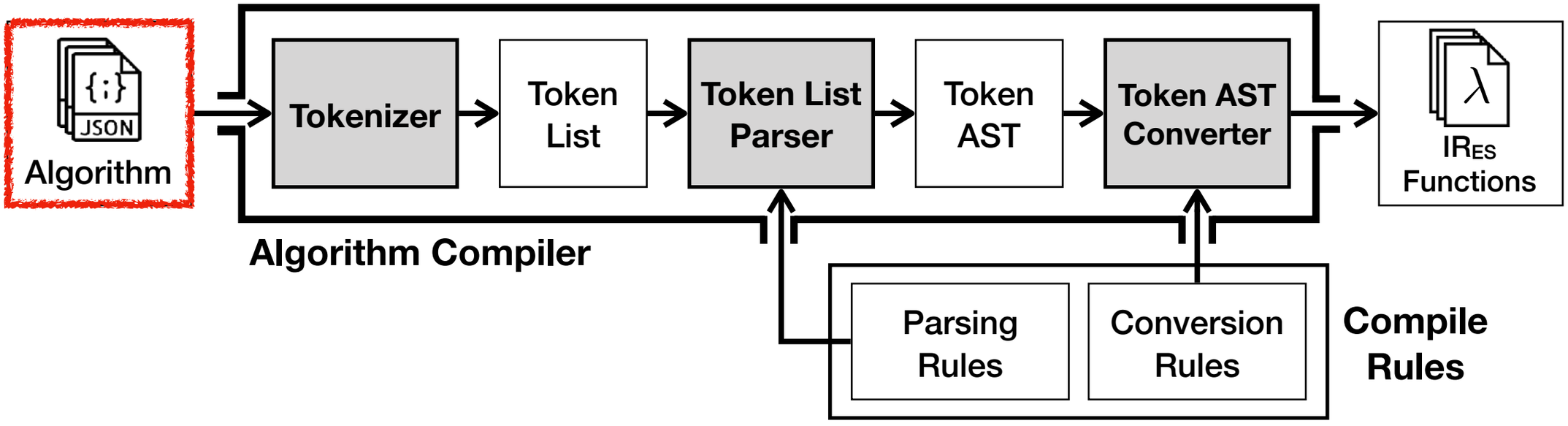
1. Let *array* be ! ArrayCreate(0).
2. Let *pad* be the ElisionWidth of *Elision*; if *Elision* is not present, use the numeric value zero.
3. Perform Set(*array*, "length", ToUint32(*pad*), **false**).
4. NOTE: The above Set cannot fail because of the nature of the object returned by ArrayCreate.
5. Return *array*.

The semantics of the first alternative for $ArrayLiteral$

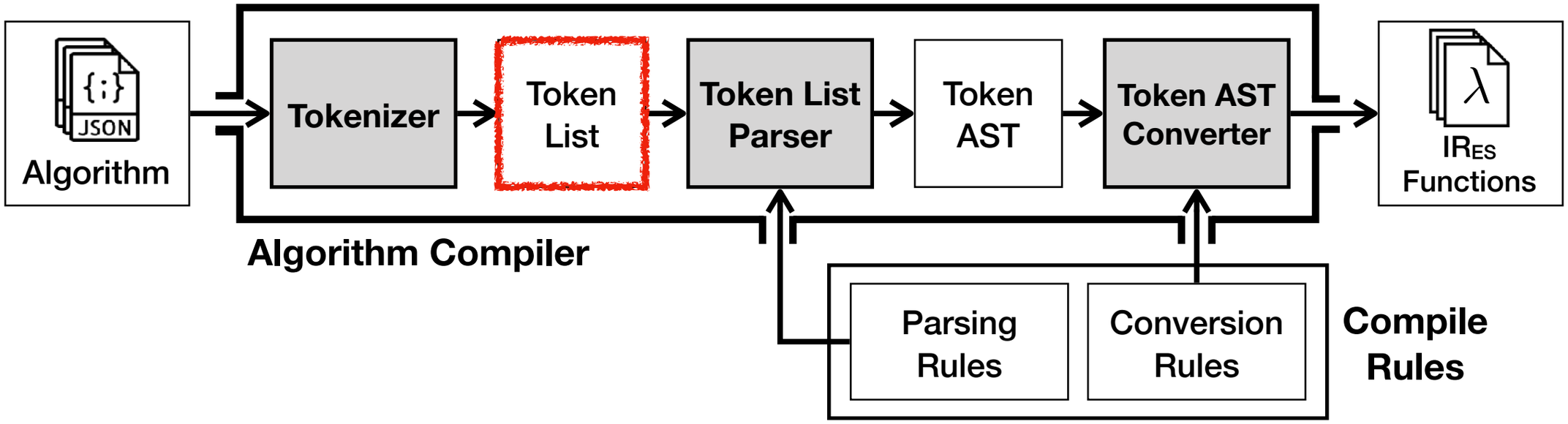
```
"ArrayLiteral0.Evaluation" (Elision) => {
  let array = ! (ArrayCreate 0)
  if (= Elision absent) let pad = 0
  else let pad = Elision.ElisionWidth
  (Set array "length" (ToUint32 pad) false)
  return array
}
```

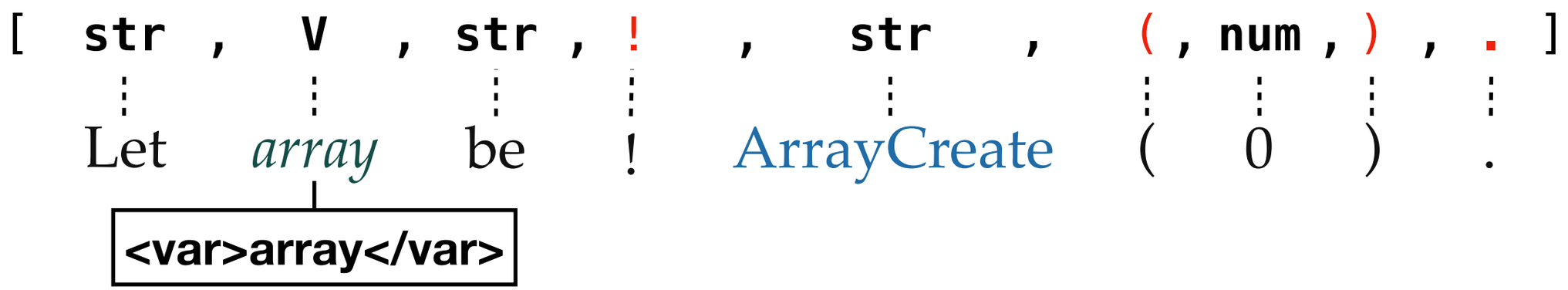
An $IR_{ES}$ function of the first alternative for $ArrayLiteral$

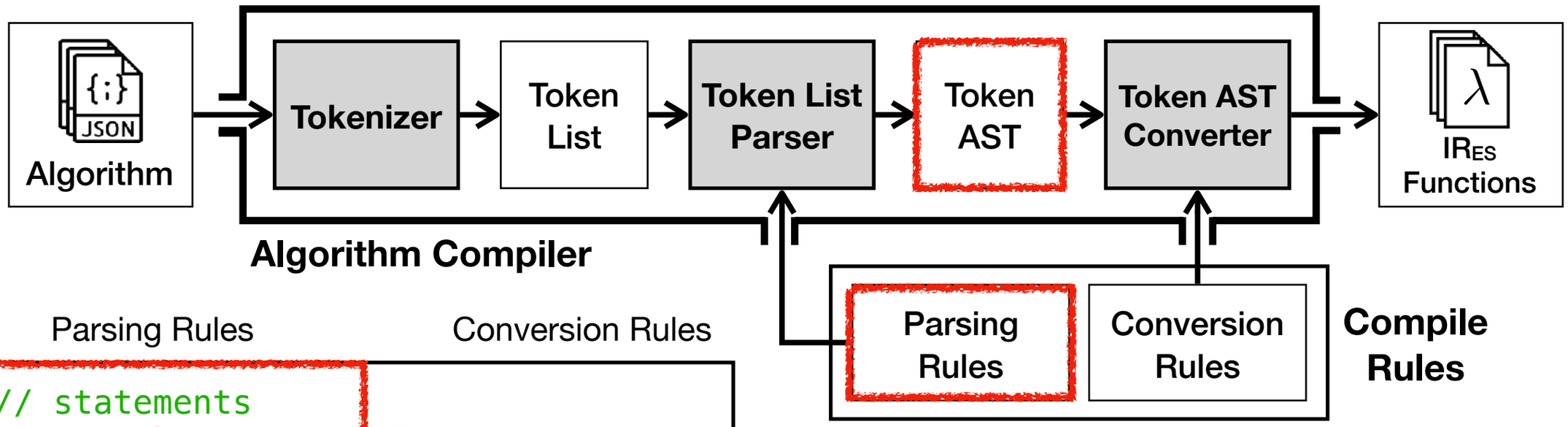





Let *array* be ! ArrayCreate ( 0 ) .


Algorithm
JSON
Tokenizer
Token List
Token List Parser
Token AST
Token AST Converter
IRES Functions
Algorithm Compiler
Parsing Rules
Conversion Rules
Compile Rules


[ str , V , str , ! , str , ( , num , ) , . ]
Let array be ! ArrayCreate ( 0 ) .
<var>array</var>



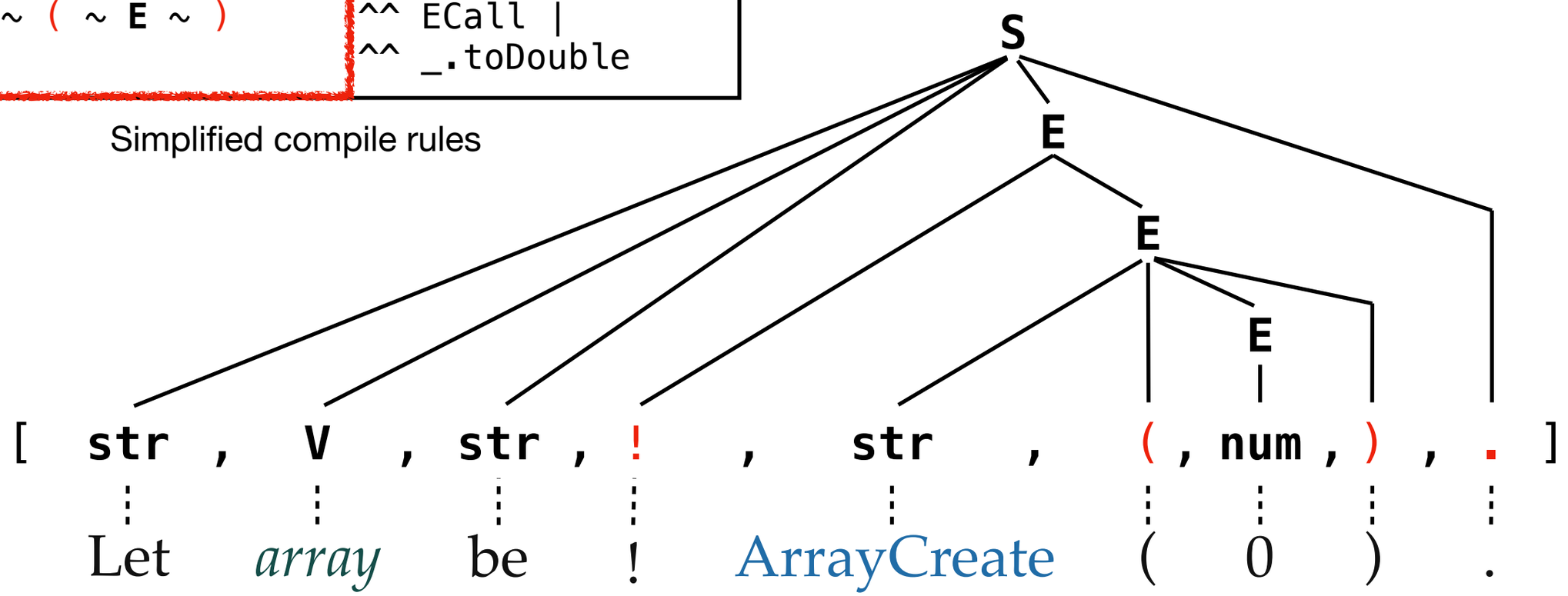

Parsing Rules

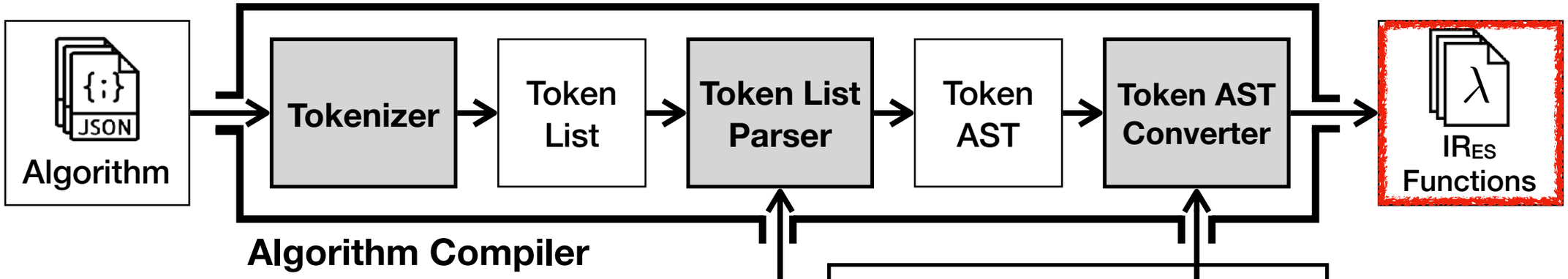
```
S = // statements
  Let ~ V ~ be ~ E ~ .

E = // expressions
  ! E
  str ~ ( ~ E ~ )
  num
```

Conversion Rules

```
^^ ILet


^^ EAbruptCheck |
^^ ECall |
^^ _.toDouble
```

Simplified compile rules





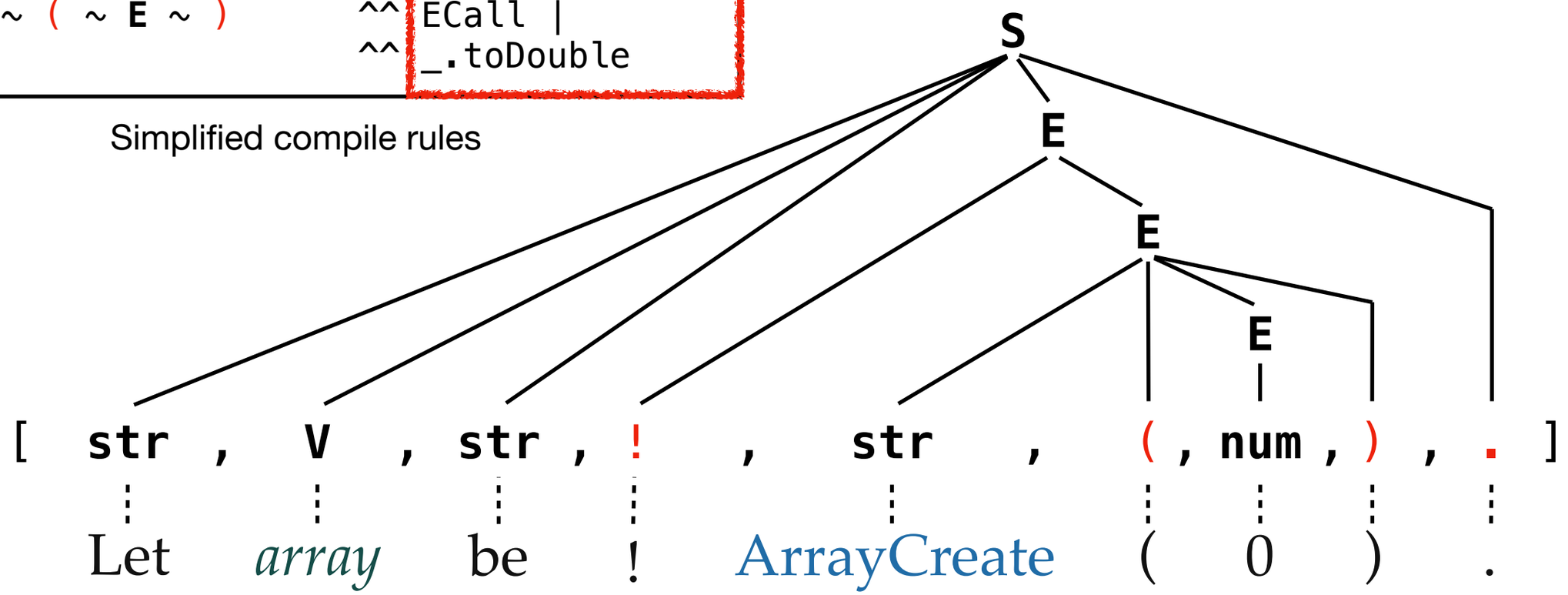

Parsing Rules

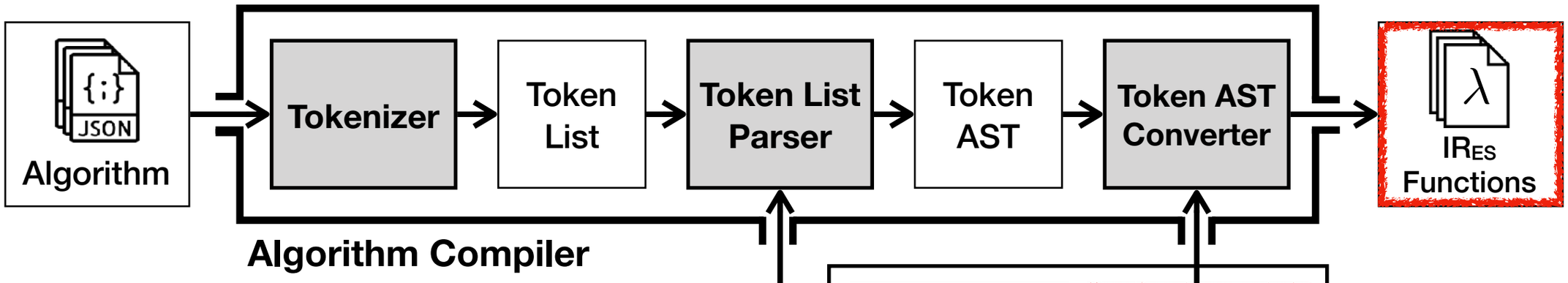
```
S = // statements
  Let ~ V ~ be ~ E ~ .
E = // expressions
  ! E
  str ~ ( ~ E ~ )
  num
```

Conversion Rules

```
^^ ILet

^^ EAbruptCheck |
^^ ECall |
^^ _.toDouble
```

Simplified compile rules






Parsing Rules

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

```
E = // expressions
  ! E                  ^^
  str ~ ( ~ E ~ )      ^^
  num                  ^^
```

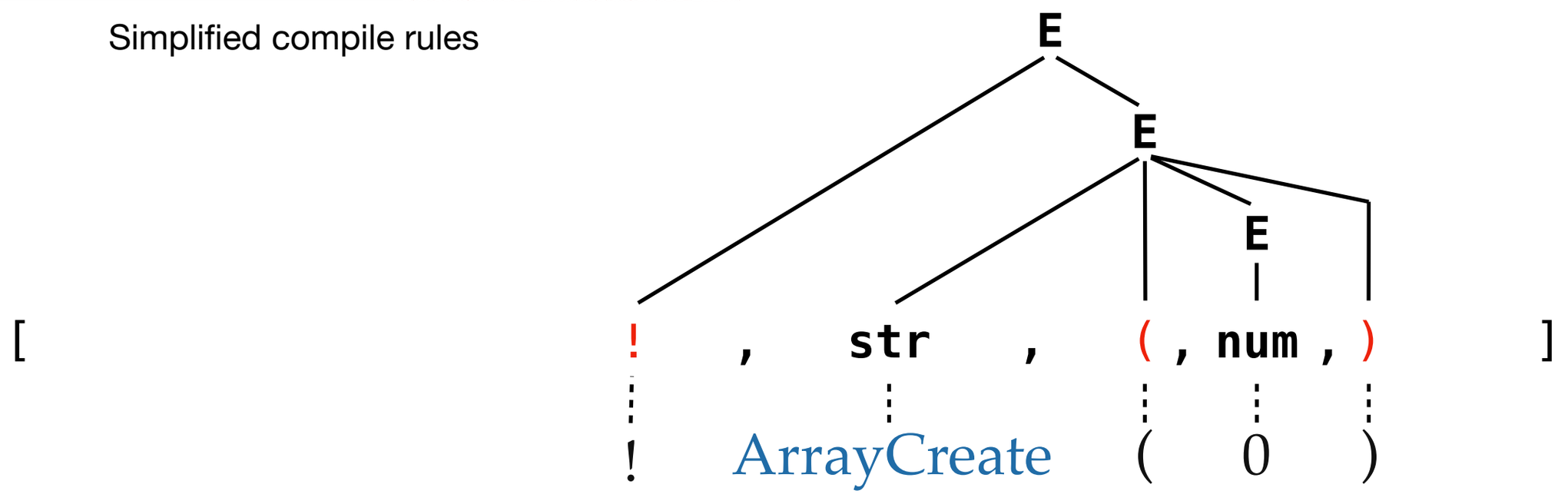
Conversion Rules

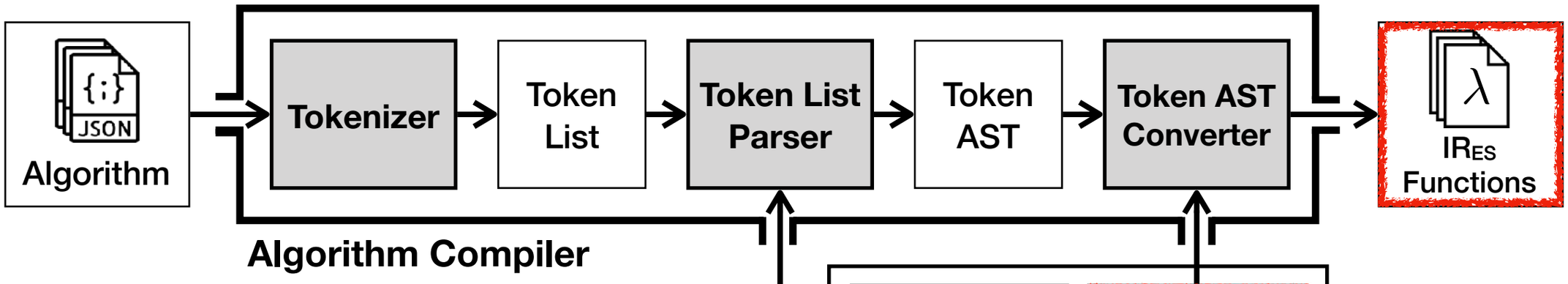
```
ILet

EAbruptCheck |
ECall |
_.toDouble
```

```
ILet(array,
```

```
)
```

Simplified compile rules






Parsing Rules

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

E = // expressions
  ! E                  ^^
  str ~ ( ~ E ~ )      ^^
  num                  ^^
```

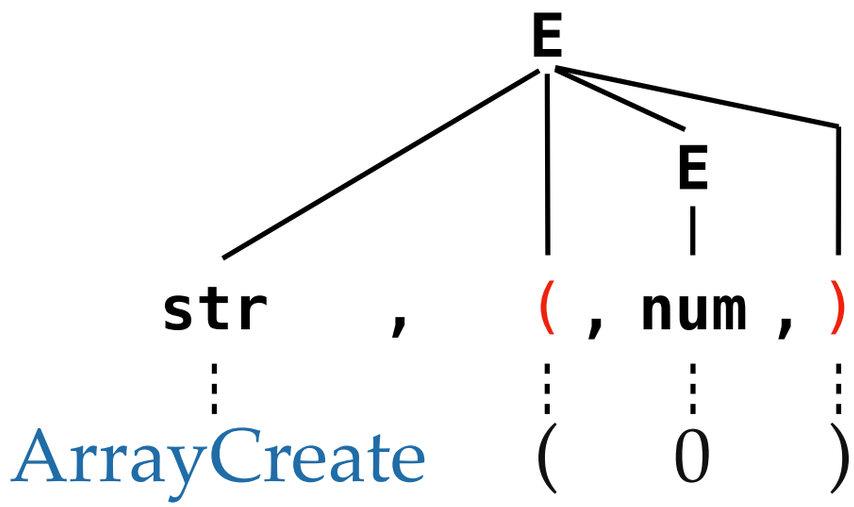
Conversion Rules

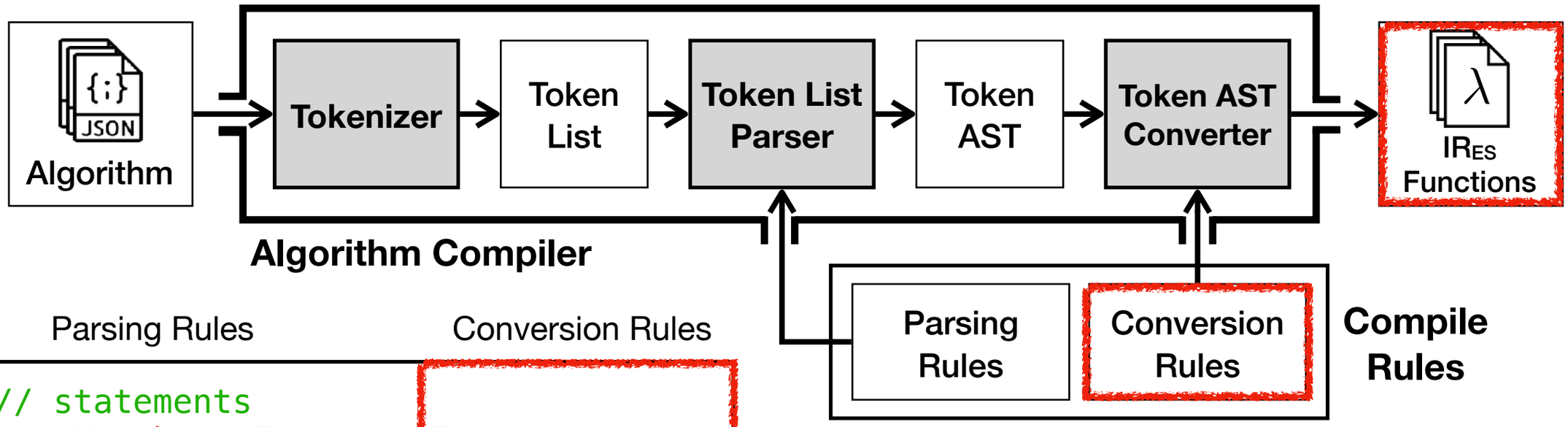
```
ILet

EAbruptCheck |
ECall |
_.toDouble
```

```
ILet(array, EAbruptCheck(
                          ))
```

Simplified compile rules

```
[                                                    ]
```






Parsing Rules

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

```
E = // expressions
  ! E                  ^^
  str ~ ( ~ E ~ )      ^^
  num                  ^^
```

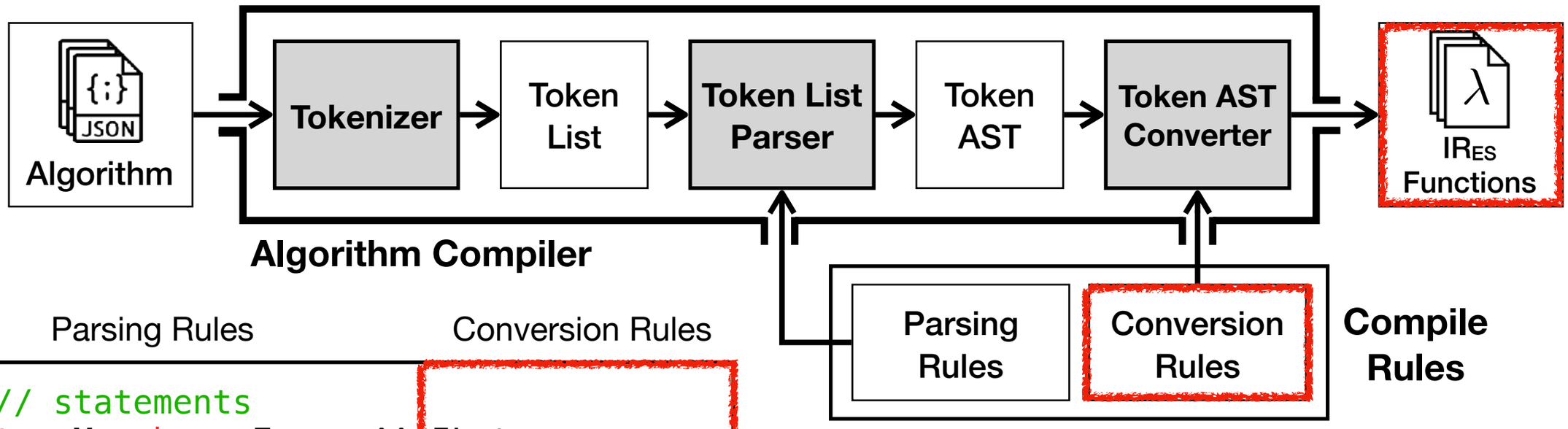
Conversion Rules

```
ILet

EAbruptCheck |
ECall |
_.toDouble
```

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

Simplified compile rules

[ **E** | **num** ⋮ 0 ]




Parsing Rules

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

Conversion Rules

```
ILet
```

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

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

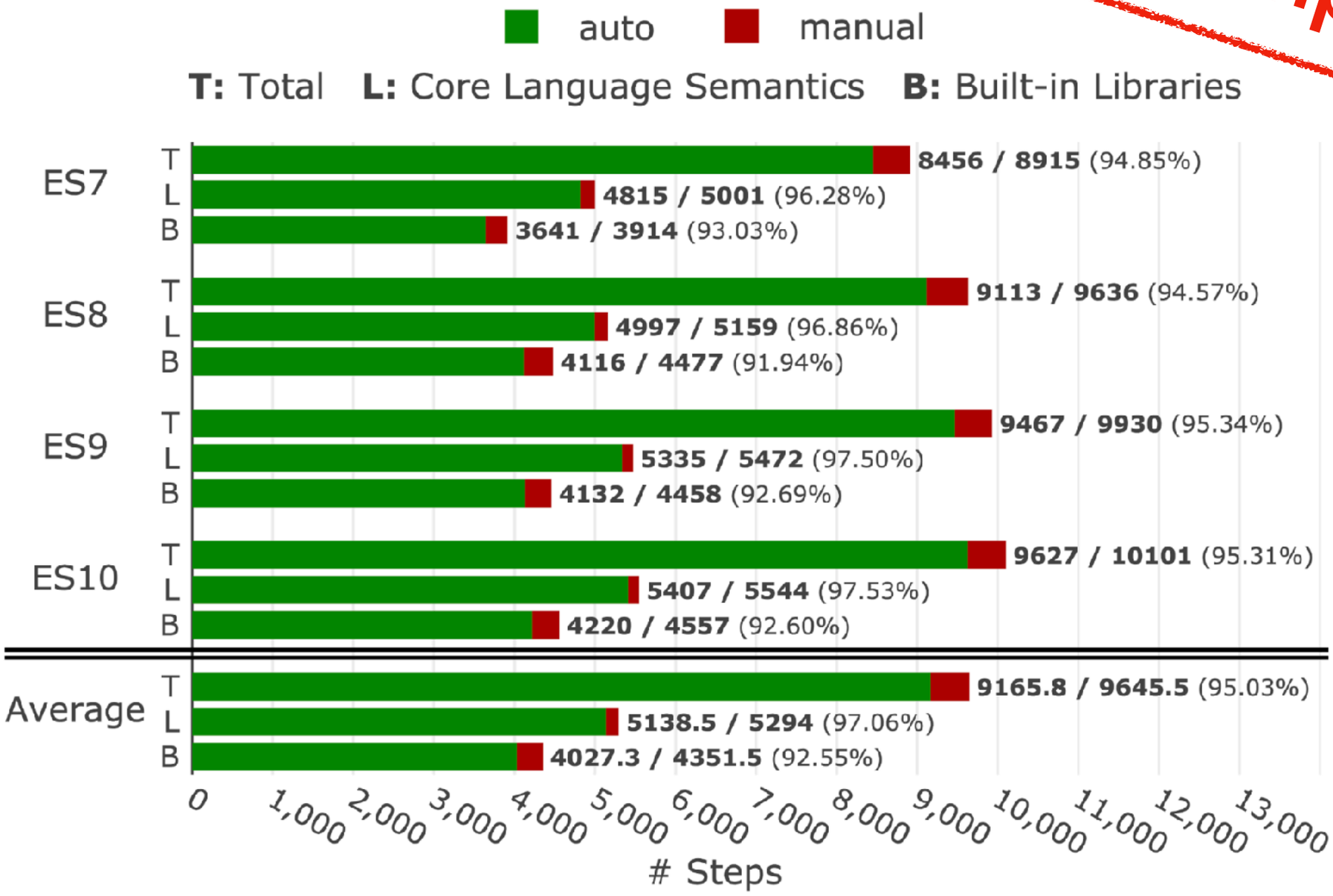
Simplified compile rules

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

# Evaluation - Semantics

| Name | Stmt | Expr | Cond | Value | Ty | Ref |
|---|---|---|---|---|---|---|
| # Rules | 21 | 27 | 16 | 11 | 34 | 9 |

≈ 95% Compiled



# Evaluation - Semantics

All Passed

- **Test262** - Official ECMAScript test suite

| | | | |
|---|---|---|---|
| **ES10** | **16,355 / 18,064 (1,709 failed tests)** | → | **18,064 / 18,064 (all passed)** |
| **ES.Next** | **292 / 303 (11 failed tests)** | → | **303 / 303 (all passed)** |

**9 spec. errors in ES10**

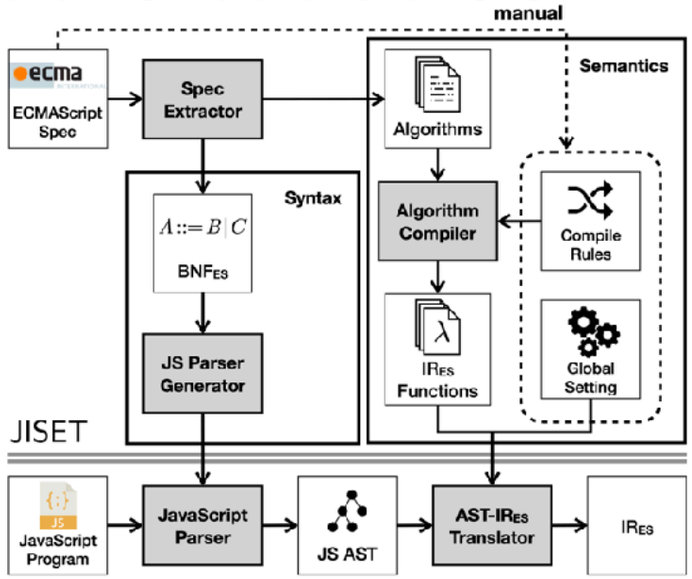
**3 spec. errors in ES.Next**

| Name | Feature | Description | Known | Created | Resolved | Existed | # Fails |
|---|---|---|---|---|---|---|---|
| ES10-1 | Iteration | Missing the `async-iterate` case in the assertion of **ForIn/OfHeadEvaluation** | X | 2018-02-16 | 2020-03-25 | 768 days | 1,116 |
| ES10-2 | Condition | Ambiguous grammar production for the dangling `else` problem in *IfStatement* | X | 2015-06-01 | TBD | TBD | 1 |
| ES10-3 | String | Wrong use of the = operator in **StringGetOwnProperty** | X | 2015-06-01 | 2020-05-07 | 1,802 days | 7 |
| ES10-4 | Completion | Unhandling abrupt completion in **Abstract Equality Comparison** | X | 2015-06-01 | 2020-04-28 | 1,793 days | 9 |
| ES10-5 | Completion | Unhandling abrupt completion in **Evaluation** of *EqualityExpression* | O | 2015-06-01 | 2019-05-02 | 1,431 days | 2 |
| ES10-6 | Await | Passing a value of wrong type to the second parameter of **PromiseResolve** | O | 2019-02-27 | 2019-04-13 | 45 days | 1,294 |
| ES10-7 | Function | No semantics of **IsFunctionDefinition** for `function(...){...}` | O | 2015-10-30 | 2020-01-18 | 1,541 days | 306 |
| ES10-8 | Function | No semantics of **ExpectedArgumentCount** for the base case of *FormalParameters* | O | 2016-11-02 | 2020-02-20 | 1,205 days | 81 |
| ES10-9 | Iteration | Two semantics of **VarScopedDeclarations** for `for await(var x of e){...}` | O | 2018-02-16 | 2019-10-11 | 602 days | 0 |
| BigInt-1 | Expression | Using the wrong variable `oldvalue` instead of `oldValue` in **Evaluation** of *UpdateExpression* | X | 2019-09-27 | 2020-04-23 | 209 days | 533 |
| BigInt-2 | Number | Using **ToInt32** instead of **ToUint32** in **Number::unsignedRightShift** | X | 2019-09-27 | 2020-04-23 | 209 days | 2 |
| BigInt-3 | Number | Unhandling BigInt values in the **Number** constructor | O | 2019-09-27 | 2019-11-19 | 53 days | 1 |

# JISET: JavaScript IR-based Semantics Extraction Toolchain

**Jihyeok Park**, Jihee Park, Seungmin An, Sukyoung Ryu - **PLRG @ KAIST**

## Overall Structure of JISET






## Evaluation - Syntax

All Success!!

| Version | ES7 | ES8 | ES9 | ES10 | Average |
|---|---|---|---|---|---|
| # Lexical productions | 78 | 78 | 78 | 81 | 78.75 |
| # Syntactic productions | 157 | 167 | 167 | 174 | 166.25 |

Test with JS programs in Test262

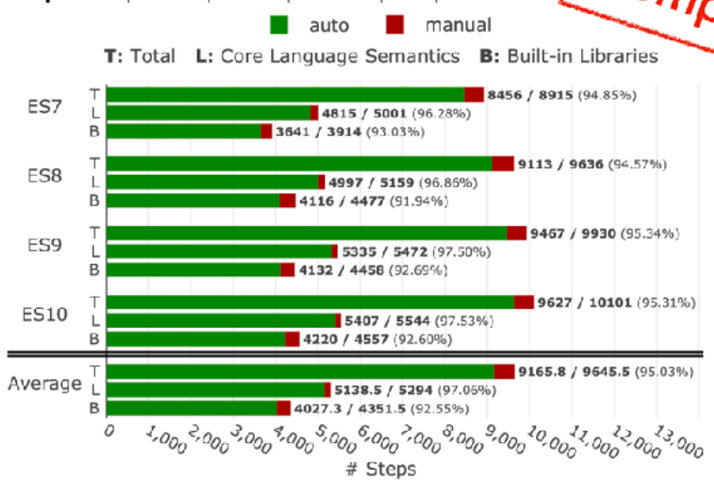
| Old version / New version | ES7 / ES8 | ES8 / ES9 | ES9 / ES10 | Average |
|---|---|---|---|---|
| Δ # Lexical productions | 3 | 5 | 6 | 4.67 |
| Δ # Syntactic productions | 140 | 15 | 8 | 54.33 |



## Evaluation - Semantics

| Name | Stmt | Expr | Cond | Value | Ty | Ref |
|---|---|---|---|---|---|---|
| # Rules | 21 | 27 | 16 | 11 | 34 | 9 |

≈ 95% Compiled






## Evaluation - Semantics

All Passed

- **Test262** - Official ECMAScript test suite

**ES10** 16,355 / 18,064 (1,709 failed tests) → 18,064 / 18,064 (all passed)

**ES.Next** 292 / 303 (11 failed tests) → 303 / 303 (all passed)

9 spec. errors in ES10

3 spec. errors in ES.Next

| Name | Feature | Description | Known | Created | Resolved | Existed | # Fails |
|---|---|---|---|---|---|---|---|
| ES10-1 | Iteration | Missing the `async-iterate` case in the assertion of **ForIn/OfHeadEvaluation** | X | 2018-02-16 | 2020-03-25 | 768 days | 1,116 |
| ES10-2 | Condition | Ambiguous grammar production for the dangling `else` problem in *IfStatement* | X | 2015-06-01 | TBD | TBD | 1 |
| ES10-3 | String | Wrong use of the = operator in **StringGetOwnProperty** | X | 2015-06-01 | 2020-05-07 | 1,802 days | 7 |
| ES10-4 | Completion | Unhandling abrupt completion in **Abstract Equality Comparison** | X | 2015-06-01 | 2020-04-28 | 1,793 days | 9 |
| ES10-5 | Completion | Unhandling abrupt completion in **Evaluation** of *EqualityExpression* | O | 2015-06-01 | 2019-05-02 | 1,431 days | 2 |
| ES10-6 | Await | Passing a value of wrong type to the second parameter of **PromiseResolve** | O | 2019-02-27 | 2019-04-13 | 45 days | 1,294 |
| ES10-7 | Function | No semantics of **IsFunctionDefinition** for `function(...){...}` | O | 2015-10-30 | 2020-01-18 | 1,541 days | 306 |
| ES10-8 | Function | No semantics of **ExpectedArgumentCount** for the base case of *FormalParameters* | O | 2016-11-02 | 2020-02-20 | 1,206 days | 81 |
| ES10-9 | Iteration | Two semantics of **VarScopedDeclarations** for `for await (var x of c) {...}` | O | 2018-02-16 | 2019-10-11 | 602 days | 0 |
| BigInt-1 | Expression | Using the wrong variable cleValue instead of oldValue in **Evaluation** of *UpdateExpression* | X | 2019-09-27 | 2020-04-23 | 209 days | 533 |
| BigInt-2 | Number | Using **ToInt32** instead of **ToUint32** in **Number::unsignedRightShift** | X | 2019-09-27 | 2020-04-23 | 209 days | 2 |
| BigInt-3 | Number | Unhandling BigInt values in the **Number** constructor | O | 2019-09-27 | 2019-11-19 | 53 days | 1 |

# Backup Slides

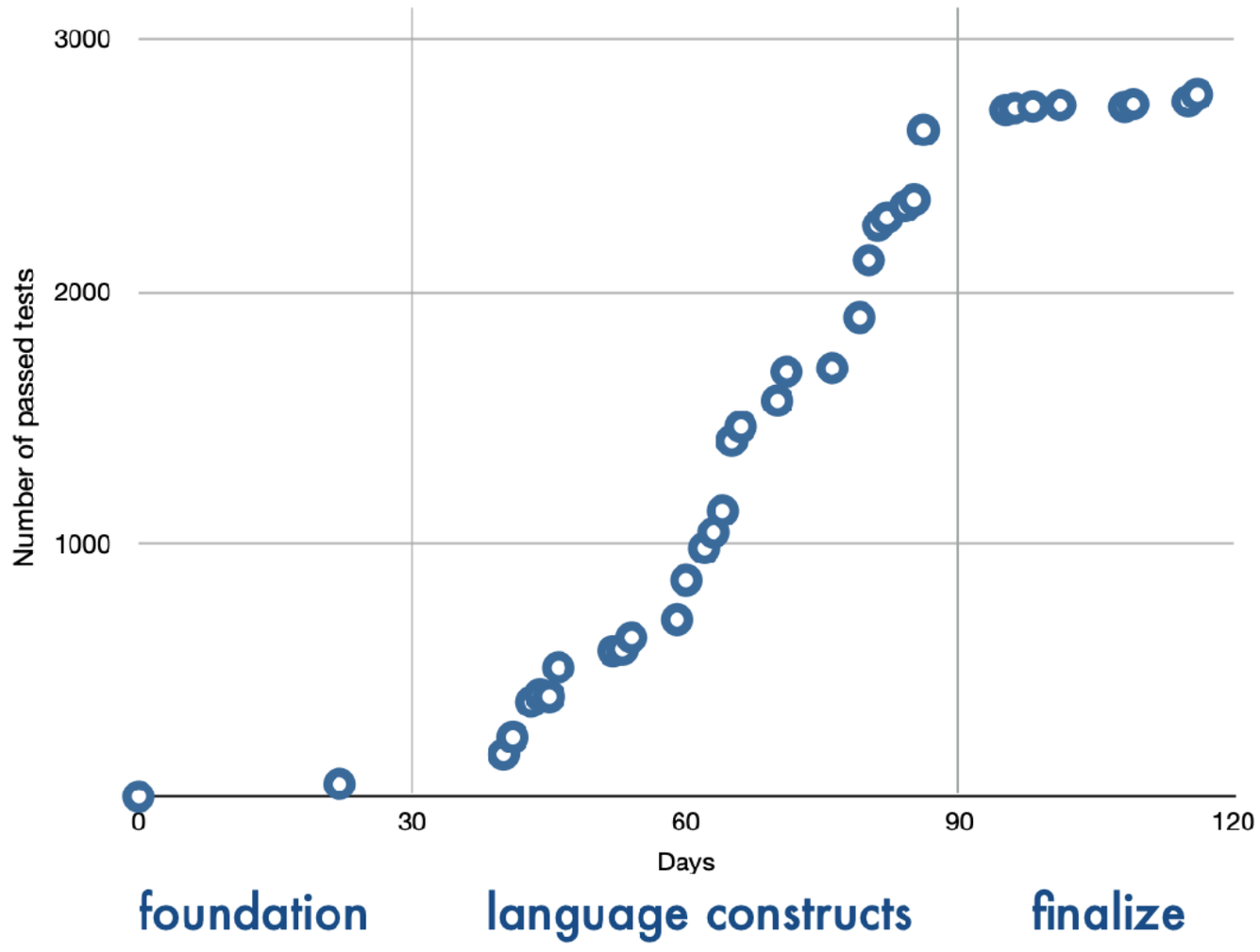
# Size of ECMAScript

| Edition | Date | # Pages |
|---|---|---|
| 1 | 1997/06 | 110 |
| 2 | 1998/06 | 117 |
| 3 | 1999/12 | 188 |
| 5 | 2009/12 | 252 |
| 5.1 | 2011/06 | 258 |
| 6 | 2015/06 | 566 |
| 7 | 2016/06 | 586 |
| 8 | 2017/06 | 885 |
| 9 | 2018/06 | 805 |
| 10 | 2019/06 | 764 |

# Development cost of KJS

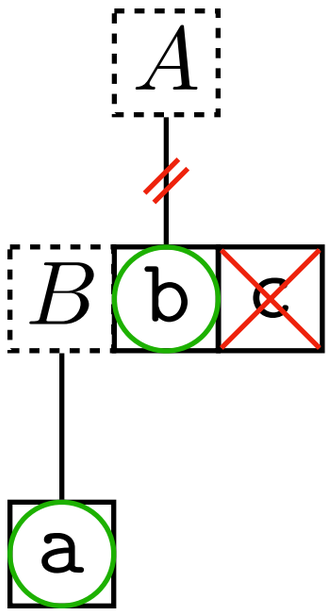
Took only *four months* by a first year PhD student.

# semantic rules: 1,370



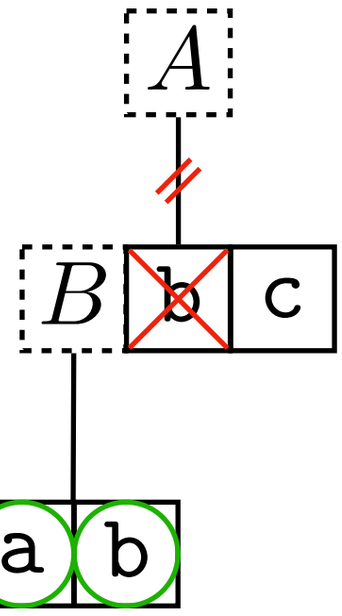
# Parsing Expression Grammar

- **Re-orderings are not always solutions**




$A ::= B\ \texttt{bc}$
$B ::= \texttt{a} \mid \texttt{ab}$

abbc




$A ::= B\ \texttt{bc}$
$B ::= \texttt{ab} \mid \texttt{a}$

abc