JSSpecVis

A JavaScript Language Specification Visualization Tool

Minseok Choe*, Kyungho Song*, Hyunjoon Kim, and Jihyeok Park

Programming Language Research Group @ Korea University

*** Both authors contributed equally to this research.**

JavaScript Language Specification

ECMA-262

- is written in english prose, somewhat like **pseudocode**
- which make **automatic processing and tooling** possible

tc39/**ecma262**

Status, process, and documents for ECMA-262

180 Contributors · 19 Used by · 15k Stars · 1k Forks

es-meta/**esmeta**

ECMAScript Specification (ECMA-262) Metalanguage

19 Contributors · 6 Issues · 226 Stars · 17 Forks

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
      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. …

JavaScript Language Specification is complex

- Evaluation of an addition expression: `□ + □`
  - includes **120+ specification steps**

Let's say you want to know: why `` `!0 + 1n` `` throws **TypeError**?

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*).

Evaluate Left (**lval** = Boolean true)

Evaluate Right (**rval** = BigInt 1n)

Boolean true + BigInt 1n

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
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

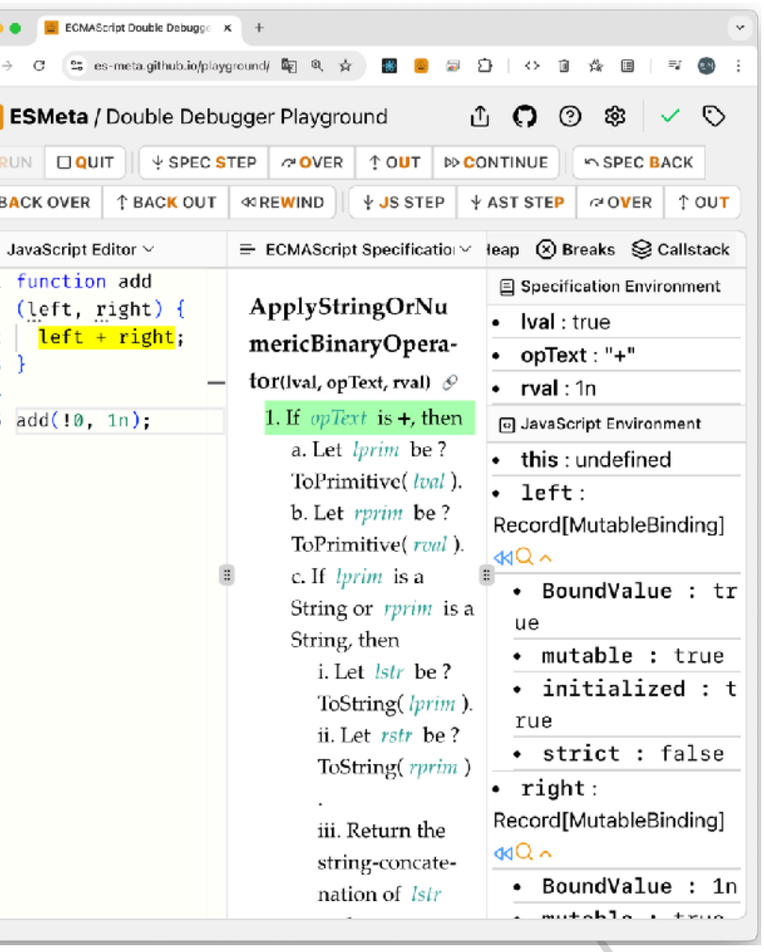
…

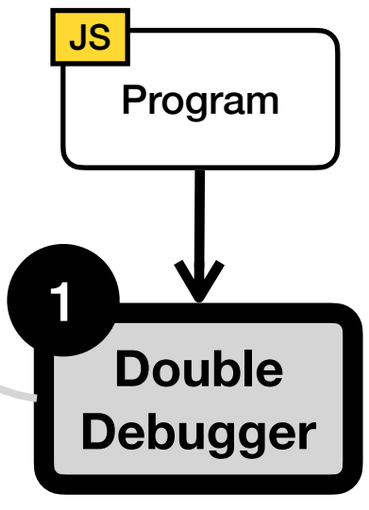
Convert to Primitive
(**lprim** = Boolean true, **rprim** = BigInt 1n)

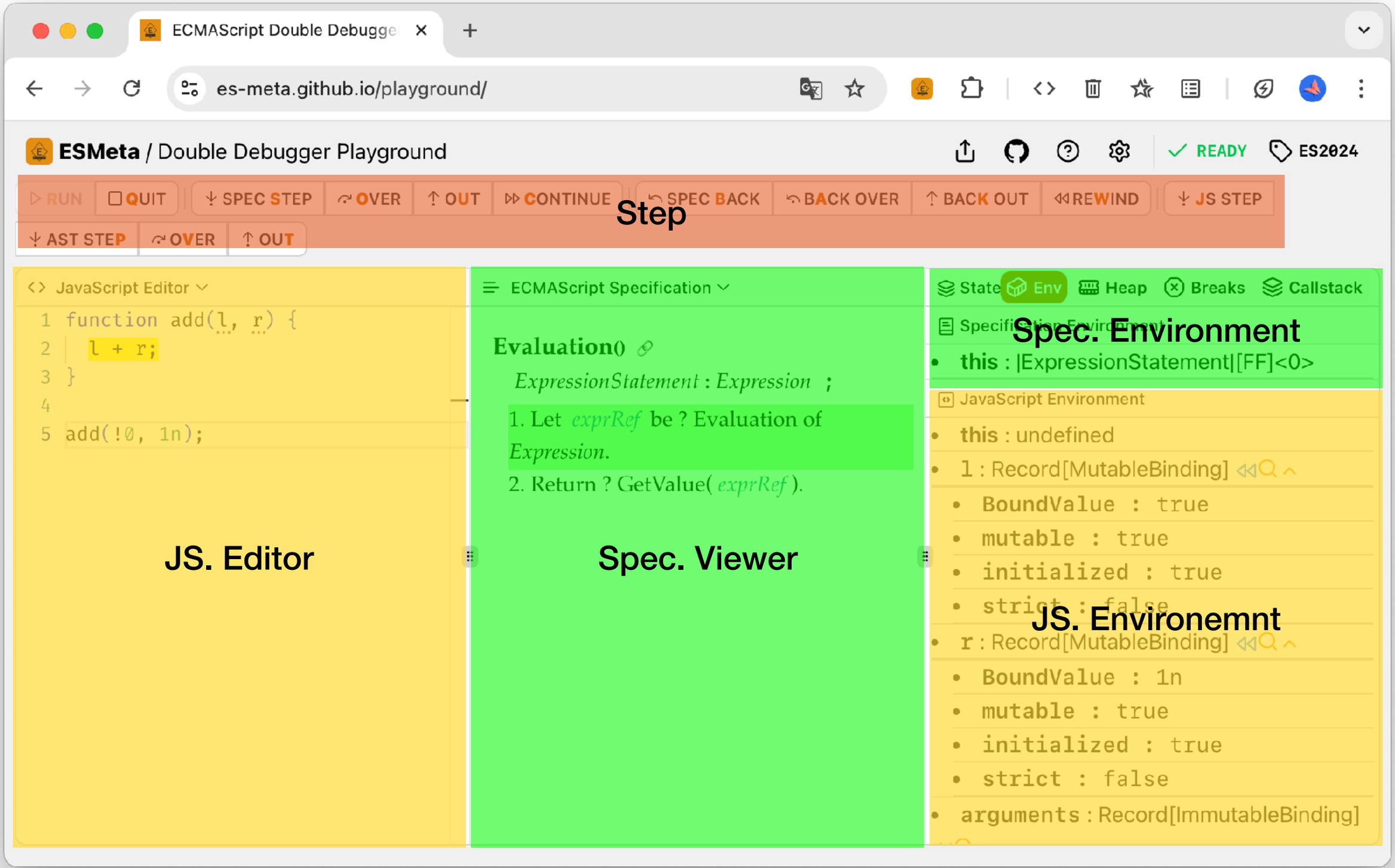
Convert to Numeric
(**lnum** = Number 1, **rnum** = BigInt 1n)

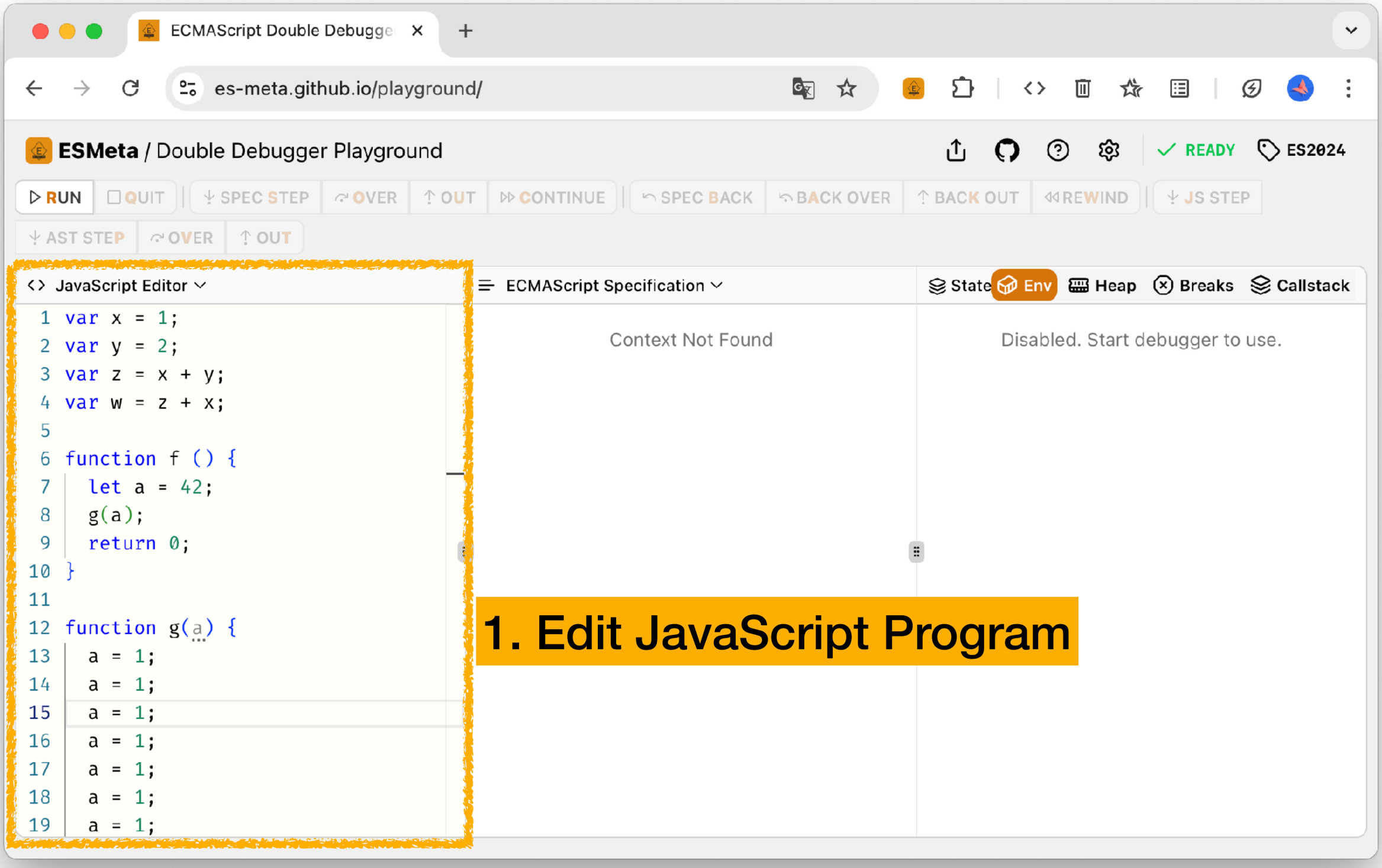
What if we can do this in a more interactive way?

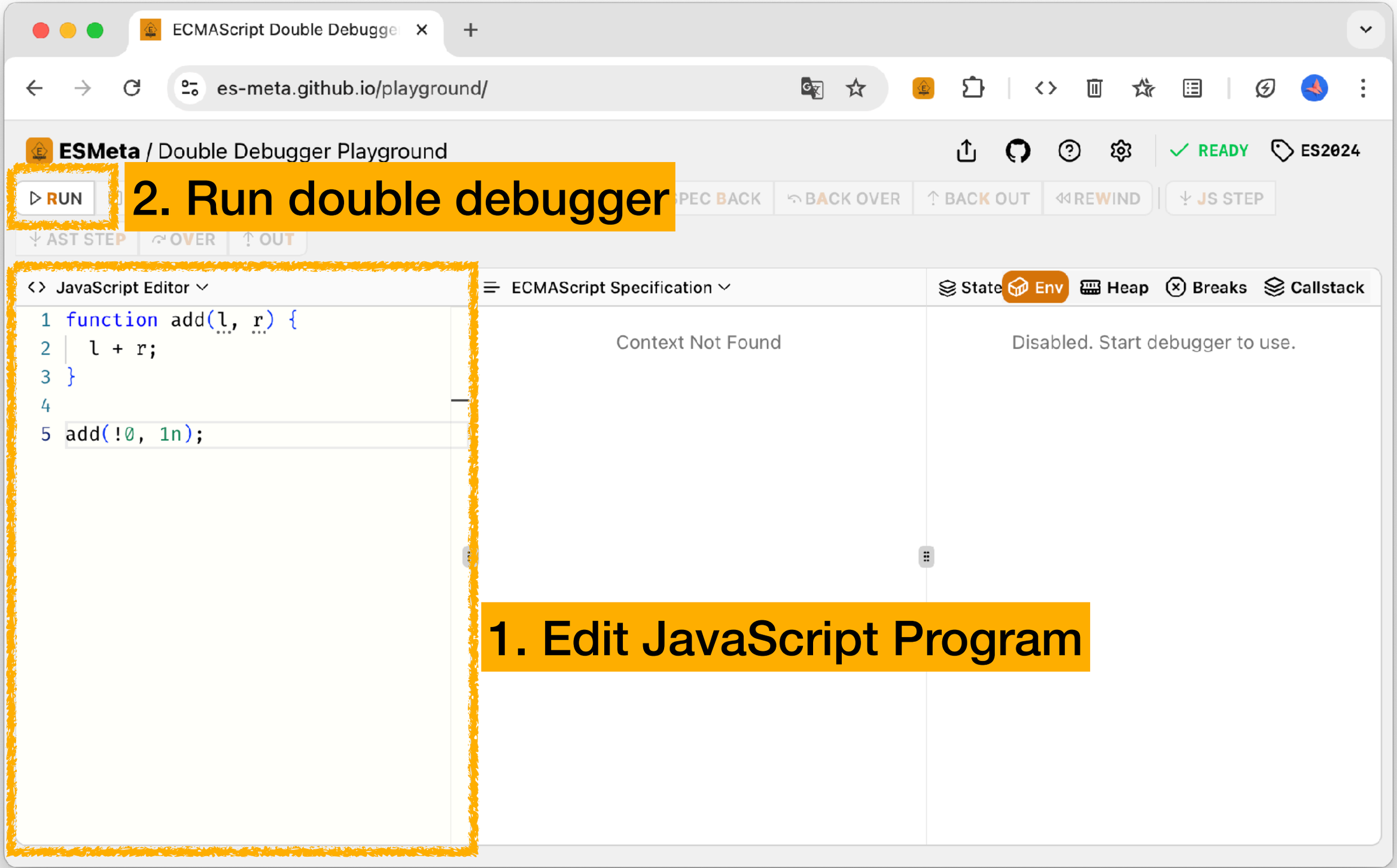
Solution 1. Double Debugger

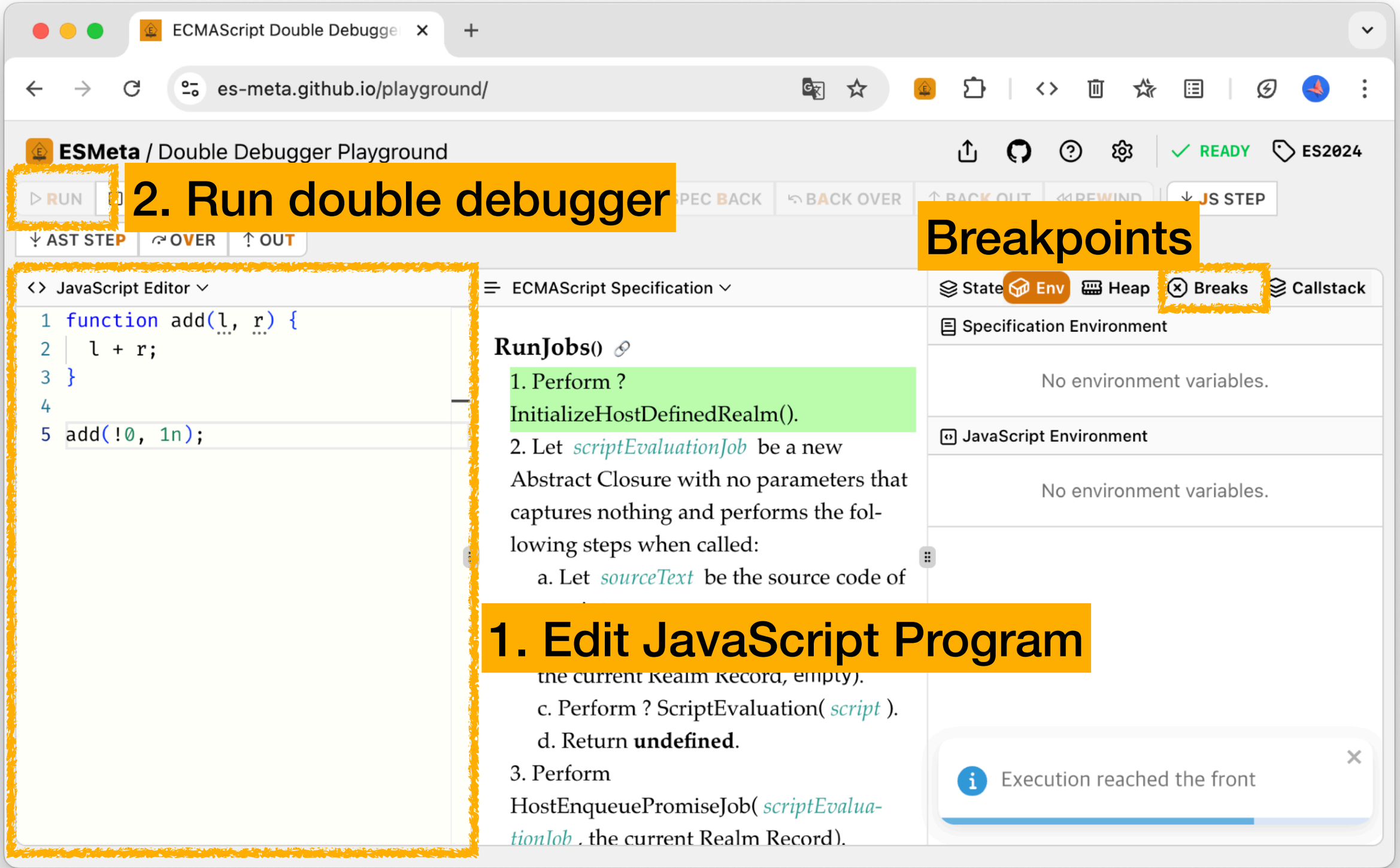


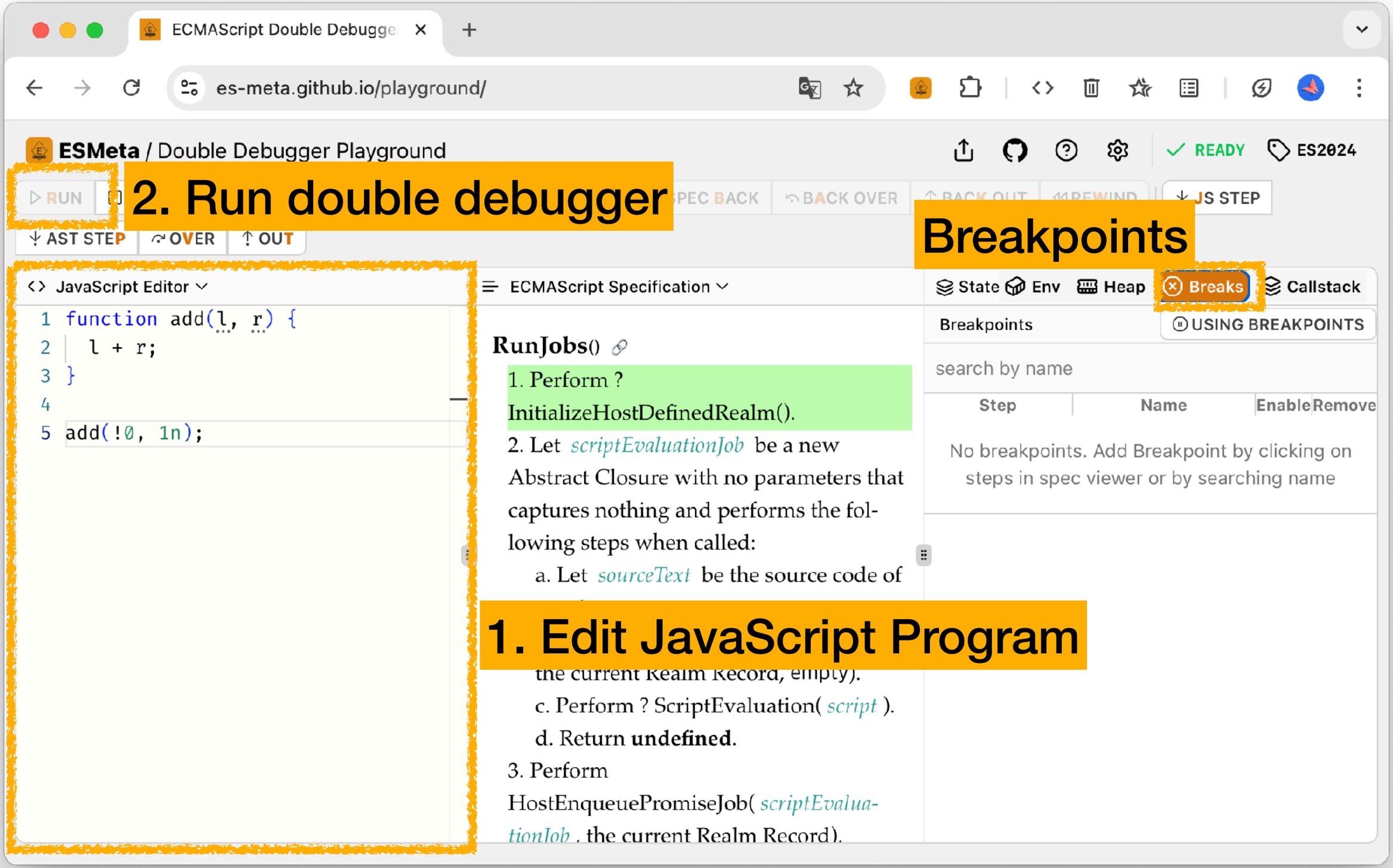


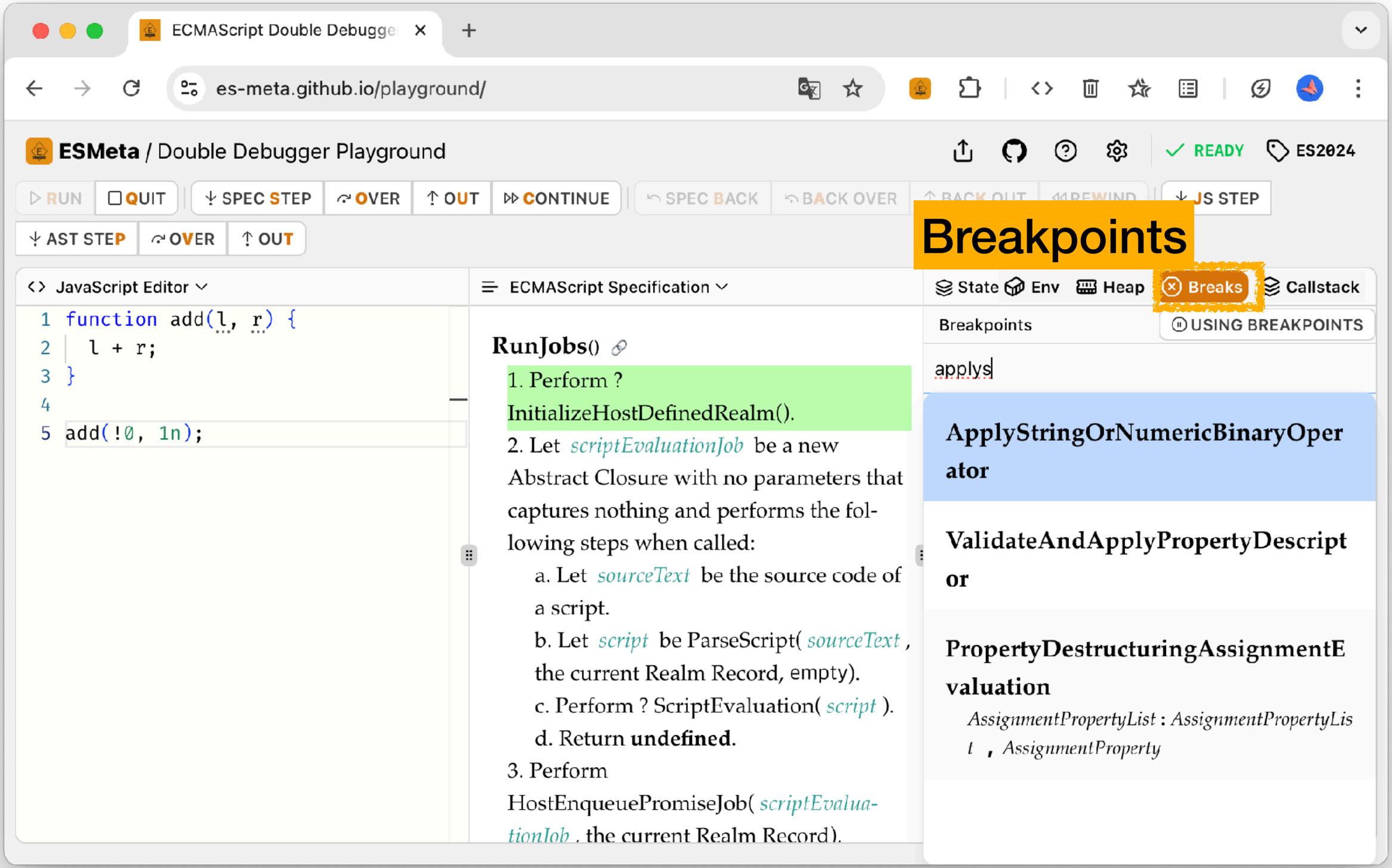

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
Step
JavaScript Editor
function add(l, r) {
l + r;
}
add(!0, 1n);
JS. Editor
ECMAScript Specification
Evaluation()
ExpressionStatement : Expression ;
1. Let exprRef be ? Evaluation of Expression.
2. Return ? GetValue(exprRef).
Spec. Viewer
State
Env
Heap
Breaks
Callstack
Specification Environment
this : |ExpressionStatement|[FF]<0>
Spec. Environment
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]
JS. Environemnt

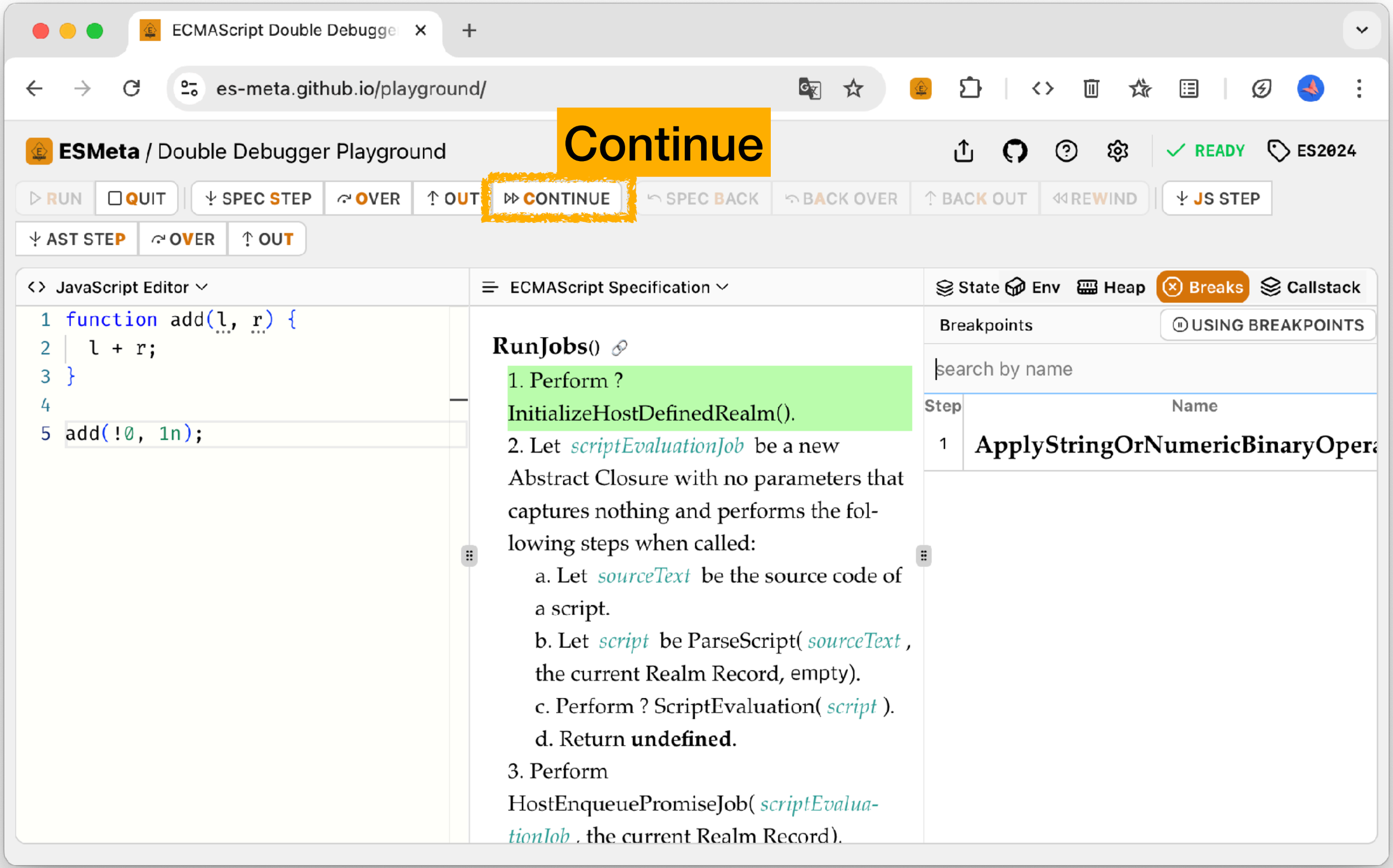

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
ECMAScript Specification
State
Env
Heap
Breaks
Callstack
Context Not Found
Disabled. 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. Edit JavaScript Program

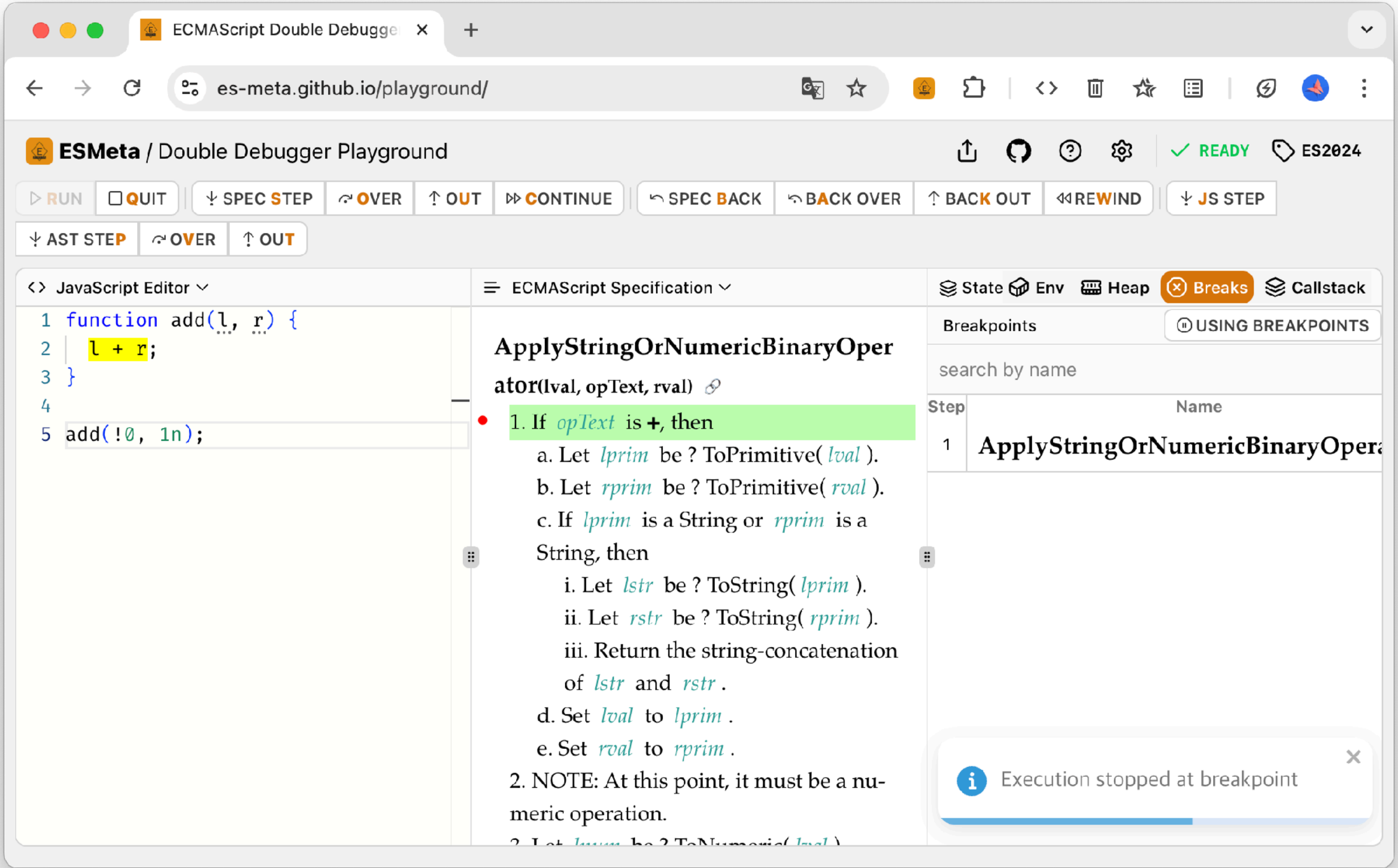

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
2. Run double debugger
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
ECMAScript Specification
State
Env
Heap
Breaks
Callstack
1 function add(l, r) {
2 l + r;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

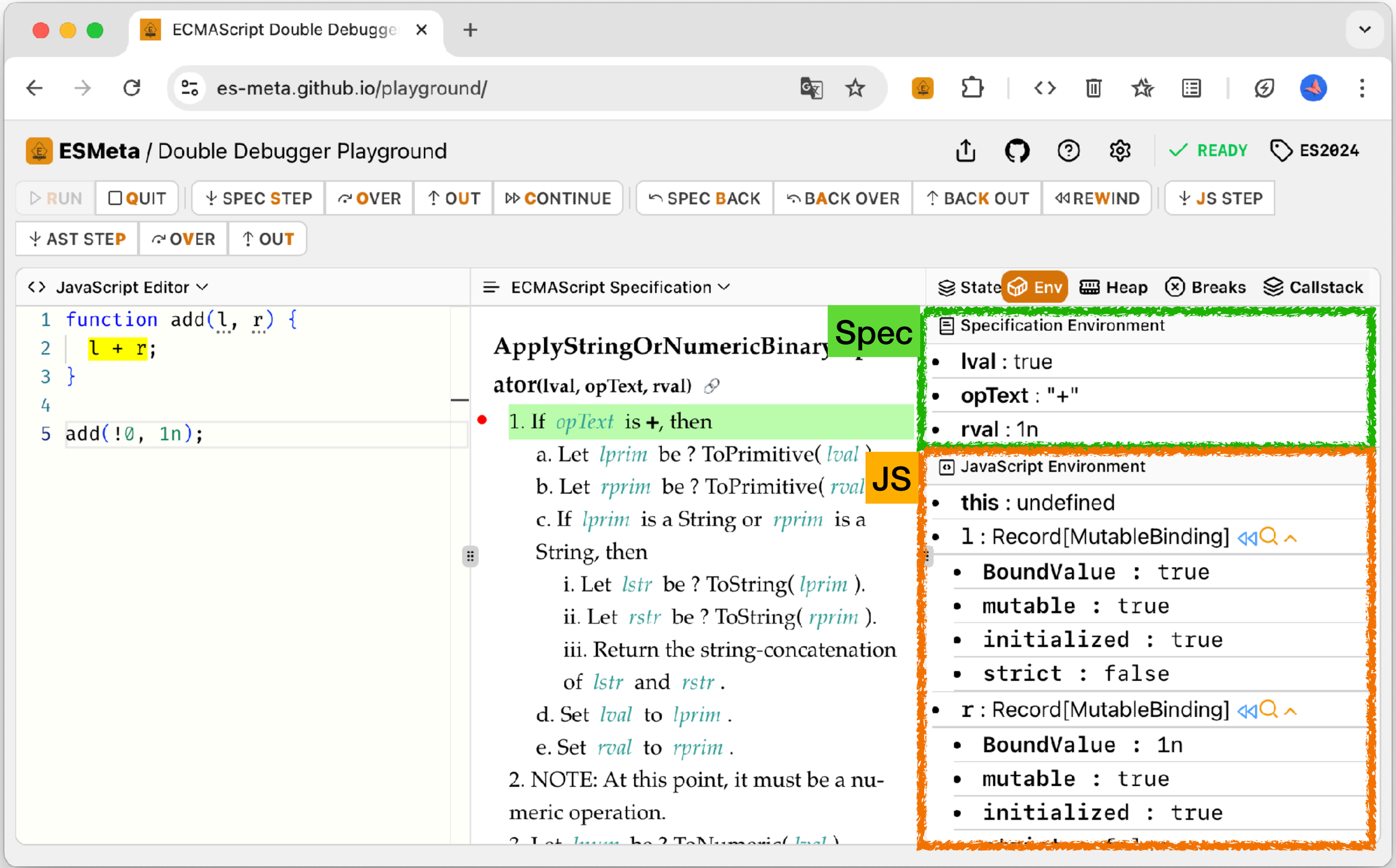

ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
2. Run double debugger
RUN
JS STEP
AST STEP
OVER
OUT
Breakpoints
JavaScript Editor
1 function add(l, r) {
2 l + r;
3 }
4
5 add(!0, 1n);
ECMAScript Specification
RunJobs()
1. Perform ? InitializeHostDefinedRealm().
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
1. Edit JavaScript Program
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.
Execution reached the front


ECMAScript Double Debugge
es-meta.github.io/playground/
ESMeta / Double Debugger Playground
READY
ES2024
RUN
JS STEP
AST STEP
OVER
OUT
2. Run double debugger
Breakpoints
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
1. Edit JavaScript Program
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


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
JS STEP
AST STEP
OVER
OUT
Breakpoints
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 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/
ESMeta / Double Debugger Playground
Continue
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
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
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
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
ApplyStringOrNumericBinary
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.
Spec
JS
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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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.

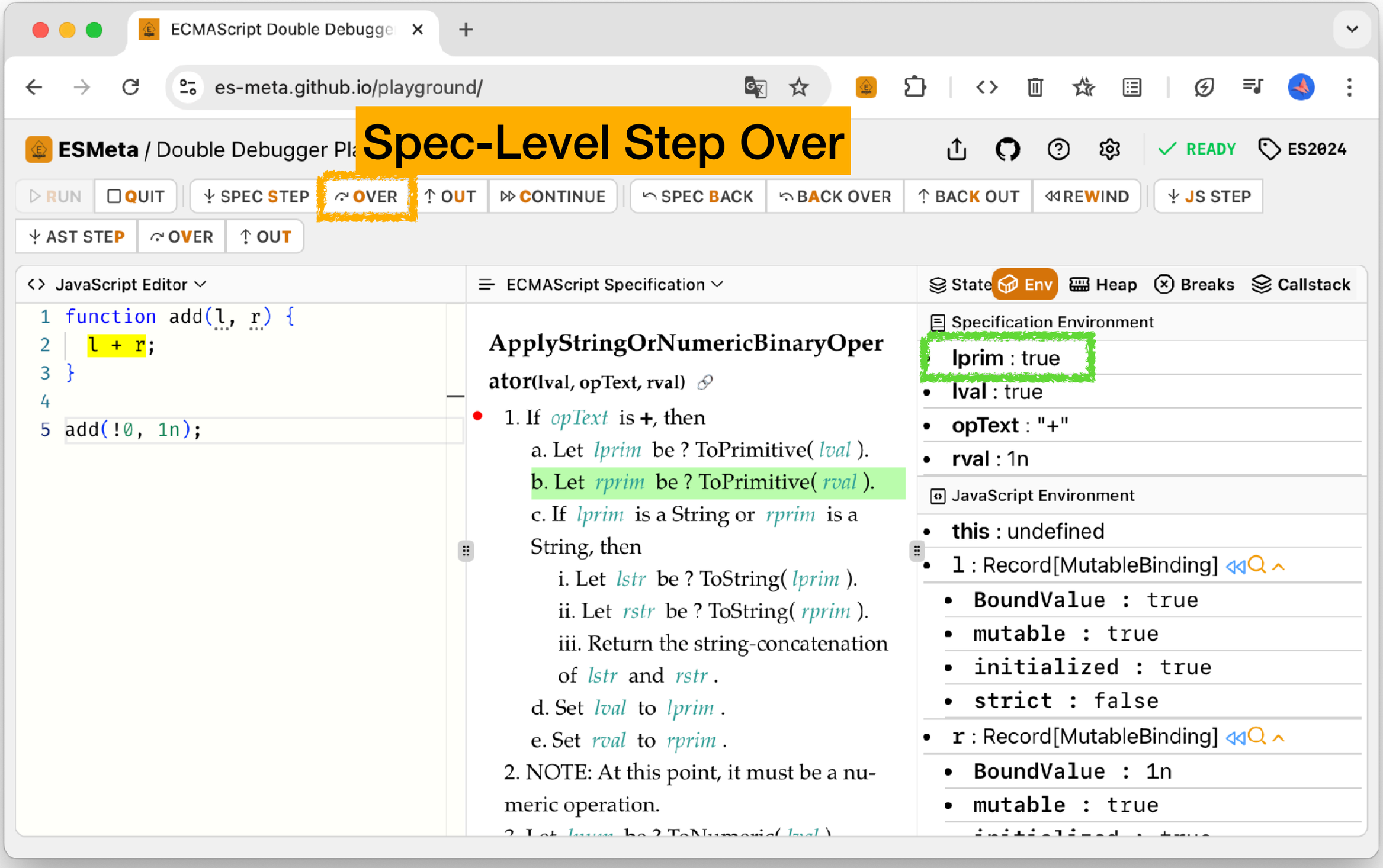
State | Env | Heap | Breaks | Callstack

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/
Spec-Level Step Over
ESMeta / Double Debugger Pl
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 : "+"
rval : 1n
JavaScript Environment
this : undefined
l : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
r : Record[MutableBinding]
BoundValue : 1n
mutable : true

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

Spec-Level Step Over

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

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]

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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

- **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

Spec-Level Step Over

ECMAScript Double Debugge

es-meta.github.io/playground/

ESMeta / Double Debugger Pl

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
- `l` : 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

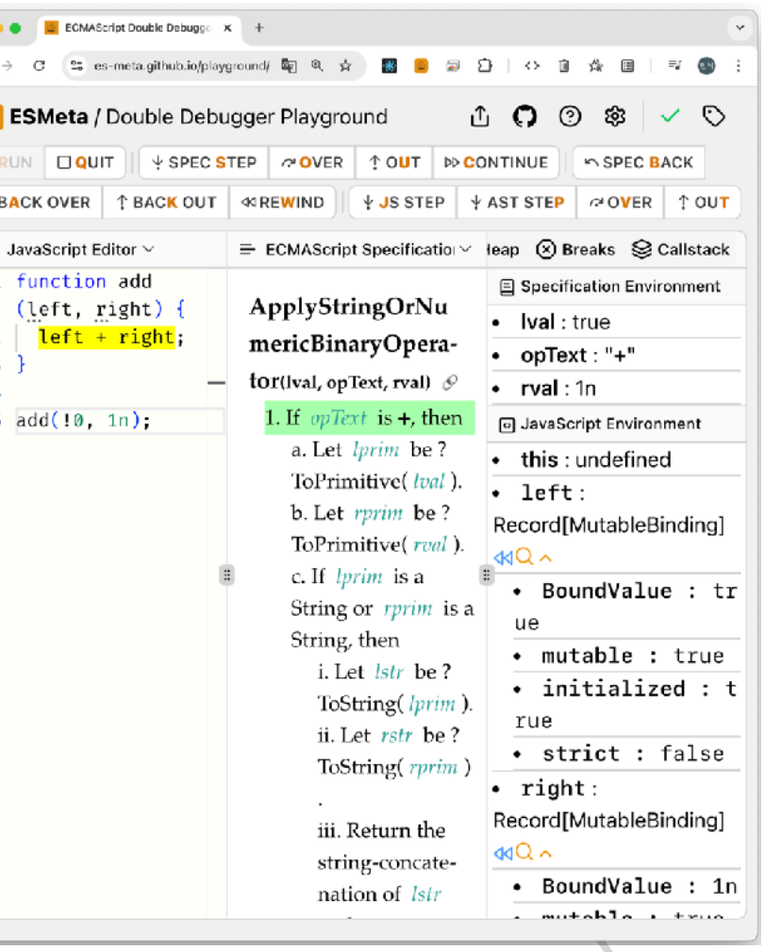
State | Env | Heap | Breaks | Callstack

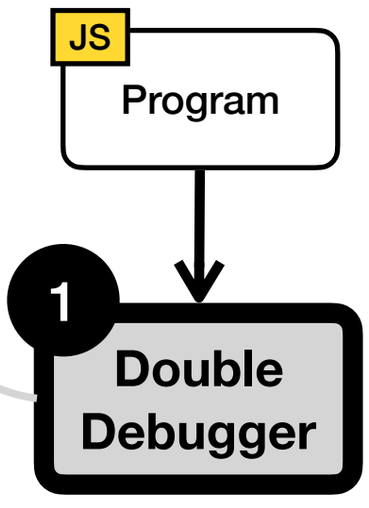
Specification Environment

- **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
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
function add
(left, right) {
left + right;
}
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
Program
1
Double
Debugger

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*).

ApplyStringOrNumeri BinaryOperator (***lval***, ***opText***, ***rval***)

1. If *opText* is +, then
   a. Let *lprim* be ? ToPrim
   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
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

…

return if [[Type]] is **NOT** normal, otherwise, use [[Value]]

syntactic sugar of:

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

Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

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

What if the spec came with example JavaScript programs?

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
      i. Let *lstr* be ? ToString(*lprim*).
      ii. Let *rstr* be ? ToString(*rprim*).
      iii. Return the string-concatenation of *lstr* and *rs*
   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

…

```
1 + 1
```

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

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

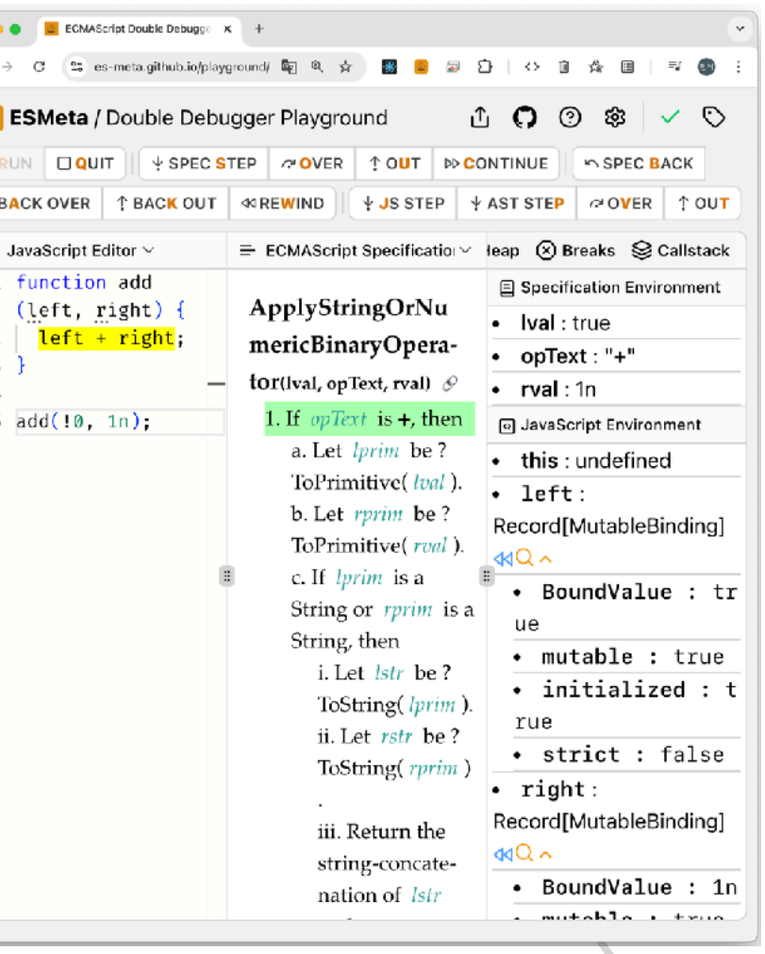
```
[] + 0
```

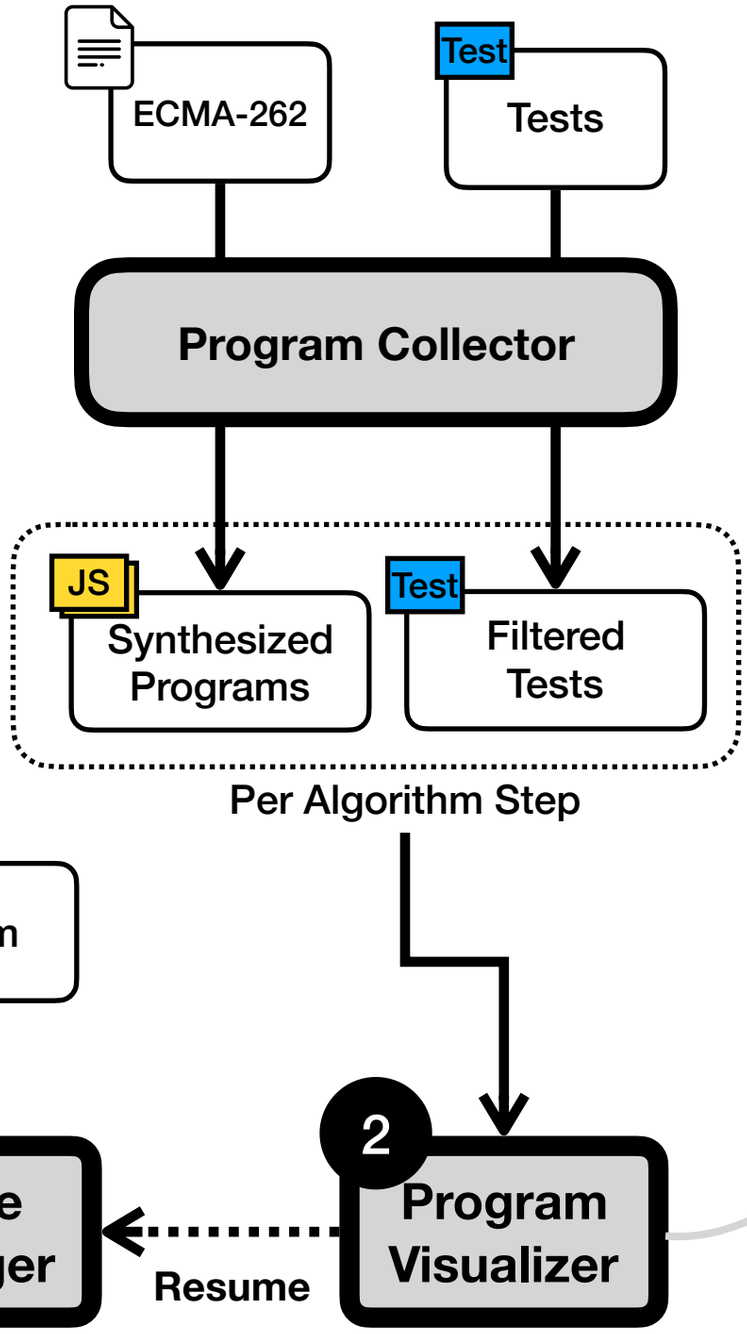
```
{ [Symbol.toPrimitive] : 0 } * 0
```

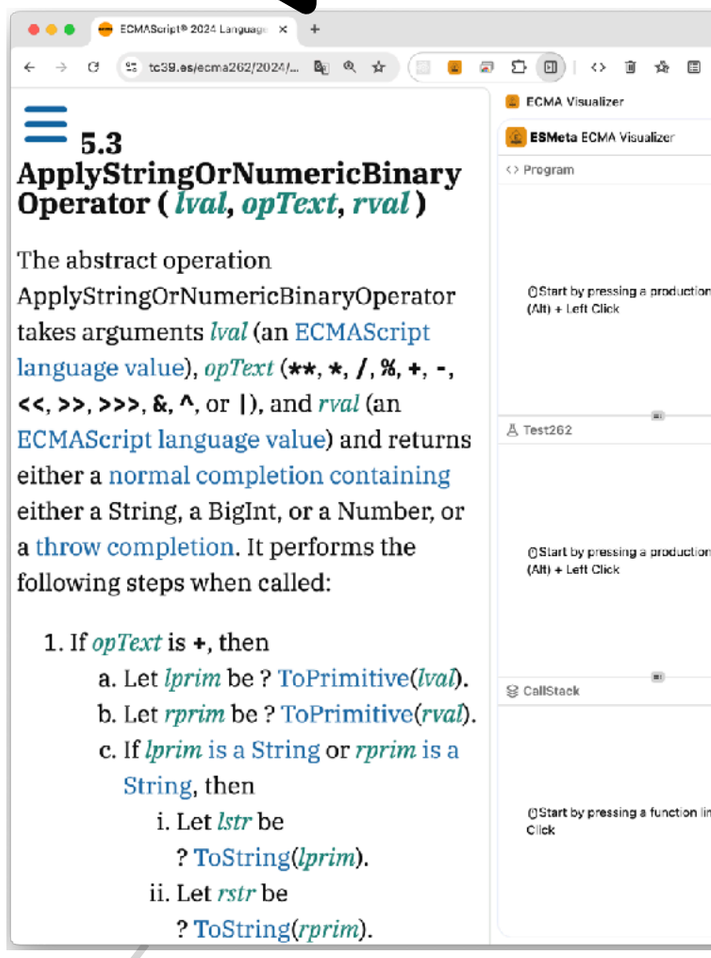
```
0 * { [Symbol.toPrimitive] : 0 }
```

```
0 + 1n
```

Solution 2. Program Visualizer







ECMAScript® 2024 Language

tc39.es/ecma262/2024/multipage/ecmascript-language...

Search...

Table of Contents

13.15.3 ApplyStringOrNu naryOperator (*lval*, *opText*, *rval*)

The abstract operation ApplyStringOrNu perator takes arguments *lval* (an ECMAScript language valu (**, *, /, %, +, -, <<, >>, >>>, &, ^, 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*).

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 ? 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

JS Program

Start by pressing a production with Option (Alt) + Left Click

Test262

Tests

Start by pressing a production with Option (Alt) + Left Click

CallStack

CallStack

Start by pressing a call stack 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 ? 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*).

Option (alt) + Click

ECMA Visualizer

ESMeta ECMA Visualizer

Program — Run on Double Debugger

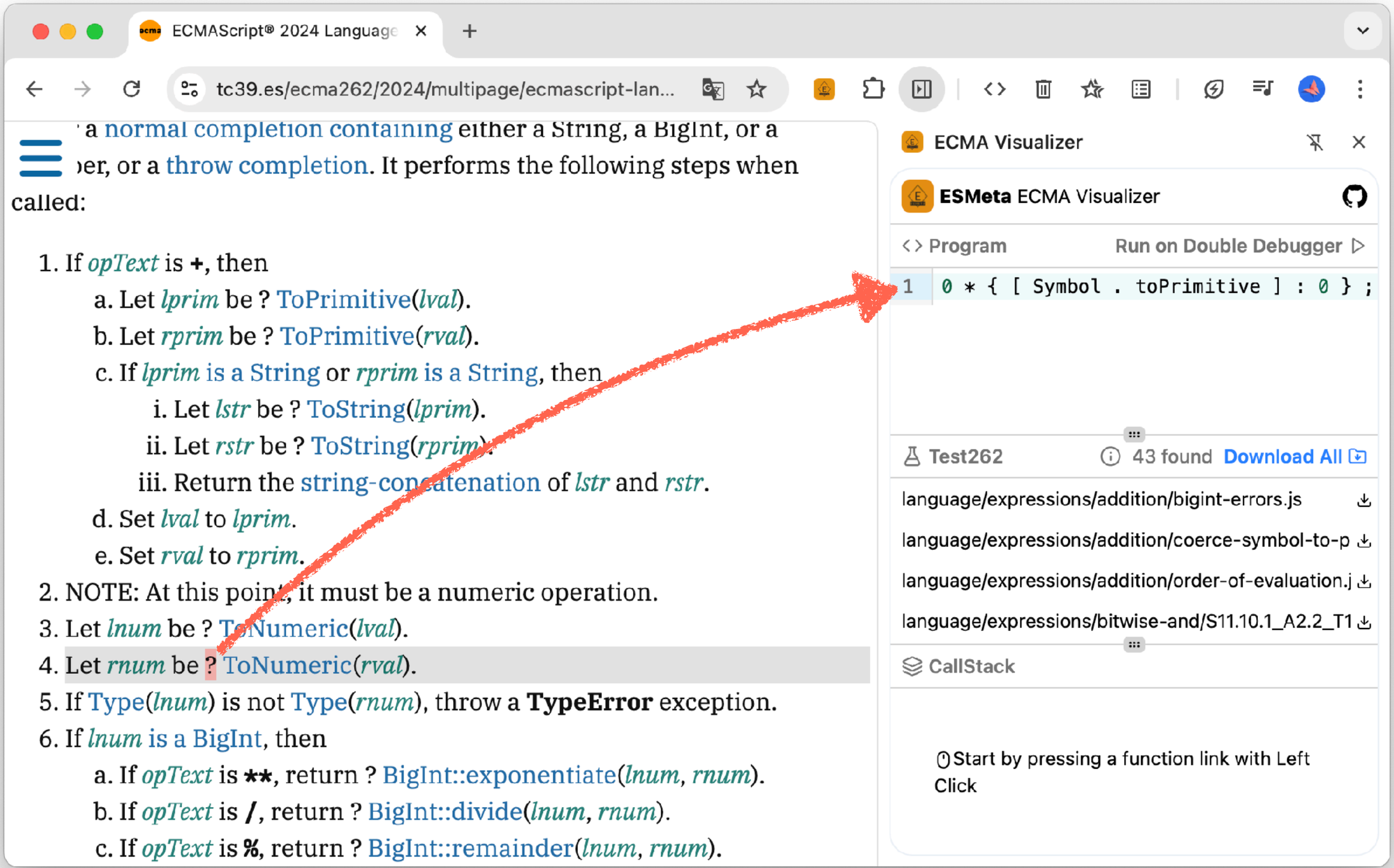
```
1  1 + 1 ;
```

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
)er, 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

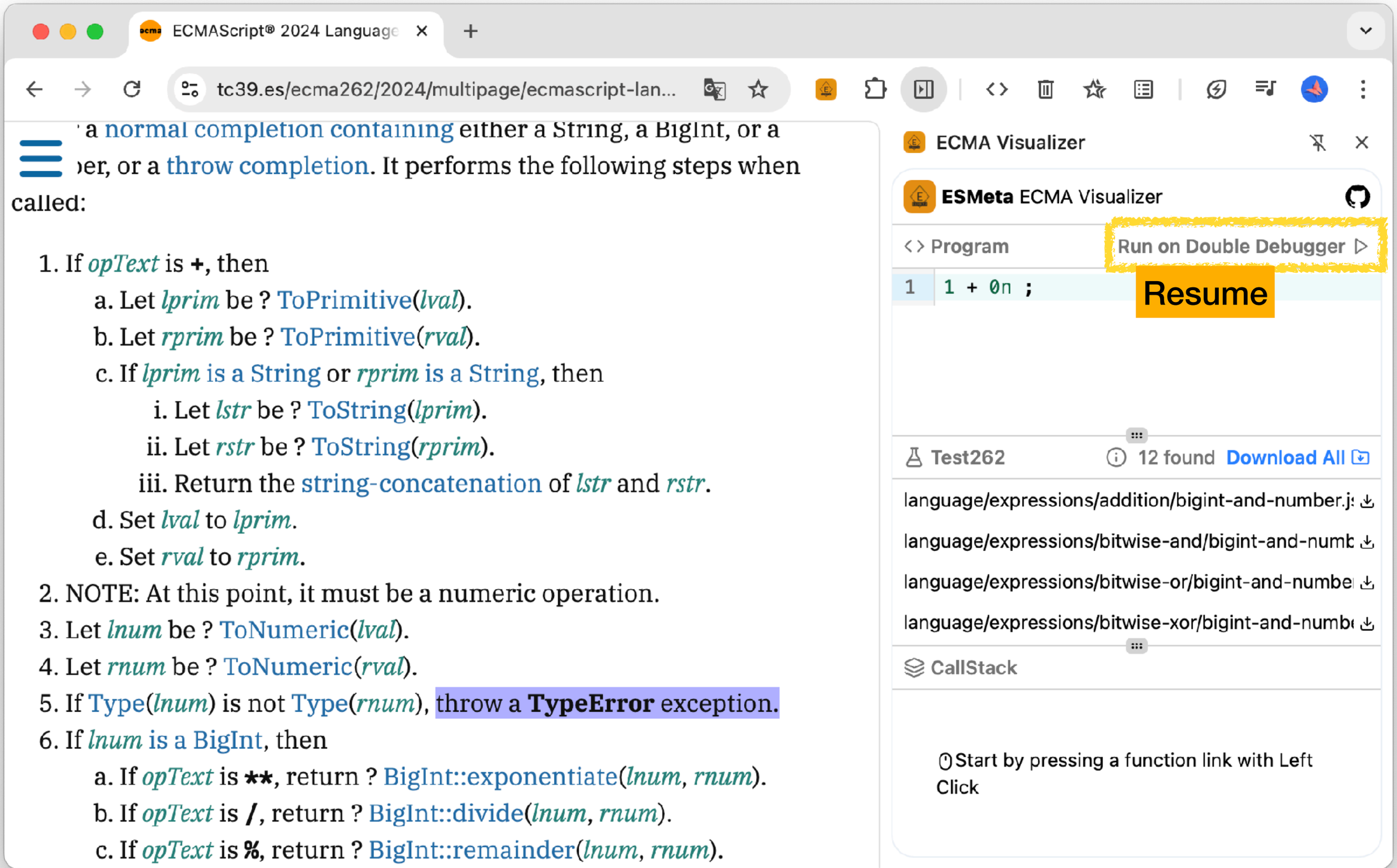
```
1  1 + 0n ;
```

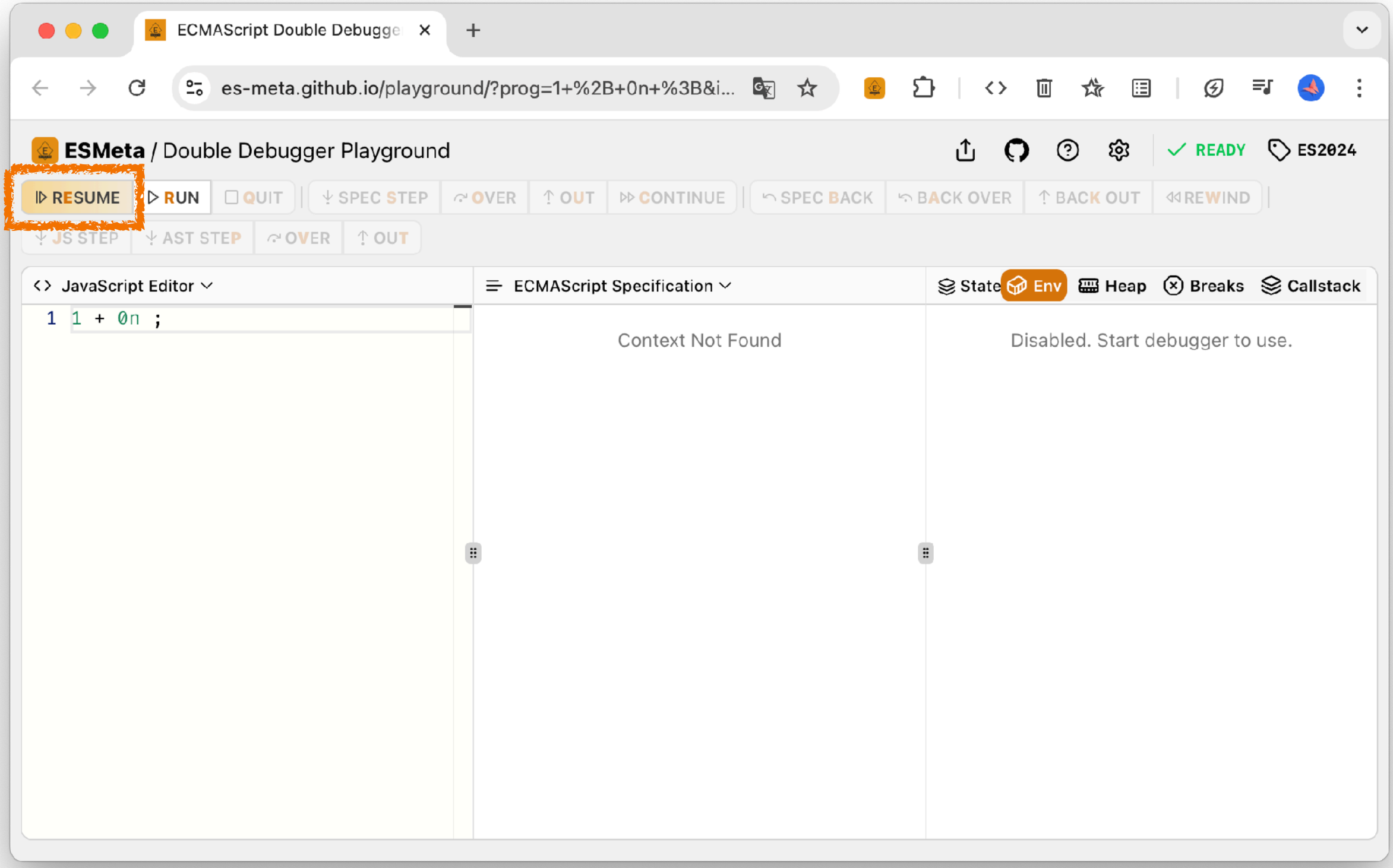
Test262 — 12 found Download All

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 ;
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

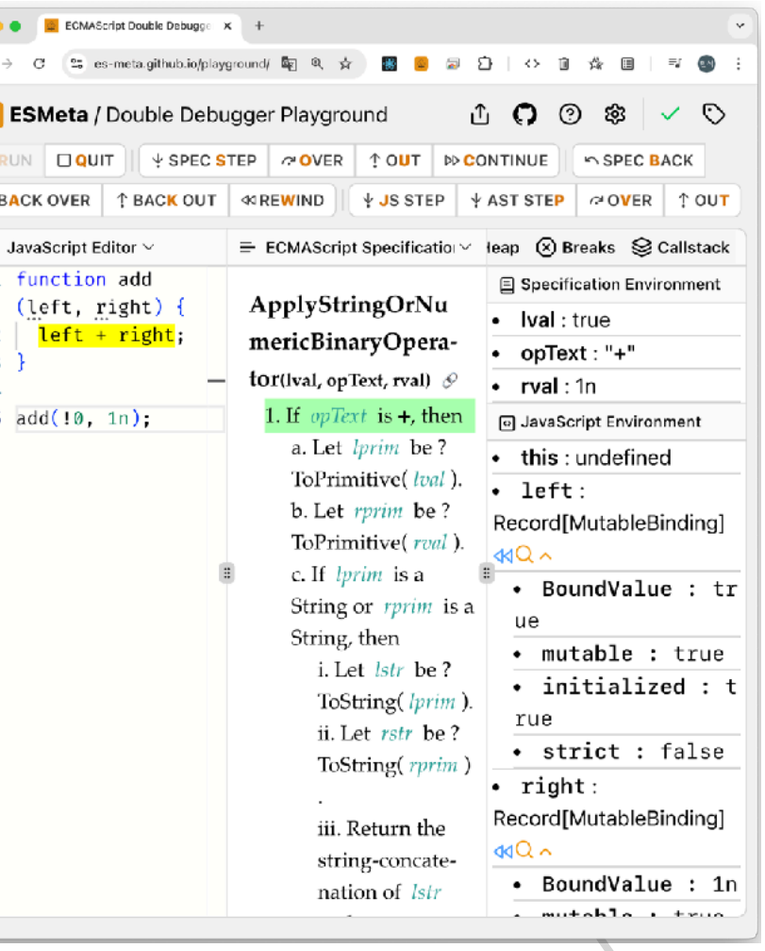
State Env Heap Breaks Callstack

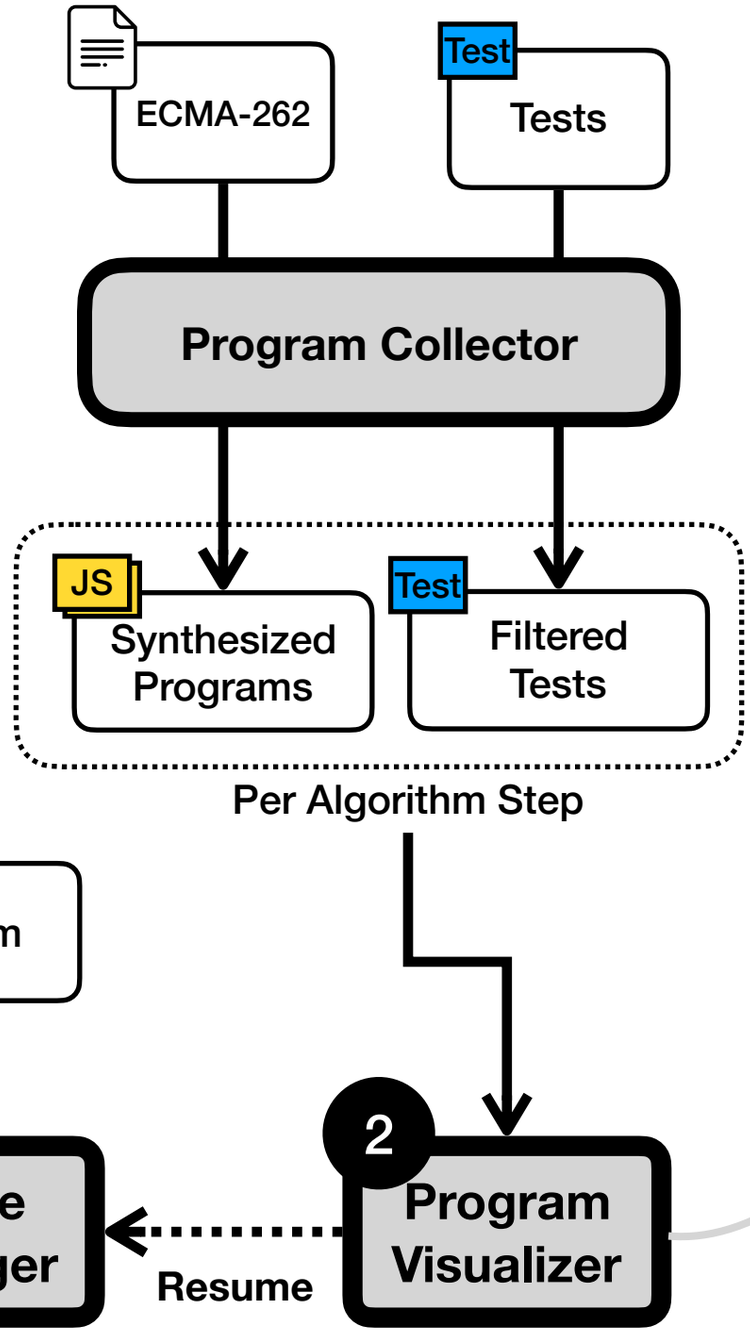
Specification Environment

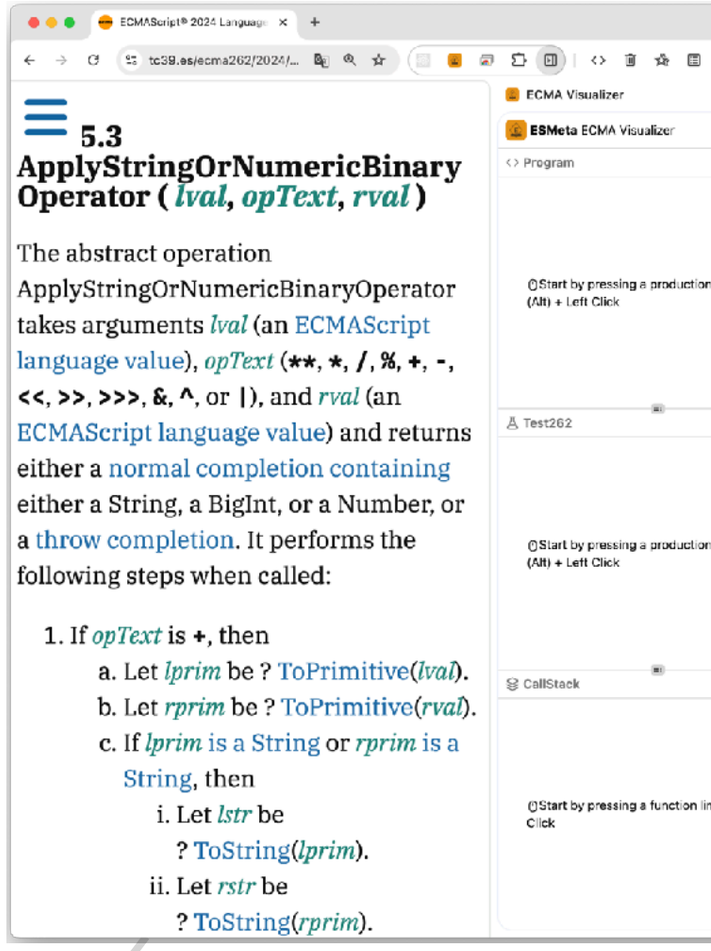
- **lnum** : 1
- **lprim** : 1
- **lval** : 1
- **opText** : "+"
- **rnum** : 0n
- **rprim** : 0n
- **rval** : 0n

JavaScript Environment

- **AggregateError** : Record[PropertyDescriptor]
  - **Value** : Record[BuiltinFunctionObject]
  - **Writable** : true
  - **Enumerable** : false


ESMeta / Double Debugger Playground
QUIT
SPEC STEP
OVER
OUT
CONTINUE
SPEC BACK
BACK OVER
BACK OUT
REWIND
JS STEP
AST STEP
OVER
OUT
JavaScript Editor
ECMAScript Specification
Breaks
Callstack
function add
(left, right) {
left + right;
}
add(!0, 1n);
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
Specification Environment
lval : true
opText : "+"
rval : 1n
JavaScript Environment
this : undefined
left : Record[MutableBinding]
BoundValue : true
mutable : true
initialized : true
strict : false
right : Record[MutableBinding]
BoundValue : 1n


ECMA-262
Test
Tests
Program Collector
JS
Synthesized Programs
Test
Filtered Tests
Per Algorithm Step
JS
Program
1
Double Debugger
Resume
2
Program Visualizer


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).
ECMA Visualizer
ESMeta ECMA Visualizer
Program
Test262
CallStack